SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 1–35, January 1991
© 1991 Society for Industrial and Applied Mathematics

LIMITS ON PARALLELISM IN THE NUMERICAL SOLUTION OF LINEAR PARTIAL DIFFERENTIAL EQUATIONS*

PATRICK H. WORLEY†

Abstract. The problem considered is that of approximating the solution of a linear scalar partial differential equation (PDE) at one or more locations in its domain. A lower bound on the amount of data required to satisfy a given error tolerance in the approximation is described. Using this bound, a lower bound on the execution time of parallel algorithms that approximate the solution is derived. The lower bound on the execution time has the form $\alpha \cdot f_{(+)} \cdot \log_2 \epsilon^{-1}$, where α is a problem-dependent constant, $f_{(+)}$ is a measure of the speed of floating point arithmetic, and ϵ is an upper bound on the error. Thus, when $\alpha > 0$, the execution time increases as ϵ decreases, independent of the number of processors, the interconnection topology, and the algorithm used. Lower bounds on the execution time are also given for the cases where the interconnection network or the number of processors is specified.

Recent research has established that it is often possible to use a large number of processors efficiently when calculating the numerical solution of a PDE if the problem is sufficiently large. In this paper, it is shown that increasing the size of such a problem will usually come at the cost of increasing the execution time. Two examples are described that verify this conclusion, an algorithm-independent analysis of an elliptic PDE and an analysis of a specific algorithm for the approximation of a hyperbolic PDE.

Key words. limits on parallelism, algorithm-independent complexity bounds, linear partial differential equations, numerical approximation, parallel complexity, parallel execution time, problem scaling

AMS(MOS) subject classifications. 65M, 65N, 65W, 68Q

1. Introduction.

Multiprocessors with a moderate number of processors (i.e., fewer than 100) have proven to be cost effective computer architectures for solving many of the computationally intensive problems in scientific computing [13], [14]. There has been some skepticism as to whether multiprocessors with significantly more processors will be as useful. There is a law of diminishing returns associated with using increasingly many processors to solve a fixed size problem. For example, a simple model of such behavior for a fixed algorithm is described by Amdahl in [2] and [3]. The recent empirical results of Gustafson, Montry, and Benner [12] indicate that the efficient use of an increasing number of processors is possible if the size of the problem can be increased as well. They observed that, when the size of the problem increases, the amount of parallel work in many algorithms grows faster than both the amount of serial work and the amount of overhead in exploiting parallelism. This is an important observation. Often the size of a problem that computational scientists want to solve is too large to be computed in a reasonable length of time on the current generation of computers. Instead, they solve the largest size problem that they can afford. Thus, given more processors, it is reasonable to consider solving a larger problem if it can be calculated in a specified amount of time. We will prove that this is not always possible.

We begin by describing information theoretic lower bounds on the execution time of an algorithm. From these results we prove that the execution time will often grow

* Received by the editors October 18, 1988; accepted for publication (in revised form) November 22, 1989. This research was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc.

† Mathematical Sciences Section, Oak Ridge National Laboratory, P.O. Box 2009, Oak Ridge, Tennessee 37831-8083.

without bound as the size of the problem increases, independent of the number of processors and of the algorithm used. This implies that increasing the problem size so as to make use of an increasing number of processors will eventually increase the execution time. These bounds are calculated for the problem of numerically approximating the solution of a linear scalar partial differential equation (PDE) that satisfies certain conditions, but the argument can be applied to other classes of problems as well. To prove the results we also require the assumption that the problem size is allowed to grow only if the solution to the problem is improved sufficiently by doing so. We contend that this assumption is the only reasonable one to make.

In §2 we describe the multiprocessor and algorithm models we use to establish the bounds. We then derive simple lower bounds on the execution time of an algorithm that are a function of how the data are used. In §3 we describe the problem assumptions. From these we derive algorithm-independent lower bounds on how the execution time grows as a function of the error in the approximation to the solution of the PDE. We then describe assumptions on how the size of the problem grows, and conclude that the execution time will usually grow as the problem grows. In §4 we calculate the lower bounds described in §3 for a model problem. An analysis of an example algorithm motivated by the work of Gustafson, Montry, and Benner [12] is also presented to illustrate the practical implications of this work.

2. Information theoretic lower bounds. Our focus in this paper is on MIMD multiprocessors[1] and on modelling parallelism at the level of concurrent execution of floating point operations. This bias is reflected in the following multiprocessor and algorithm models. Most of the assumptions are made merely to provide a structure upon which to base the proofs. Other assumptions would serve just as well.

2.1. Multiprocessor assumptions.

2.1.1. Multiprocessor model. We model a multiprocessor as a directed graph (V, E). Each vertex $v_i \in V$ represents a serial processor.[2] Each edge $e_j \in E$ represents a unidirectional communication channel in the multiprocessor. We assume that all floating point operations are computed by the composition of operators from some given set of *primitive* binary and unary floating point operators, and that the execution of a primitive operator is not spread over multiple processors. We also assume that addition is the fastest primitive binary floating point operator. We use t to refer to the minimum time required to send a single floating point number between two distinct processors in a given multiprocessor, which we will call the *minimum transmission time*. We use $f_{(+)}$ to refer to the minimum time required to add two floating point numbers in a given multiprocessor.

2.1.2. Communication capabilities. Define the *distance* from vertex v_1 to vertex v_2, $D(v_1, v_2)$, to be the minimum amount of time it takes to send a single floating point number from the processor represented by v_1 to the processor represented by v_2. Define a *center* of a subset of vertices of the graph $V' \subseteq V$ to be a vertex in V' that minimizes the maximum distance between itself and other vertices in the subset. That is, if c is a center of V', then

$$\max_{w \in V'} D(c, w) = \min_{v \in V'} \max_{w \in V'} D(v, w).$$

[1] *M*ultiple *I*nstruction *M*ultiple *D*ata is one category of Flynn's multiprocessor taxonomy [10]. If a computer is an MIMD multiprocessor, then both the instruction a processor is executing and the data it is using can be different from those of other processors at any given moment.

[2] We ignore memory devices in the model since they do not affect the lower bounds we derive.

Define the *radius* of the subset to be the distance

$$\min_{v \in V'} \max_{w \in V'} D(v, w).$$

For a given multiprocessor, consider a subset of Q processors with minimum radius over all such subsets of size Q. Define the function $r(Q)$ to be the radius of this subset. In §2.3 we will use $r(Q)$ to describe lower bounds on the execution time of algorithms.

2.1.3. Example architectures. Most multiprocessor architectures currently in use have fairly simple graphs, with essentially homogeneous processor and communication capabilities [9]. The following examples are common designs, each of whose behavior is representative of a class of architectures. All of the examples can be described as undirected graphs. If an edge exists from v_i to v_j, then an edge also exists from v_j to v_i. Additionally, all processor and communication channel capabilities are the same. Finally, all of the examples use a store-and-forward communication protocol [1]. Thus, the distance between two vertices is t times the minimum number of communication channels that must be traversed when sending a floating point number between the corresponding processors.

- *Fully connected.* The graph of the architecture is a clique, and the radius of any subset of the multiprocessor containing more than one processor is t.
- *k-dimensional array.* The graph of the architecture is a k-dimensional array. Each processor is connected to up to $2 \cdot k$ other processors. Assume that the array has equal length sides and P processors. Then the radius of a subset of Q processors is in the interval

$$\left[t \cdot k \cdot \left(\frac{Q^{1/k}}{2} - 1 \right) , t \cdot k \cdot \frac{P^{1/k}}{2} \right].$$

  The upper bound is approximately the radius of the multiprocessor. The lower bound is approached when the subset of the multiprocessor is itself a k-dimensional array with $Q^{1/k}$ processors on a side.
- *Hypercube.* The graph of the architecture is a binary hypercube. Thus, there are $P = 2^N$ processors, where N is some nonnegative integer, and each vertex is the corner of an N-dimensional cube. Each processor is connected to $\log_2 P$ other processors. The radius of a subset of Q processors is in the interval[3]

$$[t \cdot \lceil \log_2 Q \rceil , t \cdot \lceil \log_2 P \rceil].$$

  The upper bound is the radius of the multiprocessor. The lower bound is associated with a subset of the multiprocessor that approximates a Q-processor hypercube.

These examples are usually associated with distributed-memory multiprocessors, but common shared-memory architectures can also be associated with them. For example, a well-designed bus-based multiprocessor will behave like a fully connected architecture. Similarly, a shared-memory multiprocessor based on an omega interconnection network [11] will have communication capabilities similar to a hypercube.

[3] $\lceil x \rceil$ is the smallest integer greater than or equal to x. $\lfloor x \rfloor$ is the largest integer less than or equal to x.

2.2. Algorithm model.

2.2.1. Serial algorithm. We model an algorithm as a partially ordered set of *instructions* of the form

$$y = \mathit{flop}(x_1, \cdots, x_n),$$

where *flop* is a floating point operator, y is a floating point variable, and $\{x_1, \cdots, x_n\}$ are floating point constants and variables. If a floating point variable is used by two different instructions, and if one of the instructions changes the value of that variable, then the partial order specifies a precedence relationship between them. These are the only relationships established by the partial order.

We define the *serial cost* of an algorithm, C_s, to be the time spent executing the floating point operations on some standard serial processor. The standard processor is assumed to satisfy the assumptions made in the previous section about the processors in the multiprocessor. All sequential orderings of the instructions of an algorithm that are consistent with the partial ordering will have the same serial cost, and will produce the same results when executed on a serial processor. In consequence, we will also refer to the partially ordered set of instructions as a *serial algorithm.*

2.2.2. Parallel algorithm. A parallel implementation of an algorithm on a multiprocessor specifies when and on which processor each instruction is executed, and what communication takes place during the execution of the algorithm. We will refer to this information as the *scheduling* of the algorithm. Define a scheduling to be *well defined* if it is compatible with the partial order's precedence relationships, and if all demands made on the processors and communication channels are within their capabilities. We define a *parallel algorithm* to be a triple consisting of a serial algorithm, a multiprocessor architecture, and a well defined scheduling. Thus, we associate a deterministic serial algorithm with each parallel algorithm. In practice, the serial algorithm may be a function of the data. This can make determining the serial algorithm difficult, especially for chaotic parallel algorithms [6], [8]. But we can still analyze characteristics of a parallel algorithm by establishing necessary properties of the associated serial algorithm.

We define the *parallel cost*, T_p, to be the time it takes to execute a parallel algorithm on a multiprocessor. There are at least two distinct costs associated with a parallel algorithm, the *computation cost* and the *communication cost.*

The computation cost is the amount of time during which at least one of the processors is busy executing the instructions of the serial algorithm. If P processors are used by a parallel algorithm to execute instructions, then the computation cost, C_p, is bounded from below by

$$C_p \geq \frac{C_s}{P}, \tag{1}$$

where the serial cost in this expression is based on using the "fastest" of the P processors as the standard serial processor. The communication cost, W_p, is the amount of time during which at least one processor is actively sending or receiving a message or at least one of the communication channels is busy transmitting a message. While the achievable computation cost is constrained by the number of processors in the multiprocessor and by the serial algorithm, the communication cost is additionally constrained by the graph of the multiprocessor. Both the computation cost and the communication cost represent lower bounds on the parallel cost:

$$\max\{C_p, W_p\} \leq T_p. \tag{2}$$

The standard measure of the efficiency of a parallel algorithm is the fraction of the total execution time that processors spend on tasks that are present in a single processor implementation of the algorithm. If we assume that a parallel algorithm uses P identical processors, then the parallel efficiency E_p is

$$E_p = \frac{C_s}{T_p \cdot P}.$$

By (1) and (2), the efficiency is always less than or equal to one.

2.3. Lower bounds on parallel cost. A serial algorithm describes a mapping from a set of data to a set of solution values specified by the problem. Denote the set of solution values (output) of an algorithm **a** by

$$U = \{u_j \mid j = 1, \cdots, N_{a,u}\},$$

where $N_{a,u}$ is the number of these solution values and each u_j is a scalar. Denote the data (input) required to produce these solution values by

$$G = \{g_k \mid k = 1, \cdots, N_{a,g}\}.$$

That is, there is no $g_k \in G$ that can be arbitrarily changed without changing at least one of the solution values. $N_{a,g}$ is the number of these data and each g_k is a scalar.

For $u_j \in U$, define $N_a(u_j)$ to be the amount of data required by **a** to compute u_j. Again, this means that no one of these $N_a(u_j)$ data can be arbitrarily changed without changing the value of u_j. If $N_a(u_j) > 1$, then each datum must be the operand of a primitive binary floating point operator or the composition of one primitive binary floating point operator and some number of primitive unary floating point operators. This operation produces a result that is either u_j or the operand of another, similarly defined, binary floating point operator. Continuing this argument and summing the indicated number of operations leads to the conclusion that the serial cost of calculating u_j is bounded from below by $f_{(+)} \cdot (N_a(u_j) - 1)$. [4] The next lemma is a direct consequence.

LEMMA 2.1. *A lower bound on the serial cost of an algorithm* **a** *is*

$$f_{(+)} \cdot (N_a(u_j) - 1)$$

for any $u_j \in U$.

A processor *collaborates* in the computation of a solution value u_j if changing the results of the floating point operations calculated by that processor can change the value of u_j. Define $P_a(u_j)$ to be the number of processors that collaborate in the calculation of u_j for a given parallel implementation of **a**. As in §2.1.2, let $r(P_a(u_j))$ be the radius of the $P_a(u_j)$ processor subset that has the minimum radius.

LEMMA 2.2. *The communication cost of a parallel implementation of an algorithm* **a** *is bounded from below by* $r(P_a(u_j))$ *for any* $u_j \in U$.

Proof. The lemma is a consequence of the fact that information is needed from all $P_a(u_j)$ processors in order to calculate u_j. Denote the subset of processors that collaborate in the calculation of u_j by V'. Assume that a processor $v \in V'$ has been designated to calculate the final unary or binary operation that produces u_j. Each processor in V' produces a result that is crucial to the calculation. Therefore,

[4] See Lemma 1 in Kuck [17, p. 95].

whether a result travels directly to v or is used by another processor to produce a new partial result that is then sent on, the time spent communicating is never less than the distance to v from the originating processor. Therefore, the maximum distance from the processors in V' to v is a lower bound on the execution time. By definition, this is bounded from below by $r(P_a(u_j))$. □

The computation cost of calculating u_j on a multiprocessor can be no smaller than the minimum computation cost of summing its required data. The data can be reduced only by a sequence of binary operations, and binary addition is assumed to be the fastest binary floating point operator. Since addition is a binary operator, M existing summands are replaced by at most $M/2$ results during any time interval of length $f_{(+)}$. These results and any unused summands are the summands for the next step of the calculation. Thus, one lower bound on the computation cost is $f_{(+)} \cdot \lceil \log_2 N_a(u_j) \rceil$. [5] This result, Lemma 2.1, and (1) prove the following lemma.

LEMMA 2.3. *The computation cost of a parallel implementation of an algorithm* **a** *is bounded from below by*

$$f_{(+)} \cdot \max \left\{ \left\lceil \frac{N_a(u_j) - 1}{P_a(u_j)} \right\rceil , \lceil \log_2 N_a(u_j) \rceil \right\}$$

for any $u_j \in U$.

Define $N_{a,*}$ to be the maximum amount of data required by an algorithm **a** to compute a single solution value

$$N_{a,*} = \max_{u_j \in U} N_a(u_j).$$

For some parallel implementation of **a**, define $P_{a,*}$ to be the maximum number of processors that collaborate in the calculation of a single solution value

$$P_{a,*} = \max_{u_j \in U} P_a(u_j).$$

THEOREM 2.4. *The parallel cost of a parallel implementation of an algorithm* **a** *is bounded from below by*

$$\max \left\{ r(P_{a,*}) , f_{(+)} \cdot \left\lceil \frac{N_{a,*} - 1}{P_{a,*}} \right\rceil , f_{(+)} \cdot \lceil \log_2 N_{a,*} \rceil \right\}.$$

Proof. The total cost is bounded from below by $\max\{C_p, W_p\}$. The result then follows from Lemmas 2.2 and 2.3. □

COROLLARY 2.5. *For a P processor multiprocessor, a lower bound on the parallel cost of any parallel implementation of a serial algorithm* **a** *is*

$$\min_{p \in \{1, \cdots, P\}} \max \left\{ r(p) , f_{(+)} \cdot \left\lceil \frac{N_{a,*} - 1}{p} \right\rceil , f_{(+)} \cdot \lceil \log_2 N_{a,*} \rceil \right\}.$$

Proof. $P_{a,*}$ is a member of the set $\{1, \cdots, P\}$ for any parallel implementation of an algorithm **a** on this multiprocessor. The proof follows from Theorem 2.4. □

[5] See Lemma 1 in Kuck [17, p. 95].

3. Bounds on parallel algorithms for linear PDEs. By Corollary 2.5, if we know how many data are required to calculate a single solution value, then we can compute a lower bound on the execution time of a given parallel algorithm. The problem assumptions described in the following sections allow us to bound the amount of this data for all algorithms as a function of the error tolerance, and thus calculate an *algorithm-independent* lower bound.

3.1. PDE assumptions.

3.1.1. Solution operator assumptions. For any nonnegative integer k, let $\Re^k$ be the k-dimensional Euclidean vector space and let I_k be the k-dimensional unit cube

$$I_k = [0,1] \times \cdots \times [0,1] \subset \Re^k.$$

For some $d \geq 1$, let Ω be a compact subset of $\Re^d$.

1) *We assume that we are approximating the solution of a linear scalar partial differential equation defined on Ω whose solution operator can be represented by an expression of the form*

$$u(\bar{z}) = \sum_{i=1}^{S} \int_{I_{d_i}} \Psi_i(\bar{z}, \bar{x}) g_i(\bar{x})\, d\bar{x}$$

for all $\bar{z} \in \Omega$. S is a positive integer, $\{d_i \mid i \in \{1, \cdots, S\}\}$ is a set of nonnegative integers, $\{g_i \mid i \in \{1, \cdots, S\}\}$ is a set of functions representing the problem data, and $u(\bar{z})$ is the solution function. For each $i \in \{1, \cdots, S\}$ and $\bar{z} \in \Omega$, $\Psi_i(\bar{z}, \bar{x})$ is a Lebesgue integrable function in I_{d_i}.

Thus, the solution function is the sum of S *components* of the form

$$u_i(\bar{z}) = \int_{I_{d_i}} \Psi_i(\bar{z}, \bar{x}) g_i(\bar{x})\, d\bar{x}.$$

For the class of problems considered here, the kernels $\{\Psi_i\}$ are linear functionals of the *Green's function* for the PDE. See §3.1.3, §4, and Butkovskiy [7] for examples of this type of representation of the solution operator.

3.1.2. Data function assumptions. Define $C^m(R_k)$ to be the set of all functions that have continuous mth order partial derivatives on a set $R_k \subset \Re^k$. Let $\bar{\nabla}_k^{(m)} g$ be a vector whose elements are some ordering of the mth order partial derivatives of g in $\Re^k$.

2) *For each $i \in \{1, \cdots, S\}$, we assume that the data function g_i is known to be some member of a set G_i defined in the following way. G_i is the set of all functions g satisfying the properties*

(i) $g(\bar{x}) \in C^{m_i}(I_{d_i})$,

(ii) $\|\bar{\nabla}_{d_i}^{(m_i)} g(\bar{x})\|_{(i)} \leq M_i(\bar{x})$ *for all* $\bar{x} \in I_{d_i}$,

where m_i is a positive integer, $\|\cdot\|_{(i)}$ is a vector norm, and $M_i(\bar{x})$ is a nonnegative function. We also assume that any member of G_i is a permissible data function for the ith component, and that any combination of data functions from the sets $\{G_i\}$ generate a possible solution to the PDE, *with one possible exception. The inclusion of a given data function g_i in a set of data functions may force the data function for a different component, g_j, to have given function and derivative values on the boundary of I_{d_j}. We will refer to this as a compatibility condition.*

Assumption 2) states that the data functions are known to have a certain number of continuous derivatives, and that the magnitude of the largest partial derivatives is bounded by some known function. This is the only information we are assuming about the data functions. This type of assumption on the data is similar to the assumptions that are made on the data and solution functions when specifying a priori bounds on the error introduced when using finite-difference and finite-element discretizations of the PDE to approximate the solution. The particular form of these assumptions is similar to that used by Traub and Woźniakowski [19].

3.1.3. Example PDE. Let $B((0,0);1)$ be the open ball in $\Re^2$ centered on $(0,0)$ with a radius of 1. Let $\partial B((0,0);1)$ be the boundary of the ball. Let $\bar{x} = (x_1, x_2)$, and let $\bar{z} = (z_1, z_2)$. Consider the following linear PDE:

$$-\left(\frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2}\right) u(\bar{x}) = f(\bar{x}) \quad \text{for} \quad \bar{x} \in B((0,0);1),$$

$$u(\bar{x}) = h(\bar{x}) \quad \text{for} \quad \bar{x} \in \partial B((0,0);1).$$

Assume that $f \in C^2(B((0,0);1))$ and that

$$\left\| \nabla_2^{(2)} f(\bar{x}) \right\|_\infty \equiv \max\left\{ \left| \frac{\partial^2}{\partial x_1^2} f(\bar{x}) \right|, \left| \frac{\partial^2}{\partial x_1 x_2} f(\bar{x}) \right|, \left| \frac{\partial^2}{\partial x_2^2} f(\bar{x}) \right| \right\} \leq 1$$

for $\bar{x} \in B((0,0);1)$. Further assume that $h \in C^1(\partial B((0,0);1))$ and that

$$\left| \frac{\partial}{\partial t} h(\bar{x}) \right| \leq 1$$

for $\bar{x} \in \partial B((0,0);1)$, where $\partial/\partial t$ is the directional derivative tangent to the boundary of the ball.

The solution to this PDE can be represented by

$$u(\bar{z}) = \int_{B((0,0);1)} G(\bar{z}, \bar{x}) f(\bar{x})\, d\bar{x} + \int_{\partial B((0,0);1)} \left(\frac{\partial}{\partial n} G(\bar{z}, \bar{x}) \right) h(\bar{x})\, d\bar{x}$$

for $\bar{z} \in B((0,0);1)$, where $\partial/\partial n$ is the directional derivative (relative to $\bar{x}$) normal to the surface of the ball. The Green's function $G(\bar{z}, \bar{x})$ has a logarithmic singularity at $\bar{x} = \bar{z}$, and is a C^∞ function elsewhere in $B((0,0);1)$. See Butkovskiy [7, pp. 131–132] for more details.

To put this operator into the form required by Assumption 1), we first define the function $G(\bar{z}, \bar{x})$ to be identically zero for $\bar{x} \notin B((0,0);1)$. Next, define f to be a $C^2([-1,1] \times [-1,1])$ function by smoothly extending it into $([-1,1] \times [-1,1]) - B((0,0);1)$. Let

$$\varphi_1(x_1, x_2) = (2 \cdot x_1 - 1, 2 \cdot x_2 - 1), \quad \varphi_2(x) = \left(\cos\left(\frac{\pi}{2} + \pi \cdot x\right), -\sin\left(\frac{\pi}{2} + \pi \cdot x\right) \right),$$

and

$$\varphi_3(x) = \left(\cos\left(\frac{\pi}{2} - \pi \cdot x\right), \sin\left(\frac{\pi}{2} - \pi \cdot x\right) \right).$$

φ_1 maps I_2 onto the region $[-1,1] \times [-1,1]$. φ_2 maps I_1 onto the left half of $\partial B((0,0);1)$, and φ_3 maps I_1 onto the right half of $\partial B((0,0);1)$. The solution operator can now be written as

$$u(\bar{z}) = \int_{I_2} \Psi_1(\bar{z},\bar{x}) g_1(\bar{x})\, d\bar{x} \; + \; \int_{I_1} \Psi_2(\bar{z},x) g_2(x)\, dx \; + \; \int_{I_1} \Psi_3(\bar{z},x) g_3(x)\, dx,$$

where

$$\Psi_1(\bar{z},\bar{x}) = 4 \cdot G(\bar{z},\varphi_1(\bar{x}))\,, \quad g_1(\bar{x}) = f(\varphi_1(\bar{x}))\,, \quad \Psi_2(\bar{z},x) = \pi \cdot \left(\frac{\partial}{\partial n} G(\bar{z},\varphi_2(x))\right),$$

$$g_2(x) = h(\varphi_2(x))\,, \quad \Psi_3(\bar{z},x) = \pi \cdot \left(\frac{\partial}{\partial n} G(\bar{z},\varphi_3(x))\right), \quad \text{and} \quad g_3(x) = h(\varphi_3(x)).$$

Now, $g_1 \in C^2(I_2)$ and $\|\nabla_2^{(2)} g_1(\bar{x})\|_\infty \leq 4$ for $\varphi_1(\bar{x}) \in B((0,0);1)$. For other values of $\bar{x} \in I_2$, the bound will depend on how we extended the function f. Also, $g_2, g_3 \in C^1(I_1)$, $|(d/dx) g_1(x)| \leq \pi$, and $|(d/dx) g_2(x)| \leq \pi$ for $x \in I_1$. Finally, there are the following compatibility conditions between g_2 and g_3:

$$g_2(0) = g_3(1)\,, \quad \frac{d}{dx} g_2(0) = \frac{d}{dx} g_3(1)\,, \quad g_2(1) = g_3(0)\,, \quad \frac{d}{dx} g_2(1) = \frac{d}{dx} g_3(0).$$

Thus, this PDE satisfies Assumptions 1) and 2).

3.2. Problem assumptions. Numerical approximations to the solution of the PDE replace the possibly infinite-dimensional problem with a finite-dimensional problem. The dimensionality of the problem is reduced by introducing error in the following sense:

- Only a finite amount of information about the solution function is calculated. Any model of the solution based only on this information will merely approximate the true solution.
- Only a finite amount of information about the data functions is used to calculate the desired solution properties. We will refer to this as the *data function sampling.* Unless this information completely characterizes the data functions, the solution values that are calculated are also approximate.

The PDE is replaced by a relationship between the data function samples and the information about the solution function that is to be calculated. The error in the approximation to the solution is also a function of this relationship.

For the rest of this paper, we restrict ourselves to problems where

3) *values of the data functions sampled in their domains are available to be used,*
4) *values of the solution function at given locations in its domain are approximated, and*
5) *the error in approximating each solution value is bounded by a given value* ϵ.

Thus, we are specifying what type of data can be used, what solution values to approximate, and how accurately these values are to be approximated. Note that we are not stipulating where the data functions can be sampled. Rather, we are stipulating that only pointwise evaluations of the functions can be used by any algorithm used to solve this problem. Let Z represent the set of locations where the value of the solution function is needed.

We can use the following notation for any serial algorithm **a** with finite serial cost that solves a problem satisfying the above assumptions. The algorithm calculates the

value of the solution function u at some finite set of locations $\{\bar{z}_j \mid j = 1, \cdots, N_{a,u}\}$ in Ω that contains the set Z. For each data function g_i, the algorithm uses function values at some finite set of locations $X_i = \{\bar{x}_{i,k} \mid k = 1, \cdots, N_{a,i}\}$ in I_{d_i}. And, for any particular solution value $u(\bar{z}_j)$, the algorithm uses values of g_i at some set of locations

$$X_{i,j} = \{\bar{x}_{i,j,k} \mid k = 1, \cdots, N_{a,i}(u(\bar{z}_j))\} \subseteq X_i$$

in I_{d_i}.

3.3. Bounds on execution time. Assumptions 1)–5) describe the type of problems we are considering. Each problem is specified by the linear PDE, what type of data can be used, the assumptions on the data functions, the locations of the desired solution values, and the error bound to be satisfied. The following theorem describes an algorithm-independent bound on the amount of data needed to satisfy an error bound when approximating one of the desired solution values. We will then use this to describe a lower bound on the computation cost of all algorithms for a given problem.

Let $\bar{z}_j \in Z$ be one of the desired solution value locations. Let ϵ be the upper bound on the allowable error when approximating $u(\bar{z}_j)$. Let $\bar{B}_i(\bar{x}; \delta)$ denote a closed ball in $\Re^{d_i}$ centered on $\bar{x}$ with a radius of δ. If $d_i = 0$, then we define δ to be zero.

THEOREM 3.1. *Let i be the index of a component, $i \in \{1, \cdots, S\}$. If there exists a closed ball $\bar{B}_i(\bar{x}_*; \delta) \subset I_{d_i}$ on which $\Psi_i(\bar{z}_j, \bar{x})$ is either strictly positive or strictly negative, then there exist positive constants C_i and C_i' dependent only on m_i, d_i, and $\| \cdot \|_{(i)}$ with the following property. If*

$$C_i \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*; \delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*; \delta)} M_i(\bar{x}) \cdot \delta^{m_i + d_i} > \epsilon, \tag{3}$$

then data are required for at least

$$\left\lceil \left(C_i' \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*; \delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*; \delta)} M_i(\bar{x}) \cdot \delta^{m_i + d_i} \right)^{d_i/m_i} \cdot \left(\frac{1}{\epsilon} \right)^{d_i/m_i} \right\rceil$$

locations in $\bar{B}_i(\bar{x}_*; \delta)$ in order to satisfy the error bound.

This theorem is a variant of Theorem 2.2 in Worley [20, p. 45]. The full details of the proof can be found there. See Appendix A for an abbreviated version of the proof. A simple example of the argument used to establish the result follows.

Example Argument. Assume that the solution operator is

$$u(\bar{z}_j) = \int_0^1 g(x)\, dx$$

for some $\bar{z}_j \in \Omega$. Also assume that $g \in C^1(I_1)$ and $|(d/dx)g(x)| \leq 1$ for $x \in I_1$. Thus, there is only one component, $d_1 = 1$, $\Psi_1(\bar{z}_j, x) \equiv 1$, $m_1 = 1$, and $M_1(x) \equiv 1$. Assume that our numerical approximation is restricted to using data sampled from locations on a uniform grid $\{j \cdot \Delta x \mid j \in \{0, \cdots, N\}\}$, where $N = 1/\Delta x$. Further assume that $g(j \cdot \Delta x) = 0$ for all j.

Given only this information, there is an intrinsic uncertainty as to what the true data function is, and this uncertainty represents an uncontrollable error in the approximation scheme. For example, most numerical algorithms would (implicitly) assume that $g(x)$ is identically zero, and would approximate $u(\bar{z}_j)$ accordingly. But

other functions exist that are zero at these locations and satisfy the same smoothness assumptions that g is known to satisfy. For example, define $g_*(x)$ to be

$$g_*(x) = \begin{cases} x - j \cdot \Delta x & \text{for } x \in [j \cdot \Delta x, (j+0.5) \cdot \Delta x], \\ -x + (j+1) \cdot \Delta x & \text{for } x \in [(j+0.5) \cdot \Delta x, (j+1) \cdot \Delta x], \end{cases}$$

for $j \in \{0, \cdots, N-1\}$. If g_* were the true data function, then assuming that g is identically zero represents an error of size

$$\int_0^1 g_*(x)\, dx \quad = \quad \frac{\Delta x}{4}.$$

Now, g_* is not an element of $C^1(I_1)$, but it is the limit of functions satisfying the required smoothness assumptions. Therefore, the error can be arbitrarily close to this value. In fact, $\Delta x/4$ is the maximum, or *worst case*, error that can occur from assuming that $g(x) \equiv 0$. Using a different function to approximate $u(\bar{z}_j)$ will not alter the basic uncertainty, only the size of the worst case error. Since $-g_*(x)$ is also the limit of functions satisfying the assumptions, it is simple to show that assuming that $g(x)$ is identically zero minimizes the worst case error. For example, see Traub and Woźniakowski [19, pp. 12–13 and 46–47].

For a particular algorithm and set of data, there may be no error, but we have no way of knowing that given only the assumed information. In order to guarantee that the error does not exceed a given value, we must assume the worst case when choosing how much data to use. For example, if the error must be less than some error tolerance ϵ when calculating the solution to the above problem, then we require that $\Delta x \leq 4 \cdot \epsilon$, or $N \geq 1/(4\epsilon)$. This agrees with Theorem 3.1. The proof of the theorem follows the same type of reasoning we have used here, but higher dimensions and the need to bound the worst case error independent of the data sampling distribution make the proof more complicated. □

Define $N_{\epsilon,i}(u(\bar{z}_j))$ to be the minimum number of data sampling locations in I_{d_i} needed to ensure that the error in approximating $u(\bar{z}_j)$ is less than ϵ. Using this notation, an immediate corollary of Theorem 3.1 is the following.

COROLLARY 3.2. *Let i be the index of a component $i \in \{1, \cdots, S\}$. If there exists a closed ball $\bar{B}_i(\bar{x}_*; \delta) \subset I_{d_i}$ of positive radius δ on which $\Psi_i(\bar{z}_j, \bar{x})$ is either strictly positive or strictly negative and on which both $\Psi_i(\bar{z}_j, \bar{x})$ and $M_i(\bar{x})$ are bounded away from zero, then*[6]

$$N_{\epsilon,i}(u(\bar{z}_j)) = \Omega\left(\epsilon^{-d_i/m_i}\right)$$

as a function of $1/\epsilon$.

Proof. Since $\delta > 0$ we know that $d_i > 0$. Therefore, since both $|\Psi_i(\bar{z}_j, \bar{x})|$ and $M_i(\bar{x})$ are bounded away from zero, the left-hand side of (3) in Theorem 3.1 is always positive. Therefore, as $\epsilon \to 0$ (and $1/\epsilon \to \infty$), the inequality will eventually be satisfied and $N_{\epsilon,i}(u(\bar{z}_j))$ must grow at least as fast as $c\,\epsilon^{-d_i/m_i}$ for some positive constant c. □

The implication of this corollary is that the amount of data required to approximate $u(\bar{z}_j)$ will increase without bound as $\epsilon \to 0$ if the assumptions of the corollary hold for some component i. These assumptions hold for many of the linear PDEs

[6] For two real-valued functions f and g, $f(x) = \Omega(g(x))$ if and only if there exist positive constants c and x_0 such that $|f(x)| \geq c \cdot |g(x)|$ for all $x > x_0$ [15].

arising in mathematical physics. By Lemma 2.3, this implies that the computation cost of solving the problem must also increase without bound as $\epsilon \to 0$. We formally state this result in the next theorem.

Define $C_{p,\epsilon}(P, f_{(+)}, \bar{z}_j)$ to be the minimum computation cost over all algorithms (for a problem satisfying Assumptions 1)–5)) that approximate $u(\bar{z}_j)$ to within an error tolerance of ϵ on a multiprocessor with P processors and a given minimum execution time for floating point addition $f_{(+)}$.

THEOREM 3.3. *Assume that, for some $i \in \{1, \cdots, S\}$, there exists a closed ball $\bar{B}_i(\bar{x}_*; \delta) \subset I_{d_i}$ of positive radius δ on which $\Psi_i(\bar{z}_j, \bar{x})$ is either strictly positive or strictly negative and on which both $\Psi_i(\bar{z}_j, \bar{x})$ and $M_i(\bar{x})$ are bounded away from zero. Then there exist positive constants α and ϵ_* that are independent of ϵ, P, and $f_{(+)}$ such that*

$$C_{p,\epsilon}(P, f_{(+)}, \bar{z}_j) \geq \alpha \cdot f_{(+)} \cdot \max \left\{ \frac{\epsilon^{-d_i/m_i}}{P}, \log_2 \left(\epsilon^{-d_i/m_i} \right) \right\}$$

when $\epsilon \leq \epsilon_*$.

Proof. The total amount of data required to approximate the solution value $u(\bar{z}_j)$ to within an error tolerance of ϵ is at least as great as $N_{\epsilon,i}(u(\bar{z}_j))$. Therefore, by Lemma 2.3,

$$\begin{aligned} C_{p,\epsilon}(P, f_{(+)}, \bar{z}_j) &\geq f_{(+)} \cdot \min_{p \in \{1, \cdots, P\}} \max \left\{ \left\lceil \frac{N_{\epsilon,i}(u(\bar{z}_j)) - 1}{p} \right\rceil, \lceil \log_2 N_{\epsilon,i}(u(\bar{z}_j)) \rceil \right\} \\ &\geq f_{(+)} \cdot \left(\max \left\{ \frac{N_{\epsilon,i}(u(\bar{z}_j)) - 1}{P}, \log_2 \left(N_{\epsilon,i}(u(\bar{z}_j)) \right) \right\} \right). \end{aligned}$$

The result then follows from Corollary 3.2. □

This result is essentially tight for a homogeneous multiprocessor if each of the components either satisfies the assumptions of Theorem 3.3 or satisfies $d_i = 0$, and if each $M_i(\bar{x})$ is a bounded function.

THEOREM 3.4. *For each $i \in \{1, \cdots, S\}$, assume that the ith component satisfies the following conditions:*

(i) *$M_i(\bar{x})$ is a bounded, nonnegative function in I_{d_i}.*

(ii) *Either $d_i = 0$ or there exists a closed ball $\bar{B}_i(\bar{x}_*; \delta) \subset I_{d_i}$ of positive radius δ on which $\Psi_i(\bar{z}_j, \bar{x})$ is either strictly positive or strictly negative and on which both $\Psi_i(\bar{z}_j, \bar{x})$ and $M_i(\bar{x})$ are bounded away from zero.*

Also assume that $d_i \neq 0$ for at least one component. If all processors are identical and $f_{(+)} > 0$, then there exist positive constants α, β, and ϵ_* that are independent of ϵ and P and have the following property. Let

$$a = \alpha \cdot \max_{i \in \{1, \cdots, S\}} \left(\max \left\{ \left\lfloor \frac{\epsilon^{-d_i/m_i}}{P} \right\rfloor, \log_2 \left(\epsilon^{-d_i/m_i} \right) \right\} \right)$$

and

$$b = \beta \cdot \max_{i \in \{1, \cdots, S\}} \left(\max \left\{ \left\lceil \frac{\epsilon^{-d_i/m_i}}{P} \right\rceil, \log_2 \left(\epsilon^{-d_i/m_i} \right) \right\} \right).$$

Then $C_{p,\epsilon}(P, f_{(+)}, \bar{z}_j)$ is contained in the interval $[a, b]$ when $\epsilon < \epsilon_*$.

This theorem follows from Theorem 2.3 on page 51 and Lemma 3.5 on page 77 in Worley [20]. The full details can be found there. See Appendix B for an abbreviated proof. A simple example of the argument used to establish the result follows.

Example Argument. As in the previous example, assume that the solution operator is

$$u(\bar{z}_j) = \int_0^1 g(x)\,dx$$

for a given $\bar{z}_j \in \Omega$. Assume that $g \in C^m(I_1)$ for some positive m, and that $|(d^m/dx^m)g(x)| \leq 1$ for $x \in I_1$. To approximate $u(\bar{z}_j)$, we first calculate a piecewise polynomial approximation to the data function in the following way. Divide the interval $[0,1]$ into N subintervals of length $\Delta x = 1/N$,

$$\{[j \cdot \Delta x, (j+1) \cdot \Delta x] \,|\, j \in \{0, \cdots, N-1\}\}.$$

Within the subinterval $[j \cdot \Delta x, (j+1) \cdot \Delta x]$, sample the data function at the m locations

$$\left\{ \left(j + \frac{k}{m+1} \right) \cdot \Delta x \;\middle|\; k \in \{1, \cdots, m\} \right\}.$$

Denote the unique $(m-1)$st degree polynomial that interpolates g at these locations by $\tilde{g}_j(x)$. It is a standard result [5], [18] that the difference between $g(x)$ and $\tilde{g}_j(x)$ in the subinterval is bounded from above by a term of the form

$$C_m \cdot (\Delta x)^m \cdot \max_{x \in [j \cdot \Delta x, (j+1) \cdot \Delta x]} \left| \frac{d^m}{dx^m} g(x) \right|,$$

where C_m is dependent only on m. Thus, by our assumptions, the interpolation error in the subinterval is no larger than $C_m \cdot (\Delta x)^m$. Our approximation to $g(x)$ is defined by

$$\tilde{g}(x) = \begin{cases} \tilde{g}_j(x) & \text{for } x \in [j \cdot \Delta x, (j+1) \cdot \Delta x), \quad j \in \{1, \cdots, N-2\}, \\ \tilde{g}_{N-1}(x) & \text{for } x \in [(N-1) \cdot \Delta x, 1]. \end{cases}$$

Define $\tilde{u}_j$ by

$$\tilde{u}_j = \int_0^1 \tilde{g}(x)\,dx.$$

This represents an algorithm for approximating $u(\bar{z}_j)$ whose error is bounded from above by

$$|u(\bar{z}_j) - \tilde{u}_j| \;\leq\; \int_0^1 |g(x) - \tilde{g}(x)|\,dx \;\leq\; C_m \cdot (\Delta x)^m.$$

Thus, $N \geq (C_m/\epsilon)^{1/m}$ is a sufficient condition to guarantee that the error is no more than ϵ.

Using the Lagrange formulation, each polynomial $\tilde{g}_j(x)$ can be represented by an expression of the form

$$\tilde{g}_j(x) \;=\; \sum_{k=1}^{m} g\left(\left(j + \frac{k}{m+1} \right) \cdot \Delta x \right) \cdot p_k(x - j \cdot \Delta x) \;\equiv\; \sum_{k=1}^{m} g_{j,k} \cdot p_k(x - j \cdot \Delta x),$$

where p_k is a polynomial of degree at most $m-1$. Thus,

$$\begin{aligned} \tilde{u}_j &= \sum_{j=0}^{N-1} \int_{j\cdot\Delta x}^{(j+1)\cdot\Delta x} \tilde{g}_j(x)\,dx \\ &= \sum_{j=0}^{N-1} \sum_{k=1}^{m} g_{j,k} \cdot \int_0^{\Delta x} p_k(x)\,dx \\ &\equiv \sum_{j=0}^{N-1} \sum_{k=1}^{m} \alpha_k \cdot g_{j,k}. \end{aligned} \tag{4}$$

If we precompute (or already know) the constants $\{\alpha_k\}$, then the algorithm requires $m \cdot N$ multiplications and $m \cdot N - 1$ additions. We implement this algorithm on a P-processor multiprocessor in the following way. First, partition the data $\{g_{j,k}\}$ into P subsets each of which contains no more than $\lceil m \cdot N/P \rceil$ elements and allocate a different processor to each subset. Next, in parallel, use the processor allocated to a given subset to calculate the portion of the weighted sum in (4) that corresponds to the subset. Finally, sum the remaining $\min\{m \cdot N, P\}$ partial results using a parallel fan-in algorithm [17]. The computation cost of this parallel implementation of the algorithm is

$$\left\lceil \frac{m \cdot N}{P} \right\rceil \cdot (f_{(*)} + f_{(+)}) \;+\; \left(\left\lceil \log_2 (\min\{m \cdot N\,,\, P\}) \right\rceil - 1 \right) \cdot f_{(+)},$$

where $f_{(*)}$ and $f_{(+)}$ are the times required to execute a floating point multiplication and a floating point addition, respectively. If $N = \lceil (C_m/\epsilon)^{1/m} \rceil$, then this expression agrees with the upper bound in Theorem 3.4. The proof of the theorem follows the same general reasoning. The lower bound comes from Theorem 3.3, while the upper comes from constructing an algorithm whose computation cost is of the required form. The upper bound argument is somewhat more complicated than that described here because we must bound the error when interpolating in higher dimensions and when there is more than one component. □

The computation cost represents only part of the parallel cost. Using Lemma 2.2 and Theorem 3.3, we can describe a lower bound on the parallel cost that is independent of the algorithm and the number of processors in the multiprocessor, but takes into account the graph of the multiprocessor. Define $T_{p,\epsilon}(P, f_{(+)}, t, \bar{z}_j)$ to be the minimum parallel cost over all algorithms (for a problem satisfying Assumptions 1)–5)) that approximate $u(\bar{z}_j)$ to within an error tolerance of ϵ on a multiprocessor with P processors, a given minimum floating point addition execution time, $f_{(+)}$, and a given minimum transmission time, t.

THEOREM 3.5. *Assume that, for some $i \in \{1, \cdots, S\}$, there exists a closed ball $\bar{B}_i(\bar{x}_*; \delta) \subset I_{d_i}$ of positive radius δ on which $\Psi_i(\bar{z}_j, \bar{x})$ is either strictly positive or strictly negative and on which both $\Psi_i(\bar{z}_j, \bar{x})$ and $M_i(\bar{x})$ are bounded away from zero. Also assume that both $f_{(+)}$ and t are positive.*

(i) *If the multiprocessor is a hypercube or fully connected architecture, then there exist positive constants α and ϵ_* that are independent of ϵ and P such that*

$$T_{p,\epsilon}(P, f_{(+)}, t, \bar{z}_j) \;\geq\; \alpha \cdot \log_2 \left(\epsilon^{-d_i/m_i} \right)$$

when $\epsilon \leq \epsilon_*$.

(ii) *If the multiprocessor is a k-dimensional array architecture, then there exist positive constants β and ϵ_{**} that are independent of ϵ and P such that*

$$T_{p,\epsilon}(P, f_{(+)}, t, \bar{z}_j) \geq \beta \cdot \epsilon^{-d_i/((k+1)\cdot m_i)}$$

when $\epsilon \leq \epsilon_{**}$.

Proof. Statement (i) is just a weakened form of Theorem 3.3. By the definition of a k-dimensional array, Corollary 2.5, and Corollary 3.2, if ϵ is sufficiently small, then

$$\begin{aligned} T_{p,\epsilon}(P, f_{(+)}, t, \bar{z}_j) &\geq \min_{p\in\{1,\cdots,P\}} \max\left\{ r(p)\,,\ \eta \cdot \frac{\epsilon^{-d_i/m_i}}{p}\,,\ \nu \cdot \log_2\left(\epsilon^{-d_i/m_i}\right) \right\} \\ &\geq \min_{p\in\{1,\cdots,\infty\}} \max\left\{ \tau \cdot (p^{1/k} - 2)\,,\ \eta \cdot \frac{\epsilon^{-d_i/m_i}}{p}\,,\ \frac{\nu \cdot d_i}{m_i} \cdot \log_2 \epsilon^{-1} \right\} \end{aligned}$$

for positive constants η, ν, and τ. For small ϵ, the minimum in this lower bound on $T_{p,\epsilon}(P, f_{(+)}, t, \bar{z}_j)$ is achieved with

$$p > \left(\frac{\eta}{\tau}\right)^{k/(k+1)} \cdot \epsilon^{-k\cdot d_i/((k+1)\cdot m_i)}.$$

Therefore,

$$T_{p,\epsilon}(P, f_{(+)}, t, \bar{z}_j) > \eta^{1/(k+1)} \cdot \tau^{k/(k+1)} \cdot \epsilon^{-d_i/((k+1)\cdot m_i)} - 2\cdot\tau,$$

and statement (ii) follows immediately. □

Thus, for a k-dimensional array architecture, the lower bound on the parallel cost grows significantly faster than the lower bound on the computation cost.

3.4. Problem scaling. In §1 we introduced the idea of increasing the problem size (or *scaling* the problem) in order to increase the amount of easily exploitable parallelism. A formal definition of problem size can be made [20, p. 64], but it is sufficient to treat it as a measure of the serial cost of the algorithms used to solve a problem. Thus, increasing the size of the problem implies changing the parameters of the problem so that the amount of work to be done in solving it increases. Scaling the problem defines a family of similar problems all approximating the solution to the PDE. We will refer to a given set of parameters as a problem *instance*. The problem parameters that are normally free to be varied are the solution values to be approximated, the error bound to be satisfied, and the available data. Commonly, increasing the size of the problem indicates that the number of solution values and the amount of data used increase, and that the error bound decreases. For some applications the solution is desired at only a fixed set of locations, and only the other two parameters will vary. But in both cases, increasing the size of the problem results in a better approximation to the solution function. It is simple to increase the size of a problem without suffering an increased parallel cost if only the number of solution values is increased and if enough idle processors are available. But continually increasing the number of approximate solution values will not lead to a better solution unless the error in these approximate values also decreases. We do not consider it reasonable to increase the size of the problem unless there is some advantage gained by doing so. This motivates the following assumption on how the size of the problem is increased.

6) *For a given problem instance, denote the set of locations where the solution is to be approximated by Z, and denote the error bound on these approximations by ϵ. If the size of the problem grows and Z' and ϵ' are the corresponding parameters of the new instance, then we assume that $Z \subseteq Z'$ and*

$$\epsilon' \leq \epsilon \cdot (|Z|/|Z'|)^{\gamma}$$

for some positive γ independent of the scaling. Here $|Z|$ represents the number of locations in the set.

Note that this assumption is not on the problem, but rather on how we permit the problem to grow. By this assumption the problem size is allowed to grow only if the error bound also decreases. Moreover, if the problem size grows without bound, then the error bound goes to zero. Assumption 6) is unnecessarily restrictive, but it is sufficient to establish the following theorem. In particular, the assumption that $Z \subseteq Z'$ is made merely to simplify the proof. See Worley [20, pp. 65–66] for a less restrictive assumption.

THEOREM 3.6. *Assume that, for some $i \in \{1, \cdots, S\}$ and a given problem instance, there exists a $\bar{z}_j \in Z$ with the following property. There exists a closed ball $\bar{B}_i(\bar{x}_*; \delta) \subset I_{d_i}$ of positive radius δ on which $\Psi_i(\bar{z}_j, \bar{x})$ is either strictly positive or strictly negative and on which both $\Psi_i(\bar{z}_j, \bar{x})$ and $M_i(\bar{x})$ are bounded away from zero. Also assume that $f_{(+)}$ is positive and bounded away from zero for all permissible multiprocessor architectures. Then, if the size of the problem increases without bound, so will the parallel cost, independent of the algorithm and of the number of processors used.*

Proof. This theorem follows directly from the assumption on how the problem scales, from Theorem 3.3, and from (2). If the size of the problem increases without bound, then ϵ decreases to zero. And, once $\bar{z}_j$ is introduced as a location where the solution is approximated, the solution is approximated there for all larger instances. Therefore, Theorem 3.3 implies that the computation cost must grow at least as fast as $-c \log_2 \epsilon$ for some positive constant c, independent of the number of processors in the multiprocessor. Since this lower bound becomes infinite when $\epsilon \to 0$, and since the computation cost is a lower bound on the parallel cost, the theorem is proved. Note that, by Theorem 3.5, the lower bound grows even faster if we know that the multiprocessor is a k-dimensional array. □

4. Examples. In this section we describe two examples. First, we calculate the lower bounds described in §3 for a specific elliptic differential equation. There is nothing special about this example except that it satisfies the assumptions of §3 and the bounds are easy to calculate. Second, we describe the analysis of a particular numerical algorithm for a hyperbolic PDE to demonstrate that the predicted behavior is easily observed in practical applications.

4.1. Lower bound calculation. Consider the following elliptic differential equation:

$$-\frac{d^2}{dx^2}u(x) = g_1(x) \quad \text{for} \quad x \in [0,1],$$

$$u(0) = g_2\,, \qquad u(1) = g_3$$

for some constants g_2 and g_3. The integral representation of the solution is

$$u(z) = \int_0^1 \Psi_1(z,x) g_1(x)\, dx \;+\; g_2 \cdot (1-z) \;+\; g_3 \cdot z,$$

where

$$\Psi_1(z,x) = \begin{cases} (1-z)\cdot x & \text{if } x \le z, \\ (1-x)\cdot z & \text{if } z \le x. \end{cases}$$

There are three components:

$$u_1(z) = \int_0^1 \Psi_1(z,x) g_1(x)\, dx,$$

$$u_2(z) = g_2 \cdot (1-z),$$

$$u_3(z) = g_3 \cdot z.$$

Thus, $d_1 = 1$ and $d_2 = d_3 = 0$.

Assume that g_1 is some member of a set G_1 that is characterized by the conditions that $g \in C^2([0,1])$ and

$$\left| \frac{d^2}{dx^2} g(x) \right| \le 2 \quad \forall x \in [0,1]$$

if $g \in G_1$. For example, $x^2 \in G_1$. (Since we are assuming only that g_1 has two continuous derivatives, some bound on the second derivative is needed to bound the error.) Assume that $.5 \in Z$. That is, we are interested in approximating $u(.5)$.

The function $\Psi_1(.5,x)$ is strictly positive in any interval of the form $[\alpha, 1-\alpha]$ when $\alpha \in (0,.5)$. Let

$$\begin{aligned} \epsilon(\alpha) &= C_1 \cdot \inf_{x \in [\alpha, 1-\alpha]} |\Psi_1(.5,x)| \cdot 2 \cdot (.5-\alpha)^3 \\ &= C_1 \cdot \alpha \cdot (.5-\alpha)^3, \end{aligned}$$

where C_1 is the constant from Theorem 3.1. By Theorem 3.1, if $\epsilon < \epsilon(\alpha)$, then

$$\begin{aligned} (5) \quad N_{\epsilon,1}(u(.5)) &\ge \left\lceil \left(C_1' \cdot \inf_{x \in [\alpha, 1-\alpha]} |\Psi_1(.5,x)| \cdot 2 \cdot (.5-\alpha)^3 \right)^{1/2} \cdot \left(\frac{1}{\epsilon} \right)^{1/2} \right\rceil \\ &= \left\lceil \left(C_1' \cdot \alpha \cdot (.5-\alpha)^3 \right)^{1/2} \cdot \left(\frac{1}{\epsilon} \right)^{1/2} \right\rceil. \end{aligned}$$

On page 89 in Worley [20] we show that $C_1' = C_1 = \frac{1}{3}$ for this problem. Therefore, (5) becomes

$$N_{\epsilon,1}(u(.5)) \ge \left\lceil \left(\frac{\alpha}{3} \cdot (.5-\alpha)^3 \right)^{1/2} \cdot \left(\frac{1}{\epsilon} \right)^{1/2} \right\rceil.$$

This bound is maximized when $\alpha = \frac{1}{8}$. The resulting lower bound on $N_{\epsilon,1}(u(.5))$ is

$$(6) \qquad N_{\epsilon,1}(u(.5)) \ge \left\lceil \frac{3}{64} \cdot \left(\frac{1}{\epsilon} \right)^{1/2} \right\rceil$$

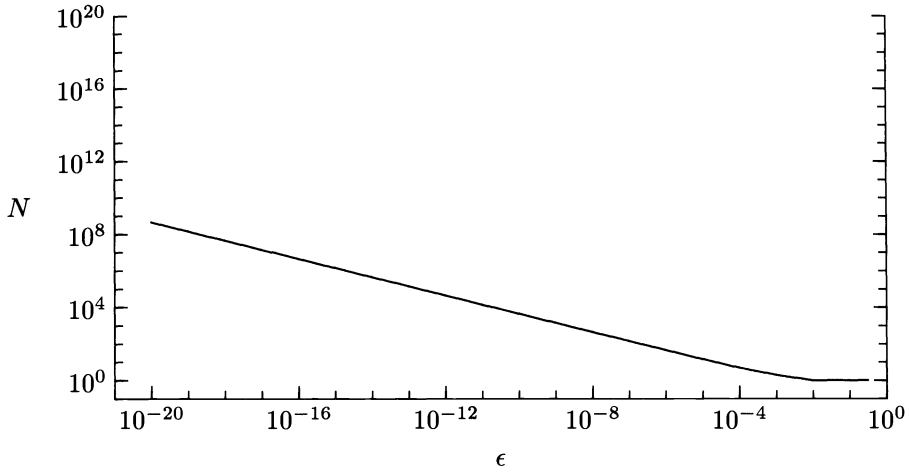


FIG. 1. *Lower bound on $N_{\epsilon,1}(u(.5))$ as a function of ϵ for elliptic example problem.*

when $\epsilon < \epsilon\left(\frac{1}{8}\right) = 9/4096$. Since it is clear that $N_{\epsilon,1}(u(.5)) \geq 1$ for all $\epsilon > 0$, it follows that (6) holds for all $\epsilon > 0$. We will refer to this bound as $N_{\epsilon,1}^{(L)}(u(.5))$. Figure 1 contains the graph of this lower bound as a function of ϵ. The constant multiplying the $\epsilon^{-1/2}$ term in (6) is relatively small, but $N_{\epsilon,1}(u(.5))$ is still guaranteed to be larger than 1000 when ϵ is less than 2.19×10^{-9}. If the functions in G_1 are instead characterized by the condition

$$\left|\frac{d^2}{dx^2}g(x)\right| \leq 200 \quad \forall x \in [0,1],$$

then the lower bound on $N_{\epsilon,1}(u(.5))$ is scaled by a factor of 10. Under this condition $N_{\epsilon,1}(u(.5))$ is guaranteed to be greater than 1000 if $\epsilon < 2.19 \times 10^{-7}$.

By Lemma 2.3,

$$f_{(+)} \cdot \left\lceil \log_2\left(N_{\epsilon,1}^{(L)}(u(.5))\right)\right\rceil \tag{7}$$

is a lower bound on the computation cost of any algorithm that satisfies an error tolerance of ϵ when approximating the solution value $u(.5)$. Figure 2 is a graph of this lower bound. Thus, the computation cost is guaranteed to be greater than $10 \cdot f_{(+)}$ when $\epsilon \leq 10^{-9}$. This bound on the optimal computation cost in (7) is very small unless ϵ is very small, but even an optimal parallel algorithm requires at least

$$\frac{N_{\epsilon,1}^{(L)}(u(.5))}{2}$$

processors in order to achieve this computation cost. If fewer processors collaborate in the approximation of $u(.5)$, then the bound on the computation cost described in Lemma 2.3 will be dominated by the term $f_{(+)} \cdot \left(N_{\epsilon,1}^{(L)}(u(.5)) - 1\right)/P$, where P is the

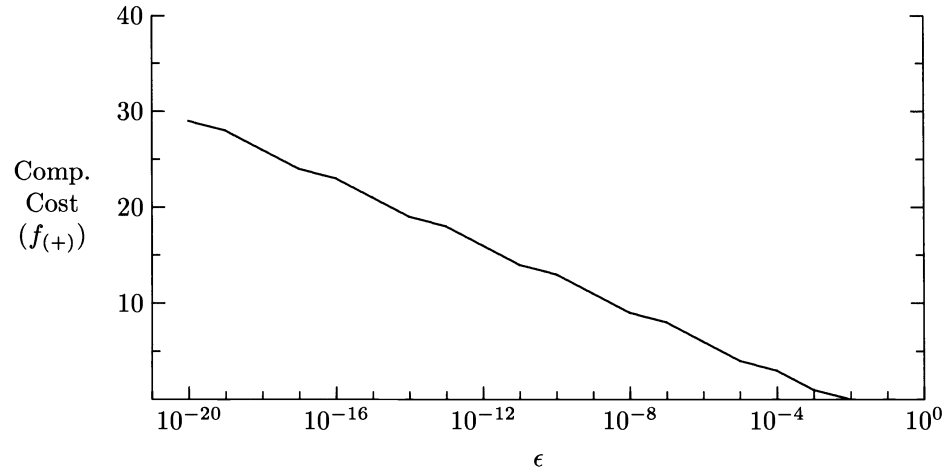


FIG. 2. *Lower bound on computation cost as a function of* ϵ *for elliptic example problem.*

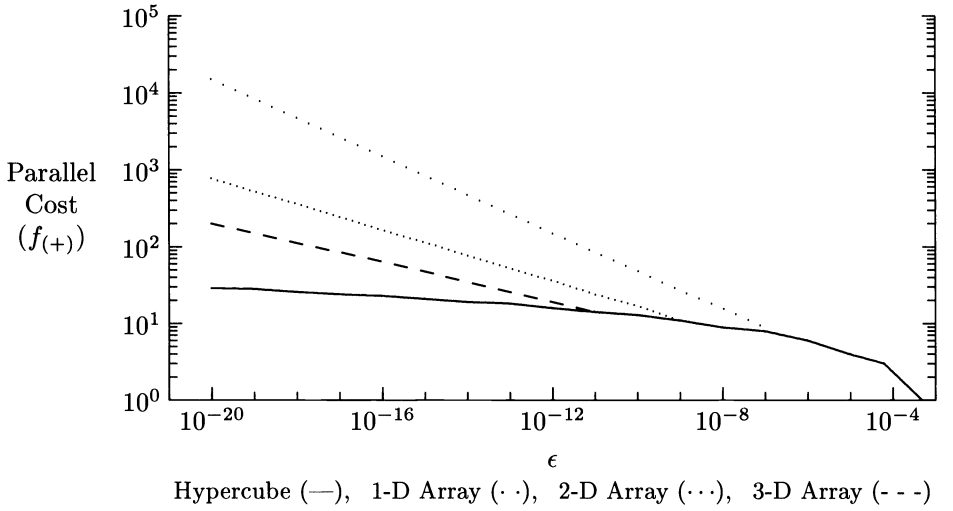


FIG. 3. *Lower bound on parallel cost as a function of* ϵ *for elliptic example problem when* $t = f_{(+)}$.

number of processors. Additionally, a parallel implementation on a multiprocessor will involve communication costs. By Corollary 2.5,

$$\min_{p\in\{1,\cdots,\infty\}} \max\left\{ r(p)\,,\; f_{(+)}\cdot\left\lceil\frac{N_{\epsilon,1}^{(L)}(u(.5))-1}{p}\right\rceil\,,\; f_{(+)}\cdot\left\lceil\log_2\left(N_{\epsilon,1}^{(L)}(u(.5))\right)\right\rceil\right\}$$

is a lower bound on the parallel cost. Assume that the minimum transmission time, t, is equal to the minimum time to execute a floating point addition, $f_{(+)}$, for all multiprocessor architectures under consideration. Then Figure 3 is a graph of this lower bound for a variety of architectures. Note that, when ϵ is small, the lower bound on the parallel cost for a k-dimensional array is significantly larger than the lower bound on the computation cost.

4.2. Algorithmic example. The example in §4.1 illustrated the type of bounds that can be derived by the analysis of §3. These algorithm-independent bounds are quite small for this problem, but Theorem 3.4 indicates that they cannot be significantly improved. This conclusion is verified in Worley [20, pp. 93–98], where the computation cost of a particular parallel algorithm is calculated for the same example and shown to be very close to the lower bound. In this section we show that, by fixing the algorithm, the growth in the execution time as a function of the problem size can be much more dramatic.

Consider the following example of the wave equation in one space dimension:

$$\frac{\partial^2}{\partial t^2}u(x,t)-\frac{\partial^2}{\partial x^2}u(x,t) \;=\; g(x,t) \quad \text{for} \quad (x,t)\in[0,1]\times[0,1],$$

$$u(x,0)=0=\frac{\partial}{\partial t}u(x,0) \quad \text{for} \quad x\in[0,1],$$

$$u(0,t)=0=u(1,t) \quad \text{for} \quad t\in[0,1].$$

Assume that we want the solution on a uniform mesh at time 1,

$$\{\,(j\cdot\Delta x,1)\,|\,j\in\{1,\cdots,N_x\}\,\},$$

where Δx is the distance between consecutive locations and N_x is the number of locations, $N_x+1=1/\Delta x$. Assume that values of the data function are available on a mesh that is uniform in the spatial direction with a separation of Δx, and uniform in the time direction with a separation of Δt,

$$\{\,(j\cdot\Delta x\,,\,k\cdot\Delta t)\;|\;j\in\{1,\cdots,N_x\}\,,\;k\in\{1,\cdots,N_t\}\,\}.$$

Here N_x is as before and $N_t=1/\Delta t$.

We approximate $u(x,t)$ on the mesh at time 1 by timestepping using the following finite-difference formula:

$$\tilde{u}_j^{k+1} \;=\; 2\left(1-\left(\frac{\Delta t}{\Delta x}\right)^2\right)\tilde{u}_j^k \;-\; \tilde{u}_j^{k-1} \;+\; \left(\frac{\Delta t}{\Delta x}\right)^2\left(\tilde{u}_{j+1}^k+\tilde{u}_{j-1}^k\right)+(\Delta t)^2 g_j^k$$

for all $j\in\{1,\cdots,N_x\}$ and for all $k\in\{1,\cdots,N_t\}$, where $\tilde{u}_j^k$ is an approximation to $u(j\cdot\Delta x,k\cdot\Delta t)$ and $g_j^k=g(j\cdot\Delta x,k\cdot\Delta t)$. Thus, the solution is approximated at all

of the locations where the data function is sampled. To ensure stability, we require that $\Delta t/\Delta x \leq 1 - \eta$ for some positive $\eta < 1$.

This scheme requires approximations to the solution at times $k \cdot \Delta t$ and $(k-1) \cdot \Delta t$ in order to calculate an approximation to the solution at time $(k+1) \cdot \Delta t$. To start the process, we define $\tilde{u}_j^{-1}$ to be zero for all j. The serial cost of this calculation is approximately

$$\left(4 \cdot f_{(+)} + 3 \cdot f_{(*)}\right) \cdot N_x \cdot N_t,$$

where $f_{(*)}$ is the time required to calculate a floating point multiplication and $f_{(+)}$ is the time required to calculate a floating point addition or subtraction.

The computation of the approximation at time $(k+1) \cdot \Delta t$ from the approximation at times $k \cdot \Delta t$ and $(k-1) \cdot \Delta t$ is highly parallel. For example, assume that a given multiprocessor has at least the interconnection topology of a one-dimensional array, that computation and communication cannot be overlapped on a given processor, and that the time required to send a message to a neighboring processor is proportional to the size of the message. If P identical processors are available for the computation and N_x/P is an integer, then partition the interval $[0,1]$ into P equal subintervals and assign the calculation of the solution locations in each subinterval to a common processor. By preserving the topology of the problem domain, we can map the subintervals and data onto the processors in such a way that each processor needs to receive only two floating point numbers from its immediate neighbors in order to finish its calculation of the next timestep. Thus, the parallel cost of computing a single timestep is

$$\left(4 \cdot f_{(+)} + 3 \cdot f_{(*)}\right) \cdot \frac{N_x}{P} \; + \; 2 \cdot t,$$

where t is now the time required to send one floating point number to a neighboring processor and receive back another. The overall parallel cost for the computation is

$$\left(\left(4 \cdot f_{(+)} + 3 \cdot f_{(*)}\right) \cdot \frac{N_x}{P} \; + \; 2 \cdot t\right) \cdot N_t. \tag{8}$$

Given our assumptions, this synchronous parallel implementation of the serial algorithm is optimal when N_x/P is an integer. The computation is perfectly parallel per timestep, the load is distributed evenly over the processors, and the time spent in interprocessor communication during a timestep is the minimum possible. If N_x/P is not an integer, then (8) is still a lower bound on the parallel cost, independent of the interconnection network. Even if we allow multiple processors to collaborate in the calculation of a single $\tilde{u}_j^{k+1}$, no more than $3 \cdot N_x$ processors can be effectively used to decrease the computation cost.[7] Thus, a lower bound on the parallel cost of any parallel implementation of this algorithm is

$$\left(f_{(+)} + f_{(*)} + 2 \cdot t\right) \cdot N_t. \tag{9}$$

Since the stability condition constrains N_t to be larger than N_x, the expression

$$\left(f_{(+)} + f_{(*)} + 2 \cdot t\right) \cdot N_x \tag{10}$$

[7] Using multiple processors to calculate a single $\tilde{u}_j^{k+1}$ will increase the communication cost beyond that indicated in this expression.

is also a lower bound on the parallel cost. For fixed processor and communication channel capabilities, the serial cost of this algorithm is a function solely of $N_x \cdot N_t$. Therefore, increasing the serial cost by a factor of K increases one of these two lower bounds by a factor of at least $\sqrt{K}$, independent of the number of processors and of the interconnection topology.

A similar result would hold even if the algorithm were unconditionally stable since increasing N_x will not significantly improve the approximation to the solution unless N_t also increases. For example, assume that this algorithm is strictly stable for all positive Δt and Δx. For a strictly stable algorithm for this PDE, the truncation error times the number of timesteps is a good approximation to the error introduced by discretizing the PDE. Without a priori knowledge of the solution function, the best approximation to this product has the form

$$C \cdot \left((\Delta t)^3 + \Delta t \cdot (\Delta x)^2 \right).$$

Thus, for Δt fixed, the error corresponding to $\Delta x = \Delta t$ is at most twice as large as the error when $\Delta x \rightarrow 0$. Since Assumption 6) requires that the error go to zero when $\Delta x \rightarrow 0$, this implies that Δt must also go to zero. Thus, by the lower bound in (9), the parallel cost grows without bound when the size of the problem grows without bound, in agreement with Theorem 3.6. To maximize the decrease in the error as the size of the problem grows, the ratio $\Delta t / \Delta x$ should be constant. In this case, N_x is a fixed proportion of N_t for any size problem, and increasing the serial cost by a factor of K again increases the lower bound on the parallel cost by a factor of $\sqrt{K}$.

In Gustafson, Montry, and Benner [12] the point is made that increasing the size of the problem often permits the efficient use of a large number of processors. This fact is also clear from this example. To use the processors efficiently when N_x/P is an integer, we simply need to ensure that

$$2 \cdot t \ll \left(4 \cdot f_{(+)} + 3 \cdot f_{(*)} \right) \cdot \frac{N_x}{P} \quad ,$$

for then $E_p \approx 1.0$. This is easily achieved by making $N_x \gg P$. If more processors become available, then the high efficiency can be maintained by keeping N_x/P constant. But, as was noted above, $N_x < N_t$, and the parallel cost is proportional to $N_x \cdot N_t / P$. Thus, maintaining the high efficiency for an increasing number of processors requires that the parallel cost increase at least as fast as the number of processors. This example algorithm is very close to one of the examples in [12], and a similar analysis applies to it. Their other examples have similar behavior as well.

5. Conclusions. The cost of calculating a numerical approximation to the solution of a partial differential equation is constrained by the amount of data that must be used in order to calculate an accurate enough approximation. For the simple assumptions about the data functions and the partial differential equation described in Assumptions 1)–5) and Theorem 3.3 of §3, these constraints limit how quickly the approximation can be calculated. As ϵ, the bound on the error in the approximation, decreases, there exists an algorithm-independent lower bound on the computation cost that must increase at least as fast as $-c \log_2(\epsilon)$ for some positive constant c. When the additional assumptions of Theorem 3.4 also hold, this bound is tight in the sense that there exists a parallel algorithm whose computation cost grows no faster than this. When the interconnection network or the number of processors in the multiprocessor are taken into account, the lower bound on the execution time may increase much more rapidly.

These theoretical results imply that increasingly large instances of a problem cannot be solved in a fixed amount of time no matter how many processors are available. Thus, enabling the use of an increasing number of processors by increasing the size of the problem will conflict with a requirement that the parallel cost remain below a given threshold. This conclusion becomes obvious if any given algorithm is considered, but these results are algorithm-independent. If algorithms are additionally required to be efficient on a multiprocessor as the number of processors increases, then it is clear that problem size must grow even larger, resulting in an even greater increase in the parallel cost.

The results of this paper are easily extended to classes of problems that do not satisfy the assumptions of §3. We must simply show that the error in the individual solution values must go to zero in order for the error in the approximation to go to zero, and that at least one solution value cannot be calculated exactly without an infinite amount of data. This is a common situation when numerically approximating the solution of differential equations.

A. Proof of Theorem 3.1.

A.1. Introduction. Let $u(\bar{z})$ be the solution function for a problem satisfying Assumptions 1)–5). Therefore, $u(\bar{z})$ can be represented by

$$u(\bar{z}) = \sum_{i=1}^{S} \int_{I_{d_i}} \Psi_i(\bar{z}, \bar{x}) g_i(\bar{x})\, d\bar{x} \qquad \forall \bar{z} \in \Omega.$$

Let $u(\bar{z}_j)$ be one of the desired solution function values, $\bar{z}_j \in \Omega$. Let **a** be an algorithm that solves the given problem.

For each $i \in \{1, \cdots, S\}$, let $G_{a,i,j}(g_i)$ be the subset of G_i satisfying the following conditions. If $g \in G_{a,i,j}(g_i)$, then $g \in G_i$,

$$g(\bar{x}_{i,j,k}) = g_i(\bar{x}_{i,j,k}) \qquad \forall \bar{x}_{i,j,k} \in X_{i,j},$$

and g and all derivatives of g are identical to g_i and the corresponding derivatives of g_i on the boundary of I_{d_i}. Thus, algorithm **a** cannot distinguish between the functions in $G_{a,i,j}(g_i)$ because they have the same values at all sampling locations and they satisfy the same compatibility conditions. Let

$$U_{a,j}(u) = \left\{ v(\bar{z}) \;\middle|\; v(\bar{z}) = \sum_{i=1}^{S} \int_{I_{d_i}} \Psi_i(\bar{z}, \bar{x}) \gamma_i(\bar{x})\, d\bar{x}\,, \text{ where } \gamma_i(\bar{x}) \in G_{a,i,j}(g_i)\ \forall i \right\}.$$

Similarly, algorithm **a** cannot distinguish between $u(\bar{z}_j)$ and the other elements of the set $\{v(\bar{z}_j) \,|\, v \in U_{a,j}(u)\}$ given only the information that algorithm **a** uses to approximate $u(\bar{z}_j)$. Define the *intrinsic error* for algorithm **a** when approximating $u(\bar{z}_j)$ to be the uncertainty introduced by using only the associated data function samplings. We will refer to this value as $E_{a,j}(u)$. Then, a lower bound on the intrinsic error is

$$E_{a,j}(u) \;\geq\; \inf_{v_2 \in U_{a,j}(u)} \sup_{v_1 \in U_{a,j}(u)} |v_1(\bar{z}_j) - v_2(\bar{z}_j)|\,. \tag{11}$$

It is clear that the right-hand side of (11) is itself bounded from below by

$$\frac{1}{2} \cdot \sup_{v_1, v_2 \in U_{a,j}(u)} |v_1(\bar{z}_j) - v_2(\bar{z}_j)|. \tag{12}$$

The difference $v_1(\bar{z}_j) - v_2(\bar{z}_j)$ can also be expressed as a sum of component differences of the form

$$\sum_{i=1}^{S} \int_{I_{d_i}} \Psi_i(\bar{z}_j, \bar{x}) \cdot (\gamma_{i,1}(\bar{x}) - \gamma_{i,2}(\bar{x}))\, d\bar{x}. \tag{13}$$

Since all functions in $G_{a,i,j}(g_i)$ satisfy the same compatibility conditions, each term in the sum (13) is independent. Thus, for every $v_1, v_2 \in U_{a,j}(u)$, there will exist $\tilde{v}_1, \tilde{v}_2 \in U_{a,j}(u)$ such that

$$\tilde{v}_1(\bar{z}_j) - \tilde{v}_2(\bar{z}_j) \;=\; \sum_{i=1}^{S} \left| \int_{I_{d_i}} \Psi_i(\bar{z}_j, \bar{x}) \cdot (\gamma_{i,1}(\bar{x}) - \gamma_{i,2}(\bar{x}))\, d\bar{x} \right|, \tag{14}$$

and an equivalent formulation of (12) is

$$\frac{1}{2} \cdot \sum_{i=1}^{S} \sup_{\gamma_{i,1}, \gamma_{i,2} \in G_{a,i,j}(g_i)} \left| \int_{I_{d_i}} \Psi_i(\bar{z}_j, \bar{x}) \cdot (\gamma_{i,1}(\bar{x}) - \gamma_{i,2}(\bar{x}))\, d\bar{x} \right|. \tag{15}$$

See Lemma 1.6 of Worley [20, p. 26] for a more detailed discussion of this equivalence.

If $\gamma_{i,1}, \gamma_{i,2} \in G_{a,i,j}(g_i)$, then the function $(\gamma_{i,1} - \gamma_{i,2})/2$ is an element of G_i, is zero at all of the sampling locations and on the boundary of I_{d_i}, and all of its derivatives are identically zero on the boundary of I_{d_i}. In consequence, if we let $\bar{0}_i$ represent the function that is identically zero in $\Re^{d_i}$, then $(\gamma_{i,1} - \gamma_{i,2})/2 \in G_{a,i,j}(\bar{0}_i)$. Using this fact, we can express the lower bound in (15) in the following form.

LEMMA A.1.

$$E_{a,j}(u) \;\geq\; \sum_{i=1}^{S} \sup_{\gamma_i \in G_{a,i,j}(\bar{0}_i)} \left| \int_{I_{d_i}} \Psi_i(\bar{z}_j, \bar{x}) \gamma_i(\bar{x})\, d\bar{x} \right|.$$

To prove Theorem 3.1, we first construct a particular function $\Gamma_i \in G_{a,i,j}(\bar{0}_i)$ for a component i satisfying the assumptions of the theorem. From this, we calculate a lower bound on $E_{a,j}(u)$. We then minimize this expression over all algorithms using the same amount of data as **a**, producing an algorithm-independent lower bound on the intrinsic error. The proof of the theorem is a simple consequence of this lower bound.

A.2. Algorithm-dependent lower bound. Let i be the index of a component satisfying the assumptions of Theorem 3.1. Therefore, there exists a closed ball $\bar{B}_i(\bar{x}_*; \delta) \subset I_{d_i}$ on which $\Psi_i(\bar{z}_j, \bar{x})$ is either strictly positive or strictly negative. The construction of Γ_i is based on the result in the following lemma. As before, $B_i(\bar{x}; \delta)$ is the open ball of radius δ and center $\bar{x}$ in $\Re^{d_i}$.

LEMMA A.2. *Assume that $d_i > 0$. For any $\bar{x}' \in I_{d_i}$ and $\delta > 0$ such that $B_i(\bar{x}'; \delta) \subset I_{d_i}$, there exists a function $\gamma \in C^\infty(I_{d_i})$ with the following properties:*

(i) $\gamma \in G_i$.

(ii) $\gamma(\bar{x}) = 0$ *for all* $\bar{x} \notin B_i(\bar{x}'; \delta)$.

(iii) $\gamma(\bar{x}) \geq 0$ *for all* $\bar{x} \in B_i(\bar{x}'; \delta)$.

(iv) *There exists a constant $C_i > 0$ dependent only on m_i, d_i, and $\| \cdot \|_{(i)}$ such that*

$$\int_{I_{d_i}} \gamma(\bar{x})\, d\bar{x} \;\geq\; C_i \cdot \delta^{m_i + d_i} \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}'; \delta)} M_i(\bar{x}).$$

The proof of Lemma A.2 is a straightforward, but somewhat lengthy, construction. See Appendix A of Worley [20, pp. 117–125] for the details.

A.2.1. No sampling locations. If there are no sampling locations in the ball, then we can bound the intrinsic error immediately using Lemmas A.1 and A.2.

LEMMA A.3. *If $d_i > 0$ and no sampling locations are located in $B_i(\bar{x}_*; \delta)$, then*

$$E_{a,j}(u) \;\geq\; C_i \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} M_i(\bar{x}) \cdot \delta^{m_i + d_i},$$

where C_i is a positive constant dependent only on m_i, d_i, and $\| \cdot \|_{(i)}$.

Proof. Let $\gamma_{\bar{x}_*;\delta}$ be the function satisfying properties (i)–(iv) of Lemma A.2 on the ball $B_i(\bar{x}_*; \delta)$. Since there are no sampling locations in $B_i(\bar{x}_*; \delta)$, and since $\gamma_{\bar{x}_*;\delta}(\bar{x}) = 0$ for $\bar{x} \notin B_i(\bar{x}_*; \delta)$, $\gamma_{\bar{x}_*;\delta}$ is a member of $G_{a,i,j}(\bar{0})$. Since $\gamma_{\bar{x}_*;\delta}(\bar{x})$ is nonnegative and $\Psi_i(\bar{x}_j, \bar{x})$ is of one sign in $B_i(\bar{x}_*; \delta)$,

$$\int_{\bar{B}_i(\bar{x}_*;\delta)} \Psi_i(\bar{z}_j, \bar{x})\, \gamma_{\bar{x}_*;\delta}(\bar{x})\, d\bar{x} = \pm \int_{\bar{B}_i(\bar{x}_*;\delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \gamma_{\bar{x}_*;\delta}(\bar{x})\, d\bar{x}.$$

Therefore, by Lemma A.1,

$$\begin{aligned}
E_{a,j}(u) &\geq \sup_{\gamma_i \in G_{a,i,j}(\bar{0})} \left| \int_{I_{d_i}} \Psi_i(\bar{z}_j, \bar{x})\, \gamma_i(\bar{x})\, d\bar{x} \right| \\
&\geq \left| \int_{\bar{B}_i(\bar{x}_*;\delta)} \Psi_i(\bar{z}_j, \bar{x})\, \gamma_{\bar{x}_*;\delta}(\bar{x})\, d\bar{x} \right| \\
&\geq \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \int_{B_i(\bar{x}_*;\delta)} \gamma_{\bar{x}_*;\delta}(\bar{x})\, d\bar{x} \\
&\geq C_i \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} M_i(\bar{x}) \cdot \delta^{d_i + m_i}.
\end{aligned}$$

This proves the lemma. □

A.2.2. Voronoi partition. To extend the bound to the case when there are sampling locations in the ball $B_i(\bar{x}_*; \delta)$, we place functions satisfying properties (i)–(iv) of Lemma A.2 between sampling locations. We use a Voronoi partition of $B_i(\bar{x}_*; \delta)$ to identify where to place these functions.

The Voronoi partition of $\Re^{d_i}$ defined by a set of locations W is the union of the set of *Voronoi cells*, $\{V(\bar{w}) \,|\, \bar{w} \in W\}$, and a closed set B. A location is in $V(\bar{w})$ if it is closer to $\bar{w}$ than it is to any other location in the set W, i.e.,

$$V(\bar{w}) = \left\{ \bar{y} \;\middle|\; \|\bar{w} - \bar{y}\|_2 < \|\bar{w}' - \bar{y}\|_2\,,\ \forall\ \bar{w}' \in W - \{\bar{w}\} \right\},$$

where $\| \cdot \|_2$ is the Euclidean norm for $\Re^{d_i}$. B is the complement of the union of the Voronoi cells. Every location in B satisfies the condition that the nearest element of W is not unique. See Worley [20, pp. 35–37] for a more complete discussion.

We refer to $\bar{w}$ as the *center* of the cell $V(\bar{w})$. The *neighbors* of a cell $V(\bar{w})$ are those cells, excepting $V(\bar{w})$, whose closures intersect with the closure of $V(\bar{w})$, i.e.,

$$\left\{ V(\bar{w}') \;\middle|\; \bar{V}(\bar{w}') \cap \bar{V}(\bar{w}) \neq \emptyset\,,\ \forall \bar{w}' \in W - \{\bar{w}\} \right\}.$$

The *neighbors* of a location $\bar{w}$ are those locations $\bar{w}' \in W$ for which $V(\bar{w}')$ is a neighbor of $V(\bar{w})$.

Let the *boundary* of a cell $V(\bar{w})$ be the intersection of B with the closure of $V(\bar{w})$, and refer to it as $\partial V(\bar{w})$. Every location in the boundary of one cell is also in the

boundary of a neighboring cell. Thus, every location in the boundary of $V(\bar{w})$ is equidistant from $\bar{w}$ and the center of one of its neighbors, and is no closer to any other center. Let $r(\bar{w})$ be the maximum distance between $\bar{w}$ and the locations in the boundary of $V(\bar{w})$, i.e.,

$$r(\bar{w}) = \max_{\bar{y} \in \partial V(\bar{w})} \|\bar{w} - \bar{y}\|_2.$$

We will use this function as a measure of the distance between data function sampling locations.

A.2.3. Multiple sampling locations. Assume that $B_i(\bar{x}_*; \delta)$ contains n sampling locations, and let W denote the set of sampling locations. Consider the Voronoi partition of $\Re^{d_i}$ generated by the sampling locations W and the locations on the surface of the ball $B_i(\bar{x}_*; \delta)$. Construct a function $\Gamma_i \in G_{a,i,j}(\bar{0})$ in the following way:

(a) For each Voronoi cell $V(\bar{w})$, $\bar{w} \in W$, identify a location $\bar{y}_{\bar{w}}$ on its boundary where the function

$$\|\bar{y} - \bar{w}\|_2 \quad \text{for } \bar{y} \in \partial V(\bar{w})$$

reaches its maximum, i.e.,

$$\|\bar{y}_{\bar{w}} - \bar{w}\|_2 = r(\bar{w}).$$

(b) Let $\tilde{W} = W$, $W' = \emptyset$, and $\Gamma_{\text{old}} = \bar{0}_i$.

(c) Choose an element $\bar{w}' \in \tilde{W}$ that maximizes $r(\bar{w})$ over the set $\tilde{W}$. Since $\bar{y}_{\bar{w}'} \in \partial V(\bar{w}')$, it is no closer than $r(\bar{w}')$ to any of the other locations in $\tilde{W}$ or $\partial B_i(\bar{x}_*; \delta)$. Therefore $B_i(\bar{y}_{\bar{w}'}; r(\bar{w}'))$ contains no sampling locations. Let $\gamma_{\bar{w}'}$ be the function described in Lemma A.2 for the ball $B_i(\bar{y}_{\bar{w}'}; r(\bar{w}'))$. Add $\bar{w}'$ to W', and set

$$\Gamma_{\text{new}} = \Gamma_{\text{old}} + \gamma_{\bar{w}'}.$$

(d) Let $\Upsilon(\bar{w}')$ be the subset of $\tilde{W}$ such that

$$\|\bar{y}_{\bar{w}'} - \bar{w}\| < 3r(\bar{w}')$$

if $\bar{w} \in \Upsilon(\bar{w}')$. Remove the locations in $\Upsilon(\bar{w}')$ from $\tilde{W}$. Set Γ_{old} equal to Γ_{new}.

(e) If $\tilde{W}$ is the empty set, then set Γ_i equal to Γ_{new} and stop. Otherwise, return to step (c).

Upon termination, $\tilde{W} = \emptyset$ and

$$\Gamma_i(\bar{x}) = \sum_{\bar{w}' \in W'} \gamma_{\bar{w}'}(\bar{x}). \tag{16}$$

Let n' be the number of elements in W'. It is straightforward to show that this algorithm for constructing Γ_i is well defined and that Γ_i has the following properties.

LEMMA A.4. *$\Gamma_i(\bar{x})$ is a nonnegative function in $G_{a,i,j}(\bar{0})$ that can be represented as the sum of a finite number of nonnegative functions $\{\gamma_k\}$ each of which satisfies the following two conditions:*

(i) $\gamma_k \in G_{a,i,j}(\bar{0})$.

(ii) *The support of γ_k is disjoint from the support of the other functions in the set $\{\gamma_k\}$.*

See Lemmas 2.6, 2.7, and 2.8 in Worley [20, pp. 40–42] for the proof. We can also show that the support of Γ_i covers a significant portion of $B_i(\bar{x}_*; \delta)$.

LEMMA A.5.

$$B_i(\bar{x}_*; \delta) \subseteq \bigcup_{\bar{w}' \in W'} B_i(\bar{y}_{\bar{w}'}; 5r(\bar{w}')).$$

See Appendix B in Worley [20, pp. 127–130] for the proof.

A.2.4. Lower bound on intrinsic error. Using Γ_i, we can now describe a nontrivial lower bound on $E_{a,j}(u)$ for any number of sampling locations in $B(\bar{x}_*; \delta)$.

THEOREM A.6. *If $d_i > 0$ and n sampling locations lie in $B_i(\bar{x}_*; \delta)$, then $E_{a,j}(u)$ is bounded from below by*

$$C_i \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*; \delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*; \delta)} M_i(\bar{x}) \cdot \begin{cases} \delta^{m_i + d_i} & \text{if } n = 0, \\ \sum_{k=1}^{n} r_k^{m_i + d_i} & \text{if } n > 0 \end{cases}$$

where

$$\sum_{k=1}^{n} r_k^{d_i} \geq \left(\frac{\delta}{5}\right)^{d_i} \quad \text{if } n > 0$$

and

$$0 \leq r_k \leq \delta \quad \text{for } 1 \leq k \leq n.$$

C_i is the constant from Lemma A.2 for a d_i-dimensional domain.

Proof. If $n = 0$, then the theorem follows directly from Lemma A.3. If $n > 0$, then let $W' = \{\bar{w}'_k \mid k \in \{1, \cdots, n'\}\}$ be the sampling locations used in the construction of Γ_i.

By (16) and Lemmas A.1, A.2, and A.4,

$$\begin{aligned}
E_{a,j}(u) &\geq \sup_{\gamma_i \in G_{a,i,j}(\bar{0})} \left| \int_{I_{d_i}} \Psi_i(\bar{z}_j, \bar{x})\, \gamma_i(\bar{x})\, d\bar{x} \right| \\
&\geq \left| \int_{\bar{B}_i(\bar{x}_*; \delta)} \Psi_i(\bar{z}_j, \bar{x})\, \Gamma_i(\bar{x})\, d\bar{x} \right| \\
&\geq \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*; \delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \left| \int_{B_i(\bar{x}_*; \delta)} \Gamma_i(\bar{x})\, d\bar{x} \right| \\
&\geq \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*; \delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \sum_{k=1}^{n'} \int_{B_i(\bar{y}_{\bar{w}'_k}; r(\bar{w}'_k))} \gamma_{\bar{w}'_k}(\bar{x})\, d\bar{x} \\
&\geq C_i \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_k; \delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*; \delta)} M_i(\bar{x}) \cdot \sum_{k=1}^{n'} r(\bar{w}'_k)^{d_i + m_i}.
\end{aligned}$$

By Lemma A.5,

$$B_i(\bar{x}_*; \delta) \subseteq \bigcup_{k=1}^{n'} B_i(\bar{y}_{\bar{w}'_k}; 5r(\bar{w}'_k)).$$

Therefore,

$$\mathrm{Vol}\left(B_i(\bar{x}_*;\delta)\right) \leq \bigcup_{k=1}^{n'} \mathrm{Vol}\left(B_i(\bar{y}_{\bar{w}'_k}; 5r(\bar{w}'_k))\right),$$

or

$$\delta^{d_i} \leq \sum_{k=1}^{n'} (5r(\bar{w}'_k))^{d_i}.$$

Setting $r_k = r(\bar{w}'_k)$ for $1 \leq k \leq n'$ and $r_k = 0$ for $k > n'$ produces the desired bound, and proves the theorem. □

A.3. Algorithm-independent lower bound. We can use Theorem A.6 to establish a lower bound on $E_{a,j}(u)$ that is a function only of the number of sampling locations used by the algorithm. The following lemma establishes how to do this.

LEMMA A.7. *If $m_i \geq 1$ and $d_i \geq 1$, then the problem*

$$\min \sum_{k=1}^{n} r_k^{d_i+m_i}$$

subject to

$$\sum_{k=1}^{n} r_k^{d_i} \geq \left(\frac{\delta}{5}\right)^{d_i} \quad \text{and} \quad r_k \geq 0 \qquad \forall k \in \{1, \cdots, n\},$$

is solved when

$$r_k = \frac{\delta}{5} \cdot \left(\frac{1}{n}\right)^{1/d_i} \qquad \forall k \in \{1, \cdots, n\}.$$

The minimum value of the objective function is

$$\left(\frac{1}{5}\right)^{d_i+m_i} \cdot \left(\frac{\delta}{n^{1/d_i}}\right)^{m_i} \cdot \delta^{d_i}.$$

Proof. Add the constraint that $\sum_{k=1}^{n} r_k^{d_i} = q$ for some given $q \geq (\delta/5)^{d_i}$. By Lagrange's method [4] and inspection of the objective function, we know that the minimum of the augmented problem must satisfy

$$\frac{\partial}{\partial r_k}\left(\sum_{l=1}^{n} r_l^{d_i+m_i} + \lambda \cdot \sum_{l=1}^{n} r_l^{d_i}\right) = 0 \qquad \forall k$$

$$\lambda \cdot \sum_{l=1}^{n} r_l^{d_i} = q.$$

The unique solution to this condition is

$$r_k = (q/n)^{1/d_i} \qquad \forall k. \tag{17}$$

The corresponding value of the objective function is

$$q \cdot (q/n)^{m_i/d_i}. \tag{18}$$

The expression (18) is minimized by choosing $q = (\delta/5)^{d_i}$. For this value of q, (18) is a lower bound on the objective function for any feasible set of $\{r_k\}$. Since this lower bound is achieved for the r_k values described in (17), this must be the minimum value for the objective in the original minimization problem. □

The following theorem is an immediate consequence of Lemma A.7 and Theorem A.6.

THEOREM A.8. *If $d_i > 0$ and $n > 0$, then*

$$E_{a,j}(u) \geq C_i \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} M_i(\bar{x}) \cdot \left(\frac{1}{5}\right)^{d_i+m_i} \cdot \left(\frac{\delta}{n^{1/d_i}}\right)^{m_i} \cdot \delta^{d_i}$$

for any algorithm **a** *with n sampling locations in $B_i(\bar{x}_*;\delta)$. C_i is the constant from Lemma* A.2.

A.4. Proof of theorem. If $d_i = 0$, then $\delta \equiv 0$ and Theorem 3.1 is trivially true. Therefore, assume that $d_i > 0$. By Lemma A.3, if no sampling locations are located in $B_i(\bar{x}_*;\delta)$, then the intrinsic error will be at least as large as

$$C_i \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} M_i(\bar{x}) \cdot \delta^{m_i+d_i}.$$

If this is too large to satisfy the given error tolerance ϵ, then at least one data function sampling location must be located in $B_i(\bar{x}_*;\delta)$. In this case, by Theorem A.8,

$$C_i \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} M_i(\bar{x}) \cdot \left(\frac{1}{5}\right)^{d_i+m_i} \cdot \left(\frac{\delta}{n^{1/d_i}}\right)^{m_i} \cdot \delta^{d_i} \leq \epsilon$$

must hold in order to satisfy the error bound. In consequence, there must be at least

$$\left\lceil \left(C_i \cdot \left(\frac{1}{5}\right)^{d_i+m_i} \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} |\Psi_i(\bar{z}_j, \bar{x})| \cdot \inf_{\bar{x} \in \bar{B}_i(\bar{x}_*;\delta)} M_i(\bar{x}) \cdot \delta^{m_i+d_i} \right)^{d_i/m_i} \cdot \left(\frac{1}{\epsilon}\right)^{d_i/m_i} \right\rceil$$

sampling locations in $B(\bar{x}_*;\delta)$. This proves Theorem 3.1. □

B. Proof of Theorem 3.4.

B.1. Introduction. The lower bound in the statement of Theorem 3.4 is a direct consequence of Theorem 3.3. The upper bound is established by constructing an algorithm that satisfies the error tolerance and for which there exists a parallel implementation whose computation cost is of the indicated form. The algorithm is based on using piecewise polynomial interpolants of the data functions in the following way. For each i, let $\tilde{g}_i$ be an approximation to the data function g_i calculated from values of g_i taken at a finite number of sampling locations in I_{d_i}. Define the approximation $\tilde{u}(\bar{z}_j)$ to be the sum of components $\{\tilde{u}_i(\bar{z}_j) \mid i \in \{1, \cdots, S\}\}$, where $\tilde{u}_i(\bar{z}_j)$ is defined by

$$\tilde{u}_i(\bar{z}_j) = \int_{I_{d_i}} \Psi_i(\bar{z}_j, \bar{x}) \tilde{g}_i(\bar{x})\, d\bar{x}.$$

Then, this construction corresponds to an algorithm **a** satisfying Assumptions 3), 4), and 5) if

$$|u_i(\bar{z}_j) - \tilde{u}_i(\bar{z}_j)| \leq \frac{\epsilon}{S}$$

for all i. Since

$$\begin{aligned} |u_i(\bar{z}_j) - \tilde{u}_i(\bar{z}_j)| &= \left| \int_{I_{d_i}} \Psi_i(\bar{z}_j, \bar{x}) \cdot (g_i(\bar{x}) - \tilde{g}_i(\bar{x}))\, d\bar{x} \right| \\ &\leq \int_{I_{d_i}} |\Psi_i(\bar{z}_j, \bar{x})| \cdot |g_i(\bar{x}) - \tilde{g}_i(\bar{x})|\, d\bar{x}, \end{aligned} \tag{19}$$

the error in approximating the solution can be bounded by a function of the error in approximating the data functions. The next section describes bounds on the error incurred in using interpolants to approximate the data.

B.2. Interpolation results. Let $K_{d_i}(m_i)$ be the set of d_i-vectors whose components are nonnegative integers that sum to m_i, i.e.,

$$K_{d_i}(m_i) = \left\{ \bar{\mu} \,\middle|\, \bar{\mu} = (\mu_1, \cdots, \mu_{d_i}) \text{ and } \sum_{l=1}^{d_i} \mu_l = m_i \right\}.$$

Let $\bar{x} = (x_1, \cdots, x_{d_i})$. Let $\bar{x}^{\bar{\mu}} = x_1^{\mu_1} x_2^{\mu_2} \cdots x_{d_i}^{\mu_{d_i}}$. Represent a particular m_ith order partial derivative of a function $g \in C^{m_i}(\Re^{d_i})$ by

$$\frac{\partial^{\bar{\mu}}}{\partial \bar{x}^{\bar{\mu}}} g(\bar{x}) = \frac{\partial^{\mu_1}}{\partial x_1^{\mu_1}} \cdots \frac{\partial^{\mu_{d_i}}}{\partial x_{d_i}^{\mu_{d_i}}} g(\bar{x}),$$

where $\bar{\mu} \in K_{d_i}(m_i)$. This is the multi-index notation of Schwartz. See John [16].

Let $C_i(\bar{x}'; \delta)$ be the open d_i-dimensional cube centered on $\bar{x}'$ with volume δ^{d_i}. Let $h = \delta/(m_i + 1)$. Let

$$x_{(s,k)} = (\bar{x}')_s - \frac{\delta}{2} + k \cdot h$$

for $1 \leq s \leq d_i$, where $(\bar{x}')_s$ is the sth component of the vector $\bar{x}'$. Let π_s be the following set of equally spaced locations in the interval $[(\bar{x}')_s - \delta/2, (\bar{x}')_s + \delta/2]$,

$$\{x_{(s,1)}, \cdots, x_{(s,m_i)}\}.$$

Define $\pi(d_i)$ to be the following set of $m_i^{d_i}$ locations in $C_i(\bar{x}'; \delta)$.

$$\begin{aligned} \pi(d_i) &= \pi_1 \times \cdots \times \pi_{d_i} \\ &= \{\bar{x} \,|\, (\bar{x})_s \in \pi_s\}. \end{aligned} \tag{20}$$

LEMMA B.1. *Let g be a function defined in a d_i-dimensional cube $C_i(\bar{x}'; \delta)$. Define $\pi(d_i)$ to be the set of equidistributed locations in $C_i(\bar{x}'; \delta)$ described by* (20). *Then there exists a unique polynomial $\tilde{g}(\bar{x})$ of degree at most $m_i - 1$ in each variable that interpolates g at the locations in $\pi(d_i)$. If g has (m_i)th order continous partial derivatives in $C_i(\bar{x}'; \delta)$, then the pointwise error in using the interpolant to approximate g in $C_i(\bar{x}'; \delta)$ is bounded from above by a term of the form*

$$\tilde{C}_i \cdot \delta^{m_i} \cdot \max_{\bar{x} \in \bar{C}(\bar{x}'; \delta)} \max_{\bar{\mu} \in K_{d_i}(m_i)} \left| \frac{\partial^{\bar{\mu}}}{\partial \bar{x}^{\bar{\mu}}} g(\bar{x}) \right|,$$

where $\tilde{C}_i$ is independent of δ and $\bar{x}'$.

This is a variation of the results in Chapter 5 of Prenter [18]. See Appendix C in Worley [20, pp. 131–134] for a discussion of the proof.

Let g_i be a function in G_i. Let $C_i(\bar{x}';\delta)$ be some d_i-dimensional cube in I_{d_i}. By Lemma B.1, there exists a unique polynomial that interpolates the function $g_i(\bar{x})$ at the locations $\pi(d_i)$ described above. Call this function $\tilde{g}_i(\bar{x})$. By the equivalence of finite-dimensional vector norms [5, p. 414], there exists a constant $\tilde{C}_i'$ such that

$$\tilde{C}_i \cdot \max_{\bar{\mu}\in K_{d_i}(m_i)} \left| \frac{\partial^{\bar{\mu}}}{\partial \bar{x}^{\bar{\mu}}} g_i(\bar{x}) \right| \leq \tilde{C}_i' \cdot \|\nabla_{d_i}^{(m_i)} g_i(\bar{x})\|_{(i)}$$

for all $\bar{x} \in I_{d_i}$, where $\tilde{C}_i$ is defined in Lemma B.1. Therefore, the error bound on the approximation described in Lemma B.1 becomes

$$|g_i(\bar{x}) - \tilde{g}_i(\bar{x})| \leq \tilde{C}_i' \cdot \delta^{m_i} \cdot \sup_{\bar{x}\in\bar{C}(\bar{x}';\delta)} M_i(\bar{x}) \tag{21}$$

when $\bar{x} \in C(\bar{x}';\delta)$.

B.3. Serial algorithm. Define an algorithm **a** for approximating $u(\bar{z}_j)$ in the following way. For each i such that $d_i = 0$, compute the component $u_i(\bar{z}_j)$ exactly:

$$\tilde{u}_{a,i}(\bar{z}_j) \equiv u_i(\bar{z}_j) = \Psi_i(\bar{z}_j) \cdot g_i.$$

For each i such that $d_i > 0$, approximate $u_i(\bar{z}_j)$ in the following way.

(a) Divide I_{d_i} into cubes of the form $C_i(\bar{x}_l; 2^{-\nu_{a,i}})$, for some positive integer $\nu_{a,i}$.
(b) In each subcube $C_i(\bar{x}_l; 2^{-\nu_{a,i}})$, let $\tilde{g}_{a,i,l}(\bar{x})$ be the approximation to $g_i(\bar{x})$ defined in Lemma B.1.
(c) Approximate $u_i(\bar{z}_j)$ by the following expression:

$$\tilde{u}_{a,i}(\bar{z}_j) = \sum_{l=1}^{2^{\nu_{a,i}\cdot d_i}} \int_{C_i(\bar{x}_l;2^{-\nu_{a,i}})} \Psi_i(\bar{z}_j, \bar{x}) \tilde{g}_{a,i,l}(\bar{x})\, d\bar{x}. \tag{22}$$

Finally, approximate $u(\bar{z}_j)$ by the following expression:

$$\tilde{u}_a(\bar{z}_j) = \sum_{i=1}^{S} \tilde{u}_{a,i}(\bar{z}_j).$$

Let $\pi_l(d_i) = \{\bar{x}_{i,l,k}\}$ be the set of sampling locations in $C_i(\bar{x}_l; 2^{-\nu_{a,i}\cdot d_i})$. Let $\Lambda_{i,l,k}(\bar{x})$ be the unique polynomial of degree at most $m_i - 1$ in each variable such that

$$\Lambda_{i,l,k}(\bar{x}) = \begin{cases} 1 & \text{if } \bar{x} = \bar{x}_{i,l,k}\,, \\ 0 & \text{if } \bar{x} \in \pi_l(d_i)\,,\ \bar{x} \neq \bar{x}_{i,l,k}\,. \end{cases}$$

Then $\tilde{g}_{a,i,l}(\bar{x})$ can be represented by

$$\sum_{k=1}^{m_i^{d_i}} g_i(\bar{x}_{i,l,k}) \cdot \Lambda_{i,l,k}(\bar{x}).$$

See Appendix C in Worley [20, pp. 131–134] and Prenter [18] for a more detailed discussion of this representation of the interpolating polynomial. Using this notation, the expression for $\tilde{u}_{a,i}(\bar{z}_j)$ in (22) becomes

$$\begin{aligned}\tilde{u}_{a,i}(\bar{z}_j) &= \sum_{l=1}^{2^{\nu_{a,i}\cdot d_i}} \int_{C_i(\bar{x}_l;2^{-\nu_{a,i}\cdot d_i})} \Psi_i(\bar{z}_j,\bar{x}) \cdot \left(\sum_{k=1}^{m_i^{d_i}} g_i(\bar{x}_{i,l,k}) \cdot \Lambda_{i,l,k}(\bar{x}) \right) d\bar{x} \\ &= \sum_{l=1}^{2^{\nu_{a,i}\cdot d_i}} \sum_{k=1}^{m_i^{d_i}} \left(g_i(\bar{x}_{i,l,k}) \cdot \int_{C_i(\bar{x}_l;2^{-\nu_{a,i}\cdot d_i})} \Psi_i(\bar{z}_j,\bar{x}) \cdot \Lambda_{i,l,k}(\bar{x})\, d\bar{x} \right).\end{aligned}$$

Each factor $\int_{C_i(\bar{x}_l;2^{-\nu_{a,i}\cdot d_i})} \Psi_i(\bar{z}_j,\bar{x}) \cdot \Lambda_{i,l,k}(\bar{x})\, d\bar{x}$ is independent of the data, and can be precomputed. Call it $r_{a,i,l,k}(\bar{z}_j)$. The expression for $\tilde{u}_{a,i}(\bar{z}_j)$ is then

$$\tilde{u}_{a,i}(\bar{z}_j) = \sum_{l=1}^{2^{\nu_{a,i}\cdot d_i}} \sum_{k=1}^{m_i^{d_i}} r_{a,i,l,k}(\bar{z}_j) \cdot g_i(\bar{x}_{i,l,k}).$$

Similarly, the expression for the approximation to $u(\bar{z}_j)$ is now

$$\tilde{u}_a(\bar{z}_j) = \sum_{i=1}^{S} \sum_{l=1}^{2^{\nu_{a,i}\cdot d_i}} \sum_{k=1}^{m_i^{d_i}} r_{a,i,l,k}(\bar{z}_j) \cdot g_i(\bar{x}_{i,l,k}). \tag{23}$$

B.4. Bounds on sample size. A simple upper bound on the error in the approximation to $u_i(\bar{z}_j)$ generated by **a** follows directly from (19) and (21).

LEMMA B.2.

$$|u_i(\bar{z}_j) - \tilde{u}_{a,i}(\bar{z}_j)| \leq \tilde{C}_i' \cdot 2^{-\nu_{a,i}\cdot m_i} \cdot \sup_{\bar{x}\in I_{d_i}} M_i(\bar{x}) \cdot \int_{I_{d_i}} |\Psi_i(\bar{z}_j,\bar{x})|\, d\bar{x}.$$

For each component, algorithm **a** uses $2^{\nu_{a,i}\cdot d_i}$ subcubes, and a total of $m_i^{d_i} \cdot 2^{\nu_{a,i}\cdot d_i}$ sampling locations. As mentioned in §B.1, the algorithm will satisfy the error tolerance if the error in each component is no more than ϵ/S. The following lemma describes a bound on the number of sampling locations per component needed to insure this.

LEMMA B.3. *There exists a finite positive constant $\tilde{C}_i''(\bar{z}_j)$ independent of M_i and ϵ such that no more than*

$$\tilde{C}_i''(\bar{z}_j) \cdot \sup_{\bar{x}\in I_{d_i}} M_i^{d_i/m_i}(\bar{x}) \cdot \left(\frac{1}{\epsilon}\right)^{d_i/m_i} + 1$$

sampling locations in I_{d_i} are needed by algorithm **a** *for the error in the ith component to be less than ϵ/S.*

Proof. If $d_i = 0$, then a single sampling location is needed to calculate the component exactly, and the bound follows immediately for any positive $\tilde{C}_i''(\bar{z}_j)$. Assume that $d_i > 0$.

From Lemma B.2, a sufficient condition to satisfy the error criterion is

$$\tilde{C}_i' \cdot 2^{-\nu_{a,i}\cdot m_i} \cdot \sup_{\bar{x}\in I_{d_i}} M_i(\bar{x}) \cdot \int_{I_{d_i}} |\Psi_i(\bar{z}_j,\bar{x})|\, d\bar{x} \leq \frac{\epsilon}{S}.$$

Therefore,

$$\left(\frac{S\cdot \tilde{C}_i'\cdot \sup_{\bar{x}\in I_{d_i}} M_i(\bar{x})\cdot \int_{I_{d_i}} |\Psi_i(\bar{z}_j,\bar{x})|\,d\bar{x}}{\epsilon}\right)^{d_i/m_i} \leq 2^{\nu_{a,i}\cdot d_i} \tag{24}$$

is also a sufficient condition. If

$$\nu_{a,i} = \max\left\{1 + \left\lfloor \frac{1}{m_i}\cdot \log_2\left(\frac{S\cdot \tilde{C}_i'\cdot \sup_{\bar{x}\in I_{d_i}} M_i(\bar{x}) \int_{I_{d_i}} |\Psi_i(\bar{z}_j,\bar{x})|\,d\bar{x}}{\epsilon}\right)\right\rfloor, 0\right\},$$

then (24) is satisfied. Since the total number of sampling locations in I_{d_i} is $m_i^{d_i}\cdot 2^{\nu_{a,i}\cdot d_i}$, setting

$$\tilde{C}_i''(\bar{z}_j) = 2^{d_i}\cdot m_i^{d_i}\cdot \left(S\cdot \tilde{C}_i'\cdot \int_{I_{d_i}} |\Psi_i(\bar{z}_j,\bar{x})|\,d\bar{x}\right)^{d_i/m_i}.$$

is sufficient to establish the theorem. □

B.5. Parallel algorithm. Let $N_a(u(\bar{z}_j)) = \sum_{i=1}^{S} m_i^{d_i}\cdot 2^{\nu_{a,i}\cdot d_i}$. Then, using algorithm **a** to approximate $u(\bar{z}_j)$ requires $N_a(u(\bar{z}_j))$ floating point multiplications and $N_a(u(\bar{z}_j)) - 1$ floating point additions. To implement algorithm **a** on a P-processor multiprocessor, first partition the data $\{g_i(\bar{x}_{i,l,k})\}$ into P subsets each of which contains no more than $\lceil N_a(u(\bar{z}_j))/P\rceil$ elements and allocate a different processor to each subset. Next, in parallel, use the processor allocated to a given subset to calculate the portion of the weighted sum in (23) corresponding to the subset. Finally, sum the remaining $\min\{N_a(u(\bar{z}_j)), P\}$ partial results using a parallel fan-in algorithm [17]. The computation cost of this parallel implementation of the algorithm is

$$C_{p,a} \equiv \left\lceil \frac{N_a(u(\bar{z}_j))}{P}\right\rceil \cdot (f_{(*)} + f_{(+)}) + \left(\left\lceil \log_2\left(\min\{N_a(u(\bar{z}_j)), P\}\right)\right\rceil - 1\right)\cdot f_{(+)}, \tag{25}$$

where $f_{(*)}$ and $f_{(+)}$ are the times required to execute a floating point multiplication and a floating point addition, respectively.

B.6. Proof of theorem. For each i, choose the minimum nonnegative value $\nu_{a,i}$ such that

$$|u_i(\bar{z}_j) - \tilde{u}_i(\bar{z}_j)| \leq \frac{\epsilon}{S}.$$

Then, algorithm **a** satisfies Assumptions 3), 4), and 5). By Lemma B.3,

$$\begin{aligned} N_a(u(\bar{z}_j)) &= \sum_{i=1}^{S} m_i^{d_i}\cdot 2^{\nu_{a,i}\cdot d_i} \\ &\leq \sum_{i=1}^{S}\left(\tilde{C}_i''(\bar{z}_j)\cdot \sup_{\bar{x}\in I_{d_i}} M_i^{d_i/m_i}(\bar{x})\cdot \epsilon^{-d_i m_i} + 1\right) \\ &\leq S\cdot\left(1 + \max_{i\in\{1,\cdots,S\}}\left(\tilde{C}_i''(\bar{z}_j)\cdot \sup_{\bar{x}\in I_{d_i}} M_i^{d_i/m_i}(\bar{x})\cdot \epsilon^{-d_i/m_i}\right)\right). \end{aligned}$$

Let ϵ_* be the constant from Theorem 3.3. Define σ_* by

$$\sigma_* = \min\left\{\epsilon_*, 1, \max_{i\in\{1,\cdots,S\}}\left(\left(\tilde{C}_i''(\bar{z}_j)\right)^{m_i/d_i}\cdot\sup_{\bar{x}\in I_{d_i}} M_i(\bar{x})\right)\right\}.$$

Note that σ_* is always positive by the assumptions of Theorem 3.4. If $\epsilon < \sigma_*$, then

$$\begin{aligned}(26)\quad N_a(u(\bar{z}_j)) &\leq 2\cdot S\cdot\max_{i\in\{1,\cdots,S\}}\left(\tilde{C}_i''(\bar{z}_j)\cdot\sup_{\bar{x}\in I_{d_i}} M_i^{d_i/m_i}(\bar{x})\cdot\epsilon^{-d_i/m_i}\right)\\ &\leq 2\cdot S\cdot\max_{i\in\{1,\cdots,S\}}\left(\tilde{C}_i''(\bar{z}_j)\cdot\sup_{\bar{x}\in I_{d_i}} M_i^{d_i m_i}(\bar{x})\right)\cdot\max_{i\in\{1,\cdots,S\}}\epsilon^{-d_i/m_i},\end{aligned}$$

$$(27)\qquad \max_{i\in\{1,\cdots,S\}}\log_2\left(\epsilon^{-d_i/m_i}\right) \geq 0,$$

and

$$(28)\qquad \log_2\left(2\cdot S\cdot\max_{i\in\{1,\cdots,S\}}\left(\tilde{C}_i''(\bar{z}_j)\cdot\sup_{\bar{x}\in I_{d_i}} M_i^{d_i/m_i}(\bar{x})\right)\right) \leq \max_{i\in\{1,\cdots,S\}}\log_2\left(\epsilon^{-d_i/m_i}\right).$$

Let

$$Q = 2\cdot S\cdot\max_{i\in\{1,\cdots,S\}}\left(\tilde{C}_i''(\bar{z}_j)\cdot\sup_{\bar{x}\in I_{d_i}} M_i^{d_i/m_i}(\bar{x})\right).$$

Then, by (26), (27), (28), and the expression for the computation cost given in (25),

$$\begin{aligned}C_{p,a} &= \left\lceil\frac{N_a(u(\bar{z}_j))}{P}\right\rceil\cdot(f_{(*)}+f_{(+)}) + \left(\left\lceil\log_2\left(\min\{N_a(u(\bar{z}_j)), P\}\right)\right\rceil - 1\right)\cdot f_{(+)}\\ &\leq (f_{(*)}+f_{(+)})\cdot\left(\left\lceil\frac{N_a(u(\bar{z}_j))}{P}\right\rceil + (\log_2 N_a(u(\bar{z}_j)))\right)\\ &\leq (f_{(*)}+f_{(+)})\cdot\left(\left\lceil\frac{Q\cdot\max_{i\in\{1,\cdots,S\}}\epsilon^{-d_i/m_i}}{P}\right\rceil + \log_2\left(Q\cdot\max_{i\in\{1,\cdots,S\}}\epsilon^{-d_i/m_i}\right)\right)\\ &\leq (f_{(*)}+f_{(+)})\cdot\max\{\lceil Q\rceil,1\}\cdot\max_{i\in\{1,\cdots,S\}}\left(\left\lceil\frac{\epsilon^{-d_i/m_i}}{P}\right\rceil + 2\cdot\log_2\epsilon^{-d_i/m_i}\right)\\ &\leq (f_{(*)}+f_{(+)})\cdot\max\{\lceil Q\rceil,1\}\cdot 3\cdot\max_{i\in\{1,\cdots,S\}}\left(\max\left\{\left\lceil\frac{\epsilon^{-d_i/m_i}}{P}\right\rceil, \log_2\epsilon^{-d_i/m_i}\right\}\right)\end{aligned}$$

when $\epsilon < \sigma_*$. In consequence, the upper bound in Theorem 3.4 holds when

$$\beta = 3\cdot(f_{(+)}+f_{(*)})\cdot\max\left\{\left\lceil 2\cdot S\cdot\max_{i\in\{1,\cdots,S\}}\left(\tilde{C}_i''(\bar{z}_j)\cdot\sup_{\bar{x}\in I_{d_i}} M_i^{d_i/m_i}(\bar{x})\right)\right\rceil, 1\right\}$$

and $\epsilon < \sigma_*$. Since the lower bound in Theorem 3.4 follows directly from Theorem 3.3, this proves Theorem 3.4. □

Acknowledgments. We thank John Drake, Michael Heath, Esmond Ng, and the anonymous referees for their helpful suggestions on the presentation of the material in this paper.

REFERENCES

[1] G. S. ALMASI AND A. GOTTLEIB, *Highly Parallel Computing*, Benjamin/Cummings, Redwood City, CA, 1989.
[2] G. AMDAHL, *The validity of the single processor approach to achieving large scale computing capabilities*, AFIPS Conference Proceedings, 30 (1967), pp. 483–485.
[3] ———, *Limits of expectation*, Internat. J. Supercomputer Appl., 2 (1988), pp. 88–97.
[4] T. M. APOSTOL, *Mathematical Analysis*, Second Edition, Addison-Wesley, Reading, MA, 1974.
[5] K. E. ATKINSON, *An Introduction to Numerical Analysis*, John Wiley, New York, 1978.
[6] G. M. BAUDET, *Asynchronous iterative methods for multiprocessors*, J. Assoc. Comput. Mach., 25 (1978), pp. 226–244.
[7] A. G. BUTKOVSKIY, *Green's Functions and Transfer Functions Handbook*, Ellis Horwood Limited, Chichester, West Sussex, UK, 1982.
[8] D. CHAZAN AND W. MIRANKER, *Chaotic relaxation*, Linear Algebra Appl., 2 (1969), pp. 199–222.
[9] T. FENG, *A survey of interconnection networks*, IEEE Comput., 14 (1981), pp. 12–27.
[10] M. J. FLYNN, *Some computer organizations and their effectiveness*, IEEE Trans. Comput., 21 (1972), pp. 948–960.
[11] A. GOTTLIEB, R. GRISHMAN, C. P. KRUSKAL, K. P. MCAULIFFE, L. RUDOLF, AND M. SNIR, *The* NYU *Ultracomputer—designing an* MIMD *shared memory parallel computer*, IEEE Trans. Comput., 32 (1983), pp. 175–189.
[12] J. L. GUSTAFSON, G. R. MONTRY, AND R. E. BENNER, *Development of parallel methods for a 1024-processor hypercube*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 609–638.
[13] M. T. HEATH, ED., *Hypercube Multiprocessors* 1986, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.
[14] ———, *Hypercube Multiprocessors* 1987, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.
[15] E. HOROWITZ AND S. SAHNI, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1978.
[16] F. JOHN, *Partial Differential Equations*, Fourth Edition, Springer-Verlag, New York, 1982.
[17] D. J. KUCK, *The Structure of Computers and Computations*, Vol. 1, John Wiley, New York, 1978.
[18] P. M. PRENTER, *Splines and Variational Methods*, John Wiley, New York, 1975.
[19] J. F. TRAUB AND H. WOŹNIAKOWSKI, *A General Theory of Optimal Algorithms*, Academic Press, New York, 1980.
[20] P. H. WORLEY, *Information requirements and the implications for parallel computation*, Ph.D. thesis, Stanford University, Stanford, CA, June 1988.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 36–57, January 1991

A PARALLEL QR FACTORIZATION ALGORITHM WITH CONTROLLED LOCAL PIVOTING*

CHRISTIAN H. BISCHOF†

Abstract. This paper presents a new version of the Householder algorithm with column pivoting for computing a QR factorization that identifies rank and range space of a given matrix. The standard pivoting technique is not well suited for parallel computation, since it requires synchronization at every step in order to choose the next pivot column. In contrast, a restricted pivoting scheme that restricts the choice of pivot columns and avoids this synchronization constraint is employed. Incremental condition estimation is used to assess the effect that the addition of a candidate pivot column would have on the condition number of the matrix being generated. This safeguard ensures that this local strategy selects pivot columns that make sense in the global context of the computation. The resulting algorithm is well suited for implementation on a parallel machine, in particular, a MIMD machine with distributed memory. Simulations demonstrate that the numerical behavior of the restricted pivoting strategy is comparable to the traditional global pivoting strategy. Implementation results of the QR factorization algorithm without pivoting and with local and traditional pivoting on the Intel iPSC/1 and iPSC/2 hypercubes show that our scheme about halves the extra time required for pivoting.

Key words. QR factorization, column pivoting, controlled pivoting, incremental condition estimation, distributed architecture, parallel algorithms

AMS(MOS) subject classifications. 15A23, 65F20

1. Introduction.

An important problem in numerical linear algebra is the determination of a maximal set of linearly independent columns of a matrix A. In statistics this problem is often referred to as the *subset selection problem* [19], [20]. It arises in the context of identifying redundant carriers in a linear model. Other applications are the solution of underdetermined or rank-deficient least-squares problems [6], [20], [23] and nullspace methods in optimization [9].

In linear algebra terms this problem can be viewed as finding a basis for the range space of A. The common way to approach this problem is via a QR *factorization*

$$AP = QR \tag{1}$$

of A. Here P is an $n \times n$ permutation matrix, Q is an $m \times m$ matrix orthogonal matrix, and R is an upper triangular $m \times n$ matrix. If A is a dense matrix, Q is usually computed by a sequence of *Householder transformations*

$$H = I - 2u\,u^T.$$

*Received by the editors December 27, 1988; accepted for publication (in revised form) October 3, 1989. This work was partially performed when the author was a graduate student at Cornell University and was supported by the U.S. Army Research Office through the Mathematical Science Institute of Cornell University, by the Office of Naval Research under contract N00014-83-K-0640, and by National Science Foundation contract CCR 86-02310. Support at Argonne was provided by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy under contract W-31-109-Eng-38. Computations were in part performed at the facilities of the Cornell Computational Optimization Project, which is supported by the National Science Foundation under contract DMS 87-06133, at the Cornell National Supercomputer Facility, and at the Advanced Research Computing Facility at Argonne National Laboratory.

†Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, Illinois 60439 (bischof@mcs.anl.gov).

Choosing

$$u = \frac{x + \operatorname{sign}(x_1)\,\|x\|_2\, e_1}{\|x + \operatorname{sign}(x_1)\,\|x\|_2\, e_1\|_2}, \tag{2}$$

we can reduce a given vector x to a multiple of the canonical unit vector e_1, since

$$(I - 2u\,u^T)\,x = -\operatorname{sign}(x_1)\,\|x\|_2\, e_1.$$

The standard technique [7] for determining P can be viewed as choosing as the next column the one that is farthest away (in the two-norm sense) from the subspace spanned by the columns that were selected before [20, p. 168, P.6.4-5].

The hope is that in the resulting QR factorization (1) the rank of A will reveal itself by a small trailing subblock of R: if $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ are the singular values of A and we partition R into

$$\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \tag{3}$$

with an $r \times r$ lower right-hand block R_{22}, then it is easy to show [20, p. 19] that

$$\sigma_{n-r+1}(A) \leq \|R_{22}\|_2\,. \tag{4}$$

Hence if R_{22} is small, A can be considered to have numerical rank $n - r$ and the first $n - r$ columns of Q form an orthonormal basis for the range space of A. While we can construct counterexamples [20, p. 167] where the column pivoting strategy fails to reveal ill-conditioning of A, it works well in practice.

An alternative pivoting strategy has been suggested by Chan [8] and Foster [16]. It was pointed out originally in [19] that we can use the singular vector corresponding to the smallest singular value to find a permutation P that guarantees a small r_{nn} in (3) if $\sigma_n(A)$ is small. Chan [8] and Foster [16] extend this idea to higher dimensions. The idea is first to compute any QR factorization of A and then to "peel off" the small singular values of R one after the other by computing an appropriate singular vector at each step. Chan also shows how to compute upper and lower bounds $\hat{\sigma}_i$ and $\check{\sigma}_i$ of σ_i. This allows implementation of the algorithm in an adaptive fashion: it can be terminated if the lower bounds indicate that all the small singular values have been revealed. Let us from now on assume that A has r small singular values and that there is a well-defined gap between σ_{n-r} and σ_{n-r+1}. It is shown in [19] that a well-defined gap is necessary to make a sensible decision on the numerical rank of A. Then Chan proves that if r is not too large, his algorithm will compute a "rank-revealing QR factorization" in the sense that R_{22} in (3) is guaranteed to be small.

On a single processor the Householder QR factorization without pivoting (i.e., P in (1) is the identity) requires $O(mn^2)$ flops, column pivoting requires an additional n^2 flops, and the rank-revealing QR algorithm requires an additional $3rn^2$ flops on average [8]. So the computational complexity of these algorithms is comparable on a single-processor machine.

The situation is quite different on a multiprocessor machine, especially if it is based on a distributed architecture. The Householder QR algorithm without pivoting processes the columns of A in their natural order from left to right. Hence a natural approach to parallelizing such a scheme is to group the processors into a logical ring and deal out columns in a round-robin fashion. The resulting pipelined algorithm

staggers the computation across the processors and guarantees a load-balanced computation. It allows simple static assignment of data to processors and is for the most part synchronized by the flow of data between processors. The main reason for the efficiency of this algorithm is that a given processor can still be busy finishing a previous update while another processor is generating the next Householder vector. The introduction of column pivoting makes pipelining impossible since the order in which columns are processed is not known a priori. Furthermore, all processors have to synchronize to select the next pivot column. One can somewhat overlap the computation of a Householder vector with its application [11], but nonetheless global pivoting introduces considerable extra communication overhead and forces the program into a lockstep mode where essentially all processors wait for a Householder vector and then apply it at the same time. This can result in a serious loss of efficiency on MIMD machines that previously could have profited from the pipelining.

For the rank-revealing QR factorization algorithm the steps after the initial QR factorization are hard to parallelize. For each of the r small singular values of A, the algorithm computes an approximate singular vector via inverse iteration, which on the average [8] requires the solution of four triangular equation systems per small singular value. Although much progress has been made recently in solving triangular equation systems on distributed architectures [10], [15], [21], [25], [24], this problem can by no means be parallelized as efficiently as the initial QR factorization. In addition, the application of the permutation deduced from the singular vector destroys the upper triangular shape of R, which then has to be restored by a sequence of Givens rotations. Again this is essentially a sequential process that is hard to parallelize [11]. We also need $O(rn)$ extra storage to store these Givens rotations. Since r is not known beforehand, this requires either problem-specific knowledge about r or, alternatively, dynamic storage allocation. This complications does not arise in the column pivoting scheme.

Apart from their sequential nature, an inherent difficulty in parallelizing the equation solving and QR update steps is that the computational work is of the same order of magnitude as the amount of data it involves. That is, we have to perform $O(n^2)$ flops using $O(n^2)$ data. Since R is distributed throughout the system, it is hard to mask the communication overhead with the little arithmetic work to be performed. So the post-processing of R can end up being a good part of the overall computation time on a parallel machine.

In this paper we suggest a new parallel algorithm for computing a set of independent columns of A using a controlled local pivoting strategy. Each processor limits its choice of pivot columns to its local columns and thereby avoids the synchronization overhead associated with making a global pivot choice. To make this strategy numerically robust, we must decide whether a local pivot candidate is a reasonable choice in the context of the overall factorization. In other words, given the upper triangular matrix R_i already computed and a new column $\begin{pmatrix} v \\ \gamma \end{pmatrix}$ determined by the local pivot candidate column, we must decide whether

$$R_{i+1} = \begin{pmatrix} R_i & v \\ 0 & \gamma \end{pmatrix}$$

is still of full rank. Since the smallest singular value $\sigma_{\min}(A)$ of a matrix A measures the distance of A (in the two-norm sense) from the set of rank-deficient matrices [20, p. 19] it is natural to use $\sigma_{\min}(R_{i+1})$ to decide whether to accept the new column. Computing $\sigma_{\min}(R_{i+1})$ exactly is too expensive, but using incremental condition esti-

mation [4] we can obtain a good estimate for $\sigma_{\min}(R_{i+1})$ cheaply without reaccessing R_i and without disrupting the overall pipelining scheme. The resulting algorithm is well suited for a parallel machine, in particular, a MIMD machine with distributed memory.

The outline of the paper is as follows. In the next section we briefly review the pipelined algorithm for computing the QR factorization without column pivoting. In § 3 we review the QR factorization algorithm with column pivoting. Implementation results on the Intel iPSC/1 and iPSC/2 hypercubes show that—as expected—the global pivoting strategy results in considerable extra overhead. Section 4 then motivates the local pivoting scheme and shows how the reliability of this scheme can be ensured by controlling the selection of pivot columns using incremental condition estimation. Section 5 presents simulation results comparing the controlled local pivoting scheme with the traditional pivoting scheme and implementation results on the Intel iPSC/1 and iPSC/2 hypercubes. These data show that the local pivoting strategy substantially decreases the overhead associated with column pivoting while being as efficient in identifying the numerical rank of A. Lastly we summarize our contributions and outline directions for further research.

2. The pipelined Householder QR factorization algorithm. To describe the Householder QR factorization algorithm we use the primitives *genhh* (generate Householder vector) and *apphh* (apply Householder matrix):

$$[u, y] \leftarrow genhh(x)$$

returns u as defined by (2) and $y = H(u)\,x$.

$$B \leftarrow apphh(u, A)$$

returns $H(u)\,A$.

Figure 1 describes the traditional Householder QR algorithm for computing the QR decomposition of an $m \times n$ matrix A $(m \geq n)$ without any pivoting. Here $a(i:j, k:l)$ refers to the submatrix of A consisting of row entries i to j and column entries k to l. A colon (:) is used as shorthand to design a complete row or column.

for $i = 1$ **to** n **do**
  $[u_i, a(i{:}m, i)] \leftarrow genhh(a(i{:}m, i))$
  $a(i{:}m, i{+}1{:}n) \leftarrow apphh(u_i\,, a(i{+}1{:}m, i : n))$
end for

FIG. 1. *The Householder* QR *factorization algorithm without pivoting.*

A natural way to parallelize this algorithm is to distribute the columns of A to processors in a round-robin fashion. To be precise, let us assume that we have p processors $proc_0, \ldots, proc_{p-1}$ and that a_j is the jth column of A. Then processor $proc_i$ receives columns a_j where

$$i = (j - 1) \bmod p.$$

This is commonly referred to as the *column wrap mapping.* Each processor executes the algorithm given in simplified form in Fig. 2 (to save space we use the abbreviation "HH" for "Householder"). The array C is local to each processor and contains the $cols_k$ columns assigned to processor $proc_i (\sum_{i=0}^{p-1} cols_k = n)$. p_{left} and p_{right} designate

the left and right neighbor of $proc_k$, respectively. *lcnt* (local count) and *gcnt* (global count) are initially zero and count the Householder vectors generated in $proc_k$ and overall, respectively.

The application and generation of Householder vectors are staggered across processors so that the computation proceeds in what can be described as a "pipelined" fashion. To prevent processors from being idle waiting for the next Householder vector to arrive, new Householder vectors have to be generated as soon as possible. To that end a processor generating a new Householder vector applies the update $H(u)$ just received only to column $lcnt+1$, computes and sends out the new Householder vector $\hat{u}$, and only then completes the previous update $H(u)$ and applies the new update $H(\hat{u})$.

This algorithm has several attractive features. The wrap mapping is simple and makes it easy to interface the QR algorithm to other algorithms. The algorithm is also inherently load balanced since the computational work is completed in a round-robin fashion. Most important, however, is the fact that message passing overhead is low. On the average, each processor receives and sends every Householder vector once, resulting in n sends and receives and a total of $mn - n^2/2$ transmitted words per node. By computing a new Householder vector as soon as possible, we maximize the likelihood that a Householder vector will arrive at the next processor before it is actually needed, thereby avoiding processor idle time. Requiring only nearest-neighbor communication is also an important factor in decreasing communication overhead. If the parallel machine in question allows asynchronous message passing (i.e., a sender does not block while waiting for a message to be delivered to the receiver), then we can hope to further decrease communication overhead. For these reasons this pipelining technique has also been widely used [18], [21], [22], [25], [26], [28] for other factorizations. If special vector hardware can be exploited, several Householder matrices can be bundled together by using the WY factorization [5], [31] to arrive at a block pipelined algorithm [3].

3. The global pivoting strategy. In the one-processor setting column pivoting can be introduced into the Householder QR factorization algorithm at little additional cost. The resulting algorithm is shown in Fig. 3.

The vector *perm* is used to store the permutation matrix P. If $perm(i) = k$, then the kth column of A is permuted into the ith column of AP. After completing step i the values $res_j, j = i+1, \ldots, n$ are the residuals of the jth column of the currently permuted AP with respect to the span of the first i columns of AP. As a result we can consider the numerical rank of A to be determined if res_{pvt} is small. Which *threshold* we choose for termination depends heavily on the application, but in general the computation will be terminated if the distance of the next pivot column from the already chosen subspace is $O(\epsilon\|A\|_1)$ where ϵ is the machine precision. res_j can be easily updated and does not have to be recomputed at every step. Roundoff errors may make it necessary to recompute $res_j = \|(a(i:m,j))\|_2, j = i+1, \ldots, n$ periodically [13, p. 9.17] (we suppressed this detail in Fig. 3). In practice this is rarely the case, and so the additional cost for incorporating column pivoting into the QR factorization is $O(n^2)$ flops in addition to the overall flop count of $O(mn^2)$ flops for the QR factorization.

On a distributed memory machine, the determination of the pivot column introduces extra overhead. If the matrix is distributed by columns as in the pipelined QR factorization algorithm of Fig. 2, each processor can easily determine its local pivot candidate column, but to find the overall pivot column all processors have to syn-

```
processor proc_k
    lcnt ← 0; {counter for HH vectors generated in proc_k}
    gcnt ← 0; {counter for HH vectors generated globally}
    if (k == 0) then {generate initial HH vector}
      [u, c(:,1)] ← genhh(c(:,1)); send u to p_right;
      c(:,2:cols_0) ← apphh(u, c(:,2:cols_0)); lcnt ← gcnt ← 1;
    end if
    while (lcnt < cols_k) do {main loop}
      receive u from p_left; gcnt ← gcnt + 1;
      if (u not generated by p_right) then send u to p_right end if
      if (k == gcnt mod p) then { my turn to generate HH vector }
        lcnt ← lcnt + 1;
        c(gcnt:m, lcnt) ← apphh(u, c(gcnt:m, lcnt));
                          {update first column}
        [û, c(gcnt+1:m, lcnt)] ← genhh(c(gcnt+1:m, lcnt));
                                 { generate new HH vector }
        if (gcnt + 1 < n) then send û to p_right end if
        c(gcnt:m, lcnt+1:cols_k) ← apphh(u, c(gcnt:m, lcnt+1:cols_k));
                                   { complete previous HH update }
        gcnt ← gcnt + 1;
        c(gcnt:m, lcnt+1:cols_k) ← apphh(û, c(gcnt:m, lcnt+1:cols_k));
                                   { apply new HH update }
      else
        c(gcnt:m, lcnt+1:cols_k) ← apphh(u, c(gcnt:m, lcnt+1:cols_k));
                                   { simply apply HH update }
      end if
    end while
```

FIG. 2. *The pipelined Householder* QR *algorithm without pivoting.*

```
    foreach i ∈ {1, ..., n} do
      perm_i ← i; res_i ← ||a(:,i)||_2
    end foreach
    for i = 1 to n do
      Let pvt ∈ {i, ..., n} be such that res_pvt is maximal
      if (res_pvt < threshold) then
        break { A has numerical rank i − 1 }
      else { exchange columns pvt and i }
        perm_i ↔ perm_pvt; a(:,i) ↔ a(:,pvt); res_pvt ← res_i;
        [u_i, a(i:m, i)] ← genhh(a(i:m, i));
          { apply H(u) and update residuals }
        a(i:m, i+1:n) ← apphh(u_i, a(i:m, i+1:n));
        foreach j ∈ {i+1, ..., n} do
          res_j ← sqrt(res_j^2 − a(i,j)^2);
        end foreach
      end if
    end for
```

FIG. 3. *The* QR *factorization algorithm with traditional column pivoting.*

chronize. It should be noted that distributing A by rows is no quick way out either. Although the creation of a Householder vector and its application can be somewhat overlapped [11], the computation of the column norm in (2) requires collecting each processor's individual contribution necessitating $O(\log p)$ extra communication overhead.

Assuming column wrap mapping, a simplified version of the main loop of the QR factorization algorithm with column pivoting is shown in Fig. 4. To save space, we omitted the code for the generation of the first Householder vector, which is performed in an analogous fashion.

The arrays *perm* and *glblpos* encode the permutation matrix P. $perm_i = j$ if local column i was the jth column of A. $glblpos_i = j$ if local column i will be column j in AP. Hence P would permute column $perm_i$ of A into position $glblpos_i$ in AP. *myid* of $proc_i$ is i. The function *find_max* determines the processor that owns the maximal column by embedding a spanning tree [29] in the hypercube. Each processor merges packets containing a *myid* and res_{pvt} field for some processor (originally its own) so that at the end the *LEADER* node knows who possesses the new pivot column. Finding the overall pivot column in this fashion requires time about $t \log_2 p$ where t is the time to send a message containing two elements to a neighboring node. The broadcast primitive also uses the spanning tree to propagate messages and hence suffers a delay of $O(\log_2 p)$. The computation is terminated if either $\min(m, n)$ Householder vectors have been generated or the distance of the suggested new pivot column to the subspace spanned by the already selected columns is so small that the remaining columns can be considered linearly dependent.

As in the pipelined QR factorization algorithm without pivoting, we try to generate pivot columns as soon as possible. In the pipelined algorithm without pivoting, the column determining the next Householder vector is known a priori, and so it is sufficient to update just this column in order to be able to compute the next Householder vector. If we want to perform pivoting, we have to do more work before we can compute the next Householder vector. To illustrate, let $\hat{C}$ be the submatrix still left to process in a given node, let *res* be the vector of residuals that the columns defining $\hat{C}$ have with respect to the subspace already chosen, and let u be a Householder vector that is currently being received. Since the processor housing $\hat{C}$ has not seen u previously, *res* does not reflect the choice of u as a pivot column yet. So in order to be able to choose the next pivot column, we have to update *res* to reflect the choice of u as pivot column. As can be seen from Fig. 3, we need the first row of $H(u)\hat{C}$ to update the residuals. Now

$$H(u)\hat{C} = \hat{C} - 2uu^T\hat{C} = \hat{C} - 2uz^T$$

where

$$z = \hat{C}^T u.$$

So the first row of $H(u)\hat{C}$ is

$$(H(u)\hat{C})(1,:) = \hat{C}(1,:) - 2u(1)z^T.$$

Once this row has been computed, we can update the residuals, determine the local pivot column, and propagate our local pivot choice along the spanning tree in *find_max*. Nonetheless we cannot avoid the $O(\log n)$ overhead for the determination of the pivot column or the broadcasting of the Householder vector. Furthermore, the

```
processor proc_k
      { Initialization }
   foreach i ∈ {1, ..., cols_k} do
      perm_i ← myid + (i − 1)p + 1; res_i ← ||c_i||_2;
   end foreach
   { generate first Householder vector and broadcast it as below }
      { main loop }
   repeat
      repeat
         switch (message_type)
            case "Householder_vector":
                  receive u; gcnt ← gcnt + 1;
            case "You_are_winner":
                  gcnt ← gcnt + 1; lcnt ← lcnt + 1; glblpos(lcnt) ← gcnt;
                  [u, c(gcnt:m, pvt)] ← genhh(c(gcnt:m, pvt));
                  broadcast u;
                  c_lcnt ↔ c_pvt; perm_lcnt ↔ perm_pvt; res_pvt ← res_lcnt;
            case "Time_to_quit":
                  stop
         end switch
      until message is received
         { determine candidate pivot column }
      z ← c(gcnt:m, lcnt+1:cols_k)^T u;
      c(gcnt, lcnt+1:cols_k) ← c(gcnt, lcnt+1:cols_k) − 2u(1)z^T;
      res_i ← sqrt(res_i^2 − c(gcnt, i)^2), i ∈ {lcnt+1, ..., cols_k}
      Let pvt ∈ {lcnt+1, ..., cols_k} be such that res_pvt is maximal.
      [winner, res_of_winner] ← findmax(pvt, res_pvt, LEADER);
         { complete update }
      c(gcnt+1:m, lcnt+1:cols_k)
                  ← c(gcnt+1:m, lcnt+1:cols_k) − 2u(2: m−gcnt)z^T;
         { LEADER notifies the winner }
      if (myid = LEADER) then
        if ((gcnt > min(m, n)) or (res_of_winner < threshold)) then
          broadcast "Time_to_quit" message.
        else
          send "You_are_winner" message to winner.
        end if
      end if
   forever
```

FIG. 4. *The distributed Householder* QR *factorization algorithm with global pivoting.*

lockstep style of execution resulting from this synchronization point makes it impossible to mask the communication overhead as efficient as the pipelining technique of Fig. 2 allowed.

TABLE 1

The Householder QR *factorization algorithm with and without column pivoting applied to a* $500 \times n$ *matrix on a* 32*-node* iPSC/1 *hypercube.*

n	$t_{\max}$		$\bar{t}_{ohead}$	
	Global piv.	No piv.	Global piv.	No piv.
100	27.7	18.8	16.5	5.7
200	70.9	49.6	29.8	7.8
300	125	93.2	40.5	9.7
400	184	145	46.5	10.9
500	255	201	50.1	11.3
600	310	257	54.0	11.1
700	369	314	53.1	11.3
800	437	370	54.5	11.2
900	515	427	66.1	11.5

TABLE 2

The Householder QR *factorization algorithm with and without column pivoting applied to a* $500 \times n$ *matrix on a* 16*-node* iPSC/2 *hypercube.*

n	$t_{\max}$		$\bar{t}_{ohead}$	
	Global piv.	No piv.	Global piv.	No piv.
100	5.9	4.1	2.7	0.7
200	16.5	12.8	4.9	1.2
300	30.5	25.2	6.6	1.6
400	46.6	40.1	7.9	2.0
500	63.7	56.3	8.7	2.2
600	80.8	72.6	9.3	2.2
700	97.0	88.9	9.0	2.3
800	113	105	8.7	2.4
900	130	122	8.8	2.4

We compared the algorithms of Figs. 2 and 4 on a 32-node Intel iPSC/1 [14] and a 16-node iPSC/2 [1] hypercube. A Gray code mapping [29] was used to embed the ring of processors and the code was entirely written in Fortran for single precision. For the iPSC/1, we compiled with the Ryan–McFarland compiler (Version 2.20a) using the huge memory model and executed under the NX node operating system release 3.1.1. On the iPSC/2, we used the Green Hills compiler (Version 1.8.3a) and the NX/2 node operating system release 2.3. For $m = 500$ and $n = 100, 200, \ldots, 900$ we generated random matrices with singular values $1, 2, \ldots, \min(m, n)$ and reduced them to triangular form using $\min(m, n)$ Householder reductions. We observed the performance shown in Table 1 for the iPSC/1 and in Table 2 for the iPSC/2. Here $t_{\max}$ is the maximal execution time of any processor in seconds and $\bar{t}_{ohead}$ is the average time in seconds that a processor spends communicating and waiting idle.

We see that the algorithm for computing the QR factorization without pivoting is very efficient in that only a small portion of the overall execution time is spent on

communication. As predicted, we observe a noticeable performance degradation for the global pivoting scheme. In particular, for smaller problems not yet dominated by floating point work, the communication overhead increases dramatically compared to the pipelined QR factorization algorithm. This is unpleasant, especially in light of the fact that column pivoting costs only $O(n^2)$ extra flops.

4. Controlled local pivoting. The easiest way to avoid the extra overhead associated with global pivoting is to forego global pivoting altogether and have each processor limit its choice of pivot columns to the ones it houses. Whereas in the pipelined algorithm without pivoting, each processor simply picks the next available column to determine the next Householder vector, each processor now performs a *local column pivoting* step to determine the next Householder vector. This allows implementation of the algorithm in much the same fashion as in Fig. 2, and as a result we expect to reap much the same benefits as described in §2.

The problem with the strictly local pivoting strategy is obviously its reliability in identifying independent columns of A. As a pathological example, assume that all columns in processor $proc_0$ are nearly equal. As a result, processor $proc_0$ will make bad choices after it has generated the very first Householder vector. The resulting upper triangular matrix R will be nearly rank-deficient but will not necessarily have a small lower right-hand block to reveal that fact.

To guard against choosing nearly dependent pivot columns, it is natural to monitor $\sigma_{\min}(R)$ since the smallest singular value of R is the distance of R (in the two-norm sense) from the space of rank-deficient matrices. Computing $\sigma_{\min}(R)$ exactly via inverse iteration, for example, is too expensive since it would require us to reaccess R (which is distributed throughout the system) several times. In fact, it is not feasible to access the previously generated R even once when we want to decide on the suitability of a new pivot column.

Let R_i be the current upper triangular matrix R_i which is made up by the first i columns of Q^TAP after i Householder vectors have been generated and applied. Then we use the *incremental condition estimator* suggested by Bischof [4] to monitor $\sigma_{\min}(R_i)$. Given a good estimate $\hat{\sigma}_{\min}(R_i) = 1/\|x\|_2$ defined by a large norm solution x to $R_i^T x = d$ and a new column $\begin{pmatrix} v \\ \gamma \end{pmatrix}$, the incremental condition estimator allows us to obtain an estimate for $\sigma_{\min}(R_{i+1})$ where

$$R_{i+1} = \begin{pmatrix} R_i & v \\ 0 & \gamma \end{pmatrix}$$

without accessing R_i again. Defining

$$\alpha = v^T x \quad \text{and} \quad \beta = \gamma^2 x^T x + \alpha^2 - 1,$$

we have that the estimate for the smallest singular value of R_{i+1} is given by

$$\hat{\sigma}_{\min}(R_{i+1}) = \frac{1}{\|y\|_2} \tag{5}$$

where

$$y = \begin{pmatrix} sx \\ (c - s\alpha)/\gamma \end{pmatrix} \tag{6}$$

solves

$$R_{i+1}^T y = \begin{pmatrix} sd \\ c \end{pmatrix}.$$

c and s are defined as the solutions of

$$\begin{aligned} &\max \Phi(c,s) = \|y\|_2 \\ &\text{subject to } c^s + s^2 = 1. \end{aligned}$$

c and s can easily be computed, and so the cost of determining $\hat{\sigma}_{\min}(R_{i+1})$ is $3i$ flops for the inner product $v^T x$ and the scaling of x by c. Numerical experiments with this condition estimation scheme [4] show that it is reliable in producing good estimates. It overestimates the smallest singular value of a triangular matrix only by a small factor, and the results vary only a little with condition number, matrix size, and singular value distribution. Applying this condition estimator to upper triangular matrices generated by using the traditional column pivoting strategy somewhat increased its accuracy and we can confidently expect similar behavior when applying this estimator to matrices generated by the local pivoting strategy.

With the incremental condition estimator we now have the tool to ensure the reliability of the local pivoting strategy. By applying the incremental condition estimator to its local pivot candidate, a processor can decide whether its local choice is reasonable in the global context of the computation. Assuming that processor k knows the current estimate x as well as $\|x\|_2$ for the current upper triangular matrix R_{gcnt}, all that is needed for the next condition estimator step is the last column $\begin{pmatrix} v \\ \gamma \end{pmatrix}$ of R_{gcnt+1}. But

$$v = c(1 : gcnt, j)$$

has already been computed, and from the definition of u and res it follows immediately that

$$\gamma = -\text{sign}\,(c(gcnt+1, j))\, res_j.$$

So all the information for the next condition estimator step is readily at hand, and we can compute $\hat{\sigma}_{\min}(R_{gcnt+1})$ as described above. With

$$\omega = \max_{1 \le i \le n} \|a_i\|_2$$

being the norm of the largest column of A, we can then take

$$\hat{\kappa}\,(R_{gcnt+1}) = \eta\,\omega \|y\|_2 \tag{7}$$

as an estimate for the true condition number of R_{gcnt+1}. The scaling factor η reflects the trust we have in the accuracy of our estimates. A large η will result in estimates that are too pessimistic, while a small η might lead to underestimation. ω overestimates $\|A\|_2$ by at most a factor of $\sqrt{n}$, but heuristically it is a much more accurate estimate.

Comparing the estimates (5) or (7) against a chosen threshold, we will then accept or reject a candidate pivot column. If the candidate pivot column is rejected, processor k has exhausted its supply of "reasonable" columns, and from then on it will only apply Householder vectors generated by other processors to its remaining columns. If, on the other hand, we accept the candidate pivot column, then processor k will actually compute $\hat{u}$ and send $\hat{u}$ and $(y, \|y\|_2)$ to its right neighbor. So the additional work required for generating the ith Householder vector will on average be $3i$ floating point operations and the transmission of i words to a neighboring node. It is worth

emphasizing that on average y and $\|y\|_2$ have to be forwarded only to the processor that will generate the next Householder vector (which in most cases will be the right neighbor), while $\hat{u}$ will eventually be known to all processors. So the propagation of the condition estimator information will result in only a minor increase in data traffic.

This scheme continues until no processor has any acceptable pivot candidate left. Assuming that altogether we generated $\hat{n} = n - \hat{r}$ Householder vectors, we have at this point computed the incomplete QR factorization

$$AP = (Q_1, Q_2) \begin{pmatrix} R_{11} & R_{12} \\ 0 & \hat{A} \end{pmatrix} \tag{8}$$

where Q_1 is $m \times \hat{n}$, Q_2 is $m \times (m - \hat{n} + 1)$, and $Q = [Q_1, Q_2]$ is orthogonal. R_{11} is upper triangular of size $\hat{n} \times \hat{n}$ and $\hat{A}$ is of size $(m - \hat{n} + 1) \times \hat{r}$. Our controlled pivoting strategy gives us an estimate for $\sigma_{\min}(R_{11})$, and furthermore, we know that adding any of the leftover $\hat{r}$ columns of AP would result in a decrease of the smallest singular value below our chosen threshold. So we have good reason to assume that $\hat{r}$ is the dimension of the numerical null space of A.

A simplified version of the main loop of the resulting algorithm is shown in Figs. 5 and 6. We omitted the generation of the first Householder vector which is done in the same fashion as in Fig. 6 and in order not to bury the structure of the algorithm, we omitted the details concerning program termination. Each processor can be in one of three states as determined by *mystatus*. It is *ALIVE* if it still houses eligible columns; this is the initial state of every processor. Its status changes to *FINISHED* if it has processed all its local columns. Since other processors might still house eligible columns, it still must forward messages; this is its only action in this state. If a processor is *DEAD*, then all its remaining columns have been found to be unacceptable, but it still has to apply Householder vectors generated by other processors as well as forward them.

$$y \leftarrow \mathit{cond_est}(x, \|x\|_2, v, \gamma)$$

returns y as defined by (6). *trustfactor* is used to adjust for the overestimation of the smallest singular value. Since experiments with the incremental condition estimator [4] indicate that in general the condition estimator does not overestimate $\sigma_{\min}$ by a factor of more than three, we used $1/(3\|y\|_2)$ as our estimate for $\sigma_{\min}$.

The overall structure of the main loop is described in Fig. 5. Notice that a processor checks halfway through a Householder update whether a singular vector has arrived. In this case it completes the $H(u)$ update only on the pivot candidate column in order to generate a new Householder vector as soon as possible. Only after the new Householder vector $\hat{u}$ has been sent out does it complete the $H(u)$ and $H(\hat{u})$ updates. These steps are described in more detail in Fig. 6 (ignoring Part b). Figure 6 (ignoring Part a) describes the easier case of generating a pivot column when the previous Householder update has already been completed.

Notice that compared to the algorithm with global pivoting we maintain the pipelined mode of operation. By generating Householder vectors as soon as possible we attempt to keep the pipeline filled with Householder vectors and minimize processor idle time. Furthermore there are no synchronization points in this algorithm that would add extra overhead or would prevent us from masking communication overhead. We would also like to mention that this algorithm is naturally load-balanced due to its pipelined nature. Load balancing is more difficult in the algorithm employing global pivoting, since it is possible that a processor may contain all the first pivot columns

processor $proc_k$

repeat
repeat
switch (message type)
case "Householder_vector":
{ complete enough of $H(u)$ application to update res }
receive u; $gcnt \leftarrow gcnt + 1$; forward u to p_{right}.
if ($mystatus \neq FINISHED$) **then**
$z \leftarrow c(gcnt{:}m, lcnt{+}1{:}cols_k)^T u$;
$c(gcnt, lcnt{+}1{:}cols_k) \leftarrow c(gcnt, lcnt{+}1{:}cols_k) - 2u(1)z^T$;
$res_i \leftarrow \sqrt{res_i^2 - c(gcnt, i)^2}\,, i \in \{lcnt{+}1, \ldots, cols_k\}$
{ check whether another message arrived in the mean time }
if (I received another message containing a singular value) **then**
Try to generate a new Householder vector before completing
the update determined by u. See Fig. 6 and ignore part b.
else
$c(gcnt{+}1, lcnt{+}1{:}cols_k)$
$\leftarrow c(gcnt{+}1, lcnt{+}1{:}cols_k) - 2u(2{:}m{-}gcnt)z^T$;
end if
else
break to outermost repeat-loop.
end if
case "singular_vector":
receive $(x, \|x\|_2)$;
if ($mystatus = ALIVE$) **then**
determine next pivot candidate. See Fig. 6 and ignore part a.
else
forward $(x, \|x\|_2)$ to p_{right}.
end if
case "time_to_quit":
send time_to_quit message to p_{right} and quit.
end switch
until a message has been received
forever

FIG. 5. *The pipelined Householder* QR *algorithm with controlled local pivoting: main loop.*

processor $proc_k$

{ Determine new pivot candidate and estimate $\hat{\sigma}_{min}(R_{gcnt+1})$. }
Let $pvt \in \{lcnt+1, \ldots, cols_k\}$ be such that res_{pvt} is maximal.
$y \leftarrow$ cond_est$(x, \|x\|_2, c(1\!:\!gcnt, pvt), -\text{sign}(c(gcnt+1, pvt))res_{pvt})$;
if $(\|y\|_2/trust_factor > 1/threshold)$ **then**
{ new pivot candidate is not acceptable }
send $(x, \|x\|_2)$ to p_{right}. $mystatus \leftarrow DEAD$.
$c(gcnt+1, lcnt+1\!:\!cols_k) \leftarrow c(gcnt+1, lcnt+1\!:\!cols_k) - 2u(2\!:\!m-gcnt)z^T$;
else
$lcnt \leftarrow lcnt+1; gcnt \leftarrow gcnt+1; c_{lcnt} \leftrightarrow c_{pvt}$;
$perm_{lcnt} \leftrightarrow perm_{pvt}; res_{lcnt} \leftarrow res_{pvt}$;
begin of Part a
{ delay $H(u)$ update and generate new Householder vector $\hat{u}$ }
$z(1) \leftrightarrow z(pvt-lcnt+1)$;
$c(gcnt-1\!:\!m, lcnt) \leftarrow c(gcnt-1\!:\!m, lcnt) - 2z(1)u$;
$[\hat{u}, c(gcnt\!:\!m, lcnt)] \leftarrow genhh(c(gcnt\!:\!m, lcnt))$;
send $\hat{u}$ to p_{right}; send $(y, \|y\|_2)$ to p_{right}.
{ complete $H(u)$ update and then $H(\hat{u})$ update }
$c(gcnt-1\!:\!m, lcnt+1\!:\!cols_k)$
$\leftarrow c(gcnt-1\!:\!m, lcnt+1\!:\!cols_k) - 2u(2\!:\!m-gcnt)z^T$;
$c(gcnt\!:\!m, lcnt+1\!:\!cols_k) \leftarrow (I - 2\hat{u}\hat{u}^T)c(gcnt\!:\!m, lcnt+1\!:\!cols_k)$;
end of Part a
begin of Part b
{ generate new Householder vector }
$[u, c(gcnt\!:\!m, lcnt)] \leftarrow genhh(c(gcnt\!:\!m, lcnt))$;
send u to p_{right}; send $(y, \|y\|_2)$ to p_{right}.
{ complete Householder update }
$c(gcnt\!:\!m, lcnt+1\!:\!cols_k) \leftarrow (I - 2uu^T)c(gcnt\!:\!m, lcnt+1\!:\!cols_k)$;
end of Part b
$res_i \leftarrow \sqrt{res_i^2 - c(gcnt, i)^2}, i \in \{lcnt+1, \ldots, cols_k\}$
end if

FIG. 6. *Generation of a new Householder vector.*

and hence will be idle soon unless one redistributes columns dynamically. Our local pivoting scheme avoids those complications. In addition, the local pivoting scheme is—apart from the cost of startup and wind-down—insensitive to the number of processors, since communication is only between neighbors. The cost of the reduction and broadcast in the global scheme, on the other hand, depend on the diameter of the system and are likely to increase as the system grows.

5. Numerical experiments. To verify the merits of our proposed controlled local pivoting scheme, we tested its numerical reliability by comparing it with the QR factorization with traditional column pivoting. Furthermore, we tried to assess its computational performance on a distributed memory machine by implementing it on the Intel iPSC/1 and iPSC/2 hypercubes.

To assess the numerical behavior of the proposed local pivoting scheme, we simulated the parallel algorithm using PRO-MATLAB [27] and compared it with the traditional QR factorization algorithm with global column pivoting. Various 100×100 matrices were generated, and the local pivoting strategy was simulated on 8 and 32 processors.

For tests 1 to 3 we generated 50 random matrices with prescribed singular value distributions $\{\sigma_i\}$. Random orthogonal matrices U and V were generated using the method of Stewart [32], and then A was formed as $A = U\Sigma V^T$. For all matrices the largest and smallest singular values were 1 and 10^{-9}, respectively.

Break 1 *Distribution*: $\sigma_1 = \ldots = \sigma_{99} = 1;\ \sigma_{100} = 10^{-9}$.

Break 9 *Distribution*: $\sigma_1 = \ldots = \sigma_{91} = 1;\ \sigma_{92} = \ldots = \sigma_{100} = 10^{-9}$.

Exponential Distribution: $\sigma_1 = 1; \sigma_i = \alpha^{i-1} (i = 2, \ldots, 100);\ \alpha = (10^{-9})^{1/99}$.

Setting the rejection threshold for the smallest singular value to 10^{-7} and discounting the estimate for the smallest singular value (5) by a factor of 3 (i.e., *trustfactor* in Fig. 6 equals 3), we reject a candidate pivot column in the parallel algorithm if

$$\frac{1}{3\|y\|_2} = \hat{\sigma}_{min}(R_{gcnt+1}) \leq 10^{-7}.$$

For the traditional QR factorization algorithm we use the last diagonal entry of R_{gcnt+1} as an estimate for $\sigma_{\min}(R_{gcnt})$ and reject a candidate pivot column if

$$\frac{1}{3} \mid r_{gcnt+1,gcnt+1} \mid \leq 3 \cdot 10^{-7}.$$

Table 3 shows the condition numbers of the upper triangular matrices R generated by controlled local pivoting and by traditional column pivoting on those matrices. If we let σ_{cutoff} be the smallest singular value greater than 10^{-7}, then the optimal value we can theoretically achieve for $\kappa(R)$ is $\kappa_{opt}(R) = 1/\sigma_{cutoff}$. Furthermore, let $\kappa_{par}(R)$ be the condition number resulting from the parallel scheme and $\kappa_{trad}(R)$ the condition number resulting from the traditional column pivoting scheme. For $\kappa_{par}(R)$ and $\kappa_{trad}(R)$ observed minimum, average, and maximum values are displayed. These results show that guarded local pivoting is as effective as full column pivoting in generating a well-conditioned R—especially if there are more than just a few columns in each processor. Except for the break 9 example the transition from 8 to 32 processors had no noticeable effect. The break 9 example shows that the local pivoting strategy can deteriorate somewhat if the matrix is highly rank-deficient and the number of columns per processor is small. This fact is not surprising in that the pivoting choices of each processor are very limited. Due to the high dimension of the numerical null

TABLE 3
min/avg/max *values of the condition numbers of R using local and global pivoting.*

Distribution	Break 1	Break 9	Exponential
$\kappa_{par}(R)$, $p = 8$	2.7 /5.1 /8.0	6.4 /13 /46	7.2e6 /1.1e7 /1.6e7
$\kappa_{par}(R)$, $p = 32$	3.7 /7.7 /23	10 /180 /6.1e3	7.6e6 /1.3e7 /1.9e7
$\kappa_{trad}(R)$	2.8 /3.7 /4.8	4.3 /5.7 /7.8	7.2e6 /1.0e7 /1.9e7
$\kappa_{opt}(R)$	1.0	1.0	8.1e6

space, a processor may be forced to consider columns that are suboptimal but whose choice results in a matrix R_i where $\sigma_{\min}(R_i)$ is still well above the threshold.

The average values for the break 9 distribution on 32 processors in Table 3 are also somewhat misleading in that they make the local pivoting strategy look worse than it really is. If we ignore the one experiment where R had condition number 6.1e3 (which is still well below our rejection threshold), we obtain for the other 49 runs the condition number distribution shown in Fig. 7. We see that in the vast majority of cases (36 cases) the condition numbers of R were less than 50. We mention that this is a somewhat contrived example in that with only three columns per processor the pipelined algorithm would be dominated by startup and wind-down costs and as a result would not perform efficiently.

This example does, however, suggest the use of a variable threshold in accepting pivot columns if we are worried about generating as well-conditioned a matrix as possible. Starting with a conservative threshold and relaxing it as the computation proceeds, we shall in all likelihood prevent a processor from choosing a suboptimal column early on. The penalty is that we might have to consider the same candidate pivot column more than once. If we reject a candidate for the ith Householder vector, the added cost is $2i$ flops (since sx in (6) need not be computed) and the transmission of i words to a neighbor. Depending on the application this extra work might be worthwhile.

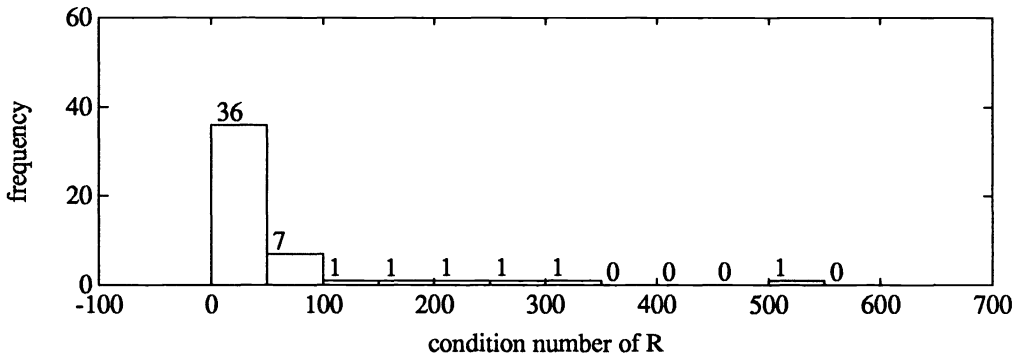


FIG. 7. *Condition number distribution of R for the "break 9" distribution with* $p = 32$.

For the sharp break distributions there is a well-defined gap between the singular values before and after the acceptance threshold, and both the local and global column pivoting strategies identify the numerical nullspace correctly for $p = 8$ and $p = 32$

TABLE 4
Frequency of accepting columns for the exponential distribution.

No. of columns accepted	72	73	74	75	76
Local pivoting, $p = 8$	2	3	12	24	9
Local pivoting, $p = 32$	5	11	23	9	2
Global pivoting	0	2	29	18	1

in all cases. As already pointed out, the determination of numerical rank becomes problematic if there is no well-defined gap between singular values that are considered "large" and "small". The exponential distribution is such a problematic case. There are 77 singular values that are larger than 10^{-7}, but there is no well-defined break. To be exact,

$$\sigma_{75} = 1.8 \cdot 10^{-7}, \quad \sigma_{76} = 1.5 \cdot 10^{-7}, \quad \sigma_{77} = 1.2 \cdot 10^{-7}, \quad \sigma_{78} = 1.0 \cdot 10^{-7}\,.$$

The column pivoting strategy reflects this difficulty in accepting less than 77 columns; the results are displayed in Table 4. So, for example, in 24 of the 50 runs we accepted 75 columns in the local pivoting scheme using 8 processors. These results show that even for an ill-defined problem the guarded local pivoting scheme is reliable in that it leans towards a small underestimate of the dimension of the numerical range space of A.

Finally, we give an example where the local pivoting strategy actually performs better than the global one. A well-known example (originally suggested by Kahan) where the traditional column pivoting strategy fails is

$$A_n = \operatorname{diag}(1, s, s^2, \ldots, s^{n-1}) \begin{pmatrix} 1 & -c & \cdots & \cdots & -c \\ 0 & 1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 1 & -c \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} + \Delta \tag{9}$$

where $\Delta = \operatorname{diag}(n\epsilon, (n-1)\epsilon, \ldots, \epsilon)$, $c^2 + s^2 = 1$ and ϵ is the machine precision. A_n is very ill-conditioned, but although each leading principal submatrix A_k $(k \leq n)$ is also ill-conditioned, there is a well-defined gap between σ_n and σ_{n-1}. As an example, in single precision for $n = 50$ and $c = 0.5$ we have $\sigma_{49} = 1.2 \cdot 10^{-3}$ and $\sigma_{50} = 3.7 \cdot 10^{-12}$. Even in floating-point arithmetic the matrix is its own QR factorization with pivoting, but no trailing block of R is small to reveal its ill-conditioning.

For this matrix both the local and global pivoting schemes select the columns in their natural order and fail. However, the condition estimator integrated into the parallel scheme detects the ill-conditioning of the leading principal submatrices A_k — in fact we observed that it never overestimates the smallest singular value by a factor of more than 1.5. So at least failure does not go unnoticed. Now let $\tilde{A}_{50}$ be the same matrix as A_{50} except that the order of columns has been reversed. For the global column pivoting scheme this permutation is without consequences and it fails. The parallel scheme on 8 or 32 processors, on the other hand, correctly identifies the numerical null space of $\tilde{A}_{50}$. While this is an exceptional occurrence resulting from the special structure of A_n, it is nonetheless surprising since intuitively we would expect the global pivoting strategy always to perform better than the local one.

To assess the computational performance of the proposed scheme, we implemented it on a 32-node iPSC/1 and 16-node iPSC/2 hypercube. When running the code on the problems used for Tables 1 and 2 and using the same notation we observed the performance shown in Table 5.

TABLE 5
Performance of the Householder QR *factorization algorithm with controlled local pivoting.*

	32-node iPSC/1		16-node iPSC/2	
n	$t_{\max}$	$\bar{t}_{ohead}$	$t_{\max}$	$\bar{t}_{ohead}$
100	25.1	13.0	4.8	1.4
200	60.2	17.2	13.9	2.0
300	108	20.6	26.9	2.6
400	165	24.0	42.4	3.1
500	227	28.1	59.4	3.8
600	280	23.6	75.2	3.2
700	336	21.0	91.6	3.1
800	392	19.3	108	3.0
900	454	20.0	125	2.9

Comparing these figures with those of Tables 1 and 2 we see that the local pivoting strategy does indeed result in a significant decrease in execution time. These gains would be even more pronounced on a bigger system as our pipelined strategy is essentially insensitive to the number of processors, whereas the broadcast and reduction operations needed in the global pivoting algorithm become more expensive. We also see that the local pivoting algorithm is efficient in that does not require significantly more flops than the QR factorization without pivoting and the communication overhead is in general a small portion of the overall execution time.

The advantages of local pivoting are even more apparent if we consider how much *more* local and standard pivoting cost us in comparison with the QR factorization without pivoting. Let $t_{\max}^{nopiv}(n)$ be the maximal execution time of the pipelined QR factorization algorithm without pivoting on a $500 \times 100 * n$ problem ($t_{\max}^{globalpiv}(n)$ and $t_{\max}^{localpiv}(n)$ analogous). Figure 8 displays

$$p_{total}^{strategy}(n) \equiv \frac{t_{\max}^{strategy}(n) - t_{\max}^{nopiv}(n)}{t_{\max}^{nopiv}(n)} * 100$$

for

$$strategy = \{globalpiv,\ nopiv\}$$

and varying n. $p_{total}^{strategy}(n)$ measures how much more time (in percent) the implementation of the two pivoting strategies costs compared to the QR factorization without pivoting. The dashed line represents the standard column pivoting strategy; the dash-dotted line represents controlled local column pivoting. Figure 9 displays

$$p_{comm}^{strategy}(n) \equiv \frac{\bar{t}_{ohead}^{strategy}(n) - \bar{t}_{ohead}^{nopiv}(n)}{\bar{t}_{ohead}^{nopiv}(n)} * 100$$

and shows the percentage increase in communication overhead due to pivoting.

These figures show that the extra running time required to implement local pivoting is on the average about half that required to implement global pivoting. This is

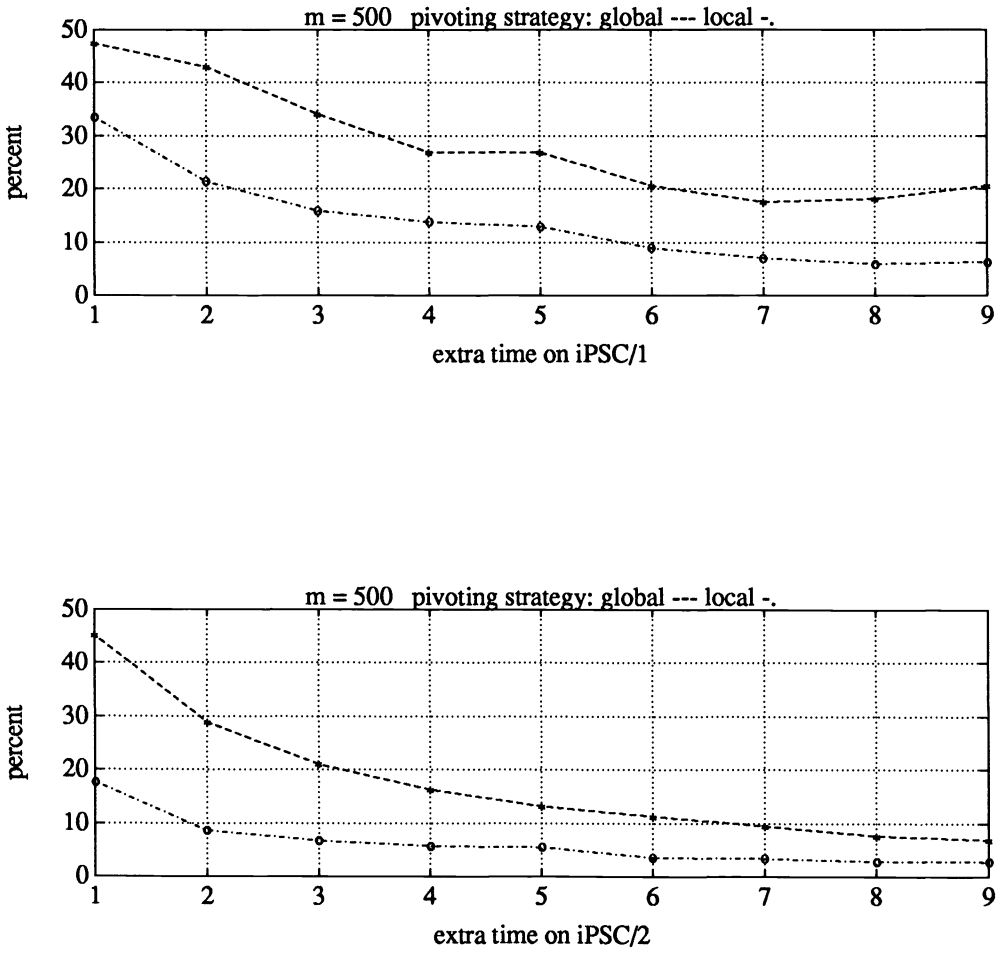


FIG. 8. *Extra time required for pivoting with respect to* QR *factorization without pivoting.*

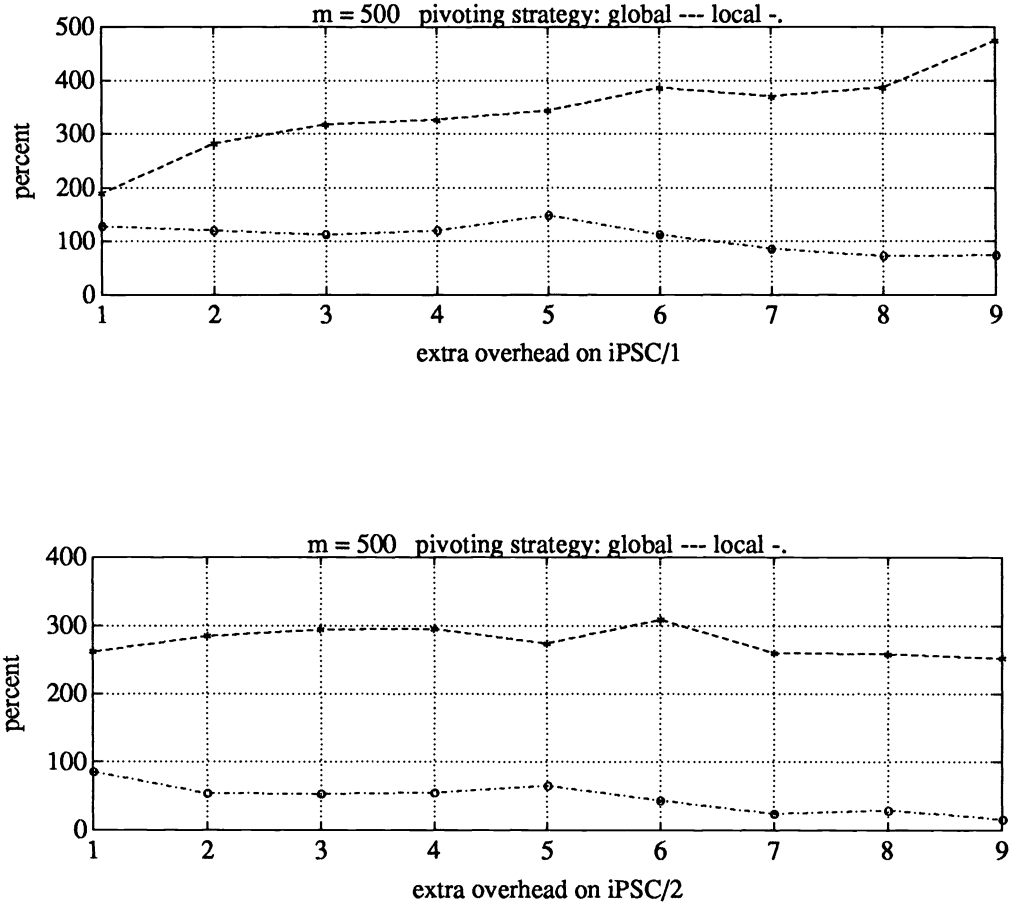


FIG. 9. *Extra communication overhead for pivoting with respect to* QR *factorization without pivoting.*

not surprising since—as shown in Figure 9—local pivoting increases the communication overhead only by about a factor of two, whereas the traditional column pivoting strategy can increase communication overhead almost fivefold. In addition, the extra communication overhead for the local pivoting strategy essentially stays flat with n whereas it grows with n for the global pivoting regime on the iPSC/1. So the local pivoting strategy is indeed superior to the traditional global pivoting strategy on MIMD machines which can profit from the pipelining scheme and for which global broadcast/gather operations are expensive.

6. Conclusions. We presented a new version of the Householder algorithm with column pivoting for computing a QR factorization which identifies rank and range space of a given matrix. To arrive at an algorithm that is better suited for parallel computation than the traditional regime, we employed a restricted pivoting scheme that restricted the choice of pivot columns to ones in local memory. The resulting algorithm is much better suited for a parallel machine, in particular, a MIMD machine with distributed memory. This is borne out by the implementation results on the Intel iPSC/1 and iPSC/2 hypercubes where the local pivoting scheme about halves the extra time required for pivoting compared to the traditional column pivoting scheme. Simulations also showed that the numerical properties of the suggested local pivoting scheme are comparable to those of the global pivoting strategy.

The key idea that made this algorithm work was the use of incremental condition estimation to restrict the choice of pivot column without giving up numerical reliability. Incremental condition estimation allowed us to "back up" when we were about to make a bad choice. Restricting pivot choices then resulted in an algorithm that displayed a much higher degree of locality of reference. We showed the resulting benefits for a MIMD machine, but obviously locality of reference is also crucial in achieving good performance on parallel machines with shared memory or one-processor machines employing a memory hierarchy. As a result, we believe restricted pivoting strategies to be advantageous in those environments as well. We are currently exploring this issue in the context of a block algorithm [2], [12], [17], [30] for computing the QR factorization with column pivoting on a shared memory machine with a memory hierarchy.

Acknowledgments. I am very grateful to Paul Plassmann from Cornell and Ed Kushner from Intel for their undaunted perseverance in porting the code from the iPSC/1 to the iPSC/2. I would also like to thank Intel Corporation for the opportunity to use one of their iPSC/2 cubes for my benchmarks.

REFERENCES

[1] R. ARLAUSKAS, *iPSC/2 system: a second generation hypercube*, in The Third Conference on Hypercube Concurrent Computers and Applications, G. Fox, ed., ACM Press, New York, 1988, pp. 38–42.

[2] M. BERRY, K. GALLIVAN, W. HARROD, W. JALBY, S.-S. LO, U. MEIER, B. PHILIPPE, AND A. SAMEH, *Parallel algorithms on the Cedar system*, in Proc. CONPAR 86, W. Händler, ed., Springer-Verlag, New York, 1986, pp. 25–39.

[3] C. H. BISCHOF, *A pipelined block QR decomposition algorithm*, in Parallel Processing for Scientific Computing, G. Rodrigue, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 3–7.

[4] ———, *Incremental condition estimation*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 312–322.

[5] C. H. BISCHOF AND C. F. VAN LOAN, *The WY representation for products of Householder matrices*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s2–s13.

[6] Å. BJÖRCK, *Difference Methods—Solutions of Equations in R^n*, Vol. II of Handbook of Numerical Analysis, Elsevier Publishers, 1989, chap. Least Squares Methods.
[7] P. A. BUSINGER AND G. H. GOLUB, *Linear least squares solution by Householder transformation*, Numer. Math., 7 (1965), pp. 269–276.
[8] T. F. CHAN, *Rank revealing* QR *factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.
[9] T. F. COLEMAN, *Large Sparse Numerical Optimization*, Lecture Notes in Computer Science 165, Springer-Verlag, New York, 1984.
[10] T. F. COLEMAN AND G. LI, *Solving systems of nonlinear equations on a message-passing multiprocessor*, Tech. Report CS–87–887, Department of Computer Science, Cornell University, Ithaca, NY, November 1987.
[11] T. F. COLEMAN AND P. PLASSMAN, *The solution of nonlinear least-squares problems on a message-passing multiprocessor*, Tech. Report CS–TR–88–923, Department of Computer Science, Cornell University, Ithaca, NY, June 1988.
[12] J. DONGARRA, A. SAMEH, AND D. SORENSEN, *Implementation of some concurrent algorithms for matrix factorization*, Parallel Comput., 3 (1986), pp. 25–34.
[13] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER, AND G. W. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.
[14] T. H. DUNIGAN, *Hypercube performance*, in Hypercube Multiprocessors, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 178–192.
[15] S. EISENSTAT, M. HEATH, C. HENKEL, AND C. ROMINE, *Modified cyclic algorithms for solving triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 589–600.
[16] L. V. FOSTER, *Rank and null space calculations using matrix decomposition without column interchanges*, Linear Algebra Appl., 74 (1986), pp. 47–71.
[17] K. GALLIVAN, W. JALBY, U. MEIER, AND A. SAMEH, *The impact of hierarchical memory systems on linear algebra algorithm design*, Tech. Report 625, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, September 1987.
[18] G. A. GEIST AND M. T. HEATH, *Parallel Cholesky factorization on a hypercube multiprocessor*, Tech. Report ORNL–6190, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, 1985.
[19] G. H. GOLUB, V. KLEMA, AND G. W. STEWART, *Rank degeneracy and least squares problems*, Tech. Report TR–456, Department of Computer Science, University of Maryland, 1976.
[20] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.
[21] M. T. HEATH AND C. H. ROMINE, *Parallel solution of triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 558–588.
[22] I. IPSEN, Y. SAAD, AND M. SCHULTZ, *Dense linear systems on a ring of processors*, Linear Algebra Appl., 77 (1986), pp. 205–239.
[23] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
[24] G. LI AND T. F. COLEMAN, *A new method for solving triangular systems on distributed memory message-passing multiprocessors*, SIAM J. Sci. Statist. Comput., 10 (1987), pp. 382–398.
[25] ———, *A parallel triangular solver for a hypercube multiprocessor*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 485–502.
[26] C. MOLER, *Matrix computation on distributed memory multiprocessors*, in Hypercube Multiprocessors 1986, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986.
[27] C. MOLER, J. LITTLE, AND S. BANGERT, PRO-MATLAB *User's Guide*, The Mathworks, Sherborn, MA, 1987.
[28] A. POTHEN, S. JHA, AND U. VEMAPULATI, *Orthogonal factorization on a distributed memory multiprocessor*, in Hypercube Multiprocessors 1987, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.
[29] Y. SAAD AND M. H. SCHULTZ, *Topological properties of hypercubes*, Tech. Report YALEU/DCS/RR-389, Department of Computer Science, Yale University, New Haven, CT, 1985.
[30] R. SCHREIBER, *Block Algorithms for Parallel Machines*, IMA Volumes in Mathematics and its Applications 13, Springer-Verlag, Berlin, 1988, pp. 197–207.
[31] R. SCHREIBER AND C. VAN LOAN, *A storage efficient WY representation for products of Householder transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 53–57.
[32] G. W. STEWART, *The efficient generation of random orthogonal matrices with an application to condition estimators*, SIAM J. Numer. Anal., 17 (1980), pp. 403–409.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 58–78, January 1991

A THEORETICAL COMPARISON OF THE ARNOLDI AND GMRES ALGORITHMS*

PETER N. BROWN†

Abstract. Two recently developed Krylov methods for solving linear systems are *Arnoldi's method* and the *Generalized Minimum Residual* (GMRES) *method.* The GMRES method has been considered superior to Arnoldi's method due in part to the fact that GMRES never breaks down in the way Arnoldi's algorithm can. However, it is shown that there is a relationship between breakdowns in the two methods. Specifically, it is shown that GMRES does exhibit breakdowns very similar to that of Arnoldi, often referred to as the "stagnation" of GMRES. A relationship between the norms of the residuals for Arnoldi and GMRES is also given which shows exactly how much larger the residual norm for Arnoldi is than that for GMRES. In general, the results in the paper suggest that if one of the methods performs poorly on a particular problem, then so will the other.

Key words. linear systems, iterative methods, stagnation, breakdown

AMS(MOS) subject classification. 65F10

1. Introduction.

Two recently developed Krylov methods for solving linear systems are *Arnoldi's method* [9] and the *Generalized Minimum Residual* (GMRES) *method* [12]. Both methods have been used successfully in applications, but no theoretical comparison has been done. We give here a theoretical analysis of the related performance aspects of the two methods. We note that Arnoldi and GMRES are equivalent to other methods for various special cases of the coefficient matrix A. For example, when A is symmetric and positive definite, Arnoldi is equivalent to the conjugate gradient (CG) method and GMRES is equivalent to the conjugate residual (CR) method. This then implies that the results presented here for Arnoldi and GMRES will also apply to the CG and CR algorithms, to the extent that they are relevant. Additionally, when the matrix A is symmetric and indefinite, Arnoldi is equivalent to the SYMMLQ algorithm and GMRES is equivalent to the MINRES algorithm. The results presented here then apply to these related methods, and are similar to those presented by Paige and Saunders in [8] for the SYMMLQ and MINRES algorithms.

One of the disadvantages of the Arnoldi algorithm is that it can break down and be unable to calculate an approximate solution. Partly for this reason, the GMRES method has been considered superior since it never breaks down in the way Arnoldi's algorithm can [12]. However, we show that GMRES does exhibit "breakdowns" very similar to that of Arnoldi, and that there is a relationship between breakdowns in the two methods. This type of breakdown for GMRES is often referred to as "stagnation." A second perceived disadvantage of Arnoldi is that it does not produce a sequence of approximations whose residual norms are nonincreasing, as GMRES does. We give a result which shows exactly how much larger the residual norm is for the Arnoldi algorithm than that for GMRES, when the approximations come from the same Krylov subspace. The results in the paper indicate that the performance of the two methods are related, and suggest that if one method performs poorly on a particular problem, then so will the other. This is consistent with the numerical results reported in Brown

* Received by the editors May 16, 1988; accepted for publication (in revised form) December 1, 1989. This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract W-7405-Eng-48, and was supported by the Department of Energy Office of Energy Research, Applied Mathematical Sciences Research Program.

† Computing and Mathematics Research Division, L-316, Lawrence Livermore National Laboratory, Livermore, California 94550 (brown@lll-crg.llnl.gov).

and Hindmarsh [3].

In §2, we review the description of the Arnoldi and GMRES algorithms and discuss some of the implementation details of the methods. In §3, we discuss how the Arnoldi algorithm can break down, and we then show the existence of a similar "breakdown" in the GMRES algorithm. We also present several results which give a relationship between breakdowns in the two methods, and then in §4 we give the proofs of these results. In §5 we prove a relationship between the size of the norms of the residuals for the Arnoldi and GMRES methods, and then in §6 we consider some examples illustrating the results in the paper. Finally, in §7 we make some concluding remarks.

2. The Arnoldi and GMRES algorithms. In this section we will review the Krylov subspace methods under consideration, and discuss their main properties. We start with a brief description of the Arnoldi and GMRES algorithms. Next, we comment on some implementation details of Arnoldi and GMRES, and then briefly discuss restarted versions of the two methods.

We are interested in solving linear systems of the form

$$Ax = b, \tag{2.1}$$

where A is an $n \times n$ real matrix which is nonsingular, and x and b are vectors of length n. Both Arnoldi and GMRES are iterative in nature, and so generate a sequence of iterates converging to the solution of (2.1). If x_0 is an initial guess for the true solution of (2.1), then letting $x = x_0 + z$, we have the equivalent system

$$Az = r_0, \tag{2.2}$$

where $r_0 = b - Ax_0$ is the initial *residual.* Let K_m be the *Krylov subspace*

$$K_m \equiv \text{span}\{r_0, Ar_0, \cdots, A^{m-1}r_0\}.$$

Arnoldi's method and GMRES both find an approximate solution

$$x_m = x_0 + z_m \text{ with } z_m \in K_m, \tag{2.3}$$

such that

$$(b - Ax_m) \perp K_m \text{ (equivalently } (r_0 - Az_m) \perp K_m) \tag{2.4}$$

for Arnoldi's method, and

$$\|b - Ax_m\|_2 = \min_{x \in x_0 + K_m} \|b - Ax\|_2 \ \ (= \min_{z \in K_m} \|r_0 - Az\|_2) \tag{2.5}$$

for GMRES. (Note that (2.5) is equivalent to requiring $(b - Ax_m) \perp AK_m$.) Here, $\|\cdot\|_2$ denotes the Euclidean norm on $\mathbf{R}^n$ and orthogonality is meant in the usual Euclidean sense.

The following algorithm is a version of the Arnoldi (GMRES) algorithm which generates orthonormal vectors $v_i, i = 1, 2, \cdots, m$, and then builds the vector x_m that satisfies (2.4) (or (2.5) for GMRES). In both algorithms, the v_i's are computed such that they form an orthonormal basis of the Krylov subspace K_m, where the first vector v_1 is obtained by normalizing r_0. The parameter ϵ below is a user-supplied tolerance on the norm of the residual which is used in deciding when to stop the iteration, and $(\cdot, \cdot)$ denotes the Euclidean inner product.

Algorithm: Arnoldi (GMRES)

(1) *Arnoldi process:*
- For an initial guess x_0, form $r_0 = b - Ax_0$.
- Compute $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.
- For $j = 1, 2, \cdots$, do:
  (a) Form Av_j and orthogonalize it against the previous $v_1, \cdots, v_j$ via

$$\begin{aligned} h_{i,j} &= (Av_j, v_i), \qquad i = 1, 2, \cdots, j, \\ \hat{v}_{j+1} &= Av_j - \sum_{i=1}^{j} h_{i,j} v_i, \\ h_{j+1,j} &= \|\hat{v}_{j+1}\|_2, \quad \text{and} \\ v_{j+1} &= \hat{v}_{j+1}/h_{j+1,j}. \end{aligned} \tag{2.6}$$

  (b) Compute the residual norm $\rho_j = \|b - Ax_j\|_2$ of the solution x_j that would be obtained if we stopped at this step.
  (c) If $\rho_j \leq \epsilon$ set $m = j$ and go to (2).

(2) *Form the approximate solution:*

Arnoldi: Define H_m to be the $m \times m$ (Hessenberg) matrix whose nonzero entries are the coefficients $h_{i,j}$, $1 \leq i \leq \min\{j+1, m\}$, $1 \leq j \leq m$ and define $V_m \equiv [v_1, v_2, \cdots, v_m]$.
- Compute $x_m^A = x_0 + z_m^A$, where $z_m^A = \beta V_m H_m^{-1} e_1$.

GMRES: Define $\bar{H}_m$ to be the $(m+1) \times m$ (Hessenberg) matrix whose nonzero entries are the coefficients $h_{i,j}$, $1 \leq i \leq j+1$, $1 \leq j \leq m$ and define $V_m \equiv [v_1, v_2, \cdots, v_m]$.
- Find the vector y_m^{GM} which minimizes $\|\beta e_1 - \bar{H}_m y\|_2$ over all vectors y in $\mathbf{R}^m$, where $e_1 = [1, 0, \cdots, 0]^T$.
- Compute $x_m^{GM} = x_0 + V_m y_m^{GM}$.

The first stage of the algorithms above is an Arnoldi loop which builds an orthonormal basis $V_m = [v_1, v_2, \cdots, v_m]$ of the Krylov subspace K_m. If we denote by V_j the $n \times j$ matrix with column vectors $v_1, v_2, \cdots, v_j$, then it follows immediately from (2.6) that (see also [9] and [12])

$$AV_m = V_m H_m + \hat{v}_{m+1} e_m^T, \tag{2.7}$$

where $e_m = [0, \cdots, 0, 1]^T \in \mathbf{R}^m$. This relation can be rewritten as

$$AV_m = V_{m+1} \bar{H}_m, \tag{2.8}$$

and is crucial in the development of the Arnoldi and GMRES methods.

The details on the practical implementation of the methods above are discussed at length in [9], [3], and [12]. We discuss here only those details relevant to the current situation. First, step (2) above computes the approximate solution x_m^A or x_m^{GM} in $x_0 + K_m$ that solves either (2.4) or (2.5). For Arnoldi, first note that condition (2.4) is equivalent to

$$V_m^T (b - Ax_m^A) = 0.$$

Since x_m^A is in the affine space $x_0 + K_m$, we have $x_m^A = x_0 + V_m y$ for some $y \in \mathbf{R}^m$. The above system is then equivalent to

$$V_m^T A V_m y = V_m^T r_0. \tag{2.9}$$

Next, from (2.7) it follows that $H_m = V_m^T A V_m$, using the fact that $\hat{v}_{m+1}$ is orthogonal to K_m. Also, since $r_0 = \beta v_1$, the right-hand side reduces to $V_m^T r_0 = \beta e_1$. Hence, we must solve the m-dimensional linear system

$$H_m y = \beta e_1. \tag{2.10}$$

This is accomplished by finding an LU decomposition of the Hessenberg matrix H_m, and we denote the solution of (2.10) by y_m^A. The solution x_m^A is then given by $x_m^A = x_0 + V_m y_m^A$.

For the GMRES algorithm, first let $z = V_m y$ for $y \in \mathbf{R}^m$. Then $\|r_0 - Az\|_2 = \|\beta v_1 - AV_m y\|_2$. Using (2.8), we have

$$\begin{aligned} \|\beta v_1 - AV_m y\|_2 &= \|V_{m+1}(\beta e_1 - \bar{H}_m y)\|_2 \\ &= \|\beta e_1 - \bar{H}_m y\|_2, \end{aligned}$$

since V_{m+1} has orthonormal columns. We denote by y_m^{GM} the solution of the minimization problem

$$\min_{y \in \mathbf{R}^m} \|\beta e_1 - \bar{H}_m y\|_2. \tag{2.11}$$

Then the optimal x is given by $x_m^{GM} = x_0 + V_m y_m^{GM}$. Note that (2.11) is a least squares problem of size $m+1$, and its coefficient matrix is upper Hessenberg. Problem (2.11) is solved using a QR factorization of the matrix $\bar{H}_m$, and in §4 we give a detailed discussion of its solution. See [3], [9], [10], and [12] for more specifics about Arnoldi and GMRES.

One important property of both algorithms is that the residual norm ρ_j referred to in step (1) above does not require the computation of the approximate solution x_j at every step. Instead an inexpensive formula is updated at each step while the factorization of the Hessenberg matrix H_m or $\bar{H}_m$ is updated. We derive this result for GMRES in §4 while discussing the QR factorization of $\bar{H}_m$. For the Arnoldi algorithm, see [2], [9], and [10]. In §5 we also give an alternative formula for Arnoldi which uses a QR factorization of H_m.

Another aspect of the above algorithms we have not yet considered is the ability to restart the methods. In a typical implementation of the above Krylov methods, a maximum value of m is dictated by storage considerations. If we let $m_{\max}$ be this value, then it is possible that $m = m_{\max}$ in the Arnoldi process, and yet ρ_m is still greater than ϵ, the tolerance on the residual r_m. In this case, we can set x_0 equal to x_m^A (or x_m^{GM}) and restart the Arnoldi process, effectively restarting the Krylov method. The convergence of such a procedure is not always guaranteed, but the idea seems to work well in practice.

Finally, we note that the Arnoldi algorithm is theoretically equivalent to the conjugate gradient method when A is symmetric and positive definite, and to the Lanczos method for solving linear systems when A is symmetric [9]. The GMRES algorithm is theoretically equivalent to the conjugate residual method when A is symmetric and positive definite, to GCR [4], [5], and to ORTHODIR [7] when A is only positive definite. Both Arnoldi and GMRES are guaranteed to obtain the exact solution of (2.1) in at most n steps using exact arithmetic, for any nonsingular $n \times n$ matrix A and right-hand side vector b.

3. Breakdowns. In this section we discuss how the Arnoldi and GMRES algorithms can break down, and then present several relationships between breakdowns in the two methods. The proofs of these results are then given in the following section.

We first discuss the two ways in which the Arnoldi algorithm can break down. This can happen if either $\hat{v}_{m+1} = 0$ so that v_{m+1} cannot be formed, or $H_{m_{\max}}$ is singular, which means that the maximum number of Arnoldi steps has been taken, but the final iterate cannot be formed. The first situation has often been referred to as a *happy* breakdown, since $\hat{v}_{m+1} = 0$ implies H_m is nonsingular and x_m^A is the exact solution of (2.1) (cf. Brown [1] or Saad [9]). The second case is more serious in that it causes a convergence failure of the algorithm. One possible recourse is to hope that H_j is nonsingular for some j among $1, \cdots, m_{\max} - 1$. If such a j exists, we can compute x_j^A and then restart the algorithm using x_j^A as the new initial guess x_0. It may not always be possible to do this, however, as the following example shows.

Example 3.1. Let A be the permutation matrix sending $e_1 \to e_2 \to \cdots \to e_n \to e_1$, where e_i is the ith standard basis vector in $\mathbf{R}^n$. Then

$$A = \begin{pmatrix} 0 & & & & 1 \\ 1 & \ddots & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{pmatrix}.$$

Consider solving (2.1), where $b = e_1$ and $x_0 = 0$. Then the true solution is $x_* = A^{-1}b = e_n$. We have $V_m = [e_1, \cdots, e_m]$, and H_m is given by

$$H_m = \begin{pmatrix} 0 & & & & 0 \\ 1 & \ddots & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{pmatrix} \quad \text{for } m < n,$$

with $H_n = A$. Hence, H_m is singular for $m < n$, and the Arnoldi iterates x_m^A do not exist for $m < n$, but $x_n^A = x_*$. Next, for GMRES

$$\bar{H}_m = \begin{pmatrix} H_m \\ 0 \cdots 0 \; 1 \end{pmatrix},$$

and the solution of (2.11) is given by

$$y_m^{GM} = (\bar{H}_m^T \bar{H}_m)^{-1} \bar{H}_m^T V_{m+1}^T b.$$

Since $V_{m+1}^T b = e_1$, and since $\bar{H}_m^T e_1 = 0$ $(m = 1, \cdots, n-1)$ because the first row of $\bar{H}_m$ is all zeros, the GMRES iterates satisfy

$$x_m^{GM} = V_m y_m^{GM} = 0 \; (m = 1, \cdots, n-1), \qquad x_n^{GM} = x_*.$$

Thus, for this example neither Arnoldi nor GMRES make any progress until $m = n$. Note that restarting would be of no use here. This example was also reported on in [3].

A second possibility when H_m is singular would be to try to find the minimum norm solution of

$$\min_{x \in x_0 + K_m} \|V_m^T (b - Ax)\|_2 = \min_{y \in \mathbf{R}^m} \|\beta e_1 - H_m y\|_2. \tag{3.1}$$

For the H_m in the example, it is trivial to form its singular value decomposition. We have $H_m = U_m \Sigma_m V_m^T$, where $U_m = [e_2, \cdots, e_m, e_1]$, $V_m = I_m$ (the $m \times m$ identity matrix), and Σ_m is given by

$$\Sigma_m = \begin{pmatrix} & & 0 \\ & I_{m-1} & \vdots \\ & & 0 \\ 0 \cdots 0 & & 0 \end{pmatrix}.$$

The solution of (3.1) is then given by $y_m^{LS} = \beta H_m^+ e_1$, where $H_m^+ = \Sigma_m U_m^T$ is the pseudoinverse of H_m. This then gives $y_m^{LS} = \beta \Sigma_m U_m^T e_1 = \beta \Sigma_m e_m = 0$ for all $m < n$. Thus, this second possibility produces no better results than GMRES. We show below that this will always happen. That is, when H_m is singular, solving (3.1) gives exactly the same solution as solving (2.11).

The example above illustrates how Arnoldi and GMRES can have catastrophic breakdowns. Even though the GMRES solution can always be calculated, the approximations are of no use since they are all zero until the very last one, which gives the exact solution. This example is also indicative of several relationships between the singularity of the Hessenberg matrix H_m and the solution of the least squares problem (2.11). We first state the results and discuss their theoretical importance, and then give their proofs in the next section.

THEOREM 3.1. *Suppose m steps of the Arnoldi process have been taken, and assume that H_m is singular. Then*

$$\min_{y \in \mathbf{R}^m} \|\beta e_1 - \bar{H}_m y\|_2 = \min_{y \in \mathbf{R}^{m-1}} \|\beta e_1 - \bar{H}_{m-1} y\|_2. \tag{3.2}$$

If y_j^{GM} is the solution of (2.11) *with m replaced by j, $j = m$ or $m-1$, then $y_m^{GM} = ((y_{m-1}^{GM})^T, 0)^T$, and it follows that $x_m^{GM} = x_{m-1}^{GM}$. Conversely, suppose m steps of the Arnoldi process have been taken and that* (3.2) *holds. Then H_m is singular.*

This theorem gives a clear indication of how the performance of the Arnoldi and GMRES algorithms are interrelated. If H_m is singular, so that x_m^A does not exist, then GMRES was unsuccessful in reducing the norm of the residual, and $x_m^{GM} = x_{m-1}^{GM}$. The converse is also true. Thus, either both methods make progress, or both fail. However, the result says nothing about the errors in the two approximations. If one uses the size of the residual norm as a measure of accuracy, then clearly x_m^{GM} is to be preferred over x_m^A in view of (2.5).

Next, define the function $g(x)$ by

$$g(x) \equiv \frac{1}{2} \|Ax - b\|_2^2.$$

We say that the vector v is an *ascent* direction for g at x if

$$g(x + \lambda v) > g(x)$$

for all nonzero λ. We also have the following result which gives an equivalent condition for the matrix H_m to be singular.

COROLLARY 3.2. *Suppose that m steps of the Arnoldi process have been taken. Then H_m is singular if and only if v_m is an ascent direction for g at x_{m-1}^{GM}.*

These two results indicate what happens when the Arnoldi and GMRES algorithms begin performing poorly. If H_m is nearly singular, or equivalently if $\|r_m^{GM}\|_2$ is

only slightly smaller than $\|r_{m-1}^{GM}\|_2$, then the new Krylov vector v_m is close to being an ascent direction for g at x_{m-1}^{GM}.

THEOREM 3.3. *Suppose m steps of the Arnoldi process have been taken, and assume that H_m is singular. Then*

$$\min_{x \in x_0 + K_m} \|V_m^T(b - Ax)\|_2 = \min_{y \in \mathbf{R}^m} \|\beta e_1 - H_m y\|_2 = \min_{y \in \mathbf{R}^m} \|\beta e_1 - \bar{H}_m y\|_2. \tag{3.3}$$

Furthermore, $y_m^{LS} = y_m^{GM}$, where y_m^{LS} and y_m^{GM} are the solutions to the last two problems (respectively) in (3.3).

This last result also suggests that the GMRES algorithm may be the better of the two methods when using the residual norm as the measure of accuracy in the approximate solutions.

4. Proofs. Before we begin the proofs of the theorems stated in the last section, it will be necessary to review some of the details involved in the QR factorization of the Hessenberg matrix $\bar{H}_m$. This is accomplished using Givens rotations. As Saad and Schultz have noted [12], it is desirable to be able to update the factorization of $\bar{H}_m$ progressively as each column of it appears. This allows the residual norm $\|b - Ax_m^{GM}\|_2$ to be calculated without computing x_m^{GM} at each step.

From Brown and Hindmarsh [3], let $F_{j,k} \in \mathbf{R}^{(k+1)\times(k+1)}$ be the rotation matrix which rotates e_j and e_{j+1} by the angle θ_j, namely,

$$F_{j,k} = \begin{pmatrix} I_{j-1} & & & \\ & c_j & -s_j & \\ & s_j & c_j & \\ & & & I_{k-j} \end{pmatrix} \leftarrow \text{row } j+1,$$

where $c_j = \cos(\theta_j)$ and $s_j = \sin(\theta_j)$ for $j \le k$. From the notation, θ_j is assumed independent of k, and so the rotations $F_{j,k}$ and $F_{j,k+1}$ differ by only the addition of a column and row of the identity matrix. Now, suppose that the rotations $F_{1,j}, \cdots, F_{j,j}$ have been applied to $\bar{H}_j$, giving

$$F_{j,j} F_{j-1,j} \cdots F_{1,j} \bar{H}_j = R_j \in \mathbf{R}^{(j+1)\times j},$$

where R_j is upper triangular with its last row containing all zeros. The next step of the Arnoldi process produces the last row and column of the matrix $\bar{H}_{j+1}$. Let $d = (d', h)^T$, where $d' \in \mathbf{R}^{j+1}$ and $h = h_{j+2,j+1}$, be the new column. R_{j+1} is obtained by first forming $\bar{d} = F_{j,j+1} F_{j-1,j+1} \cdots F_{1,j+1} d$ (note that the last component h is unaffected by this multiplication), and then choosing the rotation $F_{j+1,j+1}$ to eliminate h in $\bar{d}$. This gives

$$\begin{aligned} c_{j+1} &= r/\sqrt{r^2 + h^2}, \\ s_{j+1} &= -h/\sqrt{r^2 + h^2}, \end{aligned}$$

letting r denote the next-to-last component of $\bar{d}$. After m steps, the resulting decomposition is

$$Q_m^T \bar{H}_m = R_m, \tag{4.1}$$

where $Q_m^T = F_{m,m} F_{m-1,m} \cdots F_{1,m} \in \mathbf{R}^{(m+1)\times(m+1)}$, and $R_m \in \mathbf{R}^{(m+1)\times m}$ is upper triangular with zeros in its last row. We note that if m steps of the Arnoldi loop

have been possible, then $h_{j+1,j} \neq 0$ for $j = 1, \cdots, m-1$. If $h_{m+1,m} = 0$, then H_m is nonsingular from the remarks at the beginning of §3, and so $\bar{H}_m$ would have full column rank. If $h_{m+1,m} \neq 0$, then $\bar{H}_m$ still has full column rank due to its upper Hessenberg structure. Thus, the solution of the least squares problem (2.11) is unique, and it is found using (4.1). We have

$$\begin{aligned} \|\beta e_1 - \bar{H}_m y\|_2 &= \|\beta e_1 - Q_m R_m y\|_2 \\ &= \|\beta Q_m^T e_1 - R_m y\|_2, \end{aligned}$$

since Q_m is orthogonal. Letting

$$R_m = \begin{pmatrix} \bar{R}_m \\ 0 \cdots 0 \end{pmatrix},$$

and $g_m = \beta Q_m^T e_1 = (\bar{g}_m, g)^T$ with $\bar{g}_m \in \mathbf{R}^m$ and $g \in \mathbf{R}$, we have that the value of y which solves (2.11) is then

$$y_m^{GM} = (\bar{R}_m)^{-1} \bar{g}_m, \tag{4.2}$$

and it follows that

$$\|b - Ax_m^{GM}\|_2 = \|\beta Q_m^T e_1 - R_m y_m^{GM}\|_2 = |g|.$$

We have $g = \beta e_1^T Q_m e_{m+1}$, and letting $Q_m = [q_1^m, \cdots, q_{m+1}^m]$ gives

$$\|b - Ax_m^{GM}\|_2 = |g| = \beta |(q_{m+1}^m)^T e_1|.$$

In [3] it is shown (cf. [3, eq. (2.26)]) that

$$q_{m+1}^m = \left(\prod_{i=1}^{m} s_i, c_1 \prod_{i=2}^{m} s_i, c_2 \prod_{i=3}^{m} s_i, \cdots, c_{m-1} s_m, c_m \right)^T. \tag{4.3}$$

It now follows that $|g| = \beta |s_1 s_2 \cdots s_m|$, and thus we have

$$\|b - Ax_m^{GM}\|_2 = \beta |s_1 s_2 \cdots s_m|. \tag{4.4}$$

This last equation gives an inexpensive way to calculate the norm of the residual associated with x_m^{GM}. See [3] and [12] for more details.

Proof of Theorem 3.1. First, note that if m steps of the Arnoldi process have been possible, then it must be the case that $h_{j+1,j} \neq 0$ for $j = 1, \cdots, m-1$. Otherwise, the Arnoldi process would have broken down at an earlier stage. From above, if $h_{m+1,m} = 0$ (i.e., if $\hat{v}_{m+1} = 0$), then H_m is nonsingular and $x_m^A = x_m^{GM}$ is the exact solution to (2.1). Thus, H_m singular implies that $h_{m+1,m} \neq 0$. Also, when H_m is singular the dimension of its null space is one due to its upper Hessenberg structure and $h_{j+1,j} \neq 0$ for $j = 1, \cdots, m-1$.

Next, consider the QR factorizations of $\bar{H}_m$ and $\bar{H}_{m-1}$. Let

$$\bar{H}_m = Q_m R_m \quad \text{and} \quad \bar{H}_{m-1} = Q_{m-1} R_{m-1},$$

where

$$R_m = \begin{pmatrix} \bar{R}_m \\ 0 \cdots 0 \end{pmatrix} \quad \text{and} \quad R_{m-1} = \begin{pmatrix} \bar{R}_{m-1} \\ 0 \cdots 0 \end{pmatrix}.$$

Note that

$$\bar{H}_m = \begin{pmatrix} & & h_{1,m} \\ & \bar{H}_{m-1} & \vdots \\ & & h_{m,m} \\ 0 \cdots 0 & & h_{m+1,m} \end{pmatrix} = \begin{pmatrix} \bar{H}_{m-1} & h_m \\ 0 \cdots 0 & h_{m+1,m} \end{pmatrix}$$

and $H_m = [\bar{H}_{m-1}, h_m]$, if we let $h_m = (h_{1,m}, \cdots, h_{m,m})^T$. Since H_m is singular, and from the fact that $\bar{H}_{m-1}$ has full column rank, we must have $h_m = \bar{H}_{m-1}\tilde{y}$ for some nonzero $\tilde{y} \in \mathbf{R}^{m-1}$. Thus,

$$h_m = Q_{m-1}R_{m-1}\tilde{y} = Q_{m-1}\begin{pmatrix} \bar{R}_{m-1}\tilde{y} \\ 0 \end{pmatrix}. \tag{4.5}$$

As noted above, the QR factorizations of $\bar{H}_m$ and $\bar{H}_{m-1}$ are related, and we can write $Q_m = \tilde{Q}_{m-1}F_{m,m}^T$, where $F_{m,m}$ is the last Givens rotation defined by

$$F_{m,m} = \begin{pmatrix} I_{m-1} & & \\ & c_m & -s_m \\ & s_m & c_m \end{pmatrix} \quad \text{and} \quad \tilde{Q}_{m-1} = \begin{pmatrix} & & 0 \\ & Q_{m-1} & \vdots \\ & & 0 \\ 0 \cdots 0 & & 1 \end{pmatrix}.$$

Hence, we have

$$F_{m,m}\tilde{Q}_{m-1}^T\bar{H}_m = R_m$$

or

$$F_{m,m}\begin{pmatrix} Q_{m-1}^T\bar{H}_{m-1} & Q_{m-1}^T h_m \\ 0 \cdots 0 & h_{m+1,m} \end{pmatrix} = R_m. \tag{4.6}$$

From (4.5), we have $Q_{m-1}^T h_m = (\tilde{y}^T\bar{R}_{m-1}^T, 0)^T$. Using this last equation, and the fact that $Q_{m-1}^T\bar{H}_{m-1} = R_{m-1}$, we have

$$F_{m,m}\begin{pmatrix} R_{m-1} & \begin{pmatrix} \bar{R}_{m-1}\tilde{y} \\ 0 \end{pmatrix} \\ 0 \cdots 0 & h_{m+1,m} \end{pmatrix} = R_m.$$

Thus, the c_m and s_m in $F_{m,m}$ are chosen so that

$$\begin{pmatrix} c_m & -s_m \\ s_m & c_m \end{pmatrix}\begin{pmatrix} 0 \\ h_{m+1,m} \end{pmatrix} = \begin{pmatrix} \pm h_{m+1,m} \\ 0 \end{pmatrix},$$

with $c_m^2 + s_m^2 = 1$. The values of c_m and s_m must then satisfy $c_m = 0$ and $|s_m| = 1$. If y_m^{GM} is the solution to (2.11), then from (4.4) we have

$$\begin{aligned} \|\beta e_1 - \bar{H}_m y_m^{GM}\|_2 &= \beta \cdot |s_1 s_2 \cdots s_m| = \beta \cdot |s_1 s_2 \cdots s_{m-1}| \\ &= \|\beta e_1 - \bar{H}_{m-1} y_{m-1}^{GM}\|_2. \end{aligned} \tag{4.7}$$

It follows from the uniqueness of y_m^{GM} that we must have $y_m^{GM} = ((y_{m-1}^{GM})^T, 0)$. This then also gives $x_m^{GM} = x_{m-1}^{GM}$.

For the converse, suppose m steps of the Arnoldi process have been taken and assume that (3.2) holds. It follows from (4.4) that $c_m = 0$ and $s_m = \pm 1$. Using (4.6), we can write

$$F_{m,m} \begin{pmatrix} R_{m-1} & Q_{m-1}^T h_m \\ 0 \cdots 0 & h_{m+1,m} \end{pmatrix} = R_m, \tag{4.8}$$

since $R_{m-1} = Q_{m-1}^T \bar{H}_{m-1}$. (Note that (4.6) holds whether or not H_m is singular.) Now, the last Givens rotation $F_{m,m}$ is chosen so that R_m is upper triangular with its last row containing all zeros. But we know that $F_{m,m}$ has the form

$$F_{m,m} = \begin{pmatrix} I_{m-1} & & \\ & 0 & \pm 1 \\ & \mp 1 & 0 \end{pmatrix}.$$

Hence, it must be the case that the last component of the m-vector $Q_{m-1}^T h_m$ is zero. Now, $Q_{m-1}^T h_m = (q_{1,m-1}^T h_m, \cdots, q_{m-1,m-1}^T h_m, 0)^T \equiv (w, 0)^T$. If $w = 0$, then $h_m = 0$ because Q_{m-1}^T is orthogonal, and so $H_m = [\bar{H}_{m-1}, h_m]$ is singular. So assume that $w \neq 0$ in the remainder of the proof.

Let $s = (\bar{s}^T, t)$, with $\bar{s} \in \mathbf{R}^{m-1}$ and $t \in \mathbf{R}$. We then have

$$\begin{aligned} H_m s &= [\bar{H}_{m-1}, h_m] s \\ &= [Q_{m-1} R_{m-1}, h_m] s \\ &= Q_{m-1} [R_{m-1}, Q_{m-1}^T h_m] s \\ &= Q_{m-1} (R_{m-1} \bar{s} + t Q_{m-1}^T h_m). \end{aligned}$$

Using $R_{m-1} = (\bar{R}_{m-1}^T, 0)^T$ and the definition of w, we have

$$H_m s = Q_{m-1} \begin{pmatrix} \bar{R}_{m-1} \bar{s} + tw \\ 0 \end{pmatrix}. \tag{4.9}$$

Since $\bar{R}_{m-1}$ is nonsingular, given any t there exists an $\bar{s}$ such that the right-hand side of (4.9) is zero. Thus, H_m is singular. □

Before we begin the proof of Corollary 3.2, recall that the function $g(x)$ is defined by

$$g(x) \equiv \frac{1}{2} \|Ax - b\|_2^2,$$

and that the new Krylov basis vector v_m is an ascent direction for g at x_{m-1}^{GM} whenever

$$g(x_{m-1}^{GM} + \lambda v_m) > g(x_{m-1}^{GM})$$

for all nonzero λ.

Proof of Corollary 3.2. Note that x_m^{GM} is the solution to the minimization problem

$$\min_{x \in x_0 + K_m} g(x).$$

So, if H_m is singular we also have $x_m^{GM} = x_{m-1}^{GM}$ and hence

$$g(x_{m-1}^{GM} + \lambda v_m) \geq g(x_{m-1}^{GM}) \tag{4.10}$$

for all λ. Defining $h(\lambda)$ to be the function

$$h(\lambda) = g(x_{m-1}^{GM} + \lambda v_m),$$

we have

$$\begin{aligned} h'(\lambda) = \frac{dh(\lambda)}{d\lambda} &= \nabla g(x_{m-1}^{GM} + \lambda v_m)^T v_m \\ &= (A^T(A(x_{m-1}^{GM} + \lambda v_m) - b), v_m) \\ &= (r_{m-1}^{GM} + \lambda A v_m, A v_m) \\ &= (r_{m-1}^{GM})^T A v_m + \lambda \|A v_m\|_2^2, \end{aligned}$$

where $(\cdot,\cdot)$ is the Euclidean inner product and $\nabla = (\partial/\partial x_1, \cdots, \partial/\partial x_n)^T$. It follows from (4.10) that $h'(0) = 0$, and so we must have $(r_{m-1}^{GM})^T A v_m = 0$, or that r_{m-1}^{GM} is orthogonal to Av_m. It then follows from the nonsingularity of A that when λ is nonzero there is strict inequality in (4.10).

Next, suppose that (4.10) holds with strict inequality when λ is nonzero. Then it again follows that $(r_{m-1}^{GM})^T A v_m = 0$. Since it is already known that r_{m-1}^{GM} is orthogonal to AK_{m-1} (see [11]), r_{m-1}^{GM} is also orthogonal to AK_m. But r_m^{GM} is the unique residual which is orthogonal to AK_m. Hence, $r_{m-1}^{GM} = r_m^{GM}$, which gives H_m singular by Theorem 3.1. □

One interesting fact in the proof above is that whenever H_m is singular the residual r_{m-1}^{GM} is already orthogonal to AK_m. Since requiring the residual r_m^{GM} to be orthogonal to AK_m uniquely determines the GMRES solution, we must then have $x_m^{GM} = x_{m-1}^{GM}$.

Proof of Theorem 3.3. From the proof of Theorem 3.1, we have $H_m = [\bar{H}_{m-1}, h_m]$, where $h_m = (h_{1,m}, \cdots, h_{m,m})^T$. Using the QR factorization of $\bar{H}_{m-1}$ it is trivial to obtain the corresponding factorization of H_m. Indeed, we have

$$\begin{aligned} H_m = [\bar{H}_{m-1}, h_m] &= [Q_{m-1}R_{m-1}, h_m] \\ &= Q_{m-1}[R_{m-1}, Q_{m-1}^T h_m] \\ &= Q_{m-1}\tilde{R}_m, \end{aligned} \tag{4.11}$$

letting $\tilde{R}_m = [R_{m-1}, Q_{m-1}^T h_m] \in \mathbf{R}^{m\times m}$. When H_m is singular, $\tilde{R}_m$ will also be singular. Thus, to find the minimum norm solution of

$$\min_{y\in\mathbf{R}^m} \|\beta e_1 - H_m y\|_2,$$

we must further factor $\tilde{R}_m$. Following Golub and Van Loan [6], we can use a sequence of Householder transformations $E_1, \cdots, E_m$ to perform the final reduction, giving

$$E_m E_{m-1} \cdots E_1 \tilde{R}_m^T = S_m^T = \begin{pmatrix} & & 0 \\ & \bar{S}_m^T & \vdots \\ & & 0 \\ 0 \cdots 0 & & 0 \end{pmatrix},$$

where $\bar{S}_m \in \mathbf{R}^{(m-1)\times(m-1)}$ is nonsingular and upper triangular. Then letting $Z_m = E_m \cdots E_1$, we have

$$\begin{aligned} \beta e_1 - H_m y &= \beta e_1 - Q_{m-1}\tilde{R}_m y \\ &= Q_{m-1}(\beta Q_{m-1}^T e_1 - S_m Z_m y) \\ &= Q_{m-1}(\beta Q_{m-1}^T e_1 - S_m w), \end{aligned}$$

where $w = Z_m y$. Thus,

$$\begin{aligned}
\|\beta e_1 - H_m y\|_2^2 &= \|\beta Q_{m-1}^T e_1 - S_m w\|_2^2 \\
&= \left\| \beta Q_{m-1}^T e_1 - \begin{pmatrix} & & 0 \\ & \bar{S}_m & \vdots \\ & & 0 \\ 0 \cdots 0 & & 0 \end{pmatrix} w \right\|_2^2 \\
&= |g|^2 + \|\bar{g} - \bar{S}_m \bar{w}\|_2^2,
\end{aligned}$$

letting $Q_{m-1}^T e_1 = (\bar{g}, g)^T$, and $w = (\bar{w}, t)^T$. Within w, it is clear that $\bar{w}$ is uniquely determined. The last component t contributes nothing to the residual, and hence the minimum norm solution would have $w = (\bar{w}, 0)^T$. Hence, the minimum norm solution of (3.1) is $y_m^{LS} = Z_m^T(\bar{w}, 0)^T$. Since $g = \beta s_1 \cdots s_{m-1}$ from (4.3), we have

$$\min_{y \in \mathbf{R}^m} \|\beta e_1 - H_m y\|_2 = |g| = \beta |s_1 \cdots s_{m-1}|.$$

Thus, relation (3.3) follows immediately from (4.7). To see that $y_m^{LS} = y_m^{GM}$, note first that by (3.2) and (4.7)

$$\min_{y \in \mathbf{R}^m} \|\beta e_1 - H_m y\|_2 = \min_{y \in \mathbf{R}^{m-1}} \|\beta e_1 - \bar{H}_{m-1} y\|_2.$$

Since the second minimum is taken over a subset of the first minimum, and since the two are equal, it is clear (by uniqueness) that $y_m^{LS} = ((y_{m-1}^{GM})^T, 0)^T = y_m^{GM}$. □

5. A relationship between the residual norms for Arnoldi and GMRES. In this section we first discuss a variant of the Arnoldi algorithm which uses a QR factorization of the upper Hessenberg matrix H_m to solve (2.10) instead of the usual LU factorization. We next derive an expression for the norm of the residual r_m^A analogous to that for GMRES. This expression is then used in deriving the main result of this section which gives an equation for $\|r_m^A\|_2$ in terms of $\|r_m^{GM}\|_2$.

As noted above, there is a relationship between H_m and $\bar{H}_{m-1}$, namely,

$$H_m = [\bar{H}_{m-1}, h_m],$$

where $h_m = (h_{1,m}, \cdots, h_{m,m})^T$. Using the QR factorization of $\bar{H}_{m-1}$ it is trivial to obtain the corresponding factorization of H_m. We have

$$\begin{aligned}
H_m = [\bar{H}_{m-1}, h_m] &= [Q_{m-1} R_{m-1}, h_m] \\
&= Q_{m-1}[R_{m-1}, Q_{m-1}^T h_m] \\
&= Q_{m-1} \tilde{R}_m,
\end{aligned}$$

letting $\tilde{R}_m = [R_{m-1}, Q_{m-1}^T h_m] \in \mathbf{R}^{m \times m}$. Assuming H_m is nonsingular, the solution of (2.10) is found by solving the upper triangular system

$$\tilde{R}_m y = \beta Q_{m-1}^T e_1 = \beta \begin{pmatrix} q_{1,m-1}^T e_1 \\ \vdots \\ q_{m,m-1}^T e_1 \end{pmatrix}. \tag{5.1}$$

Let y_m^A denote the solution of this system. Next, using (2.7) we have

$$\begin{aligned} b - Ax_m^A &= r_0 - AV_m y_m^A \\ &= r_0 - V_m H_m y_m^A - \hat{v}_{m+1} e_m^T y_m^A \\ &= V_m(\beta e_1 - H_m y_m^A) - \hat{v}_{m+1} e_m^T y_m^A \\ &= -\hat{v}_{m+1} e_m^T y_m^A. \end{aligned}$$

Thus,

$$\|b - Ax_m^A\|_2 = h_{m+1,m} |e_m^T y_m^A|,$$

since $h_{m+1,m} = \|\hat{v}_{m+1}\|_2$. From (4.3) and (5.1), it follows that

$$e_m^T y_m^A = \beta s_1 \cdots s_{m-1} / \tilde{r}_{m,m},$$

where $\tilde{r}_{m,m}$ denotes the (m,m) element of $\tilde{R}_m$. Therefore,

$$\|b - Ax_m^A\|_2 = \beta |s_1 \cdots s_{m-1} / \tilde{r}_{m,m}| \cdot h_{m+1,m}. \tag{5.2}$$

This last equation gives an economical way to update the residual norm $\|r_m^A\|_2$ as the QR factorization of H_m is updated in a manner very similar to that done for the GMRES algorithm. This version of the Arnoldi algorithm could then easily handle the case where $H_{m_{\max}}$ is singular by using Householder transformations as in the proof of Theorem 3.3. Although this implementation would be more expensive, it would nonetheless never break down in the way the LU factorization version of the algorithm can.

Our main result in this section makes use of (5.2), and gives a relationship between the norms of the residuals for Arnoldi and GMRES.

THEOREM 5.1. *Suppose m steps of the Arnoldi process have been taken, and H_m is nonsingular. Then*

$$\|r_m^A\|_2 = \|r_m^{GM}\|_2 \cdot \sqrt{1 + (h_{m+1,m}/\tilde{r}_{m,m})^2}. \tag{5.3}$$

Proof. It is clear from (4.4) and (5.2) that (5.3) is equivalent to

$$\beta |s_1 \cdots s_{m-1} / \tilde{r}_{m,m}| \cdot h_{m+1,m} = \beta |s_1 \cdots s_m| \sqrt{1 + (h_{m+1,m}/\tilde{r}_{m,m})^2},$$

or that

$$|s_m| = h_{m+1,m} / \sqrt{\tilde{r}_{m,m}^2 + h_{m+1,m}^2}.$$

But this last equation follows easily from (4.6) and the definition of the Givens rotation $F_{m,m}$. □

From (5.2) and (4.4), we also have

$$\|r_m^A\|_2 = \frac{h_{m+1,m}}{|\tilde{r}_{m,m}|} \cdot \|r_{m-1}^{GM}\|_2.$$

Next, from (5.3)

$$\frac{h_{m+1,m}}{|\tilde{r}_{m,m}|} = \sqrt{\frac{\|r_m^A\|_2^2}{\|r_m^{GM}\|_2^2} - 1}.$$

Thus,

$$\frac{\|r_m^A\|_2^2}{\|r_{m-1}^{GM}\|_2^2} = \frac{\|r_m^A\|_2^2}{\|r_m^{GM}\|_2^2} - 1,$$

from which it follows that

$$\|r_m^A\|_2 = \|r_m^{GM}\|_2(1 - s_m^2)^{-1/2}. \tag{5.4}$$

This last result then shows that the residual norm for Arnoldi grows without bound as $s_m \to 1$, which corresponds to the stagnation of GMRES.

6. Numerical results. In this final section we first present a somewhat more realistic example which illustrates the main results of the paper. This example also exhibits poor performance of the Arnoldi and GMRES algorithms, and we present it partly because of the lack of tractable examples when studying the Arnoldi and GMRES algorithms. As we show below, for this problem it is possible to write down explicit expressions for the Krylov basis vectors, and for the residual norm calculation. There are very few examples of this type, and so we present it in an effort to facilitate the investigation of the peformance of Krylov subspace projection methods. We then consider Arnoldi and GMRES applied to the solution of a related two-point boundary value problem. We note that these examples are indicative of the fact claimed above that either both methods (Arnoldi and GMRES) will perform well or both will perform poorly. However, we make no claim as to why the methods perform as they do. This will be the subject of future work.

Example 6.1. Let A be a skew-symmetric matrix of the form

$$A = \begin{pmatrix} 0 & 1 & & & \\ -1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 0 & 1 \\ & & & -1 & 0 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \\ -\alpha \end{pmatrix},$$

where $\alpha = 1/\sqrt{2}$. Consider solving (2.1), where n is even and $x_0 = 0$. Then the true solution is $x_* = A^{-1}b = (\alpha, \cdots, \alpha)^T$. An easy calculation gives

$$H_m = \begin{pmatrix} 0 & -1 & & & \\ 1 & 0 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 0 & -1 \\ & & & 1 & 0 \end{pmatrix} \in \mathbf{R}^{m \times m}$$

for $m \leq n$, and $h_{m+1,m} = 1$ for $m < n$, with the Krylov basis vectors given by

$$v_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ (-1)^{i+1}\alpha \\ 0 \\ \vdots \\ 0 \\ -\alpha \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{matrix} \\ \\ \\ \leftarrow \text{element } i \\ \\ \\ \\ \leftarrow \text{element } n+1-i \\ \\ \\ \\ \end{matrix} \qquad \text{for } i = 1, \cdots, n.$$

Since A is skew symmetric, so is $H_m = V_m^T A V_m$ for all m. Hence, H_m is singular for odd m, and the Arnoldi iterates x_m^A do not exist for m odd. For GMRES, we then have $\|r_{2k}^{GM}\|_2 = \|r_{2k+1}^{GM}\|_2$ for $k = 1, \cdots, n/2 - 1$. For $k = 1, \cdots, n/2 - 1$, the Givens sine coefficients are $s_{2k} = -(k/(k+1))^{1/2}$ and $s_{2k-1} = -1$. The element $\tilde{r}_{2k,2k}$ of $\tilde{R}_{2k}$ in the QR factorization of H_{2k} is given by $\tilde{r}_{2k,2k} = 1/\sqrt{k}$. Thus, we have

$$\|r_{2k}^A\|_2 = \beta |s_1 \cdots s_{2k-1}/\tilde{r}_{2k,2k}| h_{2k+1,2k} = 1, \quad \text{and}$$
$$\|r_{2k}^{GM}\|_2 = \|r_{2k+1}^{GM}\|_2 = \beta |s_1 \cdots s_{2k}| = 1/\sqrt{k+1}$$

for all $k = 1, \cdots, n/2 - 1$, since $\beta = 1$. Thus, Arnoldi is ineffective in reducing the norm of the residual until the very last iterate, and GMRES makes very slow progress until the last iterate. The restarted algorithms also perform poorly on this example, and we note that the poor performance does not seem to be linked to the condition number of A, as Fig. 6.1 shows a plot of the 2-norm condition number of A versus n for various even values of n from 2 to 40. (For this example, the eigenvalues of A are $\lambda_k = i\cos(k\pi/(n+1))$, with $k = 1, \cdots, n$, and so the condition number is $\cos(\pi/(n+1))/\cos(n\pi/(2n+2))$.)

We also consider A above perturbed by a small scalar times the identity matrix to give $A(\epsilon) = A + \epsilon I$, for $\epsilon > 0$ and small. Arnoldi applied to the linear system $A(\epsilon)x = b$ has a sawtooth graph for iteration count versus the residual norm, while GMRES will have a nonincreasing staircase graph. While both methods perform poorly for this perturbed problem, GMRES would be preferred due to the fact that the residual norm will never increase. Figs. 6.2 and 6.3 show the behavior of GMRES and Arnoldi for various values of ϵ when $n = 40$. Notice the sawtooth behavior of Arnoldi for $\epsilon = 0.05$.

We should also note that the poor performance here is not necessarily linked to skew-symmetric matrices, since a similar situation occurs for the linear system $|A|x = |b|$ with A and b defined as above, and where the absolute value sign simply means to take the absolute value of all the elements in A and b. In this case, the spectrum of $|A|$ is that of A rotated by 90 degrees, and it is possible to derive simple relationships for H_m and the v_i's similar to those obtained above for $Ax = b$. It happens that with a zero initial guess the initial residual $r_0 = |b|$ is orthogonal to half of the eigenspace of $|A|$, and so Arnoldi and GMRES will take at most $n/2$ iterations to obtain the solution in exact arithmetic. Figs. 6.4 and 6.5 show the behavior of

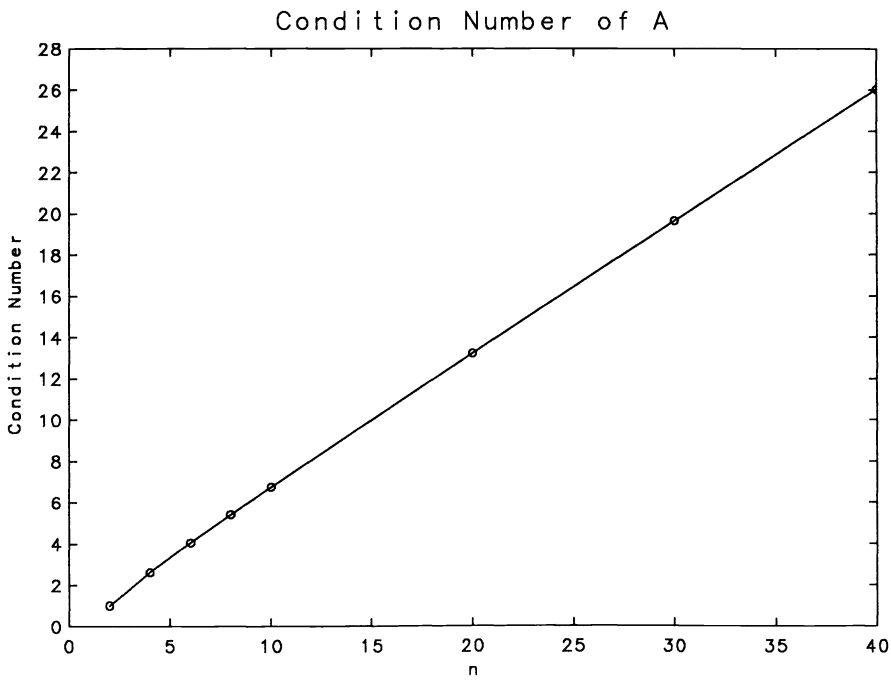


FIG. 6.1. *Condition number of A versus n.*

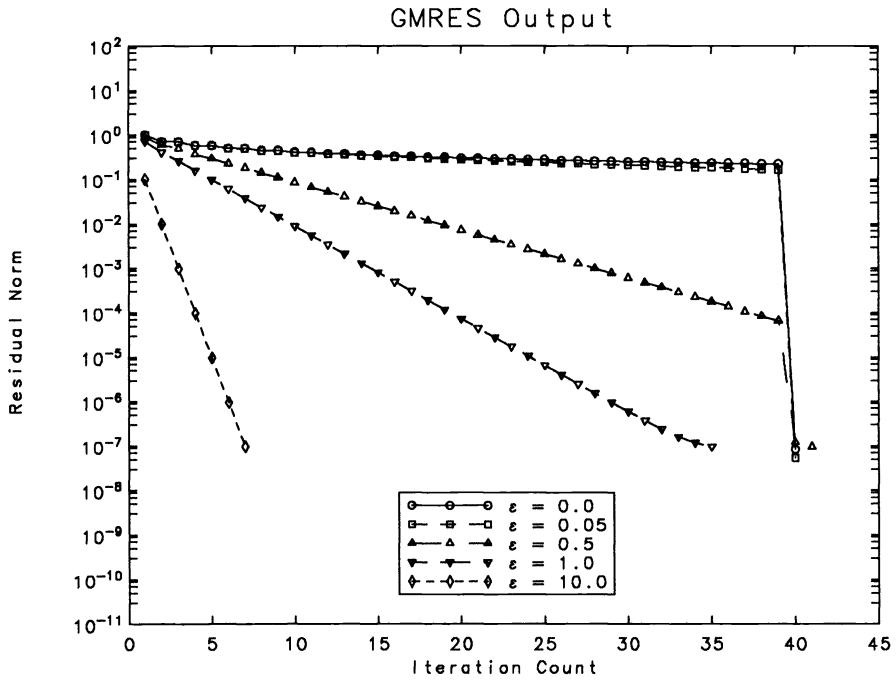


FIG. 6.2. *GMRES applied to* $(A + \epsilon I)x = b$.

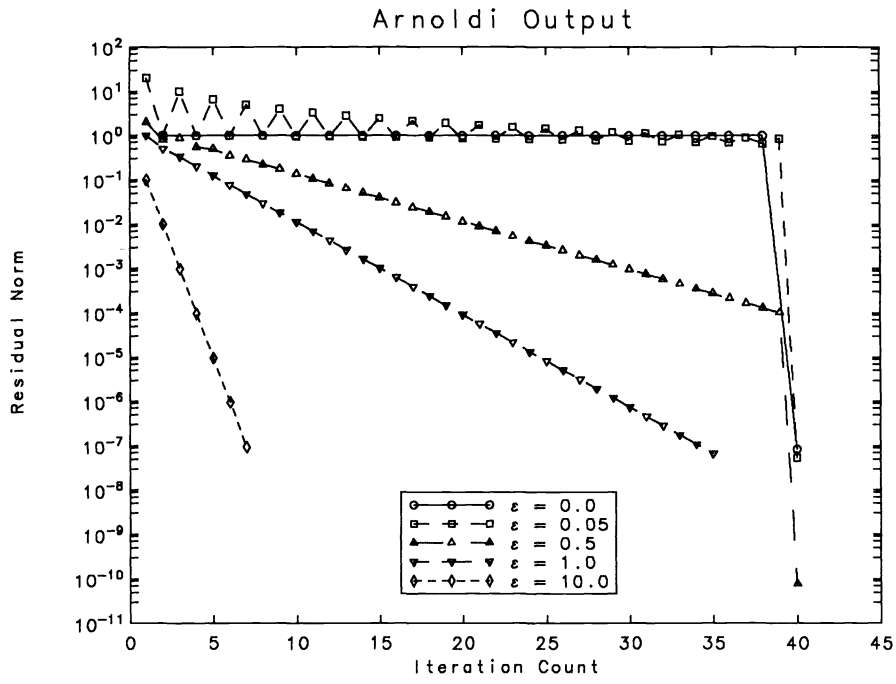


FIG. 6.3. *Arnoldi applied to* $(A + \epsilon I)x = b$.

GMRES and Arnoldi applied to the problem $(|A| + \epsilon I)x = |b|$ for various values of ϵ when $n = 40$. Again note the sawtooth behavior of the Arnoldi iteration.

Finally, the above matrix A (up to a scalar factor) results when letting $a \to 0$ in the two-point boundary value problem

$$au_{xx} + u_x = f \quad \text{on } 0 < x < 1,$$
$$u(0) = u(1) = 0,$$

where $f(x) \equiv 1$ and $a > 0$ is a constant, and if centered finite differences are used to approximate the first- and second-order derivatives. Figs. 6.6 and 6.7 show the behavior of GMRES and Arnoldi on this problem for various values of the scalar a when the interval $0 < x < 1$ is discretized into 40 subintervals (i.e., $n = 40$). Figs. 6.8 and 6.9 are magnified portions of Figs. 6.6 and 6.7, respectively. Again, note the sawtooth behavior of Arnoldi for small a. We also remark that, in general, centered differences for the first derivative terms are not recommended. However, it is worth pointing out that both GMRES and Arnoldi perform poorly for the two largest values of a, and for these two values the approximate solution has at least four significant digits.

7. Concluding remarks. In this paper we have given several relationships between the residual norms for Arnoldi and GMRES which suggest that either both methods will perform well on a particular problem, or both will perform poorly. We have also given a relationship which shows exactly how much larger the residual norm for the Arnoldi algorithm is than that for the GMRES algorithm. Unfortunately, these results give no indication of how well either method will perform on a particular

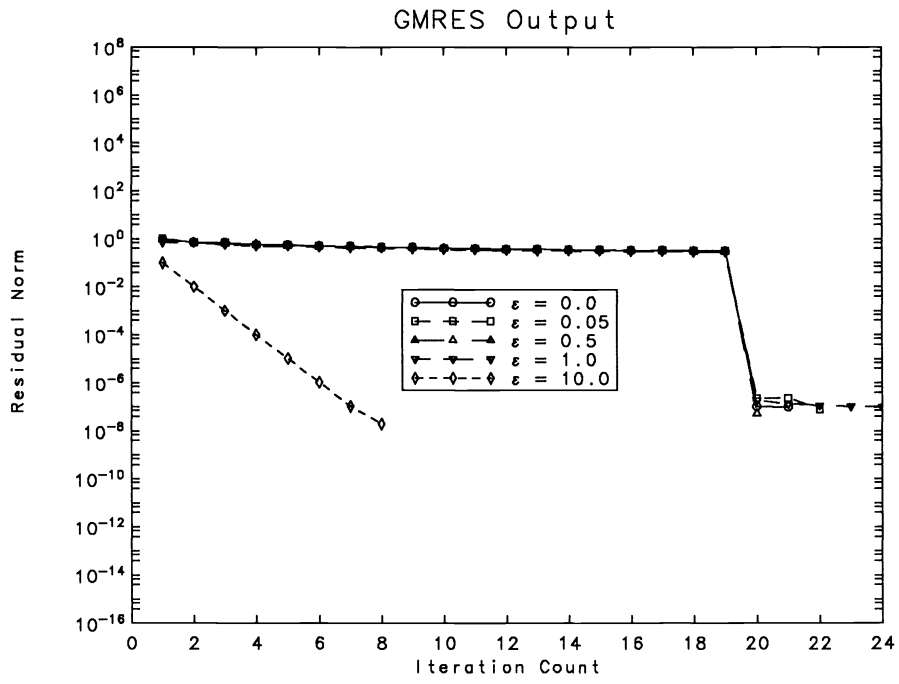


FIG. 6.4. *GMRES applied to* $(|A| + \epsilon I)x = |b|$.

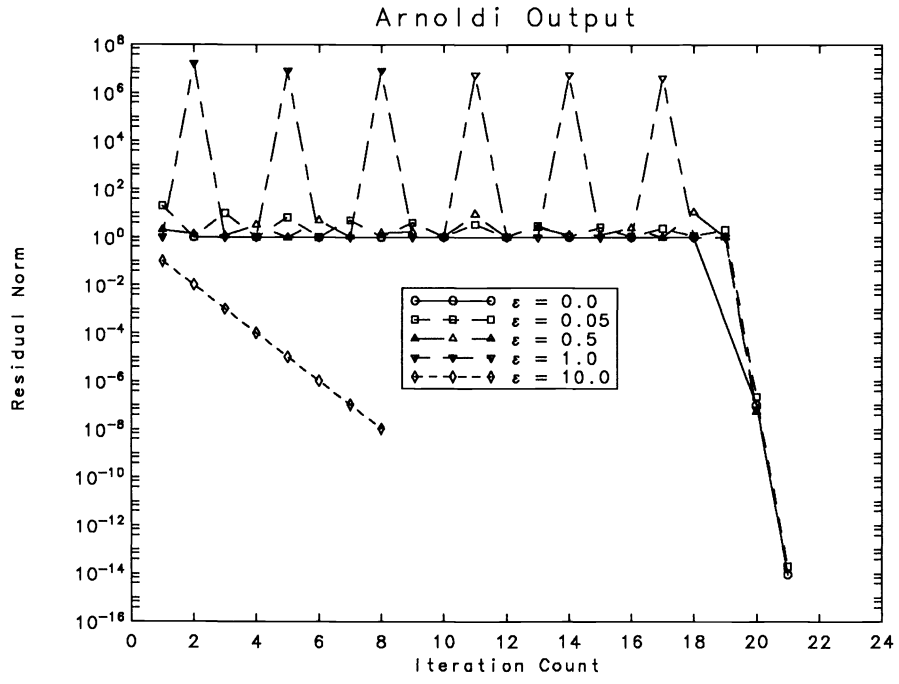


FIG. 6.5. *Arnoldi applied to* $(|A| + \epsilon I)x = |b|$.

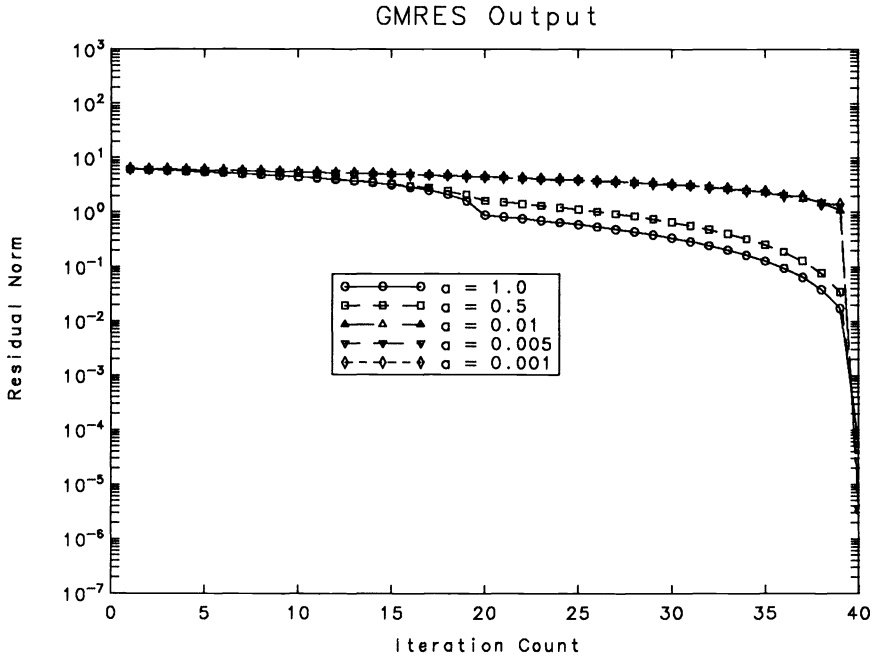


FIG. 6.6. *GMRES applied to* $au_{xx} + u_x = f$.

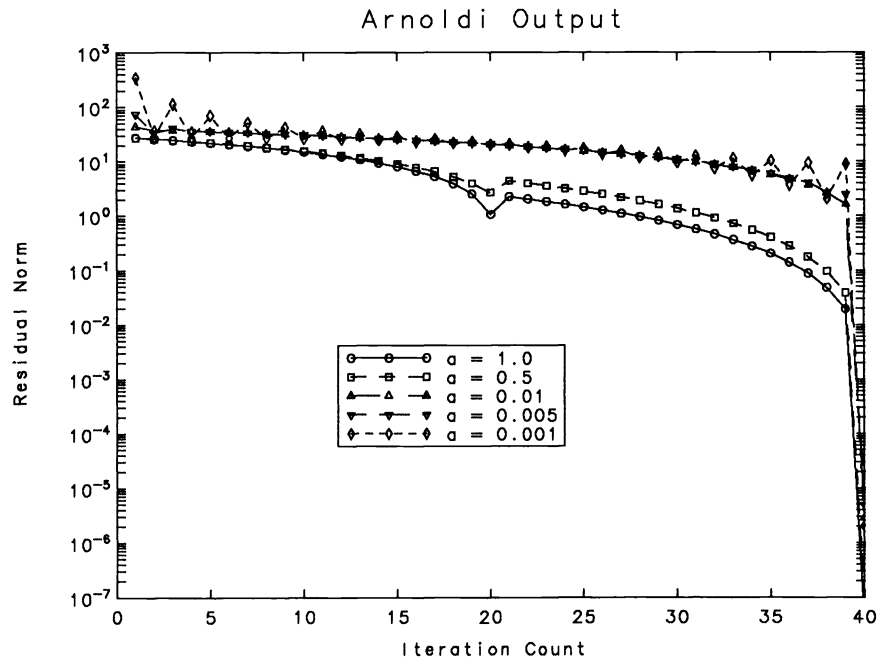


FIG. 6.7. *Arnoldi applied to* $au_{xx} + u_x = f$.

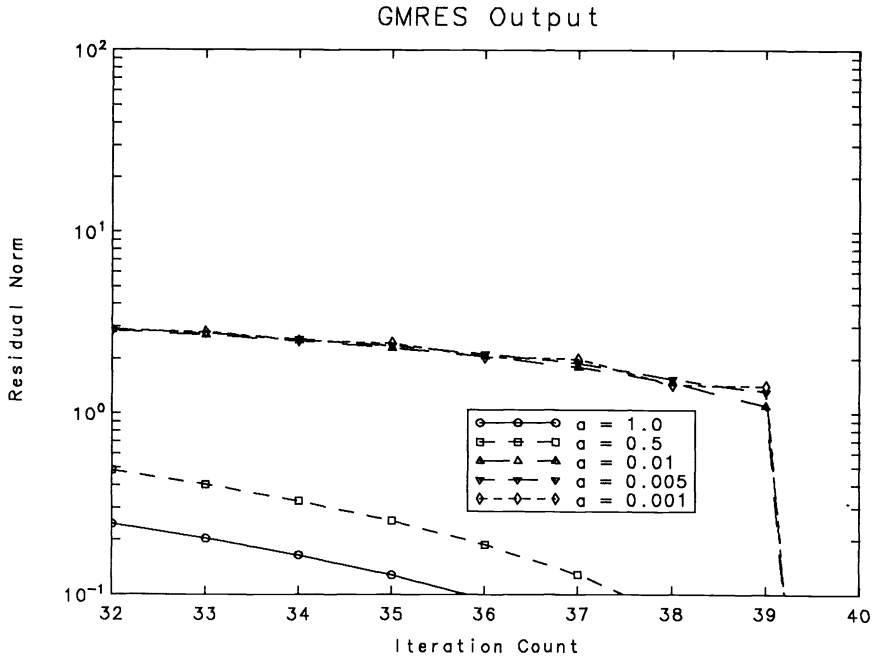


FIG. 6.8. *GMRES applied to* $au_{xx} + u_x = f$.

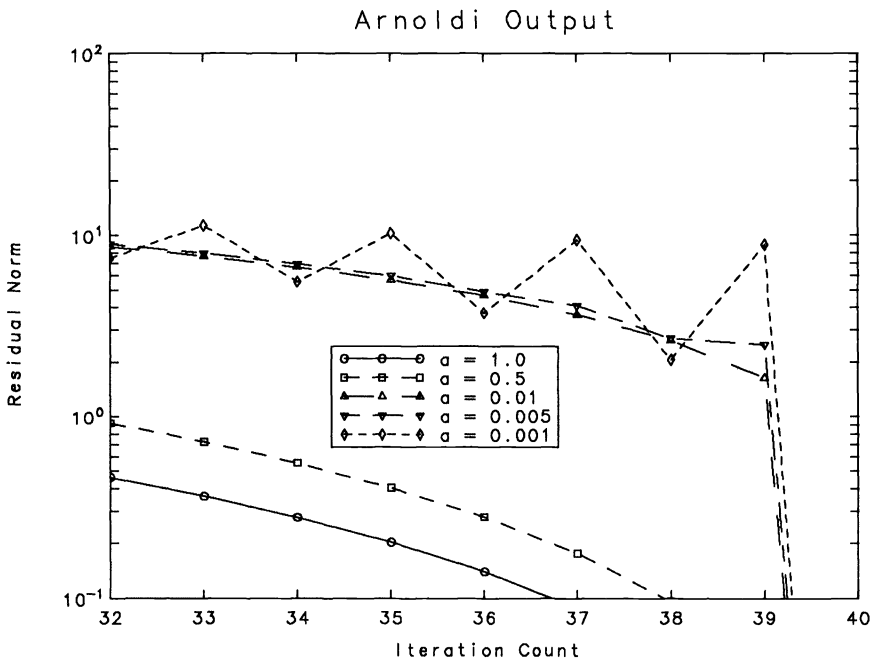


FIG. 6.9. *Arnoldi applied to* $au_{xx} + u_x = f$.

problem. They do, however, show that when Arnoldi or GMRES begins breaking down or stagnating, the latest Krylov basis vectors are poor descent directions for reducing the residual norm. In general, we prefer the GMRES algorithm due to its minimal norm property (and we would recommend GMRES over Arnoldi based upon this fact), but much more work is needed in order to better understand the performance characteristics of these methods. This will also be important in understanding what determines a good preconditioner for a particular problem.

Acknowledgment. The author would like to thank Alan Hindmarsh for his careful reading of a preliminary version of this manuscript.

REFERENCES

[1] P. N. BROWN, *A local convergence theory for combined inexact-Newton/finite-difference projection methods,* SIAM J. Numer. Anal., 24 (1987), pp. 407-434.

[2] P. N. BROWN AND A. C. HINDMARSH, *Matrix-free methods for stiff systems of* ODE*'s,* SIAM J. Numer. Anal., 23 (1986), pp. 610-638.

[3] _____, *Reduced storage methods in stiff* ODE *systems,* J. Appl. Math. Comput., 31 (1989), pp. 40-91.

[4] S. C. EISENSTAT, H. C. ELMAN, AND M. H. SCHULTZ, *Variational iterative methods for nonsymmetric systems of linear equations,* SIAM J. Numer. Anal., 20 (1983), pp. 345-357.

[5] H. C. ELMAN, *Iterative methods for large sparse nonsymmetric systems of linear equations,* Ph.D. thesis, Department of Computer Science, Yale University, New Haven, CT, 1982.

[6] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations,* The Johns Hopkins University Press, Baltimore, MD, 1983.

[7] A. L. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods,* Academic Press, New York, 1981.

[8] C. C. PAIGE AND M. A. SAUNDERS, *Solution of sparse indefinite systems of linear equations,* SIAM J. Numer. Anal., 12 (1975), pp. 617-629.

[9] Y. SAAD, *Krylov subspace methods for solving unsymmetric linear systems,* Math. Comp., 37(1981), pp. 105-126.

[10] _____, *Practical use of some Krylov subspace methods for solving indefinite and nonsymmetric linear systems,* SIAM J. Sci. Statist. Comput., 5 (1984), pp. 203-228.

[11] Y. SAAD AND M. H. SCHULTZ, *Conjugate gradient-like algorithms for solving nonsymmetric linear systems,* Math. Comp., 44/170(1985), pp. 417-424.

[12] _____, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems,* SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856-869.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 79–94, January 1991
© 1991 Society for Industrial and Applied Mathematics
004

THE FAST GAUSS TRANSFORM*

LESLIE GREENGARD† AND JOHN STRAIN‡

Abstract. Many problems in applied mathematics require the evaluation of the sum of N Gaussians at M points in space. The work required for direct evaluation grows like NM as N and M increase; this makes it very expensive to carry out such calculations on a large scale. In this paper, an algorithm is presented which evaluates the sum of N Gaussians at M arbitrarily distributed points in $C \cdot (N + M)$ work, where C depends only on the precision required. When $N = M = 100,000$, the algorithm presented here is several thousand times faster than direct evaluation. It is based on a divide-and-conquer strategy, combined with the manipulation of Hermite expansions and Taylor series.

Key words. Gauss transform, Hermite polynomials, fast algorithms

AMS(MOS) subject classifications. 65R10, 33A65, 35K05, 62G20, 65P05

1. Introduction.

Many problems in applied mathematics involve the *Gauss transform*

$$G_\delta f(x) = \int_\Gamma e^{-|x-y|^2/\delta} f(y)\, dy \qquad (\delta > 0) \tag{1}$$

of a function f defined on $\Gamma \subset \mathbf{R}^d$. The simplest example is the heat equation. The solution of the pure initial value problem

$$\begin{aligned} u_t(x,t) &= \Delta u(x,t) && \text{for } t > 0 \\ u(x,0) &= f(x) && \text{for } x \in \mathbf{R}^d \end{aligned}$$

$$\begin{aligned} u_t(x,t) &= \Delta u(x,t) && \text{for } t > 0 \\ u(x,0) &= f(x) && \text{for } x \in \mathbf{R}^d \end{aligned}$$

is given by

$$u(x,t) = (4\pi t)^{-d/2} G_{4t} f(x),$$

with Γ equal to the whole space. A similar transform, with Γ a lower-dimensional subset of $\mathbf{R}^d$, occurs when one solves any initial/boundary value problem for the heat equation by means of potential theory [1], [6], [9], [13], [14]. Other examples occur in vortex methods [3], tomography [11], and nonparametric statistics [7], [15]. Finally, a common analytical tool is *mollification*; one approximates an arbitrary function f by the family of smooth, rapidly decreasing functions

$$f_\delta(x) = (\pi\delta)^{-d/2} G_\delta f(x),$$

* Received by the editors July 12, 1989; accepted for publication November 22, 1989.

† Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. The work of this author was supported by the Office of Naval Research grant N00014-89-J-1527, by IBM grant P00038437, and by a National Science Foundation Mathematical Sciences Postdoctoral Fellowship.

‡ Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. Present address, Department of Mathematics, Princeton University, Princeton, New Jersey 08540. The work of this author was supported by the Defense Advanced Reasearch Project Agency/Air Force Office of Scientific Research under contract F-49620-87-C-0065.

which converge to f as $\delta \to 0$.

For numerical purposes, one must discretize $G_\delta f$. Given the values of f at a set of points $s_j \in \mathbf{R}^d$, one can approximate the integral (1) by means of a quadrature formula. A reasonable approximation to $G_\delta f$ might then take the form of a *discrete Gauss transform*

$$G(x) = \sum_{j=1}^{N} q_j \, e^{-|x-s_j|^2/\delta} \,, \tag{2}$$

where the coefficients q_j depend on the values $f(s_j)$ and the weights of the chosen quadrature formula. This paper will focus on the problem of evaluating the sum of Gaussians (2) as efficiently as possible. It will often be convenient to speak of (2) as the Gaussian "field" due to sources of strengths q_j at the points s_j, evaluated at the "target" x.

Suppose now that we want to evaluate (2) directly from the definition at M targets $x = t_i$. In other words, we want to apply the rectangular matrix with entries

$$G_{ij} = e^{-|t_i - s_j|^2/\delta}$$

to the vector $q = (q_1, \cdots, q_N)^T$. This requires $O(NM)$ work, which grows rapidly as M and N increase, and makes large scale calculations prohibitively expensive.

In this paper, we present an algorithm for evaluating sums of the form (2) at M points in $O(N+M)$ work. The constant in $O(N+M)$ depends only on the dimension d and the desired precision. The amount of memory required is also proportional to $N + M$, so that the algorithm is asymptotically optimal in terms of both work and storage. Furthermore, the sources and the targets can be placed anywhere; they need not be restricted to a regular grid. Even if the function f were given at N equispaced points and $G_\delta f$ evaluated at N equispaced points, fast convolution by means of the Fast Fourier Transform (FFT) would require $O(N \log N)$ operations, whereas our algorithm requires only $O(N)$.

2. Hermite expansions. This section describes the properties of the Gaussian kernel and Hermite expansions which we will need. Good references for this material are [4], [5], [8], and particularly Hille's paper [10].

The Hermite polynomials $H_n(t)$ may be defined by the Rodrigues formula

$$H_n(t) = (-1)^n e^{t^2} D^n e^{-t^2} \,, \qquad t \in \mathbf{R} \,,$$

where $D = d/dt$. We will make use of this definition as well as the generating function for Hermite polynomials

$$e^{2ts - s^2} = \sum_{n=0}^{\infty} \frac{s^n}{n!} H_n(t) \,.$$

Multiplication of each side of the preceding expression by e^{-t^2} yields

$$e^{-(t-s)^2} = \sum_{n=0}^{\infty} \frac{s^n}{n!} h_n(t),$$

where the Hermite functions $h_n(t)$ are defined by

$$h_n(t) = e^{-t^2} H_n(t). \tag{3}$$

(Note that these are not the usual orthonormal Hermite functions; the definition here is the right one for this situation.) In practice, we will use a shifted and scaled version of this formula: for $s_0 \in \mathbf{R}$ and $\delta > 0$, we have

$$\begin{aligned} e^{-(t-s)^2/\delta} &= e^{-(t-s_0-(s-s_0))^2/\delta} \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} \left(\frac{s-s_0}{\sqrt{\delta}} \right)^n h_n \left(\frac{t-s_0}{\sqrt{\delta}} \right) \\ &= e^{-(t-s_0)^2/\delta} \sum_{n=0}^{\infty} \frac{1}{n!} \left(\frac{s-s_0}{\sqrt{\delta}} \right)^n H_n \left(\frac{t-s_0}{\sqrt{\delta}} \right). \end{aligned}$$

This formula tells us how to evaluate the Gaussian field $e^{-(t-s)^2/\delta}$ at the target t due to the source at s, as a Hermite expansion centered at s_0. Thus we are shifting a Gaussian at s to a sum of Hermite polynomials times a Gaussian, all centered at s_0.

We can also interchange t and s to write

$$e^{-(t-s)^2/\delta} = \sum_{n=0}^{\infty} \frac{1}{n!} h_n \left(\frac{s-t_0}{\sqrt{\delta}} \right) \left(\frac{t-t_0}{\sqrt{\delta}} \right)^n. \tag{4}$$

Looked at this way, the expansion expresses a Gaussian with target t as a Taylor series about a nearby target t_0; the coefficients of the Taylor series are the Hermite functions evaluated at t_0. Thus the same expansion serves as both a near-field (Taylor) and a far-field (Hermite) expansion. The final one-dimensional results we will need are the recurrence relation

$$h_{n+1}(t) = 2t\, h_n(t) - 2n\, h_{n-1}(t), \qquad t \in \mathbf{R},$$

for Hermite functions and Cramer's inequality for Hermite polynomials:

$$|H_n(t)| \le K 2^{n/2} \sqrt{n!} e^{t^2/2},$$

where K is a numerical constant less than 1.09 in value. Cramer's inequality immediately implies the following useful bound for Hermite functions:

$$\frac{1}{n!} |h_n(t)| \le K 2^{n/2} \frac{1}{\sqrt{n!}} e^{-t^2/2}.$$

We will also need the straightforward extensions of these facts to higher dimensions. Thus, let t and s lie in d-dimensional Euclidean space $\mathbf{R}^d$, and consider the Gaussian

$$e^{-|t-s|^2} = e^{-(t_1-s_1)^2-\ldots-(t_d-s_d)^2}.$$

We will find it convenient to make use of multi-index notation. A multi-index $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_d)$ is a d-tuple of nonnegative integers, playing the role of a multidimensional index. For any multi-index α and any $t \in \mathbf{R}^d$, we define

$$\begin{aligned} |\alpha| &= \alpha_1 + \alpha_2 + \cdots + \alpha_d, \\ \alpha! &= \alpha_1! \alpha_2! \cdots \alpha_d!, \\ t^\alpha &= t_1^{\alpha_1} t_2^{\alpha_2} \cdots t_d^{\alpha_d}, \\ D^\alpha &= \partial_1^{\alpha_1} \partial_2^{\alpha_2} \cdots \partial_d^{\alpha_d}, \end{aligned}$$

where ∂_i is differentiation with respect to the ith coordinate in $\mathbf{R}^d$. If p is an integer, we say $\alpha \geq p$ if $\alpha_i \geq p$ for $1 \leq i \leq d$.

The multidimensional Hermite polynomials and Hermite functions are then defined by

$$H_\alpha(t) = H_{\alpha_1}(t_1) \cdots H_{\alpha_d}(t_d),$$

$$h_\alpha(t) = e^{-|t|^2} H_\alpha(t) = h_{\alpha_1}(t_1) \cdots h_{\alpha_d}(t_d), \tag{5}$$

where $|t|^2 = t_1^2 + \cdots + t_d^2$.

The Hermite expansion of a Gaussian in $\mathbf{R}^d$ is then simply

$$e^{-|t-s|^2} = \sum_{\alpha \geq 0} \frac{(t-s_0)^\alpha}{\alpha!} h_\alpha(s-s_0). \tag{6}$$

Cramer's inequality generalizes to

$$\frac{1}{\alpha!}|h_\alpha(t)| \leq K e^{-|t|^2/2} 2^{|\alpha|/2} \frac{1}{\sqrt{\alpha!}}, \tag{7}$$

where K is less than $(1.09)^d$.

Finally, our algorithm will require the Taylor expansion of the Hermite function $h_\alpha(t)$ about an arbitrary point $t_0 \in \mathbf{R}^d$. Since h_α is defined by

$$\begin{aligned} h_\alpha(t) &= e^{-t^2} H_\alpha(t) \\ &= (-1)^{|\alpha|} D^\alpha e^{-t^2}, \end{aligned} \tag{8}$$

applying D^β gives immediately

$$D^\beta h_\alpha(t) = (-1)^{|\beta|} h_{\alpha+\beta}(t). \tag{9}$$

Thus the Taylor series of h_α is

$$h_\alpha(t) = \sum_{\beta \geq 0} \frac{(t-t_0)^\beta}{\beta!} (-1)^{|\beta|} h_{\alpha+\beta}(t_0). \tag{10}$$

We now present the three lemmas on which our algorithm relies. The first describes how to transform the field due to all sources in a box into a single rapidly converging Hermite expansion about the center of the box.

LEMMA 2.1. *Let N_B sources s_j lie in a box B with center s_B and side length $r\sqrt{2\delta}$, with $r < 1$. Then the Gaussian field due to the sources in B,*

$$G(t) = \sum_{j=1}^{N_B} q_j e^{-|t-s_j|^2/\delta}, \tag{11}$$

is equal to a single Hermite expansion about s_B:

$$G(t) = \sum_{\alpha \geq 0} A_\alpha h_\alpha\left(\frac{t-s_B}{\sqrt{\delta}}\right).$$

The coefficients A_α are given by

$$A_\alpha = \frac{1}{\alpha!} \sum_{j=1}^{N_B} q_j \left(\frac{s_j - s_B}{\sqrt{\delta}} \right)^\alpha . \tag{12}$$

The error $E_H(p)$ due to truncating the series after p^d terms satisfies the bound:

$$|E_H(p)| = \left| \sum_{\alpha \geq p} A_\alpha h_\alpha \left(\frac{t - s_B}{\sqrt{\delta}} \right) \right| \leq K Q_B \left(\frac{1}{p!} \right)^{d/2} \left(\frac{r^{p+1}}{1-r} \right)^d ,$$

where $Q_B = \sum |q_j|$ and $K = (1.09)^d$.

Proof. Use (6) to expand each Gaussian in the sum (11) into a Hermite series about s_B and interchange the sums over α and j. The truncation error bound follows from Cramer's inequality (7) and the formula for the tail of a geometric series. □

The second lemma shows how to convert a Hermite expansion about s_B into a Taylor expansion about t_C. The Taylor series converges rapidly in a box of side $r\sqrt{2\delta}$ about t_C, with $r < 1$.

LEMMA 2.2. *The Hermite expansion*

$$G(t) = \sum_{\alpha \geq 0} A_\alpha \, h_\alpha \left(\frac{t - s_B}{\sqrt{\delta}} \right) \tag{13}$$

has the following Taylor expansion, about an arbitrary point t_C:

$$G(t) = \sum_{\beta \geq 0} B_\beta \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta . \tag{14}$$

The coefficients B_β are given by

$$B_\beta = \frac{(-1)^{|\beta|}}{\beta!} \sum_{\alpha \geq 0} A_\alpha \, h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) . \tag{15}$$

If the A_α are given by (12)*, then the error $E_T(p)$ in truncating the Taylor series after p^d terms is bounded, in the box C with center t_C and side length $r\sqrt{2\delta}$, by*

$$|E_T(p)| = \left| \sum_{\beta \geq p} B_\beta \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \right| \leq K Q_B \left(\frac{1}{p!} \right)^{d/2} \left(\frac{r^{p+1}}{1-r} \right)^d ,$$

if $r < 1$.

Proof. Each Hermite function in (13) can be expanded into a Taylor series by means of (10). The expansion (14) is then obtained by interchanging the order of summation. The truncation error bound is only a little more difficult: By the formula (12) for A_α, we have

$$\begin{aligned} B_\beta &= \frac{(-1)^{|\beta|}}{\beta!} \sum_{\alpha \geq 0} A_\alpha h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) \\ &= \frac{(-1)^{|\beta|}}{\beta!} \sum_{\alpha \geq 0} \frac{1}{\alpha!} \sum_{j=1}^{N_B} q_j \left(\frac{s_j - s_B}{\sqrt{\delta}} \right)^\alpha h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) \\ &= \frac{(-1)^{|\beta|}}{\beta!} \sum_{j=1}^{N_B} q_j \sum_{\alpha \geq 0} \frac{1}{\alpha!} \left(\frac{s_j - s_B}{\sqrt{\delta}} \right)^\alpha h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) . \end{aligned}$$

But by (10), the inner sum is the Taylor expansion of $h_\beta((s_j - t_C)/\sqrt{\delta})$. Thus

$$B_\beta = \frac{(-1)^{|\beta|}}{\beta!} \sum_{j=1}^{N_B} q_j \, h_\beta \left(\frac{s_j - t_C}{\sqrt{\delta}} \right) , \tag{16}$$

and Cramer's inequality implies

$$|B_\beta| \leq \frac{1}{\beta!} K Q_B 2^{|\beta|/2} \sqrt{\beta!} \leq K Q_B \frac{2^{|\beta|/2}}{\sqrt{\beta!}} . \tag{17}$$

The truncation error bound follows, as in Lemma 2.1, from summing the tail of a geometric series. □

For our algorithm, we will need a variant of Lemma 2.2 in which the Hermite series is *truncated* before converting it to a Taylor series. This essentially means that in addition to truncating the Taylor series itself, we are also truncating the infinite sum expression (15) for the coefficients. Fortunately, however, the error due to this approximation of the coefficients turns out to be much smaller than the truncation error of the Taylor series.

LEMMA 2.3. *A truncated Hermite expansion*

$$G(t) = \sum_{\alpha \leq p} A_\alpha h_\alpha \left(\frac{t - s_B}{\sqrt{\delta}} \right)$$

has the following Taylor expansion about an arbitrary point t_C*:*

$$G(t) = \sum_{\beta \geq 0} C_\beta \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta .$$

The coefficients C_β *are given by*

$$C_\beta = \frac{(-1)^{|\beta|}}{\beta!} \sum_{\alpha \leq p} A_\alpha \, h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) . \tag{18}$$

If the A_α *are given by* (12)*, then the error* $E_T(p)$ *in truncating the Taylor series after* p^d *terms is bounded, in the box* C *with center* t_C *and side length* $r\sqrt{2\delta}$*, by*

$$|E_T(p)| = \left| \sum_{\beta \geq p} C_\beta \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \right| \leq K' Q_B \left(\frac{1}{p!} \right)^{d/2} \left(\frac{r^{p+1}}{1 - r} \right)^d ,$$

where $K' \leq K \left(1 + (p!)^{-d/2}\right) \leq 2K$ *if* $r \leq \frac{1}{2}$.

Proof. The proof is an application of the triangle inequality. Write C_β as the coefficient B_β from Lemma 2.2 plus the tail of a series

$$\begin{aligned}
C_\beta &= \frac{(-1)^{|\beta|}}{\beta!} \sum_{j=1}^{N_B} q_j \sum_{\alpha \leq p} \frac{1}{\alpha!} \left(\frac{s_j - s_B}{\sqrt{\delta}} \right)^\alpha h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) \\
&= \frac{(-1)^{|\beta|}}{\beta!} \sum_{j=1}^{N_B} q_j \left(\sum_{\alpha \geq 0} - \sum_{\alpha > p} \right) \frac{1}{\alpha!} \left(\frac{s_j - s_B}{\sqrt{\delta}} \right)^\alpha h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) \\
&= B_\beta - \frac{(-1)^{|\beta|}}{\beta!} \sum_{j=1}^{N_B} q_j \sum_{\alpha > p} \frac{1}{\alpha!} \left(\frac{s_j - s_B}{\sqrt{\delta}} \right)^\alpha h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) \\
&= B_\beta + (C_\beta - B_\beta) .
\end{aligned}$$

Then

$$|E_T(p)| \leq \left| \sum_{\beta \geq p} B_\beta \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \right| + \left| \sum_{\beta \geq p} (C_\beta - B_\beta) \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \right| . \tag{19}$$

From Lemma 2.2, we know that the first sum is bounded by

$$K Q_B \left(\frac{1}{p!} \right)^{d/2} \left(\frac{r^{p+1}}{1-r} \right)^d .$$

Hence we need only bound the second sum. For this, we have

$$\begin{aligned} &\left| \sum_{\beta \geq p} (C_\beta - B_\beta) \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \right| \\ &\quad \leq Q_B \sum_{\beta > p} \left| \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \right| \frac{1}{\beta!} \sum_{\alpha > p} \frac{1}{\alpha!} \left| \left(\frac{s_j - s_B}{\sqrt{\delta}} \right)^\alpha \right| \left| h_{\alpha+\beta} \left(\frac{s_B - t_C}{\sqrt{\delta}} \right) \right| \\ &\quad \leq K Q_B \sum_{\alpha > p} \sum_{\beta > p} \frac{r^{|\alpha|}}{\sqrt{\alpha!}} \sqrt{\frac{(\alpha+\beta)!}{\alpha!\beta!}} \frac{r^{|\beta|}}{\sqrt{\beta!}} . \end{aligned}$$

But

$$\frac{(\alpha+\beta)!}{\alpha!\beta!} \leq 2^{|\alpha+\beta|} ,$$

so the lemma follows immediately. □

Remark 2.1. The alternate expression (16) for B_β which appears in the proof of Lemma 2.2 has a simple meaning. Rather than using Lemma 2.1 to accumulate all the Gaussians into a single Hermite expansion and then shifting it to t_C, we can use Lemma 2.3 to shift each Gaussian individually to t_C and add up the resulting Taylor coefficients. (A Gaussian is a one-term Hermite series, after all, and can therefore be shifted just like any other truncated Hermite series.) Thus, a Gaussian

$$G(t) = q e^{-|t - s_j|^2/\delta}$$

has the following Taylor expansion about t_C:

$$G(t) = \sum_{\beta \geq 0} B_\beta \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta .$$

The coefficients B_β are given by

$$B_\beta = q \frac{(-1)^{|\beta|}}{\beta!} h_\beta \left(\frac{s_j - t_C}{\sqrt{\delta}} \right) \tag{20}$$

and the error in truncation after p^d terms is

$$|E_T(p)| = \left| \sum_{\beta \geq p} B_\beta \left(\frac{t - t_C}{\sqrt{\delta}} \right)^\beta \right| \leq K q \left(\frac{1}{p!} \right)^{d/2} \left(\frac{r^{p+1}}{1-r} \right)^d$$

for $r < 1$.

3. The fast Gauss transform. We now have the tools necessary to construct and analyze a fast algorithm for evaluating the discrete Gauss transform

$$G(t_i) = \sum_{j=1}^{N} q_j e^{-|t_i - s_j|^2/\delta} \tag{21}$$

for $1 \leq i \leq M$ in $O(M+N)$ work. By shifting the origin and rescaling δ if necessary, we can assume (as a convenient normalization) that the sources s_j and targets t_i all lie in the unit box $B_0 = [0,1]^d$.

The algorithm is based on subdividing B_0 into smaller boxes with sides of length $r\sqrt{2\delta}$ parallel to the axes, with a fixed $r \leq \frac{1}{2}$. We can then assign each source s_j to the box B in which it lies and each target t_i to the box C where it lies. For the sake of clarity, we maintain a notational distinction between source boxes B and target boxes C even though they may be the same.

For each target box C, we need to evaluate the total field due to sources in all boxes B, at each target in C. Because the range of the Gaussian $e^{-|t-s|^2/\delta}$ is $O(\sqrt{\delta})$ and the boxes have side lengths $r\sqrt{2\delta}$, only a fixed number of source boxes B can contribute more than $Q\epsilon$ to the field in a given target box C, where $Q = \sum_{j=1}^{N} |q_j|$ and ϵ is a specified precision. Indeed, if we cut off the sum over all B after including the $(2n+1)^d$ nearest boxes to C, we incur an error bounded by $Qe^{-2r^2n^2}$. We can always choose n depending only on r and ϵ to make this less than $Q\epsilon$. For example, if $r = \frac{1}{2}$ we get single precision accuracy relative to Q with $n = 6$ and double precision with $n = 8$.

Suppose now that we want to evaluate the field due to a box$B$ with N_B sources at M_C targets in a box C. There are two ways in which B can transmit its influence, and two ways in which C can handle the information it receives. B can directly send out the strengths and centers of all N_B Gaussians located in B, or it can use Lemma 2.1 to collect them into a single truncated Hermite expansion. C can then directly evaluate all fields (Gaussians or Hermite expansions) sent to it, at the M_C target locations in C, or it can use Lemma 2.3 to convert the fields sent to it into a single truncated Taylor expansion about the center t_C of C. Evaluation of this Taylor series then yields the total field at each target location.

Thus, there are four possible ways in which B can influence C:

1. N_B Gaussians $\rightarrow$ directly evaluated.
2. N_B Gaussians $\rightarrow$ accumulated in Taylor series via (20).
3. Hermite series $\rightarrow$ directly evaluated.
4. Hermite series $\rightarrow$ accumulated in Taylor series via (18).

A fast algorithm can be based on any one of the second through fourth alternatives, because they all decouple the number of sources from the number of targets. Methods 1–4 require the following work to evaluate $G(t)$ at M targets t_i.

1. The cost of evaluating N Gaussians at M points is of the order $O(NM)$.

2. Consider a fixed source box B. For each target box C within range, we must compute p^d Taylor series coefficients

$$C_\beta(B) = \frac{(-1)^{|\beta|}}{\beta!} \sum_{s_j \in B} h_\beta\left(\frac{s_j - t_C}{\sqrt{\delta}}\right) . \tag{22}$$

Each coefficient requires $O(N_B)$ work to evaluate, resulting in a net cost of the order $O(p^d N_B)$. Since there are at most $(2n+1)^d$ boxes within range, the total work for

forming all Taylor series is of the order $O((2n+1)^d p^d N)$. Now, for each target t_i, we must evaluate the p^d term Taylor series corresponding to the box C in which t_i lies. The total cost of this algorithm is, therefore,

$$O((2n+1)^d p^d N) + O(p^d M) .$$

3. In the third approach, we form a Hermite series for each box B and evaluate it at all targets. First, using Lemma 2.1, we write

$$\begin{aligned} G(t) &= \sum_B \sum_{s_j \in B} q_j e^{-|t-s_j|^2/\delta} \\ &= \sum_B \sum_{\alpha \geq 0} A_\alpha(B)\, h_\alpha\left(\frac{t-s_B}{\sqrt{\delta}}\right) + E_H(p) , \end{aligned}$$

where $|E_H(p)| \leq Q\epsilon$ and

$$A_\alpha(B) = \frac{1}{\alpha!} \sum_{s_j \in B} q_j \left(\frac{s_j - s_B}{\sqrt{\delta}}\right)^\alpha . \tag{23}$$

Computing each $A_\alpha(B)$ costs $O(N_B)$ work, so forming all the Hermite expansions requires $O(p^d N)$ work. Evaluating at most $(2n+1)^d$ expansions at each target t_i costs $O((2n+1)^d p^d)$ work per target, so this approach results in a total work

$$O(p^d N) + O((2n+1)^d p^d M) .$$

4. Finally, suppose we accumulate all sources into truncated Hermite expansions and transform all Hermite expansions into Taylor expansions via Lemma 2.3. Thus, we approximate $G(t)$ in C by

$$\begin{aligned} G(t) &= \sum_B \sum_{s_j \in B} q_j e^{-|t-s_j|^2/\delta} \\ &= \sum_{\beta \leq p} C_\beta \left(\frac{t-t_C}{\sqrt{\delta}}\right)^\beta + E_T(p) + E_H(p) , \end{aligned}$$

where $|E_H(p)| + |E_T(p)| \leq Q\epsilon$,

$$C_\beta = \frac{(-1)^{|\beta|}}{\beta!} \sum_B \sum_{\alpha \leq p} A_\alpha(B)\, h_{\alpha+\beta}\left(\frac{s_B - t_C}{\sqrt{\delta}}\right) , \tag{24}$$

and the coefficients $A_\alpha(B)$ are given by (23). As we saw under the third approach, it costs $O(p^d N)$ work to form all the Hermite expansions, i.e., to compute the coefficients $A_\alpha(B)$ for $\alpha \leq p$ and all source boxes B. Because of the product form (5) of $h_{\alpha+\beta}$, the computation of the p^d coefficients C_β involves only $O(d\,p^{d+1})$ operations for each box B in range. Therefore, a total of $O((2n+1)^d\, d\, p^{d+1})$ work per target box C is required. Finally, evaluating the appropriate Taylor series for each target t_i requires $O(p^d M)$ work. Hence this algorithm has net CPU requirements of the order

$$O((2n+1)^d\, d\, p^{d+1} N_{\text{box}}) + O(p^d N) + O(p^d M) ,$$

where the number of boxes N_{box} is bounded by $\min((r\sqrt{2\delta})^{-d/2}, M)$. Note that the factor $(2n+1)^d$ no longer multiplies either the $O(N)$ or $O(M)$ terms. The work is now

decoupled into three parts: $O(p^d N)$ to form Hermite expansions, $O(p^d M)$ to evaluate Taylor series, and a constant term depending on the number of box–box interactions and the cost of transforming a Hermite expansion into a Taylor series.

Thus, we really have four algorithms for evaluating $G(t)$, three of which are asymptotically optimal. We can try to minimize the constants in the work estimate by varying the choice of algorithm from box to box. Clearly an optimal strategy for this choice is global, but a reasonable strategy can be constructed in which each box decides independently what action to take. For this purpose, let N_B sources in a box B be within range of M_C targets in a box C. Choose cutoff parameters N_F and M_L. Then

1. If $N_B < N_F$ then B sends out N_B Gaussians.
2. If $N_B \geq N_F$ then B sends out a Hermite expansion.
3. If $M_C < M_L$ then C evaluates all fields sent to it immediately.
4. If $M_C \geq M_L$ then C transforms all fields sent to it into Taylor series, accumulates the coefficients, and only then evaluates the Taylor series.

The work in this algorithm can be broken down as follows:

1.
$$\sum_{N_B \geq N_F} O(p^d N_B)$$
to evaluate Hermite expansions;

2.
$$+ \sum_{M_C < M_L} \sum_{N_B < N_F} O(N_B M_C)$$
to evaluate Gaussians;

3.
$$+ \sum_{M_C < M_L} \sum_{N_B \geq N_F} O(p^d M_C)$$
to evaluate Hermite expansions;

4.
$$+ \sum_{M_C \geq M_L} \sum_{N_B < N_F} O(p^d N_B)$$
to transform Gaussians into Taylor series;

5.
$$+ \sum_{M_C \geq M_L} \sum_{N_B \geq N_F} O(d\, p^{d+1})$$
to transform Hermite series into Taylor series;

6.
$$+ \sum_{M_C \geq M_L} O(p^d M_C)$$
to evaluate Taylor series.

Clearly we can achieve a rough balance of work by taking $N_F = O(p^{d-1})$ and $M_L = O(p^{d-1})$. The total work then has the form

$$O(p^d N) + O(d\,p^{d+1}(2n+1)^d \min((r\sqrt{2\delta})^{-d/2}, M)) + O(p^d M).$$

This is linear in N and M, with a constant depending only on the precision. The complexity estimate is similar to the fourth algorithm above, but the advantage here is that when there are only a few particles in a box, the overhead associated with transformation of Hermite series to Taylor series is avoided.

4. Formal description of the algorithm. In this section, we describe the fast Gauss transform in a more procedural form.

ALGORITHM.

Comment [Choose the largest $r \leq \frac{1}{2}$ such that $1/r\sqrt{2\delta}$ is an integer N_{side}. Subdivide the unit box into N_{side}^d boxes. Choose the number n of boxes to go out in each direction based on r and the desired precision ϵ. Each source sends to $(2n+1)^d$ boxes. Choose the number of terms p^d based on r and ϵ. Choose the cutoffs N_F and M_L.]

Step 1.

Assign sources and targets to boxes. Determine number of boxes containing more than M_L targets. For each such box, allocate storage for a Taylor series with p^d terms and initialize to zero.

Step 2.

Comment [Loop through boxes, computing interactions between sources in box and targets within range n. For each pair of source and target boxes, one of the four options summarized in §3 is used.]

do $i = 1, \cdots, N_{\text{side}}^d$
  N_B = number of sources in ith box B.
  Form the *interaction list* of $(2n+1)^d$ target boxes C within range of B.
  if $N_B \leq N_F$ **then**
    do $j = 1, \cdots, (2n+1)^d$
      M_C = number of targets in jth box C in *interaction list.*
      if $M_C \leq M_L$ **then**
        Compute source/target interactions by direct evaluation of Gaussians.
      else
        Convert each of the N_B sources into a Taylor series about the center of box C via (20) and add to Taylor series for box C.
      end if
    end do
  else $(N_B > N_F)$
    Form Hermite expansion about center of box B due to N_B sources via equation (20).
    do $j = 1, \cdots, (2n+1)^d$
      M_C = number of targets in jth box C.
      if $M_C \leq M_L$ **then**
        Evaluate Hermite expansion at each target location and add to accumulated potential.
      else
        Convert Hermite expansion into a Taylor series about the center of box C by means of (12) and add to Taylor series for box C.
      end if
    end do
  end if
end do

Step 3.

Comment [Loop through boxes evaluating Taylor series for boxes with more than M_L targets.]

do $i = 1, \cdots, N_{\text{side}}^d$

M_C = number of targets in ith box C.

if $M_C > M_L$ **then**

Evaluate Taylor series for box C at each of the M_C target positions to obtain the desired potential.

end if

end do

5. Numerical experiments. In this section, we present the results of numerical experiments with the fast Gauss transform and demonstrate dramatic speedups over the direct calculation for realistic problems. A two-dimensional version of the algorithm was programmed in Fortran and run on a Sun-4 workstation, using up to 100,000 sources and targets and δ lying in the range .0001 to 1.0.

We examined the cost of the fast algorithm as compared to the direct evaluation of all the Gaussians, as N and M increased. Two distributions of targets and surces were tried, uniformly distributed in the unit box and equally spaced on a circle, and several values of δ were used. For the uniform distribution, strengths were uniformly distributed between -1 and 1. For the circle, the strengths were specified to be $\cos(\theta)$, where θ is the angle. We asked for an error relative to the total charge Q of $\epsilon = 10^{-6}$, which required $p^d = 8^2$ terms in the Hermite and Taylor expansions. The results are given in Tables 1–4.

TABLE 1

Table of cost and errors for $\delta = 1.0$, with targets and sources distributed uniformly in the unit box. CPU times are given in seconds for the fast and direct algorithms. Times for the direct algorithm were estimated by evaluating $G(t)$ at 100 targets and extrapolating. All information was rounded to two significant digits.

Case	$N = M$	Fast	Direct	Error/Q
1	100	0.42	0.59	.627E-08
2	200	0.62	2.3	.306E-08
3	400	1.1	9.7	.175E-08
4	800	1.8	38	.157E-08
5	1600	3.4	150	.126E-08
6	3200	6.5	601	.135E-08
7	6400	12.8	2407	.114E-08
8	12800	26.0	9702	.563E-09
9	25600	51.9	38790	.563E-09
10	51200	103	155550	.337E-09
11	102400	205	622780	.237E-09

With 102,400 sources and targets equispaced on a circle and $\delta = .01$, the fast algorithm is more than 3,000 times faster than the direct calculation. Typically the performance of the fast algorithm improves when the source distribution is spatially nonuniform, as it is in many practical problems. There are then more particles in each of a smaller number of occupied boxes, reducing overhead costs.

TABLE 2

Table of cost and errors for $\delta = 0.01$, with targets and sources distributed uniformly in the unit box.

Case	$N = M$	Fast	Direct	Error/Q
1	100	0.65	0.70	.115E-08
2	200	1.840	2.76	.616E-09
3	400	5.8	10.9	.478E-09
4	800	20.6	43.6	.291E-08
5	1600	115	174	.274E-08
6	3200	349	697	.443E-08
7	6400	344	2792	.249E-08
8	12800	353	11173	.177E-08
9	25600	383	44650	.144E-08
10	51200	431	179120	.120E-08
11	102400	538	716760	.501E-09

TABLE 3

Table of cost and errors for $\delta = 0.01$, with targets and sources spaced uniformly on a circle.

Case	$N = M$	Fast	Direct	Error/Q
1	100	3.25	0.67	.479E-09
2	200	4.93	2.70	.489E-09
3	400	10.9	10.8	.182E-06
4	800	12.7	42.8	.191E-06
5	1600	14.5	172	.203E-06
6	3200	18.5	684	.204E-06
7	6400	26.4	2749	.201E-06
8	12800	38.7	11003	.177E-06
9	25600	65.1	43955	.172E-06
10	51200	116	176300	.170E-06
11	102400	219	705650	.170E-06

TABLE 4

Table of cost and errors for $\delta = 0.0001$, with targets and sources spaced uniformly on a circle.

Case	$N = M$	Fast	Direct	Error/Q
1	100	4.61	0.68	.539E-10
2	200	5.31	2.72	.270E-10
3	400	6.24	10.9	.140E-10
4	800	8.71	42.9	.834E-11
5	1600	19.6	172	.298E-10
6	3200	63.9	690	.799E-08
7	6400	79.0	2785	.469E-08
8	12800	96.9	11132	.217E-09
9	25600	127	44375	.757E-10
10	51200	179	178120	.417E-10
11	102400	287	713830	.385E-10

Storage requirements for the fast algorithm are reasonable as long as N and M are large and δ is not too small. For extremely small δ, one should modify the algorithm to make use of just those boxes containing targets or sources. The interaction list for each source box can then be formed by means of an adaptive tree structure, as is done in the fast multipole method [2]. This would avoid having to loop through $N_{\text{side}}^d = (2r^2\delta)^{-d/2}$ (largely empty) boxes.

6. A generalization. The fast Gauss transform generalizes easily to sums of the form

$$(-1)^{|\gamma|} D^\gamma G(t) = \sum_{j=1}^{N} q_j h_\gamma(t - s_j), \tag{25}$$

convolutions with a fixed Hermite function. One need only apply D^γ to all the formulas presented above and use (9), relating derivatives of Hermite functions. An arbitrary multivariable polynomial $P(s)$ can be expressed as a sum of Hermite polynomials, so we can use our algorithm to evaluate sums of the form

$$\sum_{j=1}^{N} q_j P\left(\frac{t - s_j}{\sqrt{\delta}}\right) e^{-|t-s_j|^2/\delta}$$

in optimal time. As an extension of this remark, we can evaluate any convolution sum

$$K * f\,(t) = \sum_{j=1}^{N} f(s_j) K(t - s_j) \tag{26}$$

for which the kernel K has a rapidly converging Hermite series. We approximate K to within ϵ by a q^d-term truncated Hermite expansion and apply the fast algorithm of this paper to carry out each convolution with a Hermite function. This would cost $O((pq)^d(N + M))$ to evaluate (26) at M points. A better approach, however, would be to modify the algorithm so as to create, for each box, a single $(p + q)^d$-term far-field expansion which includes the effect of Hermite functions of indices up to q. The modified algorithm would evaluate (26) at M points in $O((p+q)^d(N+M))$ work; the constant $p + q$ depends only on the precision required and the smoothness of K.

Examples of convolution kernels K with rapidly converging Hermite series include any smooth function which decays at infinity faster than any power; in particular, any smooth function with compact support.

One application of this generalization is to the problem of evaluating the *continuous* Gauss transform (1), rather than the discrete sum of Gaussians. Evaluation of the continuous Gauss transform with an order of accuracy independent of δ, as would be required to evaluate the mollification of a nonsmooth function, seems to require the use of product integration. In other words, one replaces the density f with a piecewise polynomial and evaluates the resulting integrals exactly. This gives a weighted sum of values f_j which cannot be evaluated by the Gauss transform, because the integral of a Gaussian over an element is no longer a Gaussian. However, the result can be expanded in a rapidly converging Hermite series of the form (26), and this sum can be evaluated by the generalized Gauss transform just described.

However, if one really wants to evaluate (1) accurately, product integration followed by Hermite expansion is unnecessarily troublesome. A more straightforward

approach is to use the expansion on which our algorithm is based to create a "semi-continuous Gauss transform." Rather than discretizing the integral and forming the discrete moments due to the sources in each box, we simply form the continuous moments due to the sources in each box. This gives a far more accurate Hermite expansion which can then be manipulated just as in the standard Gauss transform algorithm.

Consider, for example, the problem of evaluating

$$G\mu(t) = \int_{\Gamma} e^{-|t-s|^2} \mu(s) ds \,,$$

where Γ is a hypersurface in $\mathbf{R}^d$. The semicontinuous Gauss transform can be described by the equation

$$G\mu(t) = \sum_B \int_{\Gamma \cap B} \sum_\alpha \frac{1}{\alpha!} \left(\frac{s - s_B}{\sqrt{\delta}} \right)^\alpha h_\alpha \left(\frac{t - s_B}{\sqrt{\delta}} \right) \mu(s)\, ds \tag{27}$$

$$= \sum_B \sum_\alpha M_\alpha\, h_\alpha \left(\frac{t - s_B}{\sqrt{\delta}} \right). \tag{28}$$

Here the moments M_α can very easily be evaluated to high accuracy. We then have the Hermite expansion of $G\mu(t)$, and can manipulate it just as any other Hermite expansion. The utility of this algorithm is obvious; we are currently applying it to a variety of problems in applied mathematics.

7. Conclusions. We have presented a "fast Gauss transform" algorithm for evaluating the sums

$$G(t_i) = \sum_{j=1}^{N} q_j e^{-|t_i - s_j|^2/\delta} \,, \qquad i = 1, \cdots, M \tag{29}$$

for $1 \leq i \leq M$ in $O(M+N)$ work. Direct evaluation would require $O(NM)$ work in general, so this is a substantial improvement in computational complexity. In order to evaluate the sum of 100,000 Gaussians at 100,000 points, for example, the fast algorithm requires about four minutes of CPU time on a Sun-4, while direct evaluation would take more than eight days. There are many fields of applied mathematics where such an algorithm will be a useful tool.

REFERENCES

[1] R. BROWN, *Layer potentials and boundary value problems for the heat equation on Lipschitz cylinders*, Ph.D. thesis, University of Minnesota, Minneapolis, MN, 1987.

[2] J. CARRIER, L. GREENGARD, AND V. ROKHLIN, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 669-686.

[3] G. H. COTTET, S. MAS-GALLIC, AND P.A. RAVIART, *Vortex methods for the incompressible Euler and Navier–Stokes equations*, in Proc. Workshop on Computational Fluid Dynamics and Reacting Gas Flows, Institute for Mathematics and Its Applications, Minneapolis, MN, September 1986.

[4] H. DYM AND H. P. MCKEAN, *Fourier Series and Integrals*, Academic Press, San Diego, CA, 1972.

[5] A. ERDELYI, W. MAGNUS, F. OBERHETTINGER, AND F. TRICOMI, *Higher Transcendental Functions*, Vol. II, McGraw-Hill, New York, 1953.

[6] A. FRIEDMAN, *Partial Differential Equations of Parabolic Type*, Prentice-Hall, Englewood Cliffs, NJ, 1964.
[7] S. GEMAN AND C. HWANG, *Nonparametric maximum likelihood estimation by the method of sieves*, Ann. Statist., 10 (1982), pp. 401-414.
[8] I. S. GRADSHTEYN AND I. M. RYZHIK, *Tables of Integrals, Series, and Products*, Academic Press, New York, 1980.
[9] L. GREENGARD AND J. STRAIN, *A fast algorithm for the evaluation of heat potentials*, Research Report YALEU/DCS/RR-700, Department of Computer Science, Yale University, New Haven, CT, May 1989.
[10] E. HILLE, *A class of reciprocal functions*, Ann. Math., 27(1926), pp. 427-464.
[11] N. LERNER, *Wick–Wigner functions and tomographic methods*, SIAM J. Math. Anal., 21 (1990), pp. 1084-1092.
[12] A. MCINTYRE, *Boundary integral solutions of the heat equation*, Math. Comp., 46 (1986), pp. 71-79.
[13] P. J. NOON, *The single layer heat potential and Galerkin boundary element methods for the heat equation*, Ph.D. thesis, University of Maryland, College Park, MD, 1988.
[14] W. POGORZELSKI, *Integral Equations and Their Applications*, Pergamon Press, Oxford, 1966.
[15] B. W. SILVERMAN, *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, London, 1986.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 95-106, January 1991
© 1991 Society for Industrial and Applied Mathematics
005

RECURSIVE LEAST SQUARES ON A HYPERCUBE MULTIPROCESSOR USING THE COVARIANCE FACTORIZATION*

CHARLES S. HENKEL† AND ROBERT J. PLEMMONS‡

Abstract. An efficient parallel implementation of an algorithm for recursive least squares computations based upon the covariance updating method has been developed. The target architecture is a distributed-memory multiprocessor, and test results on an Intel iPSC/2 hypercube demonstrate the parallel efficiency of the algorithm. A 64-node system is measured to execute the algorithm over 48 times as fast as a single processor for the largest problem that fits on a single node (fixed-size speedup). Moreover, the computation times increase only slightly with an increase in the number of processors when the problem size per processor remains constant. Applications include robust regression in statistics and modification of the Hessian matrix in optimization, but the primary motivation for this work is the need for fast recursive least squares computations in adaptive filtering methods in signal processing.

Key words. parallel algorithms, recursive least squares, hypercube multiprocessor, signal processing

AMS(MOS) subject classifications. primary 65F20, 65F15; secondary 65U05

1. Introduction.

1.1. Background. The problem considered in this paper is that of implementing an efficient parallel algorithm for recursive least squares computations based upon updating the covariance factorization. Our target architecture is the distributed-memory hypercube multiprocessor, and our tests are made on an iPSC/2 hypercube having an aggregate of 256 megabytes of memory and peak computational rate of 16 megaflops. Applications of recursive least squares updating include modification of the Hessian matrix in certain optimization schemes (e.g., the quasi-Newton method) [6], [19], and estimation methods schemes in adaptive signal processing and control [1], [2], [3], [5], [12], [16], [17], [20]. The primary application of interest in this paper is adaptive signal processing. In particular, we develop a parallel implementation of a recursive least squares covariance updating scheme that may be used in conjunction with Kalman adaptive filtering applications.

1.2. Least squares. The *linear least squares problem* can be posed as follows without reference to any specific application:

Given a real $m \times n$ matrix X with full column rank n and a real m-vector s, find the n-vector w that solves

$$\min \|s - Xw\|_2, \tag{1}$$

where $\|\cdot\|_2$ denotes the usual Euclidean norm. The solution to (1) is given by

$$w = (X^T X)^{-1} X^T s. \tag{2}$$

If R denotes the upper triangular *Cholesky factor* of the cross-product matrix $X^T X$, i.e., $R^T R = X^T X$, then w can be obtained by solving the triangular systems

$$R^T v = X^T s, \qquad Rw = v,$$

* Received by the editors October 24, 1988; accepted for publication (in revised form) August 16, 1989.

† Department of Nuclear Engineering, North Carolina State University, Raleigh, North Carolina 27695-8205. The research of this author was supported by the U.S. Department of Energy Nuclear Engineering Fellowship Program.

‡ Department of Mathematics and Computer Science, Wake Forest University, Winston-Salem, North Carolina 27109 (na. plemmons @ na-net. stanford. edu). The research of this author was supported by the Air Force Office of Scientific Research under grant AFOSR-88-0285 and the National Science Foundation under grant DMS-85-21154.

where v is an intermediate vector. However, in many applications where accuracy and stability are important [10], R is computed directly from X by a sequence of orthogonal transformations; that is

$$QX = \begin{bmatrix} R \\ 0 \end{bmatrix}, \qquad Q^T Q = I. \tag{3}$$

Then setting

$$Qs = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}, \quad s_1 \text{ an } n\text{-vector}, \tag{4}$$

w is computed by solving the upper triangular system

$$Rw = s_1.$$

Such schemes are sometimes called "square root" methods in the signal processing literature. In addition, the matrix

$$P \equiv (X^T X)^{-1} \tag{5}$$

is called the *covariance matrix* for (1). It measures the expected errors in w. Its inverse

$$P^{-1} = X^T X, \tag{6}$$

often called the *normal equations matrix* [10], is known as the *information matrix* for (1) in the signal processing literature [3], [12]. It measures the information content in the experiment leading to (1).

1.3. Recursive least squares. In *recursive least squares* it is required to recalculate w when observations (i.e., equations) are successively added to, or deleted from, (1). For example, in many applications information arrives continuously and must be incorporated into the solution w. This is called *updating*. Alternatively, it is sometimes important to delete old observations and have their effect excised from w. This is called *downdating*.

In some important applications the model itself is not stationary, requiring that an additional equation involving w be incorporated into the model [3], [12]. For simplicity, *we consider stationary models in this paper.*

There are two main approaches to solving recursive least square problems: the *information matrix method* based upon modifying $P^{-1} = X^T X$ and the *covariance matrix method* based upon modifying P (called information filtering and Kalman filtering, respectively, in the literature [3], [12]). Instead of modifying P^{-1} or P, directly, it is generally appropriate to modify their Cholesky factors [3], [17]. Recursive least squares algorithms based upon modifying the Cholesky factor R of $X^T X$ on a hypercube multiprocessor were already studied in [14] by Henkel, Heath, and Plemmons. *Our attention in this paper is restricted to the covariance matrix method (Kalman filtering).*

1.4. Outline of the paper. In § 2 we review methods for updating least squares problems and describe our approach to recursive least squares computations using the covariance method. Section 3 contains a description of our hypercube implementation, as well as a performance analysis given in terms of operation and communication complexities. Test results showing the actual performance of our algorithm on various configurations of the iPSC/2 are provided in § 4. Finally, § 5 contains (i) a general review of the paper and its applications, (ii) a discussion of our test findings exhibiting the efficiency of the algorithm on the hypercube architecture, and (iii) some comments on directions of future work in designing a comprehensive package of parallel algorithms for recursive least squares computations on the hypercube.

2. The RLS algorithm.

2.1. Updating computations. Recursive least squares (RLS) updating computations using the information and covariance matrix methods can be described as follows. Consider the least squares problem (1) and, without loss of generality, assume that the additional data vector y and scalar σ for the equation

$$y^T w = y_1 w_1 + \cdots + y_n w_n = \sigma$$

are appended to X and s forming

$$\tilde{X} = \begin{bmatrix} X \\ y^T \end{bmatrix}, \qquad \tilde{s} = \begin{bmatrix} s \\ \sigma \end{bmatrix}. \tag{7}$$

One now seeks to solve the modified problem

$$\min \|\tilde{s} - \tilde{X}\tilde{w}\|_2 \tag{8}$$

for the updated least squares estimate vector $\tilde{w}$ at each recursive time step. Clearly, one should be able to compute $\tilde{w}$ with the information used to compute w, along with y and σ, without performing a complete refactorization of X, i.e., in $O(n^2)$ rather than $O(n^3)$ operations, where n is the dimension of w.

We will not be concerned here with the use of exponential forgetting factors in the RLS computations (see, e.g., [12]). We take the forgetting factor to be one.

2.2. Information matrix updating. First, we review the information matrix method based upon a factorization of $P^{-1} = X^T Z$ for computing X. If R is the Cholesky factor of X, as in (3), then from (7)

$$\tilde{X}^T \tilde{X} = X^T X + yy^T = R^T R + yy^T.$$

The Cholesky factor $\tilde{R}$ of $\tilde{X}^T \tilde{X}$ can thus be calculated by applying orthogonal plane rotations to $\begin{bmatrix} R \\ y^T \end{bmatrix}$ in order to reduce the row vector y^T to zero, as described in [2], [9], [17]. Of course the same orthogonal rotations are also applied to the vector $\begin{bmatrix} s_1 \\ \sigma \end{bmatrix}$, where s_1 is the n-vector given in (4). More precisely, if

$$\tilde{Q}\begin{bmatrix} R \\ y^T \end{bmatrix} = \begin{bmatrix} \tilde{R} \\ 0^T \end{bmatrix}, \qquad \tilde{Q}\begin{bmatrix} s_1 \\ \sigma \end{bmatrix} = \begin{bmatrix} \tilde{s}_1 \\ \tilde{\sigma} \end{bmatrix}, \tag{9}$$

where $\tilde{Q}^T \tilde{Q} = I$, then

$$\tilde{R}^T \tilde{R} = R^T R + yy^T = \tilde{X}^T \tilde{X}.$$

Consequently, the updated least squares estimate vector $\tilde{w}$ is then obtained by solving the triangular system

$$\tilde{R}\tilde{w} = \tilde{s}_1. \tag{10}$$

The information matrix method described above can thus be implemented by standard procedures (see, e.g., [9]) for updating the Cholesky factor R of $P^{-1} = X^T X$ to the Cholesky factor $\tilde{R}$ of $\tilde{P}^{-1} = \tilde{X}^T \tilde{X}$, followed by the triangular solution given in (10). A hypercube implementation of this information matrix scheme is given in [14].

2.3. Covariance matrix updating. The covariance matrix scheme for updating w to $\tilde{w}$ at each recursive step is described next. Here, the covariance matrix $P = (X^T X)^{-1}$ for w is updated to $\tilde{P}$, and $\tilde{w}$ is computed from w without the necessity of a triangular solution. Essentially, one begins with the inverse Cholesky factor R^{-1} of $X^T X$, and proceeds to update w to $\tilde{w}$. Observe that since

$$P^{-1} = X^T X = R^T R,$$

the covariance matrix P has the following factorization into a product of an upper triangular matrix R^{-1} with the lower triangular matrix R^{-T}:

$$P = (X^T X)^{-1} = R^{-1} R^{-T}.$$

Thus instead of updating R for P^{-1} we now update R^{-1} for P.

The process of modifying least squares computations by updating the covariance matrix P has been used in control and signal processing for some time in the context of linear sequential filtering [3], [12], [16], [17]. One begins with estimates for $P = R^{-1}R^{-T}$ and w, and updates R^{-1} to $\tilde{R}^{-1}$ and w to $\tilde{w}$ at each recursive time step. Pan and Plemmons [18] have described a parallel scheme for these computations that forms the basis for our hypercube implementation.

Observe first that with $\tilde{X}$ given in (7), $\tilde{P}^{-1}$ is given by

$$\tilde{P}^{-1} = \tilde{X}^T \tilde{X} = X^T X + yy^T = P^{-1} + yy^T.$$

Consequently, by the Sherman-Morrison formula (see, e.g., [10]),

$$\tilde{P} = P - \frac{1}{1 + y^T P y} P y y^T P. \tag{11}$$

The updated w is given by

$$\tilde{w} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \begin{bmatrix} s \\ \sigma \end{bmatrix} = \tilde{P} \tilde{X}^T \begin{bmatrix} s \\ \sigma \end{bmatrix}. \tag{12}$$

By substituting (7) and (11) into (12) and using the representation (2) for w, there results the following basic formula for the update:

$$\tilde{w} = w + \tilde{P} y (\sigma - y^T w). \tag{13}$$

The vector

$$k \equiv \tilde{P} y \tag{14}$$

is often called the Kalman gain vector (see, e.g., [3], [12], [17]). It weights the predicted residual $\sigma - y^T w$.

Finally, we note that a discussion of the stability properties of the information and covariance methods have been given by Verhaegen and Van Dooren [20]. It is indicated there that, for problems that are not too poorly conditioned, the covariance matrix method when properly implemented performs about as well numerically as the information matrix method in the context of adaptive Kalman filtering.

2.4. The algorithm. The following algorithm for updating P to $\tilde{P}$ by updating R^{-T} to $\tilde{R}^{-T}$ and for computing the updated least squares estimate vector $\tilde{w}$ was given in Theorem 2 and Lemma 3 of [18] (see also Morf and Kailath [17]). It was indicated in Theorem 2 of [18] that the same sequence of orthogonal transformations used to update R to $\tilde{R}$ also updates R^{-T} to $\tilde{R}^{-T}$. Lemma 3 in [18] indicates that this sequence of orthogonal transformations is precisely given by (16) below. Here we write $P = L^T L$, where $L \equiv R^{-T}$.

Algorithm 1 (*Covariance Updating*). Given the current n-dimensional least squares estimate vector w, the current factor $L \equiv R^{-T}$ of $P = (X^T X)^{-1}$, and the observation $y^T w = \sigma$ being added, the algorithm computes the updated factor $\tilde{L} \equiv \tilde{R}^{-1}$ of $\tilde{P}$ and the updated least squares vector $\tilde{w}$.

(1) Form the matrix vector product

$$a = Ly. \tag{15}$$

(2) Choose orthogonal plane rotations $G_{i,n+1}$, rotating the ith component into the $n+1$st component, to form

$$G_{n,n+1} \cdots G_{1,n+1} \begin{bmatrix} -a \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \delta \end{bmatrix}, \qquad \delta = \sqrt{1 + \|a\|_2^2}, \tag{16}$$

by reducing $-a$ to 0 from the top down, and apply

$$G_{n,n+1} \cdots G_{1,n+1} \begin{bmatrix} L \\ 0^T \end{bmatrix} = \begin{bmatrix} \tilde{L} \\ u^T \end{bmatrix}, \tag{17}$$

preserving the lower triangular form of L in $\tilde{L}$.

(3) Form

$$\tilde{w} = w - \frac{1}{\delta} u(\sigma - y^T w). \tag{18}$$

Some comments about the algorithm are in order. First, as will be seen in § 3, the rotation parameters in (16) are computed successively using the components of the vector a. It follows from [18], Theorem 2, that $u = -\delta k$, where k is the Kalman gain vector given by (14). As the recursive least squares computation proceeds, $\tilde{L}$ replaces L, $\tilde{w}$ replaces w, a new equation is added, and the process returns to step (1). A parallel implementation of this algorithm on the hypercube distributed-memory system is given in the next section.

3. Hypercube implementation.

3.1. Basic principles. In solving a problem on a distributed-memory multiprocessor system such as the iPSC/2 hypercube, both the computational work and the problem data are partitioned among the processors. Ideally, the work load should be balanced so that all processors contribute equally to the computations. If a processor needs data stored in another processor, the data is sent through a network interconnecting the processors. Typically, communication is time-consuming in comparison to computation. As a result it is sometimes useful to perform redundant computations in the processors in order to avoid communication overhead. Such will be the case in our parallel implementation of the square root covariance updating algorithm.

3.2. The implementation. For our implementation of Algorithm 1, the current square root covariance factor $L \equiv R^{-T}$ of P is partitioned by columns that are distributed among the processors using a wrap mapping [15], as are the corresponding elements of the current least squares estimate vector w^T and the update vector y^T. An illustration of the storage scheme for the simple case of $n = 6$ columns and $p = 2$ processors is given in Fig. 1. In addition, the observation update σ is stored in each processor.

	(1)	(2)	(1)	(2)	(1)	(2)
	×					
	×	×				
	×	×	×			
L	×	×	×	×		
	×	×	×	×	×	
	×	×	×	×	×	×
w^T	×	×	×	×	×	×
y^T	×	×	×	×	×	×

FIG. 1. *Hypercube column wrap storage scheme for L, w^T, and y^T, with $n = 6$ columns and $p = 2$ processors.*

Our implementation can be described in terms of two stages, with the only communication occurring at the end of the first stage. A general outline of each of the stages is given first, followed by more detailed descriptions.

Basic stages.

(1) Form $a \equiv Ly$, $\alpha \equiv w^T y$ as

$$\begin{bmatrix} a \\ \alpha \end{bmatrix} = \begin{bmatrix} L \\ w^T \end{bmatrix} y$$

by a linear combination of the columns across the processors, storing the resulting accumulated vector $\begin{bmatrix} a \\ \alpha \end{bmatrix}$ in each processor.

(2) Compute the rotation parameters necessary to reduce $\begin{bmatrix} -a \\ 1 \end{bmatrix}$ to $\begin{bmatrix} 0 \\ \delta \end{bmatrix}$ from a_1 to a_n as in (16), and apply the rotations in the same order to $\begin{bmatrix} L \\ 0 \end{bmatrix}$, forming $\begin{bmatrix} L_T \\ u^T \end{bmatrix}$ across the processors. Compute $\gamma = (\sigma - \alpha)/\delta$ in each processor and form the updated vector $\tilde{w} = w - \gamma u$ across the processors.

At the end of stage 2, the new $\begin{bmatrix} L_T \\ w^T \end{bmatrix} \equiv \begin{bmatrix} \tilde{L}^T \\ \tilde{w}^T \end{bmatrix}$ is stored across the processors by columns. In the next phase of the least squares computations, the host computer receives the contributions to the desired least squares estimate vector w^T from the processors and maps the new update vector y^T and observation σ to the processors, as before. Stages 1 and 2 are then repeated, forming a recursive least squares algorithm on the hypercube.

Detailed descriptions of how we implemented stages 1 and 2 on the iPSC/2 are given next. We denote the elements and columns of L by

$$L = (l_{ij}) = [L_1, L_2, \cdots, L_n].$$

The notation "$k \in$ mycols (i)" means that processor i contains the column vector L_k and scalars w_k and y_k.

Stage 1 (For each processor i).

(i) Initialization:

$$a^{(i)} = 0 \in R^n$$

$$\alpha^{(i)} = 0$$

(ii) Linear combination:

for $k = 1, \cdots, n$
  if ($k \in$ mycols (i)) then
    for $j = k, \cdots, n$
      $a_j^{(i)} = a_j^{(i)} + y_k l_{jk}$
    $\alpha^{(i)} = \alpha^{(i)} + y_k w_k$

(iii) Global sum (gsum) to form $\begin{bmatrix} a \\ \alpha \end{bmatrix}$ across the processors, collecting the result in each processor:

$$\text{gsum} \begin{bmatrix} a^{(i)} \\ \alpha^{(i)} \end{bmatrix}.$$

Stage 2 (For each processor i).

(i) Computation of parameters ($m = \min\{k \mid k \in \text{mycols}(i)\}$):

$\beta_{m-1} = 1 + a_1^2 + \cdots + a_{m-1}^2$ (1 if $m = 1$)

$\delta_{m-1} = \sqrt{\beta_{m-1}}$

for $j = m, \cdots, n$

CompG (δ_j, c_j, s_j):

$\beta_j = \beta_{j-1} + a_j^2$

$\delta_j = \sqrt{\beta_j}$

$$c_j = \frac{\delta_{j-1}}{\delta_j}$$

$$s_j = -\frac{a_j}{\delta_j}$$

$$\delta \equiv \delta_n, \qquad \gamma = \frac{\sigma - \alpha}{\delta}$$

(ii) Application of parameters ($L \equiv (l_{ij})$):

for $k = 1, \ldots, n$

if ($k \in \text{mycols}(i)$) then

$u_k = 0$

for $j = k, \cdots, n$

ApplG (c_j, s_j, l_{jk}, u_k):

temp $= l_{jk}$

$l_{jk} = c_j l_{jk} - s_j u_k$

$u_k = c_j u_k + s_j$ temp

$w_k = w_k - \gamma u_k$

Parallelism in this implementation is achieved at each stage. In stage 1, the process of forming $\begin{bmatrix} L \\ w^T \end{bmatrix} y$ is divided among the processors, while in stage 2 the application of the Givens plane rotations to the columns of $\begin{bmatrix} L \\ 0^T \end{bmatrix}$, forming $\begin{bmatrix} L \\ u^T \end{bmatrix}$, is also divided among the processors.

The global sum operation used in stage 1, part (iii), to sum and accumulate the vector $\begin{bmatrix} a \\ \alpha \end{bmatrix}$ in each processor makes use of special iPSC/2 communication subroutines. This operation constitutes the only interprocessor communication in our implementation. Redundant computations occur during the global sum operation (see § 3.3) and in stage 2, part (i), where each processor computes the Givens rotations to be applied to the columns of L that it contains, along with the scalar γ used in updating w. These redundant calculations may appear to be inefficient; however, this calculation is a serial bottleneck in any case, whether it is done by a single node or by all the nodes. The advantage in having each node perform the calculation is that no subsequent communication is required. A similar use of redundant computations was made in Henkel, Heath, and Plemmons [14], in conjunction with the information matrix RLS implementation on the hypercube.

3.3. Performance analysis. We now give a theoretical analysis of the performance of our parallel implementation of Algorithm 1 on a hypercube multiprocessor. The actual performance of the algorithm on the iPSC/2 on a variety of test problems is reported in § 4.

Letting p denote the total number of processors, we have $p = 2^d$, where d is the dimension of the hypercube. To make the theoretical results general, we couch the performance estimates in terms of a model of computation. In our analysis we assume that the time required for a *flop* (linked scalar multiply-add pair of the form $y + ax$)

is a fixed value τ. For a given machine, we assume that one *flop* (including any necessary indexing and addressing overhead) requires τ microseconds. For simplicity, we count a square root operation as one flop. Adopting the notation in [13], we also assume that the time in microseconds required to transmit N words (four bytes each) from one processor to an immediate neighboring processor is given by a linear model

$$\alpha + N\beta,$$

where a is the start-up (latency) time for sending a message and β is the time to transmit one word. In practice, however, the model of communication time is generally nonlinear and can depend greatly on the length of the message. The parameters τ, α, and β for the iPSC/2 are given in Tables 1 and 2. It should be mentioned that these parameters are only estimates. They reflect information from [15], along with experimental timings by Dunigan on the 64 processor iPSC/2 at ORNL [7].

From Table 2 it is apparent that the iPSC/2 is most efficient on either short messages or very long messages. For our implementation of Algorithm 1, the messages have length at most $n+1$, where n is the number of least squares parameters.

The serial form of Algorithm 1 requires up to $(5/2)n^2+(15/2)n+2$ floating point multiplications, including n square roots. Since approximately the same number of additions as multiplications is required, we take

$$(\tfrac{5}{2}n^2+\tfrac{15}{2}n+2)\tau \tag{19}$$

as the serial arithmetic complexity time for Algorithm 1, in microseconds. The redundant computation performed in our parallel implementation is performed concurrently on each processor. For example, each processor computes all of the Givens plane rotation parameters concurrently. Significantly, the $(n^2/2)+O(n)$ flop operation of forming $\begin{bmatrix} a \\ \alpha \end{bmatrix} = \begin{bmatrix} L \\ w^T \end{bmatrix} y$ and the $2n^2+O(n)$ flop operation of forming $\begin{bmatrix} \tilde{L} \\ u^T \end{bmatrix} = G_{n,n+1} \cdots G_{1,n+1} \begin{bmatrix} L \\ 0^T \end{bmatrix}$ are each performed by dividing the work across the processors. Thus, we take the arithmetic complexity time for our parallel implementation as

$$\left(\frac{5}{2p}n^2+O(n)\right)\tau, \tag{20}$$

where we assume that only the work with the quadratic term is effectively performed in parallel.

TABLE 1
Approximate arithmetic timing on the iPSC/2 *hypercube.*

Parameter	
τ (μsec/flop)	6.8

TABLE 2
Approximate communication times for the iPSC/2 *hypercube.*

Parameter	Short messages (25 or fewer words)	Long messages (more than 25 words)
α (μsec)	300	894
β (μsec/4-byte word)	0	1.44

The global sum operation used in stage 1, part (iii), of the implementation makes use of the special global exchange feature described below. Some of the operations can be performed in approximately $\log_2 p$ time by using a series of *bidirectional exchanges* across the hypercube topology. Such is the case with the global sum utility subroutine gsum on the iPSC/2. Here processor contributions to the vector $[{}^{a}_{\alpha}]$ are exchanged along successive dimensions of the hypercube, interspersed with the summation of the newly acquired vectors. Fig. 2 illustrates this basic concept for the simple case of a dimension 2 hypercube (four processors).

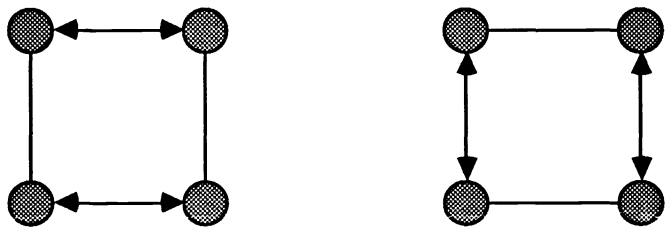


FIG. 2. *Bidirectional exchanges for global sum.*

The bidirectional exchange form of the global sum requires the optimal number of time steps, $\log_2 p$, with two communications per step that effectively take place concurrently on the iPSC/2 [7], [15]. Note that we do not perform a global fan-in where the vector $[{}^{a}_{\alpha}]$ is first collapsed to one processor and then broadcast to all the others using a global fan-out. The global sum based on bidirectional exchanges does more computations than the fan-in fan-out scheme, but requires half the time steps to produce the desired sum on each processor. The redundant computation involved in the intermediate sums is more than compensated for by this savings in communication. The communication complexity time in microseconds for our parallel implementation of the Algorithm 1 on the hypercube is given by

$$[\alpha+(n+1)\beta]\log_2 p, \tag{21}$$

where α denotes the start-up time for messages and β denotes the word transfer rate.

The performance analysis developed in this section is summarized in Table 3.

TABLE 3
Complexity times for the implementation of Algorithm 1 on a hypercube with p processors.

Parallel arithmetic complexity	$\left(\frac{5}{2p}n^2+O(n)\right)\tau$
Parallel communication complexity	$[\alpha+(n+1)\beta]\log_2 p$

4. Test results.

4.1. Framework for the computations. In this section we report the actual performance of the parallel algorithm on the iPSC/2 hypercube. The code was written in FORTRAN and compiled using the Green Hills compiler [15]. The data was generated randomly and all computations were performed in single precision. Problems ranging in size from $n=64$ to size $n=8192$ least squares estimates were considered. With four megabytes of local memory available on each processor, it was possible to run dense problems with values of n up to 1024 on a single processor in order to calculate

speedups. We consider here both standard fixed-size speedup performance as well as performance concepts introduced in [11].

4.2. Fixed-size speedup. Table 4 reports the times in seconds for a single phase of Algorithm 1 on problems of various sizes of n up to 8192. One recursive update computation for the least squares estimate vector w is performed for each problem. This reflects the addition of one equation to the least squares system (see § 2.1) and does not include data transfer times from the host processor. The best serial code was timed on one processor.

TABLE 4
Times in seconds for covariance updating on the iPSC/2 *hypercube.*

Dimension n	Number of processors p						
	1	2	4	8	16	32	64
64	.080	.044	.026	.019	.017	.016	.017
91	.157	.083	.048	.031	.024	.022	.021
128	.306	.159	.088	.053	.036	.030	.027
181	.604	.311	.166	.095	.060	.045	.039
256	1.207	.612	.319	.175	.105	.071	.056
362	2.404	1.216	.652	.332	.189	.120	.088
512	4.788	2.417	1.238	.645	.352	.210	.141
724	9.557	4.811	2.459	1.265	.672	.382	.241
1024	19.100	9.593	4.886	2.494	1.268	.681	.393
1448		19.160	9.700	4.925	2.545	1.359	.775
2048			19.330	9.767	4.996	2.624	1.444
2896				19.450	9.884	5.118	2.749
4096					19.630	10.070	5.299
5793						19.870	10.320
8192							20.260

For the largest size problem for which the algorithm can be executed on one processor ($n = 1024$), Table 5 gives the fixed-size speedup results,

$$\text{fixed-size speedup} = \frac{\text{time on 1 processor}}{\text{time on } p \text{ processors}},$$

along with the efficiencies computed by,

$$\text{efficiency} = \frac{\text{fixed-size speedup}}{p} \times 100.$$

The results given in Table 5 indicate that our implementation of the covariance matrix computations, as given in Algorithm 1, is efficient on a hypercube architecture. In particular, the test results tend to agree with the performance analysis and model for our implementation given in Table 3. The speedups here contrast with those given by Henkel, Heath, and Plemmons [14] for least squares downdating using the information matrix method, where triangular solves in the method degrade the performance of the implementation on a hypercube [8], [13].

4.3. Other efficiency concepts. In § 4.2 measurements of the parallel performance of Algorithm 1 are given in terms of fixed-size speedup: the problem size is held fixed and the number of processors is varied. However, on distributed-memory computers

TABLE 5
Fixed-size speedups and efficiency of Algorithm 1 *on the* iPSC/2 *for problem size* $n = 1024$.

Number of processors p	Fixed-size speedup	Efficiency in percent
2	1.99	99
4	3.90	98
8	7.66	96
16	15.06	94
32	28.05	88
64	48.60	76

such as the iPSC/2 hypercube, fixing the problem size can create a severe constraint, since for a relatively large number of processors it means that a problem must run efficiently even when only a portion of the memory available on each processor is used. In order to compensate for this constraint, Gustafson, Montry, and Benner [11] have introduced a performance measure based on fixing problem size per node in conjunction with their study of algorithms on a 1024 processor NCUBE at Sandia National Laboratories.

By keeping our problem size per node constant as the number of processors increases (so the total $O(n^2)$ problem size increases proportionately), the resulting timing performance chart would ideally show constant time as a function of the number of processors. The rationale here is that one is often interested in solving as large a problem as possible with a fixed number of processors. Notice that the *diagonal values in Table* 4 *remain almost constant*, increasing slightly as p increases when the problem size per node remains constant. In particular, time in seconds varies little with the hypercube dimension, as one would hope, except for the communication overhead in going from the serial version to the parallel version. Using the formulas in [11] for the case with 8,192 unknowns per node, our algorithm on 64 processors has a projected scaled speedup of over 60.

5. Summary and discussion. For the important scientific application of recursive least squares computations, we have developed an efficient parallel solution algorithm for distributed memory computers. The algorithm is based on the covariance updating method in the parallelizable form described by Pan and Plemmons [18]. Applications include robust regression in statistics, modification of the Hessian matrix in optimization, and least squares estimation methods in adaptive signal processing and control.

The parallel solution program described in § 3.2 could conceivably form the computational kernel for real-time Kalman filtering applications on distributed memory MIMD architectures (and could possibly be implemented on related special purpose systolic array architectures). One might have the following scenario: A "black box" digital filter device is connected to the host processor for the parallel system. The host processor receives the digitized signals from the device and, after initialization of the process, maps the update signals in terms of y^T and σ to the processors, receiving from the processors the updated estimates in w, which in return are communicated to the filter device.

Further work is planned for developing a comprehensive package of parallel algorithms for recursive least squares filtering on hypercube MIMD computers. For example, downdating schemes for the covariance method, as described in [18], will

be implemented. Parallel algorithms for implementations of both the information filter (based on modifying the Cholesky factor R of $P^{-1} = X^T X$) and the covariance filter (based on modifying the Cholesky factor of P) will be developed and tested.

Finally, we note that Chu and George [4] have developed an alternate hypercube implementation of Algorithm 1, which uses a novel recursive data partitioning strategy. Since no numerical tests are reported in [4], it would be of interest to compare their implementation with ours in a future paper.

Acknowledgments. We thank the ORNL Mathematics and Statistics Department, headed by Robert C. Ward, for making available to us the use of their iPSC/2 hypercube system. We also thank M. T. Heath and C. C. Paige for their helpful comments on a preliminary version of this paper. Much of the work on this paper was accomplished while we were visiting the Oak Ridge National Laboratory and the University of Tennessee during the Spring of 1988, as part of their Special Year on Numerical Linear Algebra.

REFERENCES

[1] S. T. ALEXANDER, *Adaptive Signal Processing*, Springer-Verlag, Berlin, New York, 1986.

[2] S. T. ALEXANDER, C.-T. PAN, AND R. J. PLEMMONS, *Analysis of a recursive least squares hyperbolic rotation scheme for signal processing*, Linear Algebra Appl., Special Issue on Electrical Engineering, 98 (1988), pp. 3-40.

[3] G. J. BIERMAN, *Factorization Methods for Discrete Sequential Estimation*, Academic Press, New York, 1977.

[4] E. CHU AND J. A. GEORGE, *Updating and downdating the inverse Cholesky factor on a hypercube multiprocessor*, Tech. Report CS-88-46, Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada, December 1988.

[5] J. M. CIOFFI AND T. KAILATH, *Windowed fast transversal filters, adaptive algorithms with normalization*, IEEE Trans. Acoust. Speech Signal Process., 33 (1985), pp. 607-625.

[6] J. E. DENNIS AND R. B. SCHNABEL, *Quasi-Newton Methods for Nonlinear Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[7] T. H. DUNIGAN, *Performance of a second generation hypercube*, Tech. Report ORNL/TM-10881, Oak Ridge National Laboratory, Oak Ridge, TN, August 1988.

[8] S. C. EISENSTAT, M. T. HEATH, C. S. HENKEL, AND C. R. ROMINE, *Modified cyclic algorithms for solving triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 589-600.

[9] P. E. GILL, G. H. GOLUB, W. MURRAY, AND M. A. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comput., 28 (1974), pp. 505-535.

[10] G. H. GOLUB AND C. VAN LOAN, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[11] J. L. GUSTAFSON, G. R. MONTRY, AND R. E. BENNER, *Development of parallel methods for a 1024-processor hypercube*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 609-638.

[12] S. HAYKIN, *Adaptive Filter Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[13] M. T. HEATH AND C. R. ROMINE, *Parallel solution of triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 558-588.

[14] C. S. HENKEL, M. T. HEATH, AND R. J. PLEMMONS, *Cholesky downdating on a hypercube*, in Hypercube Concurrent Computers and Applications, G. Fox, ed., ACM Press, New York, 1988, pp. 1592-1598.

[15] iPSC/2 User's Manual and Performance Report, Intel Scientific Computers, Beaverton, OR, 1988.

[16] T. KAILATH, *Signal processing applications of some moment problems*, AMS Proc. Symposium on Appl. Math., 17 (1987), pp. 71-109.

[17] M. MORF AND T. KAILATH, *Square root algorithms and least squares estimation*, IEEE Trans. Automat. Control, 20 (1975), pp. 487-497.

[18] C.-T. PAN AND R. J. PLEMMONS, *Parallel least squares modifications using inverse factorizations*, Computational and Applied Math., 34 (1989), pp. 109-127.

[19] R. J. PLEMMONS AND S. J. WRIGHT, *An efficient parallel scheme for minimizing a sum of Euclidean norms*, Linear Algebra Appl., 129 (1989), pp. 71-85.

[20] M. VERHAEGEN AND P. VAN DOOREN, *Numerical aspects of different Kalman filter implementations*, IEEE Trans. Automat. Control, 31 (1986), pp. 907-917.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 107-126, January 1991
© 1991 Society for Industrial and Applied Mathematics
006

ALGORITHMS FOR GENERATING DISCRETE RANDOM VARIABLES WITH A GIVEN GENERATING FUNCTION OR A GIVEN MOMENT SEQUENCE*

LUC DEVROYE†

Abstract. The author presents and analyzes various algorithms for generating positive integer-valued random variables when the distribution is described either through the generating function $\sum_{i=0}^{\infty} p_i s^i$ or via the sequence of moments.

Key words. generating function, random variate generation, algorithms, moment problem, rejection method, expected time analysis

AMS(MOS) subject classifications. primary 65C10; secondary 65C05

1. Introduction. Let X be a nonnegative integer-valued random variable taking the value i with probability p_i. The purpose of this paper is to suggest methods for generating X when the user is given only the probability generating function (or simply generating function)

$$k(s) = E(s^X) \triangleq \sum_{i=0}^{\infty} p_i s^i \qquad (s \in [0, 1]).$$

The p_i's are not explicitly given; if they have to be used, they should be computed from $k(s)$ (which is possible, since the generating function uniquely determines the distribution of X).

For many discrete distributions, the explicit form of the p_i's is more complicated than that of the generating function. This has greatly contributed to the popularity and importance of the generating function in probability theory and mathematical statistics. In many situations, distributions are defined in terms of a generating function first. A case in point deals with the branching processes, where the generating function of the reproduction distribution determines the extinction probability and the Malthusian (rate of growth) coefficient in case of a surviving population.

In Table 1, some generating functions are shown for well-known distributions.

In the next section, we look at some well-known and not-so-well-known properties of the generating function. In the other sections, we derive and analyze some algorithms for our problem. We are not proposing these algorithms in lieu of more classical algorithms that require the exact knowledge of the p_i's, because they are usually more time-consuming. They are primarily geared towards cases in which the p_i's are not explicitly given. Nevertheless, since the p_i's are approximated in a simple fashion, considerable time savings can be gained for distributions for which evaluations of $k(s)$ take only a fraction of the time of evaluations of p_i.

2. Properties of the generating function. The generating function $k(s)$ is nondecreasing on $[0, 1]$ and takes the value 1 at $s = 1$. For every $s \in [0, 1]$, it is defined as

* Received by the editors December 22, 1986; accepted for publication (in revised form) September 28, 1987. This research was supported in part by Natural Sciences and Engineering Research Council of Canada grant A3456 and by FCAR grant EQ-1678.

† School of Computer Science, McGill University, 805 Sherbrooke Street West, Montreal, Quebec, Canada H3A 2K6.

TABLE 1

Distribution	Generating function	p_i
Binomial (n, p)	$(1-p+ps)^n$	$\binom{n}{i}p^i(1-p)^{n-i}$ $(0 \leqq i \leqq n)$
Geometric (p)	$\dfrac{p}{1-(1-p)s}$	$(1-p)^i p$ $(i \geqq 0)$
Negative binomial (n, p)	$\left(\dfrac{p}{1-(1-p)s}\right)^n$	$\binom{n+i-1}{i}(1-p)^i p^n$ $(i \geqq 0)$
Poisson (λ)	$e^{-\lambda+\lambda s}$	$\dfrac{\lambda^i e^{-\lambda}}{i!}$ $(i \geqq 0)$
Logarithmic series (p)	$\dfrac{\log(1-ps)}{\log(1-p)}$	$\dfrac{-p^i}{i\log(1-p)}$ $(i \geqq 1)$
Uniform on $\{0, \cdots, n\}$	$\dfrac{1-s^{n+1}}{(n+1)(1-s)}$	$\dfrac{1}{n+1}$ $(0 \leqq i \leqq n)$

a convergent Taylor series

$$k(s) = \sum_{i=0}^{\infty} p_i s^i = \sum_{i=0}^{\infty} \frac{k^{(i)}(0)}{i!} s^i,$$

which implies that it is analytic on $[0, 1]$. This entails that derivatives can be taken under the summation. Thus, it is easy to see that every derivative $k^{(i)}$ is nondecreasing and possesses a convergent Taylor series for $s \in [0, 1)$.

The generating function of a convex combination (mixture) of distributions is the same convex combination of the individual generating functions. The generating function of a sum of independent random variables is the product of the individual generating functions.

The behavior of k near $s = 1$ gives us information regarding the tail behavior of the p_i's. One aspect of this is captured in the well-known property that

$$k^{(r)}(1) = E(X(X-1)\cdots(X-r+1)),$$

the rth factorial moment of X. To see this, observe that

$$k^{(r)}(s) = \sum_{i=r}^{\infty} i(i-1)\cdots(i-r+1)p_i s^{i-r}.$$

Hence all the moments of X, if they exist, can be derived from the sequence of values $k^{(r)}(1)$. The behavior of $k^{(r)}(0)$, on the other hand, provides us with information about the p_i's:

$$k^{(r)}(0) = r! p_r.$$

(This can be obtained from the series for $k^{(r)}(s)$ shown above, or by equating Taylor series expansions for $k(s)$.) Our problem would be solved if we had access to $k^{(r)}(s)$ for all r, but unfortunately we do not. One might propose a solution in which p_r is estimated by the rth derivative of $k(s)$, and this estimate is treated as the correct probability in some classical algorithm. This is unacceptable, since it introduces an error with unknown consequences in the user's application. It is therefore essential that we estimate p_r in a controlled manner (with explicit error bounds), so that we

can design an exact algorithm. To obtain such estimates, we introduce for a function $f(x)$ on the real line the *difference of order r*, defined by

$$\Delta_t^r f(x) \triangleq \sum_{i=0}^{r} (-1)^{r-i} \binom{r}{i} f(x+it),$$

which can be computed in time proportional to r if all binomial coefficients and evaluations of f are available at unit cost. It can also be obtained recursively from

$$\Delta_t^r f(x) = \Delta_t^{r-1} f(x+t) - \Delta_t^{r-1} f(x).$$

This too requires $r+1$ computations of f, but about $O(r^2)$ subtractions. Differences of order r are commonly used in approximations of functions (see, e.g., Lorentz (1986, p. 47)). The tool we will need is given in Lemma 1.

LEMMA 1 (Inequalities for the probabilities). *Assume that* $tr < 1$, $t > 0$, $r \geqq 0$. *Then,*

$$\frac{\Delta_t^r k(0)}{r!\, t^r} \geqq p_r \geqq \frac{\Delta_t^r k(0)}{r!\, t^r} - ((1-rt)^{-(r+1)} - 1) \sup_{i>r} p_i.$$

Proof of Lemma 1. We begin with a very convenient representation for differences of order r (see Lorentz (1986, p. 47)):

$$\Delta_t^r k(x) = \int_0^t \cdots \int_0^t k^{(r)}(x + y_1 + \cdots + y_r)\, dy_1 \cdots dy_r.$$

Since all $k^{(r)}$'s are $\uparrow$, it is obvious that

$$\Delta_t^r k(x) \geqq t^r k^{(r)}(x)$$

for all x, and hence,

$$\Delta_t^r k(0) \geqq t^r k^{(r)}(0) = t^r r!\, p_r.$$

The second half of the proof follows again from the integral representation, applied with $x = 0$:

$$\begin{aligned}
\Delta_t^r k(0)/t^r \leqq k^{(r)}(rt) &= \sum_{i=r}^{\infty} p_i \frac{i!}{(i-r)!} (rt)^{i-r} \\
&= r!\, p_r + r! \sum_{i=r+1}^{\infty} p_i \binom{i}{r} (rt)^{i-r} \\
&= r!\, p_r + r! \sum_{i=1}^{\infty} p_{i+r} \binom{i+r}{r} (rt)^{i} \\
&\leqq r!\, p_r + \sup_{i>r} p_i r! \sum_{i=1}^{\infty} \binom{i+r}{r} (rt)^{i} \\
&= r!\, p_r + \sup_{i>r} p_i r! ((1-rt)^{-(r+1)} - 1). \qquad \square
\end{aligned}$$

As by-products of the representation used in the proof of Lemma 1, we observe that $\Delta_t^r k(x)$ is increasing in t when r, x are kept fixed, and in x when t, r are held fixed (see also Widder (1972, p. 150)).

3. The choice of an algorithm. Lemma 1 provides us with a method for estimating every p_i to any desired accuracy based upon evaluations of $k(s)$ only, for we note that the error committed when p_r is estimated by $\Delta_t^r k(0)/(r!\, t^r)$ is not greater than

$$(1-rt)^{-(r+1)} - 1 \to 0 \quad \text{as } t \downarrow 0.$$

Interestingly, this is all we need to be able to design an exact algorithm. If p_r is estimated by $\Delta_t^r k(0)$ for some small t, there may be a serious loss of accuracy on small wordsize machines, as we need an r-times iterated difference. We will not be concerned with this issue here, preferring instead to give machine-invariant algorithms operating on real numbers.

There are basically three (partially overlapping) methods for generating discrete random variates (see, e.g., Schmeiser (1983) or Devroye (1986)): table methods, inversion methods, and rejection methods. Table methods are generally the most efficient methods (see, e.g., Marsaglia (1963), Chen and Asau (1974), Walker (1974), (1977), Ahrens and Kohrt (1981), and Peterson and Kronmal (1982)). Unfortunately, they require storage of the probabilities p_i. If they are to be used here, they should be modified in two directions, since the p_i's are known up to some small error, and since the number of possible values of X may not be finite. The way infinite tails are handled is necessarily based upon another method. The inversion and rejection methods are capable of handling infinite tails, and do not require long set-up times. They seem prime candidates for our problem. We begin with the rejection method, noting that the rejection method requires some extra knowledge in the form of a dominating probability vector (a probability vector q_i ($i \geqq 0$) is said to dominate the vector of p_i's when there exists a constant c such that $p_i \leqq cq_i$ for all i). The inversion method does not require any additional information, but is (not surprisingly) the slowest of all methods.

4. Rejection algorithms. If we knew all p_i's without error, we could use the standard rejection method:

```
REPEAT
    Generate X with probability vector q_0, q_1, ....
    Generate a uniform [0, 1] random variate U.
UNTIL Accept (U, X)
    (where Accept (U, X) ≜ [U c q_X ≦ p_X]).
RETURN X.
```

Here, as in the previous section, c is a constant with the property that $p_i \leqq cq_i$ for all i. It is known that the expected number of iterations is c. The decision $[Ucq_X \leqq p_X]$ does not require exact knowledge of p_X: it suffices to have a series of estimates of p_X approximating p_X. This technique has been developed in Devroye (1981), for example.

The function "Accept (U, X)" needed in the algorithm can be implemented as follows:

```
T ← U c q_X
t ← φ(X)/2   (φ is defined in the next section; e.g., the value φ(x) =
ε/(x(x+1)^3 ψ(x)) for some ε > 0 and a function ψ(x) = x+1 will do)
WHILE True DO
    V ← Δ_t^X k(0)/(r! t^r)
    IF T > V
        THEN RETURN False
        ELSE IF T ≦ V − (1 − Xt)^{−(X+1)} + 1
            THEN RETURN True
    t ← t/2
```

A few words of explanation are in order here. A decision is made to accept or reject only when Ucq_X does not fall in the interval around p_X defined by the inequalities of

Lemma 1, i.e.,

$$Ucq_X \notin \left[\frac{\Delta_t^X k(0)}{r!\,t^r} - (1-Xt)^{-(X+1)} + 1, \frac{\Delta_t^X k(0)}{r!\,t^r}\right].$$

Here t is varied in a geometric fashion starting with $t = \phi(X)/2$, so that it travels through the sequence $\phi(X)/2^i$ $(i = 1, 2, \cdots)$ until the interval is small enough and a decision can be reached. At this point, there is no special reason why we should consider such a sequence. The function "Accept" remains valid for any sequence $t \downarrow 0$. The proposed geometric sequence of t's leads to easily controllable expected times before halting.

A slight improvement may result in the algorithm if we bound the error not by $(1-Xt)^{-(X+1)} - 1$, but by $\min(1, \sup_{i \geqq X} cq_i)$ times this quantity. This is especially useful when the dominating probability vector is monotone $\downarrow$, or when all q_i's are small. For a similar improvement when the p_i's are known to be monotone, we refer to § 13.

The next two sections are devoted to the two key issues, the expected time needed per random variate, and the construction of dominating probability vectors q_i.

5. Expected time per random variate. The time required before halting can be measured in terms of the number of uniform $[0, 1]$ random variates needed, in terms of the number of evaluations of k, and so forth. It is helpful to isolate a counting method that is representative of the real time taken by the algorithm. One quickly realizes that the evaluations of $\Delta_t^X k(0)$ are at the heart of the matter. However, all evaluations of the difference function are not alike. If old values are not stored, then one evaluation takes time proportional to $X+1$ when binomial coefficients are available in constant time, and time proportional to $(X+1)^2$ if the difference of order X is computed recursively. To unify the treatment, let $\psi(r)$ be the time required to evaluate $\Delta_t^r k(0)$. The expected time of the algorithm, assuming that all other operations take time zero (without loss of generality) will be called $E(T)$.

For fixed $X = r$, let N be the number of iterations in the function "Accept" before a decision is reached. We observe that $P(N > i) \leqq \delta_i / cq_r$, where $\delta_i = (1-rt)^{-(r+1)} - 1$, and t is the ith value of t. This statement follows from the nature of the decision process and the fact that U is uniform $[0, 1]$. We recall that the ith value of t is $\phi(r)/2^i$. Thus, for fixed $X = r$, the expected number of evaluations of Δ_t^r is

$$\begin{aligned}
\sum_{i=0}^{\infty} P(N > i) &\leqq 1 + \sum_{i=1}^{\infty} \frac{\delta_i}{cq_r} \\
&\leqq 1 + \sum_{i=1}^{\infty} \frac{r(r+1)t}{cq_r(1-r(r+1)t)} \qquad \text{(use } (1-rt)^{r+1} \geqq 1 - r(r+1)t\text{)} \\
&\leqq 1 + \frac{r(r+1)\phi(r)}{cq_r(1-r(r+1)\phi(r))}.
\end{aligned}$$

The expected time taken by the function "Accept" for fixed $X = r$ is bounded from above by $\psi(r)$ times the previous upper bound. The expected time, averaged over all X is bounded from above by

$$c \times \sum_{r=0}^{\infty} q_r \psi(r) \left(1 + \frac{r(r+1)\phi(r)}{cq_r(1-r(r+1)\phi(r))}\right) = \sum_{r=0}^{\infty} \left(cq_r\psi(r) + \frac{r(r+1)\phi(r)\psi(r)}{1-r(r+1)\phi(r)}\right),$$

where we used Wald's equation (which explains why we can bring the c outside; see

Theorem II.3.5 of Devroye (1986)). The upper bound has two components: one that cannot be avoided,

$$cE(\psi(X)) \qquad (X \text{ has probability vector } \{q_i\})$$

(which can be considered as the absolute cost of the algorithm, regardless of how the sequence of t's is picked), and a t-dependent but distribution-invariant term

$$\sum_{r=0}^{\infty} \frac{r(r+1)\phi(r)\psi(r)}{1-r(r+1)\phi(r)},$$

which can be made as small as desired by picking the t's small enough. Consider, for example, for $X=r$,

$$t=\frac{\phi(r)}{2^i} \triangleq \frac{\varepsilon}{r(r+1)^3\psi(r)2^i} \qquad (i \geqq 1)$$

for some $\varepsilon \in (0, 1]$. Then the second contributing term in the expected time is bounded by

$$\sum_{r=0}^{\infty} \frac{\dfrac{\varepsilon}{(r+1)^2}}{1-\dfrac{\varepsilon}{\psi(r)(r+1)^2}} = \frac{\varepsilon\pi^2}{6} + O(\varepsilon^2).$$

This can be made as small as desired by the proper choice of ε.

It should not come as a surprise that even if all Δ_i^r's were precisely equal to the corresponding p_r's (and we would know this), the expected time per variate is equal to $cE(\psi(X))$, the product of the rejection constant c, and a penalty factor indicative of the size of the tail of X. The penalty factor can in some cases be infinite, but this does not invalidate the algorithm, as it still halts with probability one.

6. Dominating probability vectors. Dominating probability vectors can be found in a variety of manners. We will introduce just a few, based upon the inequalities of Lemma 2. Note that the p_i's should be bounded from above in terms of simple quantities involving the generating function only, as other information should not be assumed to be available.

LEMMA 2 (Upper bounds for probabilities). *The following exponential bounds are valid*:

$$p_i \leqq \frac{k(s)}{s^i} \qquad (s>0).$$

Furthermore, for all integers $l>0$, the following polynomial inequality is valid:

$$p_i \leqq \frac{k^{(l)}(1)}{i(i-1)\cdots(i-l+1)} \qquad (i \geqq l)$$

$$\leqq \frac{\Delta_t^l k^{(l)}(1)}{i(i-1)\cdots(i-l+1)} \qquad (\textit{all } t>0).$$

Proof of Lemma 2. The first and second inequalities follow from the fact that

$$k^{(l)}(s)=\sum_{i=l}^{\infty} i(i-1)\cdots(i-l+1)p_i s^{i-l}.$$

The first inequality is obtained by setting $l=0$ and omitting all but the ith term on the right-hand side. The inequality is obtained by setting $s=1$. □

The inequalities of Lemma 2 are but a small sampling of simple bounds depending upon the generating function only. Observe, for example, that the last inequality is a special form of Chebyshev's inequality

$$\begin{aligned} P(X=i) &\leqq P(X \geqq i) \\ &\leqq \frac{E(X(X-1)\cdots(X-l+1))}{i(i-1)\cdots(i-l+1)} \\ &= \frac{k^{(l)}(1)}{i(i-1)\cdots(i-l+1)} \qquad (i \geqq l). \end{aligned}$$

This simple argument highlights the weakness of the bounding technique employed in Lemma 2. It is usually unacceptably gross to bound individual probabilities by tail probabilities including the individual probability. There is one well-known exception to this: for exponential or subexponential tails, the probabilities are comparable in size to the tails defined from the index in question upwards. In these small-tailed cases, the first inequality of Lemma 2 is useful. Just consider an arbitrary $s>1$ to be picked further on. Then the bound $p_i \leqq k(s)s^{-i}$, when used for all i, yields a geometrically decreasing dominating probability vector. It is, of course, useful only when $k(s)<\infty$ for some $s>1$. The usefulness of the bound to us can be measured in terms of the rejection constant

$$c = \sum_{i=0}^{\infty} k(s)s^{-i} = \frac{s}{s-1}k(s).$$

This should be minimized with respect to s, if this is feasible. A random variate with the dominating probability vector $\{(s-1)/s^{i+1}(i \geqq 0)\}$ can be generated as $\lfloor E/\log(s) \rfloor$, where E is an exponential random variate. The distribution-dependent contribution to the expected time, $cE(\psi(X))$, is bounded by

$$\begin{aligned} cE\left(\psi\left(\frac{E}{\log(s)}\right)\right) &= \frac{s}{s-1}k(s)\left(1+\frac{1}{\log(s)}\right) \qquad (\psi(r) \equiv r+1) \\ &= \frac{s}{s-1}k(s)\left(1+\frac{2}{\log(s)}+\frac{2}{\log^2(s)}\right) \qquad (\psi(r) \equiv (r+1)^2). \end{aligned}$$

For distributions with heavier-than-exponential tails, the first bound of Lemma 2 is obviously useless. One should resort to a polynomial Chebyshev-like inequality, or use additional information.

7. Tables of probabilities. Let us briefly illustrate how tables of inexact probabilities can be kept to reduce the time of the rejection method. One should not confuse this with a table method, in which the table itself is needed for generating X, obviating the need for a dominating probability vector.

The idea is to fix a table size, and to store in the table records with three components, (u, δ, t), where $u = \Delta_t^r k(0)/(r!\,t^r)$ is an upper bound for p_r, t is the t value used in computing this value, and δ is the bound on the error, $(1-rt)^{-(r+1)}-1$ (thus, $u-\delta$ bounds p_r from below). In a dynamic table, there is no information at the beginning. As random variates are being generated, the table is gradually filled in. After some time, most δ's in the table will be so small that the expected time per random variate is virtually indistinguishable from that of the rejection method with a priori given p_r's.

For the sake of completeness, we indicate how the function "Accept" should be altered.

[An initially empty table of values (u_r, δ_r, t_r), $r \geqq 0$ is assumed.]
$T \leftarrow Ucq_X$
[Quick table-based decision.]
IF table entry at X present
    THEN IF $T \leqq u_X - \delta_X$
        THEN RETURN True
        ELSE IF $T > u_X$
            THEN RETURN False
            ELSE $t \leftarrow t_X/2$ (new t needed for better estimate of p_X)
    ELSE $t \leftarrow \phi(X)/2$ (for definition of ϕ, see above)
[Standard acceptance based upon converging approximations.]
WHILE True DO
    $V \leftarrow \Delta_t^X k(0)/(X!t^X)$
    $\delta \leftarrow (1-Xt)^{-(X+1)} - 1$ (for monotone $\downarrow p_i$'s, use V times this quantity)
    Update table: $(u_X, \delta_X, t_X) \leftarrow (V, \delta, t)$
    IF $T > V$
        THEN RETURN False
        ELSE IF $T \leqq V - \delta$
            THEN RETURN True
    $t \leftarrow t/2$

Dynamic tables like the one suggested here have the pleasing feature that they require work bounded above by the number of random variates generated. Thus, the dilemma of how much time should be spent (or "wasted") on setting up tables is automatically resolved. The tables suggested here improve with time. To see this, assume that $cE(\psi(X)) < \infty$. With $\phi(r) = \varepsilon/(r(r+1)^3\psi(r))$ (as suggested in the previous section), it can be shown that the expected number of evaluations of k per random variate tends to zero as more and more random variates are being generated.

8. The inversion method. We extend the ideas of the previous sections to design an inversion algorithm that is applicable to *all* generating functions, without exception. Let us recall the raw inversion method for a probability vector of p_i's:

$X \leftarrow 0$.
$S \leftarrow p_0$. (S holds $\sum_{i \leqq X} p_i$.)
Generate a uniform [0, 1] random variate U.
WHILE $U \geqq S$ DO
    $X \leftarrow X+1$
    $S \leftarrow S + p_X$
RETURN X.

The expected number of evaluations of probabilities p_i before halting is $E(1+X)$. This is an unavoidable cost. We note in passing that the inversion method is best suited for small-tailed distributions and distributions with monotonely decreasing probabilities. To show that the inversion method is not different in terms of decision making than what we have seen for the rejection method, let us rewrite the algorithm in terms of a function "Accept (U, X)," similar to the one used for the rejection method. It accepts X when $U \leqq \sum_{i \leqq X} p_i$ and rejects X otherwise.

$X \leftarrow 0$.
Generate a uniform [0, 1] random variate U.
WHILE True DO
    IF Accept (U, X)

THEN RETURN X
ELSE $X \leftarrow X+1$

The tool we will need is given in Lemma 3.

LEMMA 3 (Inequalities for cumulative probabilities). *Assume that* $r \geqq 0$, *and* $t_r = \varepsilon_r/((r+1)2^{r+1})$ *for some sequence of* ε_r*'s with* $0 < \varepsilon_r \leqq \varepsilon < 1$, *all* r. *Then,*

$$\sum_{i=0}^{r} \frac{\Delta_{t_i}^i k(0)}{i!\, t_i^i} \geqq \sum_{i=0}^{r} p_i \geqq \sum_{i=0}^{r} \frac{\Delta_{t_i}^i k(0)}{i!\, t_i^i} - \frac{2\varepsilon}{1-\varepsilon/4}.$$

Proof of Lemma 3. The upper bound for the cumulative probability is immediate from Lemma 1. For the lower bound, we again apply Lemma 1, and the fact that

$$\begin{aligned}\sum_{i=0}^{r} ((1-it_i)^{-(i+1)} - 1) &\leqq \sum_{i=0}^{r} \frac{i(i+1)t_i}{1-i(i+1)t_i} \\ &\leqq \sum_{i=0}^{r} \frac{i\varepsilon/2^{i+1}}{1-i\varepsilon/2^{i+1}} \\ &\leqq \sum_{i=0}^{\infty} \frac{i\varepsilon/2^{i+1}}{1-\varepsilon/4} = \frac{2\varepsilon}{1-\varepsilon/4}.\end{aligned}$$

□

For the implementation of the function "Accept," we immediately proceed with the dynamic table version. The table can be stored in an array, but is preferably held in a linked list. Since the inversion method increments X by one in each iteration, the linked list structure seems appropriate. Each cell in the linked list holds at least a 7-tuple, (1) the index i of the current p_i; (2) the t-value used in the computation of $\Delta_t^i k(0)$; (3) an upper bound for p_i, $\Delta_t^i k(0)/(i!\, t^i)$; (4) a lower bound for p_i, $\Delta_t^i k(0)/(i!\, t^i) - (1-it)^{-(i+1)} + 1$; (5) an upper bound for $\sum_{j \leqq i} p_j$; (6) a lower bound for $\sum_{j \leqq i} p_j$; (7) a pointer to the next cell, if not nil.

Items in the linked list are only improved as more variates are generated, i.e., t-values are replaced by smaller ones. One possible implementation is as follows:

[X also refers to the cell holding X.]
[$p_l(X)$, $p_u(X)$, $P_l(X)$ and $P_U(X)$ are the individual and cumulative lower and upper bounds referred to above.]
WHILE True DO
    IF X is not in the list (X has never been considered before)
        THEN compute all values in its cell, based upon $t = \varepsilon/((X+1)^3 2^X)$.
    IF $U \leqq P_l(X)$
        THEN RETURN True
        ELSE IF $U > P_u(X)$
            THEN RETURN False
    Update all cells $0, \cdots, X$ by halving the t's for each cell.

The size of the linked list after n random variates have been generated is precisely equal to $E(\max(X_1, \cdots, X_n))$, where the X_i's are independently and identically distributed with the given generating function k. The function "Accept" stops and returns a value in (random) finite time, since the interval around the cumulative sum shrinks with time. It is possible to generate all the information "on the fly" for each random variate generated. Obviously, it is advantageous to store information as we did in the linked list. Even without a linked list, we will now see that the expected time taken by the algorithm can be controlled very nicely.

9. Expected time per random variate for the inversion method. Let N be the number of evaluations of k needed in the course of the generation of one random variate by the inversion method. In the algorithms with memory described in the previous sections, $E(N)$ decreases with time as more and more random variates are generated. So we will take the pessimistic viewpoint that N refers to the first random variate only.

LEMMA 4. *In the inversion algorithm, $E(N)$ is bounded from above by $T_1+T_2+T_3$, where*

$$T_1=(2+\log_2 e)(\mu_3+6\mu_2+11\mu_1+3)/6, \qquad \left(\mu_i \triangleq \sum_{j=0}^{\infty} j^i p_j\right)$$

$$T_2=\sum_{i:p_i>\delta} \frac{(i^3+6i^2+11i+3)}{6}\,\delta \log_2 e,$$

$$T_3=\sum_{i:p_i\leqq\delta} \frac{(i^3+6i^2+11i+3)}{6}\,p_i \log_2\left(\frac{\delta}{p_i}\right).$$

Here $\delta=2\varepsilon/(1-\varepsilon/4)$, and ε is as in Lemma 3.

Proof of Lemma 4. For fixed U in the Xth interval (U is the uniform random variable to be inverted, and X is the generated random variable), the number of evaluations of k is $N_0+\cdots+N_X$ where N_i refers to the number due to the decision not to accept i. Now, N_i is bounded from above by $1+2+\cdots+(i+1)=(i+1)(i+2)/2=(i^2+3i+2)/2$ times the number of iterations before a decision is reached. Note that in each iteration, the t-values are halved. If this number of iterations is $n=n(U)$, we note that $n\geqq 1$, and, in view of Lemma 3, putting $u=U-\sum_{j<i} p_j$ for $i<X$ and $u=\sum_{j\leqq X} p_j-U$ for $i=X$,

$$\frac{\delta}{2^{n-1}}\leqq U,$$

where $\delta=2\varepsilon/(1-\varepsilon/4)$. Hence,

$$n\geqq \log_2\left(\frac{\delta}{u}\right)+1,$$

and thus, a safe upper bound for n is

$$2+\max\left(\log_2\left(\frac{\delta}{u}\right),0\right).$$

It is clear that u is uniformly distributed on $[0,p_X]$, so that we obtain the following upper bound for the expected number of evaluations of k:

$$E(N)\leqq \sum_{i=0}^{\infty}\int_0^{p_i}\sum_{j=0}^{i}\psi_j n(u)\,du$$

$$=\sum_{i=0}^{\infty}\Psi_i N(p_i),$$

where $\psi_j=(j^2+3j+2)/2$ (see above), $\Psi_i=\sum_{j=0}^{i}\psi_j=(i^3+6i^2+11i+3)/6$, and $N(z)=\int_0^z n(u)\,du$. It is easy to verify that

$$N(z)=\begin{cases} 2z+z\log_2\left(\dfrac{e\delta}{z}\right) & (0\leqq z\leqq\delta) \\ 2z+\delta\log_2 e & (\delta<z) \end{cases}.$$

This gives the bound

$$E(N) \leqq (2+\log_2 e) \sum_{i=0}^{\infty} \Psi_i p_i + \sum_{i:p_i>\delta} \Psi_i \delta \log_2 e + \sum_{i:p_i \leqq \delta} \Psi_i p_i \log_2 \left(\frac{\delta}{p_i}\right),$$

which was to be shown. □

Lemma 4 reveals that the expected time is finite if the entropy-related term T_3 is finite; note that this implies at the very least that the third moment of the distribution is finite. Roughly speaking, all three terms in the upper bound are indices of the size of the tail of the distribution. The second and third terms can be made as small as desired, if they are finite, by choosing ε small enough. Thus, T_1 is an unavoidable cost, inherent in the algorithm; no choice of the parameters affects T_1. The mere fact that T_1 is proportional to the third moment of the distribution renders the inversion method very inefficient; one should recall, however, that no requirements were imposed on the generating function; the inversion method is a true "black box" method that can be used as a last resort.

10. Finite distributions: Extension of the guide table method. In the case of finite probability vectors with a known bound n for the support (i.e., $p_i = 0$ for $i > n$), it is perhaps best to create a guide table of size n (Chen and Asau (1974)) for generating a random variate with probability vector proportional to $\{\Delta_t^i k(0)/(i!\, t^i),\ 0 \leqq i \leqq n\}$ for a given t, where t can possibly depend upon i. Since this vector dominates the p_i-vector (see Lemma 1), we can apply the rejection method.

[ADJUSTED GUIDE TABLE METHOD.]
[One time initialization.]
FOR $i := 0$ TO n DO $u_i \leftarrow \Delta_t^i k(0)/(i!\, t^i)$.
(t is allowed to depend upon i, see below.)

$S \leftarrow \sum_{i=0}^{n} u_i$.

Set up the necessary structures for applying the guide table method to the probability vector $\{u_i/S\}$.
[Generation.]
REPEAT
Generate X with probability vector $\{u_i/S\}$ by the guide table method.
Generate a uniform $[0, 1]$ random variate U.
UNTIL Accept (U, X) (i.e., $Uu_X \leqq p_X$)
RETURN X.

A few words about the efficiency are in order here. The set-up step requires quadratic or cubic time in n, depending upon the manner in which the Δ_t^i's are computed. This is, of course, a one-time cost. For a tight fit, it is important to take t as small as possible. The relationship between t and n is explained below. If a set-up is to be avoided altogether, one should probably start from the only guaranteed dominating probability vector, $u_i \equiv 1$, $0 \leqq i \leqq n$; or from $u_i = 1/(i+1)$, $0 \leqq i \leqq n$, in the case of nonincreasing probabilities.

The rejection constant is the expected number of uses of the guide table method before halting. It is well known that the guide table method takes expected time uniformly bounded by a constant over all distributions (see, e.g., Devroye (1986)). Hence, the rejection constant is a good index of the expected time contribution of the adjusted guide table method. A simple upper bound for the rejection constant is given below.

LEMMA 5. *The rejection constant in the adjusted guide table algorithm with $t = t_i$ is*

$$\frac{\sum_{i=0}^{n} \Delta_{t_i}^{i} k(0)}{i!\, t_i} \leqq 1 + \sup_i p_i \sum_{i=0}^{n} \frac{i(i+1)t_i}{1 - i(i+1)t_i}.$$

If we take $t_i \triangleq \varepsilon/(i(i+1)(n+1))$, where $\varepsilon \in (0, 1)$, then the rejection constant is not greater than

$$1 + \sup_i p_i \frac{\varepsilon}{1 - \varepsilon/(n+1)} \leqq 1 + \frac{\varepsilon}{1 - \varepsilon/(n+1)}.$$

This is $O(1)$ for fixed $\varepsilon > 0$, and converges to 1 as $\varepsilon \downarrow 0$.

Proof of Lemma 5. From Lemma 1, the rejection constant is not greater than

$$\sum_{i=0}^{n} \Delta_{t_i}^{i} k(0) \leqq 1 + \sup_i p_i \sum_{i=0}^{n} ((1 - it_i)^{-(i+1)} - 1)$$

$$\leqq 1 + \sup_i p_i \sum_{i=0}^{n} \frac{i(i+1)t_i}{1 - i(i+1)t_i}.$$

If we take $t_i \triangleq \varepsilon/(i(i+1)(n+1))$, where $\varepsilon \in (0, 1)$, then the least expression does not exceed

$$1 + \sup_i p_i \frac{\varepsilon}{1 - \varepsilon/(n+1)} \leqq 1 + \frac{\varepsilon}{1 - \varepsilon/(n+1)}. \qquad \square$$

Lemma 5 provides us with a simple tool for making the rejection algorithm as efficient as desired; one could, for example, take $t_i = 0.05/(i(i+1)(n+1))$.

We will see that it is even more to our advantage to take $t_i = \delta/(i(i+1)n\psi(i))$, where $\delta \in (0, 1]$ is a constant and $\psi(i)$ is a work function usually equal to $i+1$. To see this, it is necessary to analyze the second component in the total expected time, the expected number of evaluations of k. For this analysis, we give the function "Accept (U, X)" explicitly:

$V \leftarrow u_X$ (upper bound for, and approximation of p_X)
$T \leftarrow Uu_X$
$t \leftarrow t_X$ (for definition, see Lemma 5)
WHILE True DO

IF $T \leqq V - (1 - Xt)^{-(X+1)} + 1$ $\left(T \leqq V - \dfrac{X(X+1)t}{1 - X(X+1)t} \text{ will do too} \right)$

THEN RETURN True
ELSE IF $T > V$
THEN RETURN False
$t \leftarrow t/2$ (get a better approximation)
$V \leftarrow \Delta_t^X k(0)/(X!\, t^X)$

For fixed $X = i$, the expected number of evaluations of Δ_t^i is not greater than

$$\sum_{j=1}^{\infty} \frac{i(i+1)t_i/2^j}{u_i(1 - i(i+1)t_i/2^j)}$$

where we followed an argument as in § 5. Note that in the first iteration, we exit with large probability without evaluating $\Delta_t^i k(0)$. With the choice of $t_i = \delta/(i(i+1)n\psi(i))(\delta \in (0, 1])$ suggested in the discussion following Lemma 5, the upper

bound becomes

$$\sum_{j=1}^{\infty} \frac{\delta/(n\psi(i)2^j)}{u_i(1-\delta/(n\psi(i)2^j))}.$$

Each evaluation of the difference of order i takes time $\psi(i)$ (which is either $i+1$ or $(i+1)^2$, depending upon one's model); hence, the overall expected time contribution from the evaluations of the generating function is bounded from above by

$$\sum_{i=0}^{n} u_i\psi(i)\left(\sum_{j=1}^{\infty} \frac{\delta/(n\psi(i)2^j)}{u_i(1-\delta/(n\psi(i)2^j))}\right) = \sum_{i=0}^{n} \psi(i)\left(\sum_{j=1}^{\infty} \frac{\delta/(n\psi(i)2^j)}{(1-\delta/(n\psi(i)2^j))}\right)$$
$$\leqq \sum_{i=0}^{n} \left(\frac{\delta}{n-\frac{1}{2}}\right) = \delta\frac{n+1}{n-\frac{1}{2}}.$$

Here too we can control the complexity by δ: the bound can be made as small as desired by picking δ small enough. In fact, for δ very small, the performance of the algorithm should be virtually indistinguishable from that of the guide table method based upon a table of correct probabilities.

It is possible to replace the guide table by Walker tables (Walker (1974), (1977); Peterson and Kronmal (1982)). It is perhaps interesting to note that the set-up of the tables often requires much less time than for the case of known probabilities because most discrete probabilities are of a combinatorial nature and hence involve computing factorials, iterated products, Stirling numbers, and the like. Generating functions on the other hand are in many cases suprisingly "simple" functions of s.

11. The moment problem. Assume that we are not given the generating function, but that the distribution is specified indirectly in the form of values of the factorial moments

$$M_i \triangleq \sum_{j=0}^{\infty} p_j j(j-1)\cdots(j-i+1) = E(X(X-1)\cdots(X-i+1)) \qquad (i \geqq 0).$$

Under some tail conditions, these moments determine the distribution; they generally do not when one of the moments is infinite. And even if all moments are finite, the distribution is not necessarily uniquely defined: Stoyanov (1987) has constructed two families of distributions on the integers each sharing the same infinite moment sequence. We will show here how one can generate a random integer with a given moment (or factorial moment) sequence under the condition that

$$k(1+u) = E((1+u)^X) < \infty$$

for some $u > 1$. (This is also a sufficient condition for the moment sequence to determine the distribution.) The algorithm is not necessarily efficient, but at least it is conceptually *simple.*

Since there are trivial linear relationships between the ordinary moments μ_i and the factorial moments, giving the first i factorial moments is equivalent to giving the first i ordinary moments. We have

$$\mu_i = \sum_{j=0}^{i} S_{ij}M_j,$$

where S_{ij} is Stirling number of the second kind, defined, for example, through the formula

$$S_{ij} = \sum_{l=0}^{j} \binom{j}{l}(-1)^l(j-l)^i.$$

See Johnson and Kotz (1969). There are similar formulas relating M_i to the μ_j's for $0 \leqq j \leqq i$ involving Stirling numbers of the first kind (F_{ij}):

$$M_j = \sum_{i=0}^{j} F_{ij}\mu_i.$$

For easy reference, it suffices to note that $F_{ij} = 0$ when $j > 0$ and either $i > j$ or $i \leqq 0$. Furthermore, $F_{00} = 0$ and $F_{ij+1} = F_{i-1j} - jF_{ij}$ for $j \geqq i \geqq 1$. Hence we will assume without loss of generality that the factorial moments M_i are available.

Under the condition that $k(1+u) < \infty$, we see that k is a convergent power series for $|s| < 1+u$. Thus, we can expand it in a Taylor series around $s = 1$, so that for any point $s \in (-(1+u), 1+u)$,

$$k(s) = \sum_{i=0}^{\infty} \frac{(s-1)^i}{i!} M_i.$$

By the binomial theorem, this is

$$\begin{aligned} k(s) &= \sum_{i=0}^{\infty} \sum_{j=0}^{i} \binom{i}{j} s^j (-1)^{i-j} \frac{M_i}{i!} \\ &= \sum_{j=0}^{\infty} \sum_{i=j}^{\infty} s^j (-1)^{i-j} \frac{M_i}{j!(i-j)!} \\ &= \sum_{j=0}^{\infty} \frac{s^j}{j!} \sum_{i=0}^{\infty} (-1)^i \frac{M_{j+i}}{i!}. \end{aligned}$$

Under certain conditions, this series can be identified on a term-by-term basis with the original defining power series for $k(s)$. This would yield

$$p_j = \frac{1}{j!} \sum_{i=0}^{\infty} (-1)^i \frac{M_{j+i}}{i!}.$$

For sufficient conditions under which this argument is valid, see Lemma 6 below. The formula for p_j is mentioned for compact support distributions in standard textbooks, such as Kendall and Stuart (1977, p. 96). For it to be useful to us, we require some information about the error committed when the ∞ in the series is replaced by a finite integer n.

LEMMA 6. *Assume that $k(1+u) < \infty$ for some $u > 1$. Let p_{nj} be an n-term approximation of p_j defined by*

$$p_{nj} = \frac{1}{j!} \sum_{i=0}^{n} (-1)^i \frac{M_{j+i}}{i!}.$$

Then, for all even n, $p_{nj} \geqq p_j$, while for all odd n, $p_{nj} \leqq p_j$. Furthermore, for all j,

$$\lim_{n \to \infty} p_{nj} = p_j.$$

Proof of Lemma 6. $k(s)$ is absolutely summable on $A = (-(1+u), 1+u)$, and hence is analytic on A (Dieudonné (1969, p. 203)). Therefore, $k(s)$ can be expanded in a unique absolutely summable power series around every point in the interior of A. The same is true for all derivatives of k. In particular, we have

$$k(s) = \sum_{j=0}^{\infty} (s-1)^j \frac{M_j}{j!}$$

for $s<1+u$, and

$$k^{(j)}(s)=\sum_{i=0}^{\infty}\frac{(s-1)^i}{i!}k^{(j+i)}(1)$$

$$=\sum_{i=0}^{\infty}\frac{(s-1)^i}{i!}M_{j+i}.$$

We know that $j!p_j=k^{(j)}(0)$. To apply the last series expansion at $s=0$, we need to require that the radius of convergence be greater than one, and hence that $k(1+u)<\infty$ for $u>1$. It is also known that if we truncate the last sum to include only the terms $0\leqq i\leqq n$, then the remainder is

$$\frac{(s-1)^{n+1}}{(n+1)!}k^{(j+n+1)}(\theta)$$

for some $s\leqq\theta\leqq 1$. The remainder for $s=0$ is therefore negative if and only if n is even. Thus, the partial series alternately underestimates and overestimates p_j.

The last statement of Lemma 6 follows from the observation that

$$\sum_{i=0}^{\infty}\frac{u^i}{i!}M_i<\infty,$$

and thus, $M_n\leqq Cn!u^{-n}$ for some constant C. From this, for fixed j,

$$\frac{M_{j+n}}{n!j!}\leqq C\binom{n+j}{n}u^{-(n+j)}\leqq C(n+j)^j u^{-(n+j)}.$$

This tends to zero when $u>1$. This implies the convergence of p_{nj} to p_j as $n\to\infty$. □

The sandwiching inequalities in Lemma 6 are also referred to as Bonferroni's inequalities. They are, in fact, universally valid. The condition on k in fact implies that for large n the inequalities are also tight. The series representation for p_j given in Lemma 6 can be used for designing general algorithms for generating random variables with a given moment sequence. These algorithms would not involve partial solutions of the well-known moment problem (discussed, for example, in Widder (1972)). Since this has been an open problem for a long time, it is perhaps worth our while to provide clues as to how one might proceed. Note that the treatment here parallels that of the generating function. It should be noted, however, that our algorithms cannot possibly work when the moments do not uniquely define the distribution. This can be seen as a drawback, since it would seem that the additional freedom would make the generation problem simpler. The reader can verify that our algorithms do not halt when for some j, the sequence $M_{j+i}/i!$ does not tend to zero as $i\to\infty$. The algorithms are thus intimately linked to the solution of the moment problem (see, e.g., Feller (1971)). In the next two sections, we will take the liberty of referring to our generation problem as "the moment problem."

12. Rejection algorithms in the moment problem. Rejection can be dealt with as in § 4. Of course we need a dominating probability vector. Since we assume that $k(1+u)<\infty$ for some $u>1$ (Lemma 6), we can use the bound

$$p_j\leqq k(1+u)(1+u)^{-j},$$

if the k-value in question, or an upper bound for it, is available. In any case, we always have

$$p_j\leqq\min(\mu_r/j^r,1)$$

for all fixed r, where μ_r is the rth moment. We should not forget that μ_r can be computed from the first r factorial moments M_i, $1 \leqq i \leqq r$. Random variates with the polynomially decreasing probability vector can be generated by the rejection method. To see this, construct a function $g(x)$ via

$$\begin{aligned} g(x) &\triangleq 0, x < -1 \\ &= 1, -1 \leqq x < c \triangleq \lfloor \mu_r^{1/r} \rfloor. \\ &= \mu_r / x^r, x \geqq c. \end{aligned}$$

This function has the property that $g(x) \geqq p_j$ for all $j-1 < x \leqq j$ and all $j \geqq 0$. Thus, for the rejection algorithm, we can proceed as follows, after noting that the constant portion of g has integral $c+1$, and that the polynomial tail has integral $\mu_r/((r-1)c^{r-1})$:

[Rejection algorithm for $\{\min(\mu_r/j^r, 1)\}$.]
WHILE True DO
Generate iid uniform [0, 1] random variates U, V.

IF $U \leqq \dfrac{c+1}{c+1+\mu_r/((r-1)c^{r-1})}$

THEN RETURN $X \leftarrow \lceil -1+(c+1)V \rceil$
(use fact that $p_j = 1$ for $j \leqq c$)
ELSE
$Y \leftarrow c/V^{1/(r-1)}$ (Y has tail-of-g density)
$X \leftarrow \lceil Y \rceil$
Generate a uniform [0, 1] random variate W.

IF $W \leqq \left(\dfrac{Y}{X}\right)^r$ THEN RETURN X

The expected number of iterations in the algorithm is the area under g, i.e.,

$$\begin{aligned} c+1+\frac{\mu_r}{(r-1)c^{r-1}} &= \lfloor \mu_r^{1/r} \rfloor + 1 + \frac{\mu_r}{(r-1)(\lfloor \mu_r^{1/r} \rfloor)^{r-1}} \\ &\leqq \lceil \mu_r^{1/r} \rceil + 1 + \frac{\mu_r}{(r-1)(\lceil \mu_r^{1/r} \rceil)^{r-1}} \\ &\leqq 2 + \frac{r}{r-1} \mu_r^{1/r}. \end{aligned}$$

The rejection algorithm for the moment problem is as in § 4, provided that we take $cq_j \triangleq \min(\mu_r/j^r, 1)$, and that the function "Accept (U, X)" is implemented as follows:

$V \leftarrow U \min(\mu_r/X^r, 1)X!$
(get ready for testing whether $V = Ucq_X X! \leqq p_X X!$)
$n \leftarrow 0$.
$S \leftarrow M_X$
WHILE True DO
IF n even
THEN IF $V > S$ THEN RETURN False (reject)
ELSE IF $V \leqq S$ THEN RETURN True (accept)
$n \leftarrow n+1$
$S \leftarrow S+(-1)^n M_{X+n}/n!$ (next approximation)

The validity of the algorithm follows from the fact that the approximations of each p_j converge, and that consecutive approximations are underestimates and overestimates, respectively.

13. Expected time per random variate in the moment problem. In this section, we determine the efficiency of the algorithm under the assumption that the jth factorial moment is available at unit cost, or in the worst case at cost $j+1$ (when it has to be computed from the first j standard moments). For this purpose, let N be the number of evaluations of a factorial moment. We have the following lemma.

LEMMA 7. *Assume that $k(u)<\infty$ for some $u>3$, where k is the factorial moment generating function of some random variable. The expected number $E(N)$ of factorial moments required by the rejection algorithm for the moment problem is bounded by*

$$c+k(3),$$

where c is the rejection constant (see previous section).

Proof of Lemma 7. Fix $X=r$, and consider the number of moments N_r needed before a decision is reached whether to reject or accept. It is known that by Wald's theorem $E(N)=\sum_{r=0}^{\infty} cq_r E(N_r)$ where c is the rejection constant, and q_r is the dominating probability vector.

We have

$$P(N_r-1\geqq j)\leqq \frac{M_{r+j-1}}{r!(j-1)!cq_r} \qquad (j\geqq 1)$$

and

$$P(N_r-1\geqq 0)=1.$$

Here we use Lemma 6 on alternate over- and underestimates of p_r. Thus,

$$\begin{aligned} E(N_r-1) &= \sum_{j=1}^{\infty} P(N_r-1\geqq j) \\ &\leqq \sum_{j=1}^{\infty} \frac{M_{r+j-1}}{r!(j-1)!cq_r} \\ &= \sum_{j=0}^{\infty} \frac{M_{r+j}}{r!j!cq_r} \\ &= \sum_{j=0}^{\infty} \frac{k^{(r+j)}(1)}{r!j!cq_r} \\ &= \frac{k^{(r)}(2)}{r!cq_r}. \end{aligned}$$

Thus,

$$\begin{aligned} E(N) &= \sum_{r=0}^{\infty} cq_r(1+E(N_r-1)) \\ &\leqq c+\sum_{r=0}^{\infty} \frac{k^{(r)}(2)}{r!} \\ &= c+k(3), \end{aligned}$$

provided that $k(u)<\infty$ for some $u>3$. □

In practice, the bound of Lemma 7 is all but useless except for very short-tailed distributions. Indeed, by Jensen's inequality, $k(3) \geqq 3^{E(X)}$. Nevertheless, the bound is intuitively appealing, since it shows a rejection component (c), and a distributional rejectionless component $k(3)$. The latter component describes a cost we cannot do without, since it is present even if we have 100 percent acceptance probabilities ($c = 1$). In other words, it can be considered as a cost inherent in the series approximation of probabilities by means of moments or factorial moments. On the positive side, it should be mentioned that $k(u) < \infty$ (all $u > 0$) for many important distributions, such as all finite distributions and all distributions with a subexponential tail (i.e., for any $C > 0$, $p_i \leqq e^{-Ci}$ for all i large enough).

Even though the algorithm is valid when $k(u) < \infty$ for some $u > 2$, Lemma 7 does not provide a finite expected time guarantee when $k(u) < \infty$ for some $u \in (2, 3]$, but $k(u) = \infty$ for all $u > 3$.

When the rth factorial moment is not available at unit cost, but at cost $\psi(r)$, "$N_r + 1$" should be replaced in the argument by $\sum_{i=0}^{r} \psi(i)$. Although we will not pursue this any further, it suffices to mention that for $\psi(r) = r^l$ (l integer), the condition for finite expected time becomes $k^{(l)}(3) < \infty$.

Just as in the first half of this paper, the rejection algorithm can be replaced by a usually less efficient but universally applicable inversion algorithm. In the moment problem, such an algorithm can be based upon the series representation

$$P(X \geqq i) = \frac{1}{(i-1)!} \sum_{j=0}^{\infty} (-1)^j \frac{M_{i+j}}{j!(j+i)} \qquad (i \geqq 1)$$

(see, e.g., Laurent (1965, p. 437)).

The ideas presented here for the moment problem can be applied (with modifications) to continuous random variables as well. It suffices to observe that the distribution function can be represented as a converging series depending upon the moments only, provided that the moments uniquely define the distribution. See, for example, Theorem 2 on p. 227 of Feller (1971). We will present results for the general moment problem elsewhere.

14. Monotone distributions. If, in addition to the generating function $k(s)$, we know that the p_i's are monotonically $\downarrow$, it is possible to improve upon the rejection and inversion algorithms given in §§ 4–10, basically because we have the following lemma.

LEMMA 8 (Inequalities for monotone probabilities). *Assume that* $tr < 1$, $t > 0$, $r \geqq 0$. *Then,*

$$\frac{\Delta_t^r k(0)}{r!\, t^r} \geqq p_r \geqq \frac{\Delta_t^r k(0)}{r!\, t^r} (1 - ((1 - rt)^{-(r+1)} - 1)).$$

Lemma 8 is an immediate corollary of Lemma 1. The function "Accept (U, X)" used in the rejection algorithm can now be improved as follows:

$T \leftarrow Ucq_X$
$t \leftarrow \phi(X)/2$
WHILE True DO
    $V \leftarrow \Delta_t^X k(0)/(r!\, t^r)$
    IF $T > V$
        THEN RETURN False
        ELSE IF $T \leqq V(2 - (1 - Xt)^{-(X+1)})$ (improved step)
            THEN RETURN True
    $t \leftarrow t/2$

This modification has some implications on the expected time. We note that in the notation of § 5, $\Delta_t^r k(0)/(r!\,t^r) \leqq p_r/(1-\delta_i)$, so that

$$P(N>i) \leqq \frac{\Delta_t^r k(0)\delta_i}{cq_r} \leqq \frac{\delta_i p_r}{cq_r(1-\delta_i)},$$

where t is the ith value of t. A little work, too tedious to reproduce here, shows that the expected time per random variate is bounded from above by

$$cE(\psi(X)) + \sum_{r=0}^{\infty} \frac{r(r+1)\phi(r)\psi(r)p_r}{1-2r(r+1)\phi(r)},$$

where X has probability vector $\{q_i\}$. This is nearly always smaller than the corresponding bound for the original rejection algorithm (see § 5).

In terms of a dominating probability vector, we can also slightly improve on what we have been before, since now

$$p_i \leqq \frac{(s-1)k(s)}{s^{i+1}-1} \qquad (s>1;\ i \geqq 0).$$

The latter inequality is best obtained by the following argument:

$$p_i \frac{s^{i+1}-1}{s-1} = p_i \sum_{j=0}^{i} s^j \leqq \sum_{j=0}^{i} p_j s^j \leqq k(s).$$

The area under the bounding curve becomes

$$k(s) \sum_{i=0}^{\infty} \frac{1}{1+s+\cdots+s^i},$$

which is better than the corresponding area obtained without monotonicity (see § 6). Unfortunately, there does not seem to be a simple way of generating a random variable with probability vector proportional to $1/(s^{i+1}-1)$ except via rejection from a geometric random variable with probability vector proportional to s^{-i}.

REFERENCES

J. H. AHRENS AND K. D. KOHRT (1981), *Computer methods for efficient sampling from largely arbitrary statistical distributions*, Computing, 26, pp. 19-31.

H. C. CHEN AND Y. ASAU (1974), *On generating random variates from an empirical distribution*, AIIE Transactions, 6, pp. 163-166.

L. DEVROYE (1981), *The series method in random variate generation and its application to the Kolmogorov-Smirnov distribution*, Amer. J. Math. Management Sci., 1, pp. 359-379.

——— (1986), *Non-Uniform Random Variate Generation*, Springer-Verlag, Berlin, New York.

J. DIEUDONNÉ (1969), *Foundations of Modern Analysis*, Academic Press, New York.

W. FELLER (1971), *An Introduction to Probability Theory and its Applications*, Vol. 2, John Wiley, New York.

N. L. JOHNSON AND S. KOTZ (1969), *Distributions in Statistics: Discrete Distributions*, John Wiley, New York.

M. KENDALL AND A. STUART (1977), *The Advanced Theory of Statistics*, Vol. 1, Fourth Edition, Macmillan, New York.

A. G. LAURENT (1965), *Probability distributions, factorial moments, empty cell test*, in Classical and Contagious Discrete Distributions, Pergamon Press, New York, pp. 437-442.

G. G. LORENTZ (1986), *Approximation of Functions*, Second Edition, Chelsea Publishing Company, New York.

G. MARSAGLIA (1963), *Generating discrete random variables in a computer*, Comm. ACM, 6, pp. 37-38.

A. V. PETERSON AND R. A. KRONMAL (1982), *On mixture methods for the computer generation of random variables*, Amer. Statist., 36, pp. 184-191.

B. W. SCHMEISER (1983), *Recent advances in generating observations from discrete random variables*, in Computer Science and Statistics: The Interface, J. E. Gentle, ed., pp. 154–160, North-Holland, Amsterdam.

J. STOYANOV (1987), *Counterexamples in Probability*, John Wiley, Chichester, UK.

A. J. WALKER (1974), *New fast method for generating discrete random numbers with arbitrary frequency distributions*, Electron. Lett., 10, pp. 127–128.

——— (1977), *An efficient method for generating discrete random variables with general distributions*, ACM Trans. Math. Software, 3, pp. 253–256.

D. V. WIDDER (1972), *The Laplace Transform*, Princeton University Press, Princeton, NJ.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 127-144, January 1991
© 1991 Society for Industrial and Applied Mathematics
007

THE REFORMULATION AND NUMERICAL SOLUTION OF CERTAIN NONCLASSICAL INITIAL-BOUNDARY VALUE PROBLEMS*

GRAEME FAIRWEATHER† AND RICK D. SAYLOR‡

Abstract. Several physical phenomena are modeled by nonclassical parabolic or hyperbolic initial-boundary value problems in one space variable which involve an integral over the spatial domain of a function of the desired solution or of its first spatial derivative. This integral may appear in the boundary conditions and/or the governing partial differential equation itself. In this paper, several examples of problems of this type, which arise in such diverse areas as chemical diffusion and heat conduction processes, thermoelasticity, population dynamics, vibration problems, nuclear reactor dynamics, and certain biological processes, are presented. In general, these problems are either not in the form required by widely available general-purpose software packages or the application of such software would be awkward and/or inefficient. It is shown how the example problems can be converted to a form to which existing software or standard numerical procedures may be applied in a relatively straightforward manner. Special attention is devoted to one particular problem, the diffusion equation subject to the specification of mass in a portion of the domain, which has been studied quite extensively, both analytically and numerically, in recent years. As an example of an effective method for the solution of the reformulated version of this problem, a finite-difference method based on Keller's box scheme is considered. Results of numerical experiments are presented which compare this scheme with a Crank–Nicolson finite-element Galerkin method proposed recently.

Key words. nonclassical parabolic and hyperbolic initial-boundary value problems, nonlocal boundary conditions, partial integrodifferential equations, Keller's box scheme

AMS(MOS) subject classification. 65M05

1. Introduction.

Several physical phenomena are modeled by nonclassical parabolic or hyperbolic initial-boundary value problems in one space variable which involve an integral over the spatial domain of a function of the desired solution or of its first spatial derivative. Such an integral may appear in a boundary condition, in which case the boundary condition is called **nonlocal**, or in the governing partial differential equation itself, which is then often referred to as a **partial integrodifferential equation**, or in both. In this paper, we present several examples of problems of this type which arise in such diverse areas as chemical diffusion and heat conduction processes, thermoelasticity, population dynamics, vibration problems, nuclear reactor dynamics, and certain biological processes.

The presence of an integral term in a boundary condition and/or the differential equation can greatly complicate the application of standard numerical techniques, (see, for example, Cannon, Esteva, and van der Hoek (1987)). Moreover, the use of general-purpose software packages for the solution of such problems may be awkward and/or inefficient, or the problems may not even be in the proper form required by the software packages. It is therefore important to be able to convert nonclassical problems to a more desirable form, not only to improve the efficiency of software for

* Received by the editors May 9, 1988; accepted for publication (in revised form) November 3, 1989.

† Departments of Mathematics and Engineering Mechanics, University of Kentucky, Lexington, Kentucky 40506. The work of this author was supported in part by funds from National Science Foundation grant RII-8610671 and the Commonwealth of Kentucky through the University of Kentucky's Center for Computational Sciences, and also by the Commission for Educational Exchange between the United States of America and Spain while he was a Fulbright Scholar at the Universidad de Valladolid, Valladolid, Spain.

‡ Center for Applied Energy Research, University of Kentucky, 3572 Iron Works Pike, Lexington, Kentucky 40511. The work of this author was supported in part by the Institute for Mining and Minerals Research of the University of Kentucky.

solving initial-boundary value problems but also to make it more widely applicable to problems of practical interest.

Schryer (1977), (1984) gave examples of the applicability of the software package POST for solving coupled systems of partial differential equations in one space variable and time and ordinary differential equations in time using Galerkin's method with B-splines for the spatial discretization. He briefly discussed the case in which the nonlocal condition

$$\int_0^1 u(s, t)\, ds = 1 \tag{1.1}$$

is imposed on the solution $u(x, t)$ of a partial differential equation, but provided no concrete examples of such problems. Schryer outlined a procedure for approximating the integral using B-splines and Gaussian quadrature in such a way that POST can be used to solve the resulting initial-boundary value problem. In a complex problem considered by Berzins, Dew, and Furzeland (1988), a more complicated integral term arises and it too is approximated by Gaussian quadrature to convert the problem to one that can be handled by the package SPRINT (Berzins, Dew, and Furzeland (1984), (1988)). The use of quadrature approximations in these problems is not without its difficulties. First, the accuracy of the quadrature must be compatible with that of the discretization of the differential equations. Moreover, the sparsity of coefficient matrices of systems of linear algebraic equations arising in the time-stepping procedures is complicated by the use of quadrature formulae (see, for example, Fig. 5 of Berzins, Dew, and Furzeland (1988)).

In § 2, we present examples of problems with nonlocal boundary conditions and indicate how each problem can be converted to one with separated boundary conditions. The price one pays for this modification is an increase of one in the number of partial differential equations for each nonlocal condition. However, the additional equations are straightforward and the augmented problem can easily be handled using standard numerical techniques. For example, in the case of (1.1), we introduce the function $w(x, t)$ defined by

$$w(x, t) = \int_0^x u(s, t)\, ds. \tag{1.2}$$

Then

$$\frac{\partial w}{\partial x} = u(x, t), \tag{1.3}$$

which is the additional partial differential equation, and from (1.2) it is obvious that

$$w(0, t) = 0, \tag{1.4}$$

and, from (1.1),

$$w(1, t) = 1. \tag{1.5}$$

Thus the nonlocal condition (1.1) is replaced by the equation (1.3) and the boundary conditions (1.4) and (1.5).

The partial integrodifferential equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \int_0^1 c(x, s)u(s, t)\, ds, \qquad x \in [0, 1], \quad t \geqq 0, \tag{1.6}$$

is mentioned by Schryer (1981), (1984), who proposed the following procedure for reformulating this equation so that it lies in the problem class for which POST was designed. Since the semidiscretization implemented in POST is based on Galerkin's method using B-splines, suppose

$$u(x, t) = \sum_{i=1}^{N} U_i(t) B_i(x)$$

is the representation of the solution in terms of the basis functions $\{B_i\}_{i=1}^{N}$. Then the differential equation can be written in the form

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \sum_{i=1}^{N} v_i(t) c_i(x), \tag{1.7}$$

where

$$c_i(x) = \int_0^1 c(x, s) B_i(s)\, ds,$$

and

$$v_i(t) = U_i(t), \qquad i = 1, \cdots, N. \tag{1.8}$$

Thus the reformulated problem consists of the partial differential equation (1.7) and the system of N linear algebraic equations (1.8) subject to appropriate initial and boundary conditions. This problem can now be handled by POST or by SPRINT. Of course, the functions $c_i(x)$ must be determined or approximated by an appropriate quadrature rule, and in a more complicated problem, for example, when the integrand is a nonlinear function of u, quadrature would have to be used to obtain an equation of the form (1.7). Moreover, in the linear equations arising in a time-stepping procedure in POST or SPRINT, the coefficient matrices would be full in the columns corresponding to the N functions $v_i(t)$, $i = 1, \cdots, N$, because of the appearance of these functions in (1.7).

In § 3, we give examples of physical phenomena which are modeled by partial integrodifferential equations. In contrast to Schryer's problem (1.6), these examples involve integrals which are independent of the space variable x, and are functions of the time t only. As a consequence, they can be easily reformulated as a system of partial differential equations using a procedure similar to (1.2)-(1.3). For example, if in (1.6) the function c is independent of x, and we set

$$w(x, t) = \int_0^x c(s) u(s, t)\, ds, \tag{1.9}$$

then

$$\frac{\partial w}{\partial x} = c(x) u(x, t), \qquad w(0, t) = 0, \tag{1.10}$$

and (1.6) becomes

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + w(1, t). \tag{1.11}$$

The introduction of the partial differential equation (1.10) is comparable in complexity to the use of the functions (1.8) in that its semidiscretization introduces N time-dependent functions, but no quadrature approximations are involved. Furthermore,

the algebraic problem arising in the application of standard numerical techniques to the reformulated problem is much simpler, there being only one full column in the coefficient matrices, arising from the term $w(1, t)$ in (1.11). It should be noted that Schryer (1981), (1984) provided no examples of physical problems which are modeled by equations of the form (1.6) where the integral term is a function of both x and t.

In § 4, we examine in some detail the numerical solution of one particular problem involving nonlocal boundary conditions, the parabolic problem corresponding to the diffusion equation subject to the specification of mass (see Example 2.1). This problem is singled out for special attention because it has been studied quite extensively, both analytically and numerically, in recent years, and results of numerical experiments appear in the literature. As an example of a method which can solve the reformulated version of this problem systematically and in a more straightforward manner than the Crank–Nicolson Galerkin approach of Cannon, Esteva, and van der Hoek (1987), we consider a finite-difference technique based on Keller's box scheme (Keller (1971)). This method was chosen because it is ideally suited for solving problems involving first-order equations of the form (1.3), and higher order approximations can be easily obtained using Richardson extrapolation in space or time. Moreover, at each time step, the box scheme requires the solution of an almost block diagonal linear system which can be carried out very effectively using widely available software developed by Diaz, Fairweather, and Keast (1983a), (1983b). It should be noted that a software package implementing a box scheme is being developed by Furzeland for release in the NAG Library, (Berzins (1988)).

In § 4, the box scheme is compared with a Crank–Nicolson finite-element Galerkin method developed by Cannon, Esteva, and van der Hoek (1987) on the set of test problems employed by these authors. It is shown that for all but one of the problems the box scheme is the more accurate, and, in the exceptional case, the accuracy of the method can be improved by using Richardson extrapolation in space or time.

2. Initial-boundary value problems with nonlocal boundary conditions. In this section, we give examples of parabolic and hyperbolic initial-boundary value problems which involve nonlocal boundary conditions, and show how each problem can be converted to one with separated boundary conditions.

Example 2.1. Diffusion subject to the specification of mass. (Heat conduction subject to the specification of energy.) Certain chemical diffusion and heat conduction processes are modeled by the nonclassical parabolic initial-boundary value problem

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \qquad (x, t) \in I \times J, \tag{2.1.1}$$

$$u(x, 0) = f(x), \qquad x \in I, \tag{2.1.2}$$

$$\frac{\partial u}{\partial x}(1, t) = g(t), \qquad t \in J, \tag{2.1.3}$$

$$\int_0^b u(s, t)\, ds = F(t), \qquad b \in I, \quad t \in J, \tag{2.1.4}$$

where $I = (0, 1)$, $J = (0, T]$, b is a given constant, and f, g, and F are prescribed functions. For example, if u denotes the concentration of a chemical in a diffusion process, then $F(t)$ represents the mass of the chemical present in the region $0 < x < b$ at time t. Likewise, if u is the temperature in a heat conduction problem, then $F(t)$ represents the internal energy content of the region $0 < x < b$ at time t (see Cannon and van der Hoek (1986), Cannon, Esteva, and van der Hoek (1987), and Samarskii

(1980)). A similar problem arises in biochemistry in which $b = 1$ and F is a constant, and in this case (2.1.4) is known as the conservation of protein (Sauer (1988)). Analytical properties of solutions of problems of the form (2.1.1)–(2.1.4) and generalizations of it have been studied by several authors, see, for example, Cannon (1963), Ionkin (1977), Kamynin (1964), Lin (1988), and Yurchuk (1986).

Finite-difference and finite-element Galerkin methods have been formulated and analyzed for the solution of (2.1.1)–(2.1.4). Specifically, Cannon and van der Hoek (1986) analyzed a finite-difference scheme in the case in which b, the upper limit of integration in (2.1.4), is a function of time. This method, which is based on the backward-difference method, is rather involved and requires the simultaneous solution of a Volterra integral equation of the second kind to determine an approximation to a function which is essentially $\partial u(0, t)/\partial x$. Cannon, Esteva, and van der Hoek (1987) considered a semidiscrete Galerkin method based on a rather complicated weak form of the initial-boundary value problem which requires the solution of a system of linear Volterra integrodifferential equations of the second kind. They also analyzed a fully discrete Crank–Nicolson Galerkin method. Earlier, Cannon and van der Hoek (1982) analyzed a backward-difference-type method for the case in which the boundary condition (2.1.3) is replaced by

$$u(1, t) = g(t), \qquad t \in J. \tag{2.1.5}$$

All of these methods are certainly nonstandard, involving as they do the classical theta function and its first or second spatial derivative, which must be evaluated at each time step (see § 4). Moreover, it is not clear how they would apply to more general problems.

To handle the nonlocal condition (2.1.4), we define

$$w(x, t) = \int_0^x u(s, t)\, ds, \qquad (x, t) \in (0, b) \times J, \tag{2.1.6}$$

(cf. (1.2)), from which it follows that

$$\frac{\partial w}{\partial x} = u, \qquad (x, t) \in (0, b) \times J. \tag{2.1.7}$$

The nonlocal condition (2.1.4) becomes

$$w(b, t) = \int_0^b u(s, t)\, ds = F(t), \qquad t \in J, \tag{2.1.8}$$

and from (2.1.6) it is obvious that

$$w(0, t) = 0, \qquad t \in J. \tag{2.1.9}$$

Thus the reformulated problem comprises the partial differential equations (2.1.1) and (2.1.7) subject to the initial condition (2.1.2) and the (separable) boundary conditions (2.1.9), (2.1.8), and (2.1.3). The numerical solution of this initial-boundary value problem is discussed in § 4.

Example 2.2. Quasi-static theory of thermoelasticity. Parabolic problems with nonlocal boundary conditions also arise in the quasi-static theory of thermoelasticity. For example, consider a slab $0 \leqq x \leqq 1$ which is made of homogeneous, isotropic material and which undergoes a motion in which the displacement vector is parallel to the x-axis of a system of rectangular cartesian coordinates. Day (1982) has shown that $\eta(x, t)$, the entropy per unit volume, satisfies

$$(1 + \delta^2)\frac{\partial \eta}{\partial t} = \frac{\partial^2 \eta}{\partial x^2}, \qquad (x, t) \in I \times J, \tag{2.2.1}$$

subject to the nonlocal conditions

$$\eta(0, t) = \eta(1, t) = -\delta^2 \int_0^1 \eta(s, t)\, ds, \qquad t \in J,$$

and the initial condition

$$\eta(x, 0) = \eta_0(x), \qquad x \in I.$$

In (2.2.1),

$$\delta = (3\lambda + 2\mu)\alpha \left(\frac{\theta_0}{(\lambda + 2\mu)c} \right)^{1/2},$$

where λ, μ are the elastic moduli, θ is the reference temperature, c is the specific heat per unit volume, and α is the coefficient of thermal expansion.

A second example, given by Day (1983), describes the quasi-static flexure of a thermoelastic rod of unit length. In this case, the entropy η satisfies (2.2.1), subject to the nonlocal conditions

$$\eta(0, t) = -2\delta^2 \int_0^1 (2 - 3s)\eta(s, t)\, ds, \qquad t \in J,$$

$$\eta(1, t) = 2\delta^2 \int_0^1 (1 - 3s)\eta(s, t)\, ds, \qquad t \in J.$$

Here

$$\delta^2 = \frac{\theta_0 B^2}{cA},$$

where A is the flexural rigidity, the constant B is a measure of the cross-coupling between thermal and mechanical effects, and, again, θ_0 and c denote the reference temperature and specific heat per unit volume, respectively.

These examples lead Day (1983) to study properties of solutions of parabolic problems of the form

$$b(x) \frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left(a(x) \frac{\partial u}{\partial x} \right) - c(x)u, \qquad (x, t) \in I \times J, \tag{2.2.2}$$

with nonlocal conditions

$$u(0, t) = \int_0^1 f(s)u(s, t)\, ds, \qquad t \in J, \tag{2.2.3}$$

$$u(1, t) = \int_0^1 g(s)u(s, t)\, ds, \qquad t \in J, \tag{2.2.4}$$

where a, b, c, f, and g are continuous functions with a and b positive and c nonnegative. Such problems can be reformulated in the following way. First set

$$w_0(x, t) = \int_x^1 f(s)u(s, t)\, ds.$$

Then it follows that

$$\frac{\partial w_0}{\partial x} = -f(x)u(x, t), \tag{2.2.5}$$

and

$$w_0(0, t) = u(0, t), \tag{2.2.6}$$

from (2.2.3). Moreover,

$$w_0(1, t) = 0. \tag{2.2.7}$$

Likewise set

$$w_1(x, t) = \int_0^x g(s)u(s, t)\, ds$$

to obtain

$$\frac{\partial w_1}{\partial x} = g(x)u(x, t), \tag{2.2.8}$$

$$w_1(0, t) = 0, \tag{2.2.9}$$

and, from (2.2.4),

$$w_1(1, t) = u(1, t). \tag{2.2.10}$$

Thus the reformulated problem consists of the partial differential equations (2.2.2), (2.2.5), and (2.2.8) subject to the separated boundary conditions (2.2.6), (2.2.9), (2.2.7), (2.2.10).

Example 2.3. Population dynamics. First-order hyperbolic initial-boundary value problems with nonlocal boundary conditions of the form

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = -m(x)u, \qquad x \in [0, A], \quad t \geqq 0, \tag{2.3.1}$$

$$u(0, t) = f(E(t))E(t), \qquad t > 0, \tag{2.3.2}$$

$$u(x, 0) = u_0(x), \qquad x \in [0, A], \tag{2.3.3}$$

where

$$E(t) = \int_0^A b(s)u(s, t)\, ds, \tag{2.3.4}$$

arise in the study of mathematical models for population dynamics. The function $u(x, t)$ denotes the population density at time t with respect to age x, A being the maximum attainable age. Also, the functions $m(x)$ and $b(x)$ denote the age-specific mortality rate and fertility rate, respectively, and f represents a process controlling the passage of the newborn $E(t)$ into the youngest class of age $u(0, t)$. An example of such a problem is the case of fish population, where $E(t)$ corresponds to the parental egg production, $u(0, t)$ to the recruits of the population, and f is the proportion of eggs that survive to be fishes (see Di Blasio (1983)). Several examples of this type of problem are studied in Witten (1983); see also Györi (1988) who provides a theoretical analysis of the method of lines for the solution of similar first-order hyperbolic problems.

The nonlocal condition (2.3.2) can be treated in the same way as (2.2.3), that is, by setting

$$w(x, t) = \int_x^A b(s)u(s, t)\, ds,$$

from which we obtain

$$\frac{\partial w}{\partial x} = -b(x)u(x, t), \tag{2.3.5}$$

$$w(A, t) = 0. \tag{2.3.6}$$

Since

$$w(0, t) = E(t),$$

the condition (2.3.2) may be written as

$$u(0, t) = f(w(0, t))w(0, t). \tag{2.3.7}$$

The reformulated problem then consists of the equations (2.3.1) and (2.3.5) subject to the boundary conditions (2.3.7) and (2.3.6).

3. Parabolic and hyperbolic initial-boundary value problems involving partial integrodifferential equations. In this section, we briefly describe examples of physical phenomena which are modeled by parabolic or hyperbolic initial-boundary value problems in one space variable in which the governing equation is a partial integrodifferential equation. Specifically, the equation contains an integral over the spatial domain of a function of the solution or of its first derivative with respect to x, the space variable. As in § 2, we show how each problem may be reformulated by introducing an additional first-order partial differential equation similar to (1.10).

Example 3.1. Vibration problems. It is well known that the wave equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}$$

constitutes only a first approximation in the study of transverse vibrations of a string (Antman (1980)). The dimensionless nonlinear equation

$$\frac{\partial^2 u}{\partial t^2} = \left(1 + \int_0^1 \left(\frac{\partial u}{\partial s}\right)^2 ds\right)\frac{\partial^2 u}{\partial x^2}, \qquad (x, t) \in I \times J, \tag{3.1.1}$$

provides a better model in that it takes into account the increase in tension resulting from the extension of the string (Dickey (1969)).

Equation (3.1.1) subject to homogeneous Dirichlet boundary conditions was tackled numerically by Christie and Sanz-Serna (1984), (1985) who first rewrote (3.1.1) as a first-order system by setting

$$\psi = \frac{\partial u}{\partial t}, \qquad \phi = \frac{\partial u}{\partial x}.$$

Equation (3.1.1) then becomes

$$\frac{\partial \psi}{\partial t} = \left(1 + \int_0^1 [\phi(s, t)]^2\, ds\right)\frac{\partial \phi}{\partial x}, \tag{3.1.2a}$$

$$\frac{\partial \phi}{\partial t} = \frac{\partial \psi}{\partial x}. \tag{3.1.2b}$$

Christie and Sanz-Serna semidiscretized this system using Galerkin's method with piecewise linear basis functions and, for the time integration, examined predictor-corrector methods based on Crank–Nicolson-type methods. These authors recommended against the use of Newton's method in Crank–Nicolson time discretizations because the relevant Jacobians are not sparse. This difficulty may be overcome by introducing the function w defined by

$$w(x, t) = 1 + \int_0^x [\phi(s, t)]^2 \, ds. \tag{3.1.3}$$

Then

$$\frac{\partial w}{\partial x} = \phi^2, \tag{3.1.4}$$

and

$$w(0, t) = 1. \tag{3.1.5}$$

Equation (3.1.2a) then becomes

$$\frac{\partial \psi}{\partial t} = w(1, t) \frac{\partial \phi}{\partial x}. \tag{3.1.6}$$

If the Crank–Nicolson Galerkin method with piecewise linear basis functions were used to solve system (3.1.6), (3.1.2b), (3.1.4), the relevant Jacobian would be banded apart from one full column arising from the term $w(1, t)$ appearing in (3.1.6).

Another vibration problem of some interest which can be treated in a similar way involves the so-called extensible beam equation, a special case of

$$\frac{\partial^2 u}{\partial t^2} + \alpha \frac{\partial^4 u}{\partial x^4} - \left\{ \beta + k \int_0^L \left[\frac{\partial u}{\partial s}(s, t) \right]^2 ds \right\} \frac{\partial^2 u}{\partial x^2} + c(x, t, u) = f(x, t], \tag{3.1.7}$$

$$(x, t) \in (0, L) \times (0, T],$$

where α, β, k are constants, α and k being positive. This equation with $c(x, t, u) \equiv 0$, $f(x, t) \equiv 0$ was proposed by Woinowsky-Krieger (1950) as a model for the transverse deflection of an extensible beam of natural length L. Several authors have examined properties of equation (3.1.7) subject to various types of boundary conditions (see Yoshida (1985) and references cited therein).

Recently, Geveci and Christie (1989) examined the stability and convergence of a semidiscrete Galerkin method and two fully discrete Crank–Nicolson-type methods for the approximate solution of the beam equation subject to the boundary conditions

$$u(0, t) = u(L, t) = 0, \qquad \frac{\partial^2 u}{\partial x^2}(0, t) = \frac{\partial^2 u}{\partial x^2}(L, t) = 0,$$

which correspond to the case in which the ends of the beam are hinged. As in the work of Christie and Sanz-Serna (1984), (1985), predictor-corrector methods are employed to avoid the use of Newton's method for the solution of the nonlinear Crank–Nicolson equations, in which the Jacobian is essentially full. The introduction of the function w defined by (3.1.3) again avoids this difficulty.

Example 3.2. Nuclear reactor dynamics. In nuclear reactor dynamics, we encounter initial-boundary value problems of the form

$$\frac{du(t)}{dt} = -\int_0^c \alpha(s)T(s,t)\,ds, \qquad t>0, \tag{3.2.1a}$$

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(b(x)\frac{\partial T}{\partial x}\right) - q(x)T + \eta(x)\sigma(u(t)), \qquad x\in[0,c], \quad t>0, \tag{3.2.1b}$$

subject to the initial conditions

$$u(0)=u_0, \quad T(x,0)=f(x), \quad x\in[0,c], \tag{3.2.2}$$

and the boundary conditions

$$\begin{aligned} d_1T(0,t)+d_2\frac{\partial T(0,t)}{\partial x} &= 0, \qquad t>0,\\ d_3T(c,t)+d_4\frac{\partial T(c,t)}{\partial x} &= 0, \qquad t>0, \end{aligned} \tag{3.2.3}$$

where

$$|d_1|+|d_2|>0, \qquad |d_3|+|d_4|>0.$$

Such problems are of interest as dynamic models of a one-dimensional continuous medium nuclear reactor for which $\sigma(u)=-1+e^u$. In (3.2.1)-(3.2.3), x denotes position along the reactor, which is regarded as a rod of length c, t denotes the time, $u(t)$ is the logarithm of the total reactor power, $T(x,t)$ is the deviation of the temperature from equilibrium, $\alpha(x)$ is the ratio of the temperature coefficient of reactivity to the mean life of neutrons, $\eta(x)$ is the fraction of power generated at x, and $q(x)$ is a constant multiple of the coefficient of external thermal conductivity. Properties of the problem (3.2.1)-(3.2.3) and special cases of it are examined by Bronikowski, Hall, and Nohel (1972) and in references cited by these authors.

The integral appearing in (3.2.1a) is of the same form as that discussed in § 1 with respect to equation (1.6), and is treated in a similar way by setting

$$w(x,t) = -\int_0^x \alpha(s)T(s,t)\,ds.$$

Then

$$\frac{\partial w}{\partial x} = -\alpha(x)T(x,t), \tag{3.2.4}$$

$$w(0,t)=0, \tag{3.2.5}$$

and (3.2.1a) becomes

$$\frac{du(t)}{dt} = w(c,t). \tag{3.2.6}$$

The reformulated problem consisting of (3.2.6), (3.2.1b), (3.2.4) subject to (3.2.2), (3.2.3), and (3.2.5) is now in a form which can be solved by any software package capable of handling ordinary differential equations in time coupled to partial differential equations in space and time.

Example 3.3. A bubble collapse problem. Berzins, Dew, and Furzeland (1988) discussed a simplified model of the collapse of a vapor-filled bubble contained in liquid, which involves two parabolic partial differential equations and eight coupled differential-algebraic equations. In this model, the condensation rate $m(t)$ and the collapse rate of the bubble $S(t) \equiv \dot{s}(t)$, $s(t)$ being the location of the bubble wall at time t, are connected by the relation

$$3IS + \dot{I}S - 1002m = 0,$$

where

$$I(t) = \int_0^1 \xi^2 p(v(\xi, t))\, d\xi,$$

p being a given function and v the temperature inside the bubble. The integral term can be handled as in Example 3.2.

Example 3.4. A biological model. The following equations have been used for the mathematical description of the model of a capillary bed (see Györi (1988) and Jacquez (1972)):

$$\frac{\partial u(x, t)}{\partial t} + a\frac{\partial u(x, t)}{\partial x} = b[v_1(t) - u(x, t)], \qquad (x, t) \in [0, L] \times [0, T],$$

$$\frac{dv_1(t)}{dt} = c[v_2(t) - v_1(t)] + d\int_0^L [u(s, t) - v_1(t)]\, ds, \qquad t \in [0, T],$$

$$\frac{dv_2(t)}{dt} = e[v_1(t) - v_2(t)], \qquad t \in [0, T],$$

where u, v_1, and v_2 denote concentrations in the capillary, the interstitial compartment, and the intracellular compartment, respectively, of the model, and L is the length of the capillary. The given nonnegative constants a, b, c, d, e, are determined from certain parameters of the capillary bed. Note that in this example, which involves an integral term similar to that of Example 3.2, the partial differential equation is a first-order hyperbolic.

Example 3.5. Population dynamics. In nonlinear age-dependent population dynamics, the Gurtin and MacCamy (1974) model is given by equations (2.3.1)-(2.3.3) where the function m also depends on $E(t)$. This problem was studied by Marcati (1983), for example. A simpler model in which $E(t)$ occurs only in the function m, and $u(0, t)$ is a prescribed function of t was considered by Kostova (1988), who investigated the use of the Rothe method (Rektorys (1982)) to approximate solutions of such problems. (In the Rothe method, a semidiscretization is applied to the time variable, leaving the space variable unchanged.) The convergence of the new procedure was examined and results of a numerical experiment presented. The proposed method appears to be unduly involved; the problem can be easily reformulated as in previous examples, and solved in a straightforward manner, using a basic finite-difference method, for example.

A similar problem was considered by Elderkin (1983) who examined a model of dispersal in a patchy environment. In this case, the initial-boundary value problem is

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = f(u) - \delta u + \alpha \int_0^1 \delta u(s, t)\, ds, \qquad x \in [0, 1], \quad t \geqq 0,$$

$$u(0, t) = u_0, \qquad t \geqq 0,$$

$$u(x, 0) = \phi(x), \qquad x \in [0, 1].$$

4. The numerical solution of Example 2.1. The Galerkin methods discussed by Cannon, Esteva, and van der Hoek (1987) are based on the following weak form of the initial-boundary value problem (2.1.1)-(2.1.4):

$$\left(\frac{\partial u}{\partial t}, v\right)+\left(\frac{\partial u}{\partial x}, \frac{\partial v}{\partial x}\right)$$

$$= g(t)v(1, t)+\left[F'(t)-D(t)+2\left\{\int_0^t \frac{\partial^2 \theta}{\partial x^2}(b, t-\tau)u(0, \tau)\, d\tau\right.\right. \tag{4.1}$$

$$\left.\left.-\int_0^t \frac{\partial^2 \theta}{\partial x^2}(b-1, t-\tau)u(1, \tau)\, d\tau\right\}\right] v(0, t), \qquad v \in H^1(I),$$

where $H^1(I)$ denotes the usual Sobolev space, the function $\theta(x, t)$ is the classical theta function defined by

$$\theta(x, t)=\sum_{s=-\infty}^{\infty} K(x+2s, t), \tag{4.2}$$

with

$$K(x, t)=\begin{cases}\dfrac{1}{2\sqrt{\pi t}} \exp\left(\dfrac{-x^2}{4t}\right), & t>0, \quad x \in R, \\ 0, & t \leqq 0, \quad x \in R,\end{cases}$$

and

$$D(t)=2\theta(b, t)f(0)-[\theta(b-1, t)+\theta(b+1, t)]f(1)$$

$$+\int_0^1 \{\theta(b-\xi, t)+\theta(b+\xi, t)\}f'(\xi)\, d\xi. \tag{4.3}$$

The continuous-time Galerkin method is formulated in the standard way. We choose a finite-dimensional subspace of $H^1(I)$, S_h say, and define the continuous-time Galerkin approximation as the differentiable map $U:[0, T] \to S_h$ which satisfies (4.1) for all $v \in S_h$. It is easy to see that the determination of U requires the solution of a system of linear Volterra integrodifferential equations (VIDEs) of the second kind. Note that an added complication is that the function $D(t)$ defined by (4.3) must be evaluated at each time level and requires the integration of a function involving the theta function.

In the Crank-Nicolson Galerkin method formulated by Cannon, Esteva, and van der Hoek (1987), the time discretization is performed in a locally second-order fashion using an approach based on the trapezoidal rule for the approximation of the VIDEs. Under certain smoothness assumptions on the solution u of (2.1.1)-(2.1.4), L^2 and H^{-1} error estimates for this scheme are derived by these authors. In particular, it is shown that if S_h is chosen to be the space of piecewise linear functions defined on a uniform grid with mesh length h, and the timestep k is sufficiently small, then

$$\|(u-U)_{m+1}\|^2+k\sum_{j=0}^{m}\left\|\frac{\partial}{\partial x}(u-U)_{j+1/2}\right\|^2=O(h^2+k^2),$$

provided h/k (or h^2/k; see Cannon, Esteva, and van der Hoek (1987)) is a fixed constant, the norm being the usual L^2 norm, and $U_j(x) \approx u(x, jk)$. This is a suboptimal L^2 norm estimate of the error since we would expect second-order accuracy in space when piecewise linear functions are employed. Moreover, since the time discretization is performed using the trapezoidal rule, we would also expect second-order accuracy in time.

We now consider a finite-difference method based on Keller's box scheme (Keller (1971)) for solving the reformulated version of the initial-boundary value problem (2.1.1)-(2.1.4) which comprises the partial differential equations (2.1.1) and (2.1.7) subject to the initial condition (2.1.2) and the boundary conditions (2.1.9), (2.1.8), and (2.1.3) (see Example 2.1). In order to apply the box scheme, we first rewrite the initial-boundary value problem (2.1.1)-(2.1.3), (2.1.7)-(2.1.9) as a first-order system in the following way. Let

$$v = \frac{\partial u}{\partial x}. \tag{4.4}$$

Then (2.1.1) becomes

$$\frac{\partial u}{\partial t} = \frac{\partial v}{\partial x}, \qquad (x, t) \in I \times J, \tag{4.5}$$

and from (2.1.2) it follows that

$$u(x, 0) = f(x), \qquad v(x, 0) = f'(x), \qquad x \in I. \tag{4.6}$$

Also, from (2.1.3), we have

$$v(1, t) = g(t), \qquad t \in J. \tag{4.7}$$

When Keller's box scheme is applied to the problem consisting of (4.4), (4.5), and (2.1.7) subject to the initial conditions (4.6) and the boundary conditions (2.1.9), (2.1.8), (4.7), the coefficient matrix of the system of linear algebraic equations arising at the mth time step has the form

$$A_m = \begin{bmatrix} B & & & & & & & & \\ L_1 & R_1 & & & & & & & \\ & L_2 & R_2 & & & & & & \\ & & \ddots & & & & & & \\ & & & L_K & R_K & & & & \\ & & & & L_{K+1} & R_{K+1} & & & \\ & & & & & \ddots & & & \\ & & & & & & L_N & R_N & \\ & & & & & & & C & \end{bmatrix}$$

where the submatrices B and C arise from the boundary conditions, and, for the boundary conditions (2.1.9) and (4.7), we have $B = (0\ 0\ 1)$ and $C = (0\ 1)$. The submatrices L_i and R_i arise from the difference equations on the ith subinterval, $1 \leqq i \leqq N$. It should be noted that the function w is defined only on the interval $[0, b]$, and this is reflected in the fact that, if b is a gridpoint, $b = x_K$, say, then the submatrices L_i and R_i are not all of the same order. In fact, L_i is 3 by 3 when $1 \leqq i \leqq K$, and 2 by 2 otherwise, R_i is 3 by 3 when $1 \leqq i \leqq K-1$, and 2 by 2 for $K+1 \leqq i \leqq N$, while R_K is 3 by 2. For more details, see Fairweather and Saylor (1989).

The matrix A_m is **almost block diagonal** (de Boor (1978)), and systems with coefficient matrices of this form are solved very effectively using the package ARCECO (Diaz, Fairweather, and Keast (1983a), (1983b)) which implements modified alternate row and column elimination, a variant of a method proposed by Varah (1976), which is stable and generates no fill-in.

When the boundary condition (2.1.3) is replaced by (2.1.5), the only change required to the preceding is to replace the submatrix C in the matrix A_m by

$$C' = (1\ \ 0).$$

It is not at all obvious how such a boundary condition would be handled in the Galerkin approach of Cannon, Esteva, and van der Hoek (1987).

The box scheme with ARCECO was compared with the Crank-Nicolson Galerkin method of Cannon, Esteva, and van der Hoek (1987) on the set of test problems considered by these authors. In all of the problems, $b = 0.75$, and the Galerkin method employs piecewise linear functions defined on a uniform partition of $\bar{I}$ of meshwidth h, and for the evaluation of $\theta(x, t)$ and $\partial^2\theta(x, t)/\partial x^2$, the first 21 terms ($s = -10$ to 10) of the series (4.2) are used. In the first three examples, $h = k = 0.0025$, where k is the time step.

In Tables 1-3, we present the relative error, $|u - U|/|u|$, at $x = 0.25$ for the Crank-Nicolson Galerkin method and the box scheme. It should be noted that, in Cases 1-3, Cannon, Esteva, and van der Hoek presented their results to six decimal places, and to 10 decimal places in Cases 4 and 5. All of the results for the box scheme are presented to three significant figures.

In Cases 1 and 2, the box scheme produced results which are significantly more accurate than those obtained using the Crank-Nicolson Galerkin method and were determined in a much more straightforward fashion. In Case 3, the box scheme was slightly less accurate than the Galerkin method but the box scheme approximations can be easily improved using Richardson extrapolation, as we shall see in Case 4. Cannon, Esteva, and van der Hoek remark on the growth of the relative error with

TABLE 1

Case 1. $f(x) = \frac{1}{2}x^2$, $g(t) = 1.0$, $F(t) = 0.75t + \frac{1}{6}(0.75)^3$, *and* $u(x, t) = \frac{1}{2}x^2 + t$.

t	Crank-Nicolson Galerkin	Box scheme
.01	0.1929 (−2)	0.933 (−5)
.025	0.993 (−3)	0.132 (−4)
.1	0.1834 (−2)	0.497 (−5)

TABLE 2

Case 2. $f(x) = \cos x$, $g(t) = -e^{-t}\sin(1.0)$, $F(t) = e^{-t}\sin(0.75)$, *and* $u(x, t) = e^{-t}\cos x$.

t	Crank-Nicolson Galerkin	Box scheme
.01	0.89 (−4)	0.352 (−6)
.025	0.60 (−4)	0.695 (−6)
.1	0.281 (−3)	0.621 (−6)

TABLE 3

Case 3. $f(x) = \sin \pi x$, $g(t) = -\pi e^{-\pi^2 t}$, $F(t) = (1/\pi)(1/\sqrt{2}+1)\, e^{-\pi^2 t}$, *and* $u(x, t) = e^{-\pi^2 t}\sin \pi x$.

t	Crank-Nicolson Galerkin	Box scheme
.01	0.4 (−5)	0.173 (−6)
.025	0.3 (−5)	0.794 (−5)
.1	0.7 (−5)	0.325 (−4)

time in their results, which they attribute to the incomplete summation of the theta function. There is some such growth in Case 3 for the box scheme, the reason for which is unknown. It should be noted that, while the solution is linear in t and quadratic in x in Case 1, the box scheme is not exact because the function $w(x, t)$ defined by (2.1.6) is cubic in x.

In a second set of experiments, Cannon, Esteva, and van der Hoek examined the convergence of their Crank-Nicolson Galerkin method for sufficiently small k. In the following experiments, the relative errors at $x = 0.25$, $t = 0.1$ for various values of h and k are presented.

The problem considered in Case 4 is the same as that in Case 3, and the results for the Crank-Nicolson Galerkin method and the box scheme are presented in Table 4.

TABLE 4

Case 4. Relative errors at $x = 0.25$, $t = 0.1$ for (a) *Crank-Nicolson Galerkin method*; (b) *box scheme.*

k / h		.05	.01	.005	.0025	.001
.25	(a)	.621 (−1)	.734 (−1)	.711 (−1)	.715 (−1)	.715 (−1)
	(b)	.275 (0)	.361 (0)	.627 (0)	.116 (1)	.277 (1)
.05	(a)	.132 (−1)	.278 (−2)	.282 (−2)	.283 (−2)	.283 (−2)
	(b)	.187 (−1)	.424 (−2)	.432 (−2)	.783 (−2)	.937 (−2)
.01	(a)	.162 (−1)	.240 (−4)	.117 (−3)	.114 (−3)	.115 (−3)
	(b)	.130 (−1)	.523 (−3)	.243 (−3)	.172 (−3)	.153 (−3)
.005	(a)	.163 (−1)	.111 (−3)	.327 (−4)	.287 (−4)	.296 (−4)
	(b)	.128 (−1)	.410 (−3)	.130 (−3)	.605 (−4)	.409 (−4)
.0025	(a)	.163 (−1)	.133 (−3)	.113 (−4)	.729 (−5)	.812 (−5)
	(b)	.128 (−1)	.381 (−3)	.102 (−3)	.325 (−4)	.130 (−4)
.001	(a)	.163 (−1)	.139 (−3)	.510 (−5)	.105 (−5)	.351 (−7)
	(b)	.128 (−1)	.373 (−3)	.942 (−4)	.247 (−4)	.520 (−5)

In this case, the results obtained using the Galerkin method are slightly more accurate than those from the box scheme. (The error in the Galerkin approximation for $h = k = 0.001$ does seem unusually small.) Cannon, Esteva, and van der Hoek claim that their results show that, in accordance with their theoretical results, convergence occurs for a sufficiently small time step, in this case for $k \leqq 0.005$. The meaning of this statement is unclear, since for $h = 0.01$, 0.005, and 0.0025 the error for $k = 0.001$ is larger than that for $k = 0.0025$. This behaviour does not appear in the results obtained using the box scheme. In fact, the box scheme results clearly demonstrate second-order accuracy in time for $h \leqq .0025$ and $k \leqq 0.01$, and second-order accuracy in space for $h \leqq 0.01$ and $k \leqq .0025$. Moreover, more accurate approximations may be determined by performing Richardson extrapolation in time (Table 5) or in space (Table 6). The extrapolated values in Table 5 show the dominance of the spatial error, while those in Table 6 indicate that the time error dominates.

In Case 5, the results obtained using the box scheme are uniformly more accurate than the Crank-Nicolson Galerkin results (see Table 7). Cannon, Esteva, and van der Hoek claim that convergence of their method appears to begin at $k = 0.001$. For $h \leqq .005$ and $k \leqq 0.01$, the results for the box scheme exhibit second-order accuracy in time, and extrapolation in time again produces more accurate approximations (Table 8). The results do not exhibit the expected second-order accuracy in space, perhaps because of the domination of the time error.

TABLE 5

Case 4. *Relative errors at* $x = 0.25$, $t = 0.1$ *for* (a) *box scheme*; (b) *box scheme with one time extrapolation.*

h / k		.01	.005	.0025	.001
.0025	(a)	.381 (−3)	.102 (−3)	.325 (−4)	.130 (−4)
	(b)		.904 (−5)	.931 (−5)	.930 (−5)
.001	(a)	.373 (−3)	.942 (−4)	.247 (−4)	.520 (−5)
	(b)		.121 (−5)	.148 (−5)	.149 (−5)

TABLE 6

Case 4. *Relative errors at* $x = 0.25$, $t = 0.1$ *for* (a) *box scheme*; (b) *box scheme with one space extrapolation.*

h / k		.01	.005	.0025	.001
.001	(a)	.153 (−3)	.409 (−4)	.130 (−4)	.520 (−5)
	(b)		.371 (−5)	.371 (−5)	.371 (−5)

TABLE 7

Case 5. $f(x) = \sum_{n=1}^{5} \cos nx$, $g(t) = \sum_{n=1}^{5} n e^{-n^2 t} \sin n$, $F(t) = \sum_{n=1}^{5} (1/n)(\sin .75n)\, e^{-n^2 t}$, *and* $u(x, t) = \sum_{n=1}^{5} e^{-n^2 t} \cos nx$. *Relative errors at* $x = 0.25$, $t = 0.1$ *for* (a) *Crank–Nicolson Galerkin method*; (b) *box scheme.*

h / k		.05	.01	.005	.0025	.001
.25	(a)	.176 (0)	.523 (−1)	.247 (−1)	.300 (−1)	.298 (−1)
	(b)	.281 (−1)	.366 (−2)	.149 (−1)	.517 (−1)	.162 (0)
.05	(a)	.151 (0)	.262 (−1)	.442 (−2)	.140 (−2)	.123 (−2)
	(b)	.196 (−1)	.159 (−2)	.105 (−2)	.673 (−3)	.530 (−3)
.01	(a)	.150 (0)	.252 (−1)	.560 (−2)	.238 (−3)	.625 (−4)
	(b)	.190 (−1)	.744 (−3)	.213 (−3)	.809 (−4)	.440 (−4)
.005	(a)	.150 (0)	.252 (−1)	.564 (−2)	.202 (−3)	.261 (−4)
	(b)	.189 (−1)	.717 (−3)	.186 (−3)	.534 (−4)	.163 (−4)
.0025	(a)	.150 (0)	.252 (−1)	.565 (−2)	.193 (−3)	.169 (−4)
	(b)	.189 (−1)	.711 (−3)	.179 (−3)	.465 (−4)	.939 (−5)
.001	(a)	.150 (0)	.252 (−1)	.565 (−2)	.190 (−3)	.129 (−4)
	(b)	.189 (−1)	.709 (−3)	.177 (−3)	.446 (−4)	.745 (−5)

5. Conclusion. In this study, it is shown how simple techniques can be used to convert certain nonclassical initial-boundary value problems modeling a variety of physical phenomena to a more standard form. The reformulation permits the use of basic numerical techniques, or enables the user to take advantage of the availability of software packages or to make more efficient use of such packages. An approach based on Keller's box scheme is ideally suited for the solution of several of the problems described in §§ 2 and 3, because of the nature of the partial differential equations introduced in their reformulation. Its efficacy is demonstrated on one particular problem for which existing numerical methods are exceedingly involved.

TABLE 8

Case 5. Relative errors at $x = 0.25$, $t = 0.1$ for (a) box scheme; (b) box scheme with one time extrapolation.

h / k		.01	.005	.0025	.001
.005	(a)	.717 (−3)	.186 (−3)	.534 (−4)	.163 (−4)
	(b)		.905 (−5)	.919 (−5)	.923 (−5)
.0025	(a)	.711 (−3)	.179 (−3)	.465 (−4)	.939 (−5)
	(b)		.219 (−5)	.230 (−5)	.231 (−6)
.001	(a)	.709 (−3)	.177 (−3)	.446 (−4)	.745 (−5)
	(b)		.264 (−6)	.364 (−6)	.369 (−6)

Note added in proof. Problems of the form (1.6) are also discussed by Schryer (1990). A new finite-difference scheme for the solution of the initial-boundary value problem (2.1.1)-(2.1.4) is presented by Cannon, Lin, and Wang (1990). Ekolin (1989) considers three finite-difference methods for solving the initial-boundary value problem consisting of (2.1.1), (2.1.2), and nonlocal conditions of the form (2.2.3) and (2.2.4).

REFERENCES

S. S. ANTMAN (1980), *The equations for large vibrations of strings*, Amer. Math. Monthly, 87, pp. 359-370.

M. BERZINS (1988), private communication.

M. BERZINS, P. M. DEW, AND R. M. FURZELAND (1984), *Software for time-dependent problems*, in PDE Software: Modules, Interfaces and Systems, B. Enquist and T. Smedsaas, eds., IFIP, North-Holland, Amsterdam, pp. 309-324.

——— (1988), *Developing software for time dependent problems using the method of lines and differential algebraic integrators*, Appl. Numer. Math., to appear.

C. DE BOOR (1978), *A Practical Guide to Splines*, Applied Mathematical Sciences 27, Springer-Verlag, New York.

T. A. BRONIKOWSKI, J. E. HALL, AND J. A. NOHEL (1972), *Quantitative estimates for a nonlinear system of integrodifferential equations arising in reactor dynamics*, SIAM J. Math. Anal., 3, pp. 567-588.

J. R. CANNON (1963), *The solution of the heat equation subject to the specification of mass*, Quart. Appl. Math., 21, pp. 155-160.

J. R. CANNON AND J. VAN DER HOEK (1982), *An implicit finite difference scheme for the diffusion of mass in a portion of the domain*, in Numerical Solutions of Partial Differential Equations, J. Noye, ed., North-Holland, Amsterdam, pp. 527-539.

——— (1986), *Diffusion subject to the specification of mass*, J. Math. Anal. Appl., 115, pp. 517-529.

J. R. CANNON, S. P. ESTEVA, AND J. VAN DER HOEK (1987), *A Galerkin procedure for the diffusion equation subject to the specification of mass*, SIAM J. Numer. Anal., 24, pp. 499-515.

J. R. CANNON, Y. LIN, AND S. WANG (1990), *An implicit finite difference scheme for the diffusion equation subject to mass specification*, Int. J. Engrg. Sci., 28, pp. 573-578.

I. CHRISTIE AND J. M. SANZ-SERNA (1984), *A Galerkin method for a nonlinear integro-differential wave system*, Comput. Methods Appl. Mech. Engrg., 44, pp. 229-237.

——— (1985), *Finite elements for nonlinear integro-differential equations and their integration in time*, in The Mathematics of Finite Elements and Applications V, J. R. Whiteman, ed., Academic Press, London, pp. 415-420.

W. A. DAY (1982), *Extensions of a property of the heat equation to linear thermoelasticity and other theories*, Quart. Appl. Math., 40, pp. 319-330.

——— (1983), *A decreasing property of solutions of parabolic equations with applications to thermoelasticity*, Quart. Appl. Math., 40, pp. 468-475.

J. C. DIAZ, G. FAIRWEATHER, AND P. KEAST (1983a), FORTRAN *packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Software, 9, pp. 358-375.

——— (1983b) *Algorithm* 603 COLROW *and* ARCECO: FORTRAN *packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination*, ACM Trans. Math. Software, 9, pp. 376-380.

R. W. DICKEY (1969), *Infinite systems of nonlinear oscillation equations related to the string*, Proc. Amer. Math. Soc., 23, pp. 459-468.

G. DI BLASIO (1983), *Asymptotic behavior of an age-structured fish population*, Comput. Math. Appl., 9, pp. 377-381. (Also in Witten (1983).)

G. EKOLIN (1989), *Finite difference methods for a nonlocal boundary value problem for the heat equation*, Tech. Report 1989-26, Department of Mathematics, Chalmers Institute of Technology and University of Göteborg, Göteborg, Sweden.

R. H. ELDERKIN (1983), *Population dynamics with globally age-dependent dynamics: On computing the steady state*, Comput. Math. Appl., 9, pp. 371-376. (Also in Witten (1983).)

G. FAIRWEATHER AND R. D. SAYLOR (1989), *The reformulation and numerical solution of certain non-classical initial-boundary value problems*, Tech. Report CCS-89-3, Center for Computational Sciences, University of Kentucky, Lexington, KY.

T. GEVECI AND I. CHRISTIE (1989), *The convergence of a Galerkin approximation scheme for an extensible beam*, RAIRO Numer. Anal. Math. Modelling, 23, pp. 597-613.

M. E. GURTIN AND R. C. MACCAMY (1974), *Nonlinear age dependent population dynamics*, Arch. Rational Mech. Anal., 54, pp. 281-300.

I. GYÖRI (1988), *The method of lines for the solutions of some nonlinear partial differential equations*, Comput. Math. Appl., 15, pp. 635-658.

N. I. IONKIN (1977), *Solution of a boundary-value problem in heat conduction with a nonclassical boundary condition*, Differentsial'nye Uravneniya, 13, pp. 294-304. (In Russian.) Differential Equations, 13, pp. 204-211. (In English.)

J. A. JACQUEZ (1972), *Compartmental Analysis in Biology and Medicine*, Elsevier, Amsterdam.

L. I. KAMYNIN (1964), *A boundary value problem in the theory of heat conduction with a nonclassical boundary condition*, Zh. Vychisl. Mat. i Mat. Fiz., 4, pp. 1006-1024. (In Russian.) U.S.S.R. Comput. Math. and Math. Phys., 4(6), pp. 33-59. (In English.)

H. B. KELLER (1971), *A new difference scheme for parabolic problems*, in Numerical Solution of Partial Differential Equations—II, B. Hubbard, ed., Academic Press, New York, pp. 327-350.

T. V. KOSTOVA (1988), *Numerical solutions of a hyperbolic differential-integral equation*, Comput. Math. Appl., 15, pp. 427-436.

Y. LIN (1988), *Parabolic partial differential equations subject to non-local boundary conditions*, Ph.D. thesis, Washington State University, Pullman, WA.

P. MARCATI (1983), *Some considerations on the mathematical approach to nonlinear age dependent population dynamics*, Comput. Math. Appl., 9, pp. 361-370. (Also in Witten (1983).)

K. REKTORYS (1982), *The Method of Discretization in Time and Partial Differential Equations*, D. Reidel, Dordrecht, Holland.

A. A. SAMARSKII (1980), *Some problems in differential equation theory*, Differentsial'nye Uravneniya, 16, pp. 1925-1935. (In Russian.) Differential Equations, 16, pp. 1221-1228. (In English.)

N. SAUER (1988), private communication.

N. L. SCHRYER (1977), *Numerical solution of time-varying partial differential equations in one space variable*, Tech. Report No. 53, Computing Science, AT&T Bell Laboratories, Murray Hill, NJ.

——— (1981), *Numerical solution of coupled systems of partial differential equations in one space variable and time*, in Elliptic Problem Solvers, M.H. Schultz, ed., Academic Press, New York, pp. 413-417.

——— (1984), *Partial differential equations in one space variable*, Tech. Report No. 115, Computing Science, AT&T Bell Laboratories, Murray Hill, NJ.

——— (1990), *Designing software for one-dimensional partial differential equations*, ACM Trans. Math. Software, 16, pp. 72-85.

J. M. VARAH (1976), *Alternate row and column elimination for solving certain linear systems*, SIAM J. Numer. Anal., 13, pp. 71-75.

M. WITTEN, ED. (1983), *Hyperbolic Partial Differential Equations, Populations, Reactors, Tides and Waves: Theory and Application*, Pergamon Press, New York. (Originally published as a special issue of Comput. Math. Appl., 9 (1983), pp. 323-566.)

S. WOINOWSKY-KRIEGER (1950), *The effect of an axial force on the vibration of hinged bars*, J. Appl. Mech., 17, pp. 35-36.

N. YOSHIDA (1985), *Forced oscillations of extensible beams*, SIAM J. Math. Anal., 16, pp. 211-220.

N. I. YURCHUK (1986), *Mixed problem with an integral condition for certain parabolic equations*, Differentsial'nye Uravneniya, 22, pp. 2117-2126. (In Russian.) Differential Equations, 22, pp. 1457-1463. (In English.)

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 145–157, January 1991
© 1991 Society for Industrial and Applied Mathematics
008

PROGRAMMING ABSTRACTIONS FOR DYNAMICALLY PARTITIONING AND COORDINATING LOCALIZED SCIENTIFIC CALCULATIONS RUNNING ON MULTIPROCESSORS*

SCOTT B. BADEN†

Abstract. Certain software abstractions help to automate load balancing during various math-physics calculations on a team of concurrently executing processors. These abstractions have been tested on a vortex method for computational fluid dynamics. Experiments exhibited good parallel speedups of 24 and 3.6, respectively, on 32 processors of the Intel iPSC-1—a message-passing hypercube architecture—and on 4 processors of a Cray X-MP—a shared-memory vector architecture. The abstractions should apply to diverse applications, including finite difference methods, and to diverse architectures without requiring that the application be reprogrammed extensively for each new architecture.

Key words. multiprocessors, dynamic load balancing, parallel numerical computation, software portability, programming methodology, abstractions, software tools, scientific computing

AMS(MOS) subject classifications. 68U99, 65W05, 68N99, 35-04

1. Introduction.

Many hope that one day multiprocessors will be used routinely to tackle various difficult problems that arise out of science and engineering. An outstanding dilemma remains, however, which could prevent multiprocessors from assuming their expected role: how to write robust software that can run efficiently on diverse systems without having to endure major changes in programming. Time-dependent calculations that apply computational effort nonuniformly over space are particularly troublesome, since they must be dynamically partitioned. This difficulty is further compounded by the fact that multiprocessors continue to proliferate in diverse varieties, and there is no consensus on how to construct them, let alone how to program them.

An important class of scientific calculations operate on a mesh. Furthermore, they are spatially localized; they spend most of their time computing influences among nearby meshboxes and employ relatively inexpensive calculations to couple the distant boxes. This paper presents programming abstractions for dynamically partitioning and coordinating localized, mesh-based numerical computations running on a team of concurrently executing processors. Our abstractions allow the programmer to parallelize software uniformly on both local memory and shared memory multiple-instruction-multiple data (MIMD)-type architectures. However, they do not apply to nonlocalized calculations, which are outside the scope of this paper, nor do they apply to single-instruction-multiple data (SIMD)-type multiprocessors. We have installed the abstractions on two multiprocessor architectures (the Intel iPSC-1 and the Cray X-MP) that represent two opposite ends of the design spectrum, and have tested them out on a full-scale computational fluid dynamics code for computing incompressible, inviscid flow. Good parallel speedups were achieved on both target architectures. Though we

* Received by the editors March 23, 1988; accepted for publication (in revised form) October 11, 1989. This work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract DE-AC03-76SF00098, and in part by California Fellowship in Microelectronics. Additional support was provided by Cray Research, Inc., Intel Scientific Computers, the National Science Foundation, and the Defense Advanced Research Projects Agency.

† Mathematics Department, Lawrence Berkeley Laboratory, University of California, Berkeley, California 94720. Present address, Computer Science and Engineering Department, University of California at San Diego, C014, La Jolla, California 92093-0001.

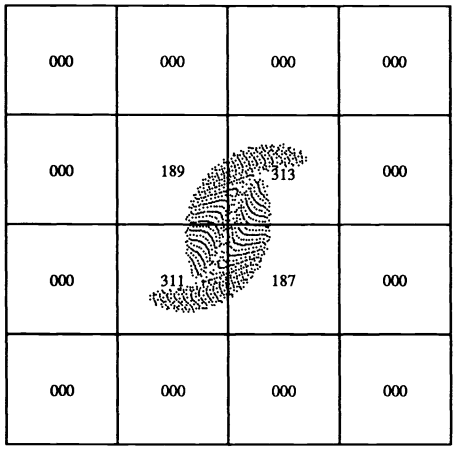
experimented with a single application, we believe that our results are reasonable, since the application we chose is typical of a wide range of computationally intensive numerical applications. Furthermore, our test code was substantial, and not merely a collection of code fragments. We invite others to try out our abstractions on their own application code; the software is in the public domain, written in standard FORTRAN 77, and is available from the author.

2. The problem. Consider the solution of a time-dependent N-body problem with a particle-particle particle-mesh type method [13]. Such a calculation arises in computational fluid dynamics, plasma physics, and particle physics; it follows a set of particles that move under mutual interaction, which congregate and disperse unpredictably with time. The particles move under the influence of a logarithmic potential that we may compute in two parts: a local part and a global part. The local part couples closely spaced particles while the global part mitigates N-body interactions in the far-field. When the problem is large, the local part does upwards of 90 percent of the computational work. The cost of the local part corresponds roughly to the local density of particles, which is a function of position and time. We ignore the global part, since our abstractions do not apply to it. We lay a mesh on top of the computational domain. Each meshbox represents a bucket containing all particles that lie in the region covered by the box. The number of particles in a box varies over the mesh and so the total amount of work required to update the state of a box will vary from box to box.

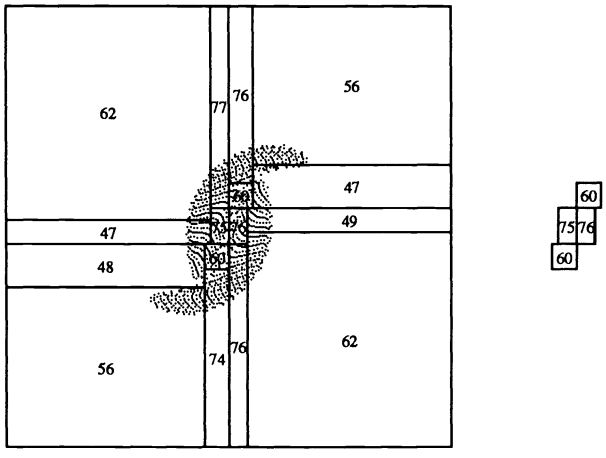
A simple way to apply a team of concurrently executing processors to our computation is to partition the mesh into rectangular regions and assign the computation and data associated with each region to a processor. There are many ways to partition the domain, two of which are shown in Fig. 1. A uniform partitioning, in which all partitions have the same area, is the most straightforward. Such a partitioning, however, utilizes only a small fraction of the total power of the processing team because work is distributed nonuniformly in space. A better way is to partition adaptively into somewhat irregularly sized regions that all complete in roughly the same time. Such a strategy compensates for the uneven distribution of work and can substantially accelerate the computation. Of course, the partitioning must be periodically recomputed, as shown in Fig. 2, or else the workloads would gradually drift out of balance as the particles redistribute themselves. Some processors would become overloaded, while others would sit idle waiting; the cost of the computation would steadily increase with time, as shown in Fig. 3.

The decision to change the work assignments dynamically, rather than to assign work statically, can substantially complicate the user's software. The trouble is that the best way to handle the communication and the bookkeeping that come as a side effect of shuffling work among the processors generally depends on various system design parameters such as communication (or memory) latency, number of processors, processor interconnection structure, and in particular, shared or nonshared memory. Thus, the code required to effectively parallelize the calculation on a shared-memory multiprocessor differs substantially from that required to run on a message-passing architecture, and code could vary even among members of one family or architecture. One way to help the programmer construct robust software is to provide a uniform interface to task partitioning, coordination, and communication activities that is insensitive to the kind of architecture in use. We propose such an interface in the next section.

3. A generic multiprocessor. GenMP stands for *generic multiprocessor.* This generic multiprocessor provides high-level operations that administer to a parallel computation. GenMP may be implemented by a layer of software and can be thought of as a virtual



T = 12.5 [Eff = 0.200]



T = 12.5 [Eff = 0.809]

FIG. 1. *Uniform (top) vs. adaptive (bottom) partitioning of a vortex dynamics calculation into sixteen subdomains. The density of the particles, which are shown as dots, is a rough measure of work. At the depicted time each subdomain's share of the workload is shown in the subdomain assigned to it, normalized to* 1,000 *units of total work. (The central region of the bottom figure is also shown without the particles superimposed.) If loads were perfectly balanced, then each subdomain would be assigned* 62.5 *units of work. The efficiency figures measure how well workloads have been balanced, and ignore any overhead costs that would be incurred in an implementation on a multiprocessor. Some of the particles have been omitted to clarify the plot.*

machine. The virtual machine provides a set of software utilities that the user invokes from application code.

GenMP applies to localized mesh-based computations. Formally, such a calculation entails updating a uniform d-dimensional *work mesh* through a sequence of states L^k, where L^0 is the initial state. In the discussions that follow we will assume that $d = 2$; however, the ideas readily generalize to an arbitrary number of dimensions. Let $L^k(i, j)$ denote the i, jth box of the mesh in the kth state of the sequence, where i, j is in the index set of L^k. Given a *local interaction distance* C, we update the $L^k(i, j)$ to the new state $L^{k+1}(i, j)$. The new state of a box depends only on its previous state and that of a small square neighborhood of boxes whose indices differ from (i, j) by no more than C. Informally, the computation makes Jacobi-style updates to the mesh,

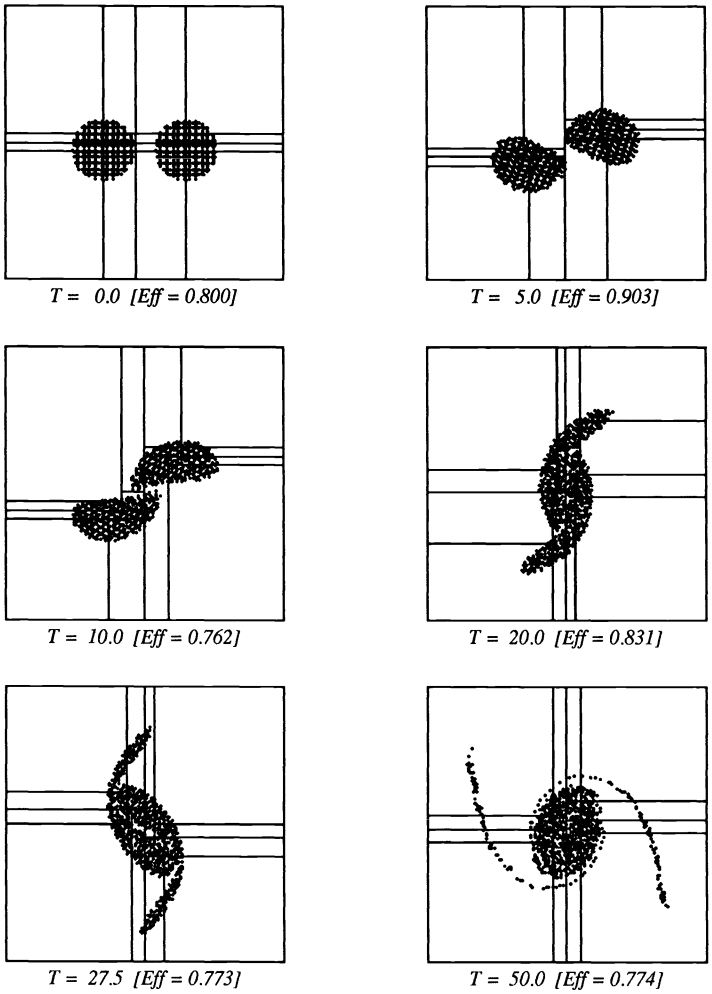


FIG. 2. *Dynamic partitioning of a vortex dynamics calculation into sixteen subdomains. Some of the particles have been omitted to clarify the plot.*

rather than (in-place) Gauss-Seidel ones. We may update the $L^k(i,j)$ in any desired order, for the outcomes that result from employing different orderings will always agree to within roundoff. The cost of updating each box depends on the state of the surrounding boxes and can be reasonably estimated with an inexpensive auxiliary computation. We call this cost function the *work estimate mapping.*

In addition to PPPM-type methods, the localized mesh model of computation assumed by genMP is a reasonable one for a variety of scientific computations; for example:

- Finite difference calculations that use a fixed rectangular mesh trivially fit the model, as do methods such as Adaptive Mesh Refinement (AMR) [4] that employ dynamic grids.
- Finite element methods may also be mapped onto a mesh [19], and divide naturally into localized and nonlocalized computation.
- Ray tracing for computer graphics may also be organized around a mesh, and has a localized communication structure (see Dippé and Swensen [11]).

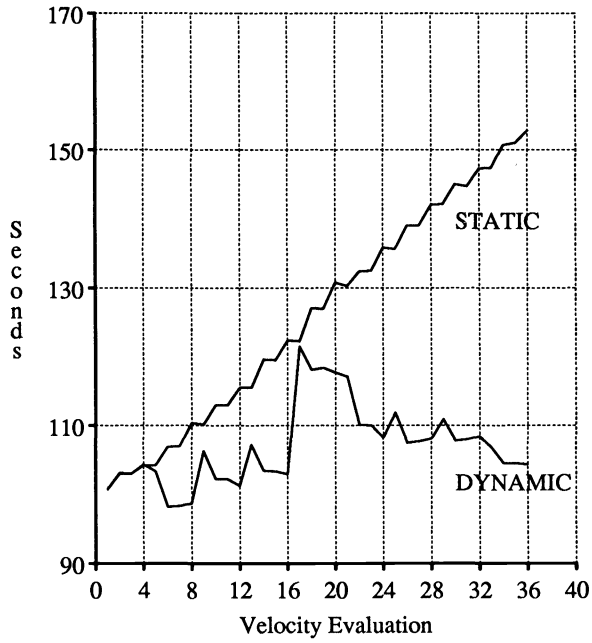


FIG. 3. *A comparison of static and dynamic load balancing of a vortex dynamics computation on thirty-two processors of the Intel* iPSC-1. *If the workloads are partitioned only at the beginning of the calculation (static load balancing, top curve), then the loads drift gradually out of balance, and the time required to perform a velocity evaluation steadly increases with time. In contrast, a dynamic load balancing strategy (bottom curve) periodically rebalances the workloads and can maintain an almost steady running time; here, loads were rebalanced every fourth velocity evaluation.*

Note that the workmesh may represent something more elaborate than a finite difference mesh. In our example, for instance, each box of the workmesh represents a list of particles. Thus, if (i, j) is in the index set of L (where we have dropped the superscript on L), then $L(i, j)$ is the set of variables (or by abuse of notation, their values) at a given box of the mesh.

To parallelize a localized mesh computation we will invoke genMP's utilities. First we use a *partitioner* utility to split the workmesh into disjoint subregions. The utility requires a nonzero integer P and the work estimate mapping computed from the initial state of the calculation. The partitioner returns a table describing P subregions $L_0, \cdots, L_{P-1}$. We next invoke an *initiation* utility to transmit the initial state across P private address spaces according to the table of subregions. We then update the subproblems in any convenient order. We are free to execute each subproblem on a separate concurrently executing processor, or we may assign more than one subproblem to a processor and interleave the execution of those subproblems. The method we choose is unimportant and to simplify the following discussions we assume that each subregion executes on its own processor.

The user computes the work estimate mapping by examining the state of the workmesh. We can think of the mapping as an integer array. In the case of a PPPM method, for example, element (i, j) of the work estimate is computed as the product of two quantities: the number of particles in box $L(i, j)$ and the number of particles in surrounding boxes. The indices of these surrounding boxes differ from i and j by no more than the interaction distance C. In most cases the computation of the work estimate mapping need not be expensive nor too involved. In cases where the cost of

computing the mapping is high, the user must trade off the cost of balancing workloads against the cost of processor idleness due to load imbalance by employing a lower-cost rough work estimate mapping. We assume that the work estimate mapping is roughly set-additive, i.e., the total amount of work done in the partitioned problem will differ only slightly from the work done on the unpartitioned problem. If the mapping is not set-additive, then the partitioner may not necessarily produce good partitionings. (Certain domain decomposition methods for elliptic partial differential equations, for example, fall into this category; see Keyes and Gropp [17].)

In partitioning the workmesh across private address spaces, we have introduced an internal boundary condition requirement. Each processor must independently satisfy the following condition prior to updating its assigned subproblem or results will be nondeterministic. For a local interaction distance C, each processor must obtain copies of all workmesh boxes lying a rectangular shell that is C boxes thick and that surrounds the subregion assigned to the processor. This surrounding region is referred to as the *external interaction region* (see Fig. 4), and it maps onto the address spaces of processors that contain locally interacting subregions of the work mesh. We may also think of the internal boundary condition requirement as a localized form of barrier synchronization. With traditional barrier synchronization, each processor must wait at a barrier until all others have signaled their readiness by reaching the barrier, too. With localized barrier synchronization, in contrast, a processor only waits on those supplying it with internal boundary conditions. Furthermore, the wait signals have a nontrivial content, namely, internal boundary conditions. More appropriately, we should think of wait signals as messages.

GenMP provides a utility called `1Bar` to handle local barrier synchronization. This utility requires three arguments: a nonzero integer C and two routines `xRdy` and

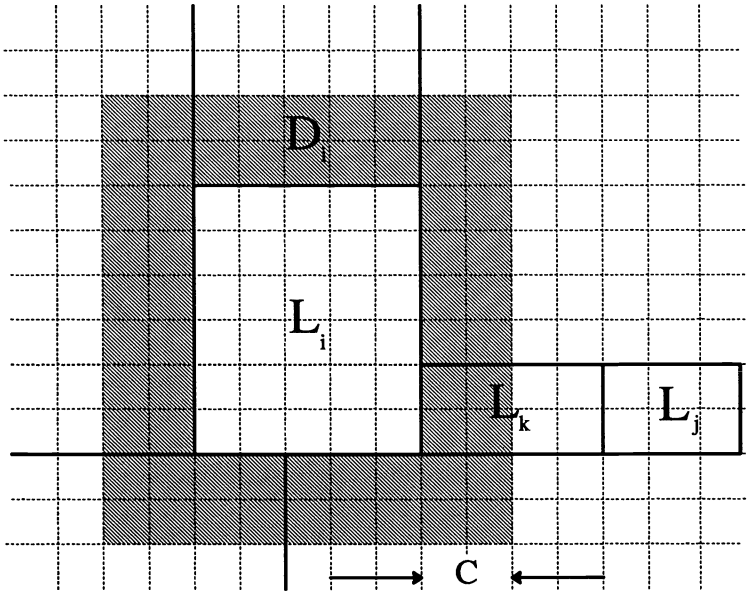


FIG. 4. *Processor i is assigned L_i, a subregion of the work mesh, and an external interaction region D_i, which we have shaded. D_i is a surrounding shell of bins and is C bins thick, where we have chosen $C = 2$. This region divides into six subregions, one for each interacting processor. Subproblems L_i and L_k are locally dependent because D_i and L_k intersect. In comparison, subproblems L_i and L_j are locally independent because D_i and L_j do not intersect.*

rRdy. This call specifies a local barrier synchronization point with an associated external interaction region that is $|C|$ bins thick. All processors will call 1Bar. Each does so at a different time, depending on how much work has been assigned to it. For each processor 1Bar will determine the set of locally interacting processors, divide the external interaction region into rectangular subregions (one for each interacting processor), set up the communication links between the processors, and transmit the signals over the links. The user is responsible for generating and handling the contents of the signals in a way that is meaningful for his application. To this end he writes the two routines xRdy and rRdy, which produce and consume the signal, respectively. These routines are similar to familiar gather and scatter operations used on vector architectures, and should not be too involved. 1Bar will call these routines in matched pairs, one call to each routine for each subregion of the external interaction region, and provide various arguments needed to handle the signals. The xRdy routine enumerates internal boundary values from a region of the work mesh specified by 1Bar and appends the values to the end of a signal whose descriptor is also provided by 1Bar. The rRdy routine reads successive values from an incoming signal passed by 1Bar and incorporates them into the user's data structures. GenMP provides the necessary operations to manipulate signals; we can think of them as sequential I/O operations. Thus the user is responsible for handling all application-dependent parts of local barrier synchronization in order to interpret the contents of the signals; while 1Bar is left to handle all the machine-dependent parts in order to transmit the signals. Note that data transmitted from one processor to another by means of 1Bar can be combined in an arbitrary, application-determined way with data in the receiving box. In cases where several processors transmit data to the same workmesh box, the order of receipt is not defined. The user must insure that his algorithm is insensitive to the ordering of data within a box.

Earlier we mentioned that the thickness of the external interaction region was a signed quantity. The sign of the thickness parameter determines whether the external interaction region acts as a source (parameter is negative) or sink (parameter is positive). Thus, when handling internal boundary conditions, in which the external interaction region acts as a buffer for storing copies of data, the thickness parameter should be positive. Conversely, when the thickness parameter is negative, 1Bar copies data out of the external interaction region into the memories of other processors. We can use this mode of 1Bar to handle the situation where a particle, for example, has migrated beyond an internal boundary into the external interaction region. This particle properly belongs to another processor and therefore must be copied into that processor's memory. (The xRdy routine is responsible for deleting each such particle.) We may also use 1Bar with a negative thickness parameter to assist in load balancing.

Because the work estimate mapping is a state-dependent quantity, we must periodically shift work from the more heavily loaded processors to the more lightly loaded ones (see Fig. 5). Under genMP there are three steps to balancing the workloads. In the first step each processor computes the work estimate mapping for its assigned subproblem. The second step calls the partitioner to recompute the work assignments based on the new work estimate. The partitioner does not actually shift the workloads, however, but only recomputes the region assigned to each processor. (The processor may determine the bounds of this region with functions provided by genMP.) Therefore we must shuffle any data finding itself under the aegis of a new processor to the correct place. The final step of load balancing calls on 1Bar to shift the data according to the new work assignments determined by the partitioner. The thickness parameter is a negative number, and the partitioner provides the appropriate value.

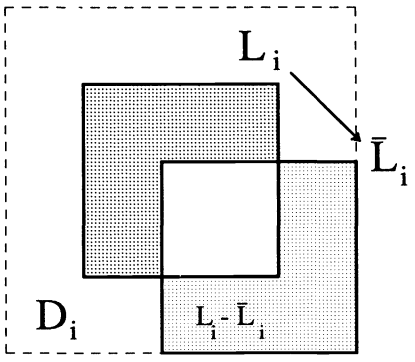


FIG. 5. *The local barrier utility will be used to handle load shifting. Processor i's assigned subproblem changes from L_i to $\bar{L}_i$ as the result of repartitioning. The processor must give up the work contained in the region $L_i - \bar{L}_i$ to other processors.*

Generally, changes to the work estimate mapping are gradual enough that each processor can be identified with a volume of space that changes size and translates by no more than small amounts over time. As a result, workloads will shift incrementally among the processors. The optimal repartitioning rate depends on the application and on various machine-dependent parameters, and must be determined through experimentation. If we rebalance too frequently, then the overhead may be too great. If we rebalance too infrequently, however, then the workload assignments will change abruptly between repartitioning phases, and the external interaction region must then be large enough to accommodate those changes. In the worst case the external interaction region could cover the entire domain, and there may not be enough memory to accommodate such a large region.

4. Evaluation. Traditional MIMD multiprocessors differ primarily according to how they handle communication, and come in two varieties: message-passing or shared-memory architectures.[1] We installed genMP on one architecture of each kind: the Intel Personal Scientific Computer iPSC-1 [15], manufactured by Intel Scientific Computers, and the Cray X-MP [9], manufactured by Cray Research Inc. The iPSC and the Cray differ in three major ways:

(1) Each iPSC-1 processor has a small local memory (0.5 megabyte), and communicates by passing messages, whereas the Cray's processors communicate through a large shared memory (up to 128 megabytes).

(2) The iPSC can support up to 128 processors, the Cray at most 4.

(3) Each iPSC CPU executes about 35 thousand floating point operations per second. Individually the Cray's CPUs are quite powerful: they provide fast vector mode arithmetic and each is capable of computing at a sustained rate of 100 megaflops or more.

As a test problem, we used a vortex blob method [7] to solve the *vorticity-stream function formulation* of Euler's equations for incompressible flow in two dimensions in an infinite domain:

$$D\omega / Dt = 0, \tag{4.1}$$

$$\omega = -\Delta\psi, \tag{4.2}$$

$$\mathbf{u} = 0 \quad \text{at} \quad \mathbf{x} = \infty, \tag{4.3}$$

[1] The IBM RP3 [20] is a notable exception. It can run as a shared-memory architecture, as a message-passing architecture, or in a hybrid mode.

where $\mathbf{u}(\mathbf{x}(t), t)$ is the velocity of the fluid at position $\mathbf{x}(t)$ at time t; ω is the vorticity, defined as $\omega = \partial u/\partial y - \partial v/\partial x$; ψ is the stream function, defined such that $\mathbf{u} = (\partial\psi/\partial y, -\partial\psi/\partial x)$; $D/Dt = \partial/\partial t + \mathbf{u} \cdot \nabla$ is the material derivative; and $\Delta = \partial^2/\partial x^2 + \partial^2/\partial y^2$ is the two-dimensional Laplacian operator. The above equations were solved for an initial vorticity distribution that was constant inside two disks of radius 0.12 centered about the origin, and zero elsewhere. The centers of the disks were separated by 0.25.

We discretized the above equations by placing N marker particles, called vortices, on a regular mesh of points confined to the places where the vorticity was nonzero. The paths of the vortices were then computed at a sequence of timesteps, according to the following system of ordinary differential equations:

$$\frac{d}{dt}\mathbf{x}_i(t) = \mathbf{u}_i(t), \qquad i = 1, \cdots, N. \tag{4.4}$$

Here $\mathbf{x}_i(t)$ is the position of the ith vortex at time t, $\mathbf{u}_i(t)$ the velocity, and ω_i is its strength, which is like a charge. The positions of the vortices were evolved by discretizing (4.4) in time with a second-order Runge–Kutta time integration scheme. Anderson's method of local corrections [1] was used to compute the mutually induced velocities on the RHS of (4.4) and was invoked twice per timestep. When there are thousands of vortices, this algorithm typically spends less than 5 percent in a global computation (a Poisson solver) and most of the remaining time in local computation. See Baden [3] for additional details concerning the local corrections algorithm.

Experiments were run with various numbers of vortices N and processors P. Owing to the differences in processor speed, numbers of processors, and memory capacity, the values of N on the Cray were different from those used on the iPSC. On the iPSC, the running program consumed nearly all of a processor's local memory (only about 300 kilobytes of a processor's 512 kilobytes of memory are directly available to the user); roughly 180 kilobytes contained data, and the remaining 120 kilobytes contained mostly code. On the Cray the running program consumed 3.6 to 4.1 megawords (depending on the number of processors used), of which only about 43 kilowords was devoted to code. The iPSC runs as a stand-alone system. The Cray X-MP normally runs as a batch system, but all measurements were obtained during blocks of dedicated time, since only then can running times be repeatedly measured with reasonable accuracy. Double precision arithmetic was used on the iPSC, single precision on the Cray. The iPSC runs were made on a model d5 with 32 processors. The software was written in FORTRAN 77, compiled with FTN286, and run under release 2.1 of the node operating system. The Cray runs were made on a model 416 with 4 processors and 16 megawords of main memory (1 word = 64 bits). The code was written in Cray X-MP FORTRAN [8], [10], compiled with version 1.14 of CFT, and run under version 1.16 of COS, the Cray Operating System.

To evaluate performance we measured the parallel efficiency, η_P, which is defined as

$$\eta_P = \frac{T_1/P}{T_P}, \tag{4.5}$$

where T_P is the time to complete on P processors. For the special case of one processor, we define the completion time to be that measured in a *uniprocessor* implementation. Thus, T_1 ignores any overhead costs that would be incurred in a *multiprocessor* implementation running on a single processor. On a local memory architecture such as the iPSC, memory capacity scales with the number of processors. Thus, a single processor lacks sufficient memory to run the larger problems. We cannot measure T_1

directly. Instead, we extrapolate T_1 in the following way. We recall from the previous section that the work estimate mapping was assumed to be set-additive. Thus, the partitioner roughly conserves the total amount of localized work that would be done in a uniprocessor computation, and that work can be reasonably approximated by summing the work (the completion times) done by all P processors. The remaining part of T_1, corresponding to the global work done by the Poisson solver, can be measured directly, since the solver has modest memory requirements. We then add this to the timing for the localized work to get the extrapolated value for T_1. Upon substituting this extrapolated value into (4.5), we obtain a *scaled* efficiency measure. We note that our extrapolation procedure ignores certain overheads that would be observed if we could measure T_1 directly, and corresponds more closely to what would be measured in an implementation optimized for this special case of one processor. However, we estimate that the overheads are so slight that they can be ignored (similar comments hold for the Cray X-MP measurements).

Tables 1 and 2 give the efficiency and speedup measurements obtained from iPSC runs and Cray runs, respectively. Efficiency was quite good on both machines. On the iPSC, η_P ranged from 90 percent with 4 processors to 74 percent with 32 processors. The efficiency on the Cray was about 90 percent on 4 processors. Thus, if efficiency were somehow increased to 100 percent, then the computations would run at most 35 percent ($(1-\eta_P^{-1})\times 100\%$) faster on the iPSC and 12 percent faster on the Cray. Note that our scheme for parallelizing the code had no effect on the CFT compiler's ability

TABLE 1

iPSC *results, where the number of vortices N varies linearly with the number of processors P.* T_1^{avg} *is the execution time for a single timestep averaged over the full run. The scaled parallel efficiency* η_P *decreases with P primarily due to the effect of increased load imbalance. By definition* $\eta_P = S_P/P$ *where* S_P *is the scaled speedup.* Local(%) *and* Global(%) *give the percentages of time spent, respectively, in local computation and in the global computation done by a Poisson solver, ignoring all overhead costs. Overhead costs are reported separately as* Lbar(%) , *the fraction of the total time spent in local barrier synchronization, and* Part(%), *the fraction spent partitioning, including the cost of producing the work estimate mapping. All runs lasted* 64 *timesteps, two velocity evaluations were done per timestep, and loads were rebalanced every other timestep.*

N	P	T_1^{avg} (minutes)	η_P (%)	S_P	Local (%)	Global (%)	Lbar (%)	Part (%)
386	4	1.3	90	3.6	96	16	0.3	1.2
796	8	1.6	85	6.8	91	10	0.8	1.2
1586	16	2.2	79	13	85	6.8	1.4	1.0
3180	32	3.5	74	24	80	4.7	1.6	0.8

TABLE 2

Timings, parallel efficiency, and speedup for the Cray X-MP *runs. The runs with* 12,848 *vortices ran for* 400 *timesteps, the larger runs for* 240 *timesteps. Loads were balanced every timestep.*

N	P	Time (sec)	S_P	η_P
12,848	1	5999	1.00	1.000
12,848	2	3081	1.95	0.973
12,848	4	1651	3.63	0.908
25,702	1	7032	1.00	1.000
25,702	4	1970	3.57	0.892

to vectorize the code, and vector lengths were not reduced. On 4 processors the Cray executed our code at a rate of 250 megaflops.

There are three reasons why efficiency fell short of 100 percent: (1) Some computations ran on only one processor and acted as serial bottlenecks; (2) processors spent some of their time communicating; (3) loads were not perfectly balanced. The cost of these effects was smaller on the Cray than on the iPSC because the former has a smaller number of processors. On the iPSC we found that the cost of load imbalance by far dominated the other factors and increased with the number of processors. But the cost was not too severe; if load imbalance were somehow reduced to zero, then the program would run at most 25 percent $((1-\eta_P^{\text{Local}})\times 100\%)$ faster. Two parameters affect load imbalance: workmesh size and partitioning frequency. The load imbalance attains its minimal value when we make the workmesh as fine as memory permits and when we repartition prior to every state change. However, our programs may actually run fastest when the load imbalance is somewhat higher than the minimal value. For example, we found that although the partitioner balanced the workloads more evenly as we shrank the workmesh boxes, eventually a diminishing point of returns was reached where the cost of manipulating still more boxes exceeded any savings due to a reduction in load imbalance. Similarly we did not repartition after every state change, though we did find that if we repartitioned too infrequently, then memory loads could become so imbalanced that a processor ran out of memory to store particles. We determined the optimal workmesh size and paritioning frequency by experimentation.

We also measured the overhead costs of genMP. We estimated the overheads on the Cray because we were unable to make detailed timing measurements of the code at the time our runs were made. Overall, genMP's overhead seems reasonable; it never exceeded 2.4 percent on the iPSC, and we estimate that it never exceeded 5 percent on the Cray. (For the complete analysis on the Cray, see [2].) Our implementation of the partitioner ran on just one processor. However, the partitioner was not a serial bottleneck. The reason is that we employed a recursive bisection algorithm to render the partitions and this algorithm is fast for the relatively small number of partitions rendered in our runs. (See Berger and Bokhari [5] for a discussion of the algorithm.) If we had been running on a machine with hundreds of processors, however, we would probably have had to reimplement the partitioner algorithm to run concurrently. Local barrier synchronization was inexpensive because the internal boundary values were transmitted in bulk rather than an element at a time. This was facilitated by restricting the partitions to having simple shapes. However, because genMP concentrates communication into brief instances of time, the underlying hardware and the operating system must be able to cope with peak loading by carefully controlling the flow of disruptive messages. Otherwise, message flow control must be handled within the `1Bar` utility.

A few implementation details deserve special mention. We refer the reader to our dissertation [2] for a more complete discussion. We made no effort to minimize communication overhead through a clever processor mapping heuristic (cf. Bokhari [6]). There was no need, since the communication overheads incurred during local barrier synchronization were so low. We made a 1 : 1 assignment of regions to processors. We felt that since loads were balanced sufficiently well there was no need of making many-to-one processor assignments in order to reduce load imbalance any further. Because genMP supports a local memory execution model, it may incur high overhead costs if run under an operating system that allows individual processors to be preemptively scheduled. On message passing systems this is not an issue, but on shared-memory systems we must either run stand-alone (as we did), or modify the

operating system's scheduler to preemptively schedule at the job level. Unless all required processors are working on our job, `1Bar` could idle for long periods of time, and the job would in effect execute serially.

5. Discussion. We have outlined a simple approach to parallelizing a class of numerical algorithms that insulates the programmer from many of the machine-dependent and low-level details. Application-dependent code and system-dependent code need not become heavily intertwined; when code is transported to a new machine, the parts that would have to change to accommodate a different communication model are restricted mostly to code the programmer never sees. The techniques were applied to a realistic application, and good parallel speedups were obtained on architectures that represent two extremes in multiprocessor design philosophy. Our approach is to provide the user with a uniform interface to processors' decomposition and communication activities that is insensitive to machine-dependent characteristics, e.g., whether or not memory is shared. This interface comes in the form of genMP. GenMP partitions work into subdomains and helps enforce correctness by synchronizing the concurrently executing processors at local barriers. The user must divide the data and computation for the local part of the problem into boxes of a regular rectangular mesh, must supply work estimates for the computation in each box, and must supply routines for converting these data to and from byte streams. GenMP will assign boxes to processors in order to even the workload and will allow each processor to access and communicate the necessary boundary data.

GenMP employs coarse-grained partitionings that preserve the locality inherent in a computation, and thus avoids the higher overheads inherent in strategies that employ fine-grained partitionings, such as processor self-scheduling [21]. In many localized computations the solution changes slowly enough that loads shift gradually among processors according to local changes in the density of computational effort, and the imbalance of loading builds up gradually enough that intermittent, rather than continual, rebalancing works well enough. GenMP can then partition work semistatically at convenient stopping points in the computation, rather than continually interleaving work-redistribution with computation. This is desirable since the amount of shared information that would otherwise be manipulated expensively during computation is reduced.

GenMP is intended for calculations that fit a simple model of spatial locality that applies to many important scientific and engineering calculations. It is neither universal nor complete, however. GenMP does not apply to SIMD-type systems, though it should apply to massively parallel MIMD-type systems as long as the calculation is sufficiently large. It applies only to the localized parts of a computation, and the user must parallelize any nonlocalized parts separately. We believe that the general question of how to produce a portable implementation of a nonlocalized computation is an open research topic. Some success has been met in dealing with certain kinds of applications (e.g., LINPACK-style calculations [14], [18]). Also, processor self-scheduling has been employed on various shared-memory architectures. See, for example, Dongarra and Sorenson's *Schedule* [12] and Jordan's *Force* [16].

Acknowledgment. The research described here was part of the author's Ph.D. dissertation research done in the Computer Science Division at the University of California at Berkeley. I gratefully acknowledge the encouragement and moral support of my thesis advisor, W. Kahan; Phillip Colella also helped to supervise the work. Thanks also go to Luigi Semenzato and David Scott for their advice on how to improve this paper.

REFERENCES

[1] C. R. ANDERSON, *A method of local corrections for computing the velocity field due to a distribution of vortex blobs*, J. Comput. Phys., 62 (1986), pp. 111–123.

[2] S. B. BADEN, *Run-time partitioning of scientific continuum calculations running on multiprocessors*, Tech. Report LBL-23625, Lawrence Berkeley Laboratory, Berkeley, CA, June 1987; Ph.D. thesis, Computer Science Division, University of California, Berkeley, CA.

[3] ———, *Very large vortex calculations in two dimensions*, in Lecture Notes in Mathematics, C. Anderson and C. Greengard, eds., Springer-Verlag, Berlin, New York, 1988; also in Proc. UCLA Workshop on Vortex Methods, University of California, Los Angeles, CA, May 20–22, 1987.

[4] M. J. BERGER AND J. OLIGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 53 (1984), pp. 484–512.

[5] M. J. BERGER AND S. BOKHARI, *A partitioning strategy for non-uniform problems on multiprocessors*, IEEE Trans. Comput., C-36 (1987), pp. 570–580.

[6] S. BOKHARI, *On the mapping problem*, IEEE Trans. Comput., C-30 (1981), pp. 207–214.

[7] A. J. CHORIN, *Numerical study of slightly viscous flow*, J. Fluid Mech., 57 (1973), pp. 785–796.

[8] Cray-1 FORTRAN (CFT) Reference Manual (2240009), Cray Research, Inc., Mendota Heights, MN, 1978.

[9] Cray X-MP Hardware Reference Manual, Cray Research, Inc., Mendota Heights, MN, 1986; order number: HR-0097.

[10] Cray X-MP Multitasking Programmer's Manual, Cray Research, Inc., Mendota Heights, MN, March 1986; order number: SN-0222.

[11] M. E. DIPPÉ AND J. A. SWENSEN, *An adaptive subdivision algorithm and parallel architecture for realistic image synthesis*, SIGGRAPH '84 Conference Proc., Minneapolis, MN, July 1984, pp. 149–158.

[12] J. J. DONGARRA AND D. C. SORENSEN, *A portable environment for developing parallel* FORTRAN *programs*, Parallel Comput., 5 (1987), pp. 175–186.

[13] J. W. EASTWOOD, R. W. HOCKNEY, AND D. N. LAWRENCE, P3M3DP—*The Three Dimensional Periodic Particle-Particle/Particle-Mesh Program*, Comput. Phys. Comm., 19 (1980), pp. 215–261.

[14] G. C. FOX AND W. FURMANSKI, *Communication algorithms for regular convolutions and matrix problems on the hypercube*, C3P-329, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, CA, August 1986.

[15] iPSC User's Guide, Intel Corporation, Beaverton, OR, October 1985; order number: 175455-003.

[16] H. F. JORDAN, M. S. BENTEN, AND N. S. ARENSTORF, Force user's manual, Department of Electrical and Computer Engineering, University of Colorado, Boulder, CO, October 1986.

[17] D. E. KEYES AND W. D. GROPP, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, YALEU/DCS/RR-448, Department of Computer Science, Yale University, New Haven, CT, December 1985.

[18] C. MOLER AND D. S. SCOTT, *Communication utilities for the* iPSC, Tech. Report, Intel Scientific Computers, Portland, OR, August 1986.

[19] B. NOUR-OMID, A. RAEFSKY, AND G. LYZENGA, *Solving finite element equations on concurrent computers*, in Proc. American Society of Mechanical Engineers Symposium on Parallel Computations and Their Impact on Mechanics, University Park, PA, December 13–18, 1987.

[20] G. F. PFISTER, W. C. BRANTLEY, D. A. GEORGE, S. L. HARVEY, W. J. KLEINFELDER, K. P. MCAULIFFE, E. A. MELTON, V. A. NORTON, AND J. WEISS, *The IBM* RP3 *introduction and architecture*, in Proc. 1985 Internat. Conference on Parallel Processing, August 1985, pp. 764–771.

[21] P. TANG AND P. YEW, *Processor self-scheduling for multiple-nested parallel loops*, in Proc. 1986 Internat. Conference on Parallel Processing, University Park, PA, August 1986, pp. 528–535.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 158–179, January 1991
© 1991 Society for Industrial and Applied Mathematics
009

A FAST ALGORITHM FOR THE EVALUATION OF LEGENDRE EXPANSIONS*

BRADLEY K. ALPERT† AND VLADIMIR ROKHLIN‡

Abstract. An algorithm is presented for the rapid calculation of the values and coefficients of finite Legendre series. Given an n-term Legendre expansion, the algorithm produces its values at n Chebyshev nodes on the interval [-1,1] for a cost proportional to $n \log n$. Similarly, given the values of a function f at n Chebyshev nodes, the algorithm produces the n-term Legendre expansion of the polynomial of degree $n-1$ that is equal to f at these nodes. The cost of the algorithm is roughly three times that of the Fast Fourier Transform of length n, provided that calculations are performed to single precision accuracy. In double precision, the ratio is approximately 5.5.

The method employed admits far-reaching generalizations and is currently being applied to several other problems.

Key words. fast transform, Legendre series, Chebyshev series, approximation theory

AMS(MOS) subject classifications. 41A58, 65D15

1. Introduction.

Legendre polynomials are widely used in applied mathematics. Their applications include quadratures, approximation theory, solution of partial differential equations, analysis of pseudospectral methods, and several other areas. However, attempts to use Legendre polynomials as a numerical tool (as opposed to an analytical apparatus) tend to meet with a serious difficulty: given a function $f : [-1,1] \to \mathcal{R}$ tabulated at n nodes, it takes order $O(n^2)$ operations to obtain the Legendre expansion of f. Similarly, given an n-term Legendre expansion, at takes order $O(n^2)$ operations to evaluate that expansion at n nodes in $\mathcal{R}$. In other words, unlike the Chebyshev expansion or the Fourier series, the Legendre series does not have a fast transform associated with it. Whenever possible, therefore, the Legendre series is avoided in favor of an expansion for which a fast transform exists. In some cases, this substitution causes relatively little difficulty (for example, in the construction of pseudospectral algorithms for the solution of partial differential equations). In other cases, it cannot be done at all (for example, in the solution of partial differential equations by the separation of variables in the spherically symmetric geometry, where the choice of Legendre polynomials as the set of basis functions is dictated by the mathematics of the problem).

In Orszag [10], a method is proposed for the rapid evaluation of a fairly wide class of eigenfunction transforms. The algorithm is based on the combination of certain analytical considerations with the Fast Fourier Transform (FFT), has the asymptotic CPU time estimate of order $n \log^2 n / \log \log n$, and becomes faster than the direct (order n^2) algorithm at $n \approx 128$ (in the case of the Legendre series).

In this paper, we present a procedure for the rapid evaluation of a Legendre expansion at Chebyshev nodes on the interval $[-1,1]$, and conversely, for the evaluation of the coefficients of a Legendre expansion from a table of its values at Chebyshev nodes. More specifically, given a function f expressed as Legendre expansion of the

*Received by the editors July 7, 1989; accepted for publication September 25, 1989. This research, performed at Yale University, was partially supported by Office of Naval Research grant N00014-86-0310 and IBM grant P00038437.

†Department of Mathematics and Lawrence Berkeley Laboratory, University of California, Berkeley, California 94720 (alpert@csr.lbl.gov).

‡Departments of Mathematics and Computer Science, Yale University, Box 2158 Yale Station, New Haven, Connecticut 06520 (rokhlin@cs.yale.edu).

form

$$f(t) = \sum_{j=0}^{n-1} \alpha_j \cdot P_j(t), \tag{1.1}$$

the algorithm evaluates f at the n Chebyshev nodes $t_0, t_1, \ldots, t_{n-1}$ on the interval $[-1, 1]$ in order $O(n \log n)$ operations. Similarly, given the values of a function $f : [-1, 1] \to \mathcal{R}$ tabulated at the nodes $t_0, t_1, \ldots, t_{n-1}$, the algorithm requires order $O(n \log n)$ operations to evaluate the coefficients $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ such that

$$f(t_i) = \sum_{j=0}^{n-1} \alpha_j \cdot P_j(t_i) \quad \text{for } i = 0, 1, \ldots, n-1. \tag{1.2}$$

The algorithm we present is based on replacing the Legendre expansion of the form (1.1) with a Chebyshev expansion of the same length, with subsequent evaluation of the latter via the fast cosine transform (see, e.g., [5, Chap. 10]). It turns out that the reduction of a Legendre expansion to a Chebyshev expansion can be performed in order $O(n)$ operations, and it is well known that the cosine transform of length n requires order $O(n \log n)$ operations. Thus, the resulting CPU time estimate of our algorithm is $O(n \log n)$.

Remark 1.1. While the asymptotic CPU time estimate of our scheme is dominated by that of the fast cosine transform it employs, in most practical situations ($n \leq 20,000$), the conversion of the Legendre series of length n into a Chebyshev series tends to be roughly twice as expensive as one fast Fourier transform of length n, provided the calculations are performed in single precision arithmetic. As a result, evaluating the series of the form (1.1) at n Chebyshev nodes by means of our algorithm is approximately three times as expensive as a single FFT of length n. In double precision arithmetic, this ratio is roughly 5.5 (see §6 below).

In the following section, we summarize several well-known facts from approximation theory to be used in the subsequent sections. In §3, the analytical properties of the linear mappings connecting the coefficients of Legendre and Chebyshev expansions are studied. Sections 4 and 5 contain the description of the algorithm and its complexity analysis, and in §6 results of several numerical experiments are presented. Finally, §7 discusses several straightforward generalizations of the algorithm of this paper. An appendix presents details of an efficient scheme for the evaluation of the function $\Gamma(x + \frac{1}{2})/\Gamma(x + 1)$, required by our algorithm.

Remark 1.2. The approach of this paper is closely related to that used in [11] to construct an order $O(n)$ scheme for the evaluation of a polynomial of order n at n arbitrary points in $\mathcal{R}$. Both algorithms can be viewed as particular implementations of a scheme for the rapid application to arbitrary vectors of matrices whose entries are sufficiently smooth functions of their indices. Such a general procedure is discussed in §7, and will be reported in detail at a later date.

2. Mathematical and numerical preliminaries.

2.1. Miscellaneous facts from approximation theory. In this section we summarize several well-known facts, the first being the classical error bound for Chebyshev approximations (see, e.g., [6]).

LEMMA 2.1. *Suppose $k \geq 2$, and let t_i denote the ith Chebyshev node of order k on $[0,1]$ and $u_i(t)$ the ith Lagrange polynomial associated with the t_i's,*

$$t_i = \frac{1}{2}\left(1 + \cos\left(\frac{(2i+1)\pi}{2k}\right)\right), \qquad u_i(t) = \prod_{j=0,\, j\neq i}^{k-1} \frac{t - t_j}{t_i - t_j}, \tag{2.1}$$

for $i = 0, 1, \ldots, k-1$. Suppose further that $f : [a,b] \to \mathcal{R}$ is a function with k continuous derivatives, and that the error $E(f,k,[a,b];t)$ of the k-node Chebyshev expansion for f at a point $t \in [a,b]$ is defined by the formula

$$E(f,k,[a,b];t) = f(t) - \sum_{i=0}^{k-1} u_i\left(\frac{t-a}{b-a}\right) \cdot f(a + (b-a)t_i). \tag{2.2}$$

Then for any $t \in [a,b]$,

$$|E(f,k,[a,b];t)| \leq \frac{2(b-a)^k}{4^k k!} \sup_{v\in[a,b]} |f^{(k)}(v)|. \tag{2.3}$$

Throughout the paper we will retain the notation of t_i for Chebyshev nodes, $u_i(t)$ for the corresponding Lagrange polynomials, and $E(f,k,[a,b];t)$ for the interpolation error.

Next we provide a bound on the derivatives of an analytic function, which follows directly from the Cauchy integral formula.

LEMMA 2.2. *Suppose that $D \subset \mathcal{C}$ is a closed disk of radius r centered at $z \in \mathcal{C}$, and that $f : D \to \mathcal{C}$ is a function continuous on D and analytic on its interior $D \backslash \partial D$. Then*

$$|f^{(k)}(z)| \leq \frac{k!}{r^k} \sup_{\theta\in[0,2\pi]} |f(z + re^{i\theta})|. \tag{2.4}$$

The next lemma provides an expression for the logarithm of the gamma function (see, e.g., [9, p. 10]) which we use in Lemma 2.4 below.

LEMMA 2.3 (BINET). *For any $z \in \mathcal{C}$ such that $\mathrm{Re}(z) > 0$,*

$$\ln\Gamma(z) = \left(z - \frac{1}{2}\right)\ln(z) - z + \frac{\ln(2\pi)}{2} + I(z), \tag{2.5}$$

where

$$I(z) = \int_0^\infty \left(\frac{1}{e^t - 1} - \frac{1}{t} + \frac{1}{2}\right)\frac{1}{t} e^{-tz} dt. \tag{2.6}$$

Furthermore,

$$|I(z)| \leq \frac{1}{|6z|}. \tag{2.7}$$

We define a function $\Lambda : \mathcal{C} \to \mathcal{C}$, by the formula

$$\Lambda(z) = \frac{\Gamma(z + \frac{1}{2})}{\Gamma(z+1)}. \tag{2.8}$$

The function Λ will often appear in the remainder of the paper. The following lemma states specific bounds on $|\Lambda(z)|$ that we will need in §3.

LEMMA 2.4. *Suppose that* $z \in \mathcal{C}$ *and* $\mathrm{Re}(z) \geq 0$. *Then*

$$\frac{e^{-1/2}}{\sqrt{|z+1|}} \leq |\Lambda(z)| \tag{2.9}$$

and

$$|\Lambda(z)| \leq \frac{e}{\sqrt{|z+1|}}. \tag{2.10}$$

Proof. Combining Binet's expression (2.5) for $\ln\Gamma$ with the definition (2.8) of Λ, we obtain

$$\ln\Lambda(z) = F(z) + \frac{1}{2}(1 - \ln(z+1)) + Q(z), \tag{2.11}$$

with the function $F : \mathcal{C} \to \mathcal{C}$ defined by the formula

$$F(z) = z \ln\left(\frac{z+\frac{1}{2}}{z+1}\right), \tag{2.12}$$

and the function $Q : \mathcal{C} \to \mathcal{C}$ defined by the formula

$$Q(z) = I\left(z+\frac{1}{2}\right) - I(z+1), \tag{2.13}$$

where $I(z)$ is given by (2.6). Combining the estimate (2.7) for $|I(z)|$ with (2.13), we see that

$$|\mathrm{Re}(Q(z))| \leq \frac{1}{2}, \tag{2.14}$$

for any z with $\mathrm{Re}(z) \geq 0$, and simple analytical manipulations show that

$$-\frac{1}{2} \leq \mathrm{Re}(F(z)) \leq 0 \tag{2.15}$$

for any z with $\mathrm{Re}(z) \geq 0$. Now, combining (2.11) with estimates (2.14) and (2.15), we obtain

$$-\frac{1}{2} \leq \mathrm{Re}(\ln\Lambda(z)) + \frac{1}{2}\ln|z+1| \leq 1,$$

from which (2.9) and (2.10) follow immediately. □

2.2. Definition of Legendre and Chebyshev polynomials. An orthogonal family of polynomials $\varphi_0, \varphi_1, \varphi_2, \ldots$, is a set of polynomials of degrees $0, 1, 2, \ldots$, in which the inner product (φ_i, φ_j) is zero if $i \neq j$ and positive if $i = j$. The inner product is defined by the formula

$$(f, g) = \int_a^b f(t)\, g(t)\, w(t)\, dt,$$

where $[a, b] \subset \mathcal{R}$ and the weight function w is continuous and nonnegative on $[a, b]$.

The Legendre and Chebyshev polynomials each form an orthogonal family of polynomials. In each case the inner-product integral is taken over the interval $[-1, 1]$; for the Legendre polynomials, the weight function is $w(t) = 1$, while for the Chebyshev polynomials, $w(t) = 1/\sqrt{1-t^2}$.

Any orthogonal polynomial family satisfies a three-term recurrence relation (for $n \geq 0$) of the form

$$\varphi_{-1}(t) = 0, \qquad \varphi_0(t) = A_0,$$

$$\frac{A_n}{A_{n+1}}\varphi_{n+1}(t) = \left[t - \frac{(t\varphi_n, \varphi_n)}{(\varphi_n, \varphi_n)}\right]\varphi_n(t) - \frac{(\varphi_n, t\varphi_{n-1})}{(\varphi_{n-1}, \varphi_{n-1})}\varphi_{n-1}(t), \tag{2.16}$$

where $A_n \neq 0$ is the leading coefficient of $\varphi_n(t)$ and can be chosen arbitrarily. For the Legendre polynomials $P_0, P_1, P_2, \ldots$, equation (2.16) takes the form

$$P_0(t) = 1, \qquad P_1(t) = t,$$

$$P_{n+1}(t) = \frac{2n+1}{n+1}t\,P_n(t) - \frac{n}{n+1}P_{n-1}(t) \qquad (n \geq 1). \tag{2.17}$$

For the Chebyshev polynomials $T_0, T_1, T_2, \ldots$, equation (2.16) becomes

$$T_0(t) = 1, \qquad T_1(t) = t,$$

$$T_{n+1}(t) = 2t\,T_n(t) - T_{n-1}(t) \qquad (n \geq 1).$$

In addition to the above recurrences which define P_n and T_n $(n = 0, 1, 2, \ldots)$, equivalent closed-form expressions are available. The Legendre polynomials can be given by the equation

$$P_n(t) = \frac{1}{2^n n!}\frac{d^n}{dt^n}(t^2 - 1)^n \qquad (n \geq 0)$$

and the Chebyshev polynomials by the equation

$$T_n(t) = \cos(n \arccos t) \qquad (n \geq 0).$$

Everything in this section can be found in standard texts (see, e.g., [6, §4.4]).

2.3. Connection between Legendre and Chebyshev expansions. We will denote by M^n, L^n a pair of $n \times n$-matrices defined by the formulae

$$M^n_{ij} = \begin{cases} \dfrac{1}{\pi}\Lambda\left(\dfrac{j}{2}\right)^2 & \text{if } 0 = i \leq j < n \text{ and } j \text{ is even,} \\ \dfrac{2}{\pi}\Lambda\left(\dfrac{j-i}{2}\right)\Lambda\left(\dfrac{j+i}{2}\right) & \text{if } 0 < i \leq j < n \text{ and } i+j \text{ is even,} \\ 0 & \text{otherwise,} \end{cases} \tag{2.18}$$

$$L^n_{ij} = \begin{cases} 1 & \text{if } i = j = 0, \\ \dfrac{\sqrt{\pi}}{2\Lambda(i)} & \text{if } 0 < i = j < n, \\ \dfrac{-j(i+1/2)}{(j+i+1)(j-i)}\Lambda\left(\dfrac{j-i-2}{2}\right)\Lambda\left(\dfrac{j+i-1}{2}\right) & \text{if } 0 \leq i < j < n, \\ & \quad i+j \text{ even,} \\ 0 & \text{otherwise,} \end{cases} \tag{2.19}$$

with Λ defined by (2.8).

Remark 2.5. While (2.18) and (2.19) define M_{ij}^n and L_{ij}^n for integer values of i, j, it is apparent from (2.8) that M^n and L^n can be naturally viewed as functions on $\mathcal{C} \times \mathcal{C}$, and we define $\mathcal{M}, \mathcal{L} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ by the formulae

$$\mathcal{M}(x,y) = \frac{2}{\pi}\Lambda\left(\frac{y-x}{2}\right)\Lambda\left(\frac{y+x}{2}\right), \tag{2.20}$$

$$\mathcal{L}(x,y) = \frac{-y(x+\frac{1}{2})}{(y+x+1)(y-x)}\Lambda\left(\frac{y-x-2}{2}\right)\Lambda\left(\frac{y+x-1}{2}\right). \tag{2.21}$$

Clearly, $M_{ij}^n = \mathcal{M}(i,j)$ if $0 < i \le j < n$ and $i+j$ is even. Similarly, $L_{ij}^n = \mathcal{L}(i,j)$ if $0 \le i < j < n$ and $i+j$ is even. It easily follows from the well-known properties of the Γ-function (see, e.g., [1]) that $\mathcal{M}$, $\mathcal{L}$ are meromorphic functions of each of their arguments with the poles of $\mathcal{M}$ located at the points $y = \pm x - 1, \pm x - 3, \pm x - 5, \ldots$ and the poles of $\mathcal{L}$ at the points $y = x+1, -x, x-1, -x-2, \ldots$.

The matrices M^n, L^n are inverses of each other; their definition is motivated by the following well-known fact (see, e.g., [7, §8.91-2]).

LEMMA 2.6. *Suppose that the function* $f : [-1,1] \to \mathcal{R}$ *has a finite Legendre expansion of the form*

$$f(\cos\theta) = \sum_{i=0}^{n-1} \alpha_i \cdot P_i(\cos\theta). \tag{2.22}$$

Then it also has a finite Chebyshev expansion of the form

$$f(\cos\theta) = \sum_{i=0}^{n-1} \beta_i \cdot T_i(\cos\theta), \tag{2.23}$$

where $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$ *and* $\vec{\beta} = \langle \beta_0, \ldots, \beta_{n-1} \rangle$ *are related by the equation*

$$\vec{\beta} = M^n \vec{\alpha}. \tag{2.24}$$

Conversely, if f *is a function given by* (2.23), *then it may be expressed in the form of* (2.22), *where* $\vec{\alpha}$ *is given by*

$$\vec{\alpha} = L^n \vec{\beta}. \tag{2.25}$$

3. Analytical properties of the mappings $\mathcal{M}$, $\mathcal{L}$.

DEFINITION 3.1. Suppose that a square $S \subset \mathcal{R} \times \mathcal{R}$ is defined by the formula $S = [x_0, x_0 + a] \times [y_0, y_0 + a]$, where $a > 0$. We will say that S is separated from the diagonal if $y_0 - x_0 \ge 2a$.

The following theorem is the principal analytical tool of this paper. It states that on any square separated from the diagonal, the entries of M^n and L^n are well approximated by Chebyshev expansions of the functions $\mathcal{M}, \mathcal{L}$ with respect to either the first or the second coordinate. For any $\tilde{x} \in \mathcal{C}$ we will define a pair of functions $\mathcal{M}_{\tilde{x}}$ and $\mathcal{L}_{\tilde{x}}$ by the formulae

$$\mathcal{M}_{\tilde{x}}(y) = \mathcal{M}(\tilde{x}, y),$$

$$\mathcal{L}_{\tilde{x}}(y) = \frac{\mathcal{L}(\tilde{x}, y)}{\tilde{x} + \frac{1}{2}}$$

for all $y \in \mathcal{C}$. Similarly, for any $\tilde{y} \in \mathcal{C}$ we will define the functions $\mathcal{M}^{\tilde{y}}$ and $\mathcal{L}^{\tilde{y}}$ by the formulae

$$\mathcal{M}^{\tilde{y}}(x) = \mathcal{M}(x, \tilde{y}),$$

$$\mathcal{L}^{\tilde{y}}(x) = \frac{\mathcal{L}(x, \tilde{y})}{x + \frac{1}{2}}$$

for all $x \in \mathcal{C}$.

THEOREM 3.2. *Suppose the square* $S = [x_0, x_0 + a] \times [y_0, y_0 + a]$, $a \geq 2$, *is separated from the diagonal and* $(x, y) \in S$. *Then*

$$|E(\mathcal{M}^y, k, [x_0, x_0 + a]; x)| \leq \frac{8e^3}{3^k} |\mathcal{M}(x, y)| \tag{3.1}$$

and

$$|E(\mathcal{M}_x, k, [y_0, y_0 + a]; y)| \leq \frac{8e^3}{3^k} |\mathcal{M}(x, y)|. \tag{3.2}$$

Similarly,

$$|E(\mathcal{L}^y, k, [x_0, x_0 + a]; x)| \leq \frac{128e^3}{3^k} \left| \frac{\mathcal{L}(x, y)}{x + \frac{1}{2}} \right| \tag{3.3}$$

and

$$|E(\mathcal{L}_x, k, [y_0, y_0 + a]; y)| \leq \frac{224e^3}{3^k} \left| \frac{\mathcal{L}(x, y)}{x + \frac{1}{2}} \right|, \tag{3.4}$$

where E *is the error for Chebyshev interpolation, as defined in* (2.2).

Proof. We will prove here only the estimate (3.1), since the proofs of all four statements (3.1)–(3.4) are quite similar. In order to prove (3.1), we will prove two inequalities:

$$|E(\mathcal{M}^y, k, [x_0, x_0 + a]; x)| \leq \frac{32e^2/\pi}{3^k \sqrt{(y - x + 2)(y + x + 2)}}, \tag{3.5}$$

$$|\mathcal{M}(x, y)| \geq \frac{4e^{-1}/\pi}{\sqrt{(y - x + 2)(y + x + 2)}}, \tag{3.6}$$

for all $(x, y) \in S$, and observe that estimate (3.1) is an immediate consequence of (3.5) and (3.6).

(a) *Proof of inequality* (3.5). In order to apply the error estimate (2.3) of Lemma 2.1 to the function $\mathcal{M}^y$, we will need a bound on the derivatives of $\mathcal{M}^y$. First, we establish a bound on $\mathcal{M}^y(x + \frac{3}{4}(y - x)e^{i\theta})$ for any $(x, y) \in S$ and $\theta \in [0, 2\pi]$. It

immediately follows from the definition (2.20) of $\mathcal{M}$ combined with estimate (2.10) that

$$\begin{aligned}\left|\mathcal{M}^y\left(x+\frac{3}{4}(y-x)e^{i\theta}\right)\right| &= \left|\mathcal{M}\left(x+\frac{3}{4}(y-x)e^{i\theta},y\right)\right| \\ &= \frac{2}{\pi}\left|\Lambda\left(\frac{y-x}{2}-\frac{3}{8}(y-x)e^{i\theta}\right)\right|\cdot\left|\Lambda\left(\frac{y+x}{2}+\frac{3}{8}(y-x)e^{i\theta}\right)\right| \\ &\le \frac{2}{\pi}\frac{e^2}{\sqrt{[(y-x)/8+1]\cdot[(y+7x)/8+1]}} \qquad (3.7)\end{aligned}$$

for any $\theta\in[0,2\pi]$. Combining estimate (3.7) with the Cauchy integral estimate (2.4) and noting that $y-x\ge a$, we obtain

$$\left|\frac{\partial^k\mathcal{M}^y}{\partial x^k}(x)\right| \le \frac{k!}{(3a/4)^k}\frac{16e^2/\pi}{\sqrt{(y-x+2)(x+y+2)}}. \qquad (3.8)$$

Now (3.5) follows from a combination of (3.8) with estimate (2.3).

(b) *Proof of inequality* (3.6). The lower bound on $\mathcal{M}$ is easily established by combining the definition (2.20) of $\mathcal{M}$ with estimate (2.9), which yields

$$|\mathcal{M}(x,y)| = \frac{2}{\pi}\left|\Lambda\left(\frac{y-x}{2}\right)\right|\cdot\left|\Lambda\left(\frac{y+x}{2}\right)\right| \ge \frac{2}{\pi}\frac{e^{-1}}{\sqrt{[(y-x)/2+1]\cdot[(y+x)/2+1]}},$$

for any $(x,y)\in S$, and inequality (3.6) immediately follows. □

Remark 3.3. Estimates (3.1)–(3.4) in Theorem 3.2 are quite pessimistic. Numerical experiments indicate that the errors in those estimates all decay approximately as 5^{-k}, as opposed to 3^{-k}. In fact, eight-point Chebyshev expansions will approximate $\mathcal{M}$ and $\mathcal{L}$ with roughly single precision accuracy (7 digits) on any square separated from the diagonal. Similarly, double precision (16 digits) is achieved with 18-point expansions. For our purposes, however, the estimates of Theorem 3.2 are adequate.

4. Informal description of the algorithm. We now define a concept closely analogous to the separation of a square from the diagonal.

DEFINITION 4.1. Suppose A is an upper-triangular $n\times n$-matrix with entries $\{A_{ij}\}$, $i,j=0,1,\ldots,n-1$. Suppose further that T is an $m\times m$-submatrix of A defined by the formula

$$T_{ij}=A_{p+i,q+j}$$

with p,q two nonnegative integers. We will say that the submatrix T of matrix A is separated from the diagonal if

$$q-p\ge 2m.$$

4.1. A simple example. Suppose that T is an $m\times m$-submatrix of matrix M^n, and that T is separated from the diagonal. We present a simple example of how the smoothness of M^n enables us to efficiently compute $\vec{w}=T\vec{v}$, where $\vec{v}=\langle v_0,\ldots,v_{m-1}\rangle$ is an arbitrary vector of length m. To compute

$$w_i=\sum_{j=0}^{m-1}\mathcal{M}(i_0+i,j_0+j)\cdot v_j \quad \text{for } i=0,\ldots,m-1, \qquad (4.1)$$

we may approximate $\mathcal{M}(i_0+i, j_0+j)$ by its Chebyshev expansion in j. We have

$$\mathcal{M}(i_0+i, j_0+j) \approx \sum_{r=0}^{k-1} \mathcal{M}(i_0+i, j_0+t_r m) \cdot u_r\left(\frac{j}{m}\right), \tag{4.2}$$

where we know from Theorem 3.2 that as k grows, the error of this approximation shrinks as 3^{-k}. Substituting (4.2) into (4.1) and changing the order of summation, we obtain

$$w_i \approx \sum_{r=0}^{k-1} \mathcal{M}(i_0+i, j_0+t_r m) \sum_{j=0}^{m-1} u_r\left(\frac{j}{m}\right) \cdot v_j = \sum_{r=0}^{k-1} \mathcal{M}(i_0+i, j_0+t_r m) \cdot b_r,$$

where $b_r = \sum_{j=0}^{m-1} u_r(j/m) \cdot v_j$ $(r = 0, 1, \ldots, k-1)$.

The number of operations required to evaluate $\vec{w}$ in this manner is $O(km)$. Indeed, evaluating $b_0, \ldots, b_{k-1}$ requires order $O(km)$ operations (k coefficients at m operations per coefficient). Evaluating the vector $\vec{w}$ given the coefficients $b_0, \ldots, b_{k-1}$ is also an order $O(km)$ procedure (evaluating a k-term expansion at m nodes). For a fixed precision ϵ, the number k of Chebyshev nodes required is $\log_3(1/\epsilon)$, and is independent of m. Thus the cost of the evaluation of $\vec{w} = T\vec{v}$ has been reduced from order $O(m^2)$ to $O(m \log(1/\epsilon))$.

This example represents only a part of the computation required to apply matrix M^n to an arbitrary vector, since the submatrix T is assumed to be separated from the diagonal. The actual algorithm is constructed by extending the above example in order to apply the entire matrix. The matrix M^n can be divided into square submatrices, each of which is separated from the diagonal (Fig. 4.1). To apply the matrix M^n to the vector $\vec{\alpha}$, each submatrix T is applied separately, as suggested in the example. The remaining undivided portion of M^n near the diagonal is applied directly. Before we can introduce the algorithm, however, we will need some additional notation.

4.2. Notation. The algorithm to be described will input an arbitrary vector $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$ and compute an approximation $\vec{\gamma} = \langle \gamma_0, \ldots, \gamma_{n-1} \rangle$ to the vector $\vec{\beta} = M^n \vec{\alpha}$. Theorem 3.2 will be used to ensure the desired quality of approximation.

Suppose that s is a positive integer such that $h = \log_2(n/s) - 1$ is also a positive integer. For any integer $l \in \{1, \ldots, h\}$ and $j \in \{0, \ldots, n/(2^{l-1}s) - 1\}$, we define the interval $I_{l,j} \subset \mathcal{R}$ by the formula

$$I_{l,j} = [j \cdot 2^{l-1}s, (j+1) \cdot 2^{l-1}s].$$

For any integer $l \in \{1, \ldots, h\}$, $i \in \{0, \ldots, n/(2^{l-1}s) - 3\}$, and $j \in \{i+2, i+3\}$ (for i even), $j = i+2$ (for i odd) we define the square ${}_lS_{i,j} \subset \mathcal{R}^2$ by the formula

$${}_lS_{i,j} = I_{l,i} \times I_{l,j}.$$

The definition of the squares ${}_lS_{i,j}$ is illustrated in Fig. 4.2.

For $l \in \{1, \ldots, h\}$, $m \in \{0, \ldots, 2^{l-1}s - 1\}$, and $r \in \{0, \ldots, k-1\}$, we define the Chebyshev interpolation coefficient $u^l_{r,m}$ by the formula

$$u^l_{r,m} = u_r\left(\frac{m}{2^{l-1}s}\right), \tag{4.3}$$

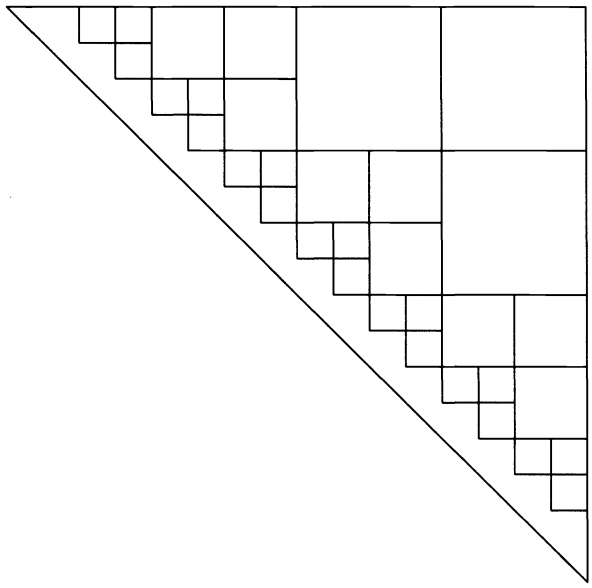


FIG. 4.1. *Each submatrix T in the subdivision of M^n is separated from the diagonal (see Definition 4.1). Here the subdivision is shown to three levels.*

where u_r is given by (2.1). For each interval $I_{l,j}$ and $r \in \{0, \ldots, k-1\}$ we define the coefficient $b_{l,j}^r$ by the formula

$$b_{l,j}^r = \sum_{m=0}^{2^{l-1}s-1} u_{r,m}^l \cdot \alpha_{m+j\cdot 2^{l-1}s} \tag{4.4}$$

and observe that the definition of $b_{l,j}^r$ is analogous to the definition of b_r in the example above. For each square ${}_lS_{i,j}$ we define a $k \times k$-matrix ${}_l\mathcal{M}_{i,j}$ of which each element ${}_l\mathcal{M}_{i,j}^{r,m}$ $(r, m \in \{0, \ldots, k-1\})$ is given by the formula

$${}_l\mathcal{M}_{i,j}^{r,m} = \mathcal{M}((i+t_r)\cdot 2^{l-1}s, (j+t_m)\cdot 2^{l-1}s). \tag{4.5}$$

We further define, for each square ${}_lS_{i,j}$ and $r \in \{0, \ldots, k-1\}$, the coefficient ${}_lc_{i,j}^r$ by the formula

$${}_lc_{i,j}^r = \sum_{m=0}^{k-1} {}_l\mathcal{M}_{i,j}^{r,m} \cdot b_{l,j}^m.$$

For each interval $I_{l,j}$ and $r \in \{0, \ldots, k-1\}$, we define the coefficient $c_{l,j}^r$ by the formulae

$$\begin{array}{lcll} c_{l,2j}^r & = & {}_lc_{2j,2j+2}^r + {}_lc_{2j,2j+3}^r & \\ c_{l,2j+1}^r & = & {}_lc_{2j+1,2j+3}^r & (j \le n/(2^l s) - 2), \\ c_{l,n/(2^{l-1}s)-2}^r & = & c_{l,n/(2^{l-1}s)-1}^r = 0. & \end{array} \tag{4.6}$$

For each interval $I_{l,j}$ and $m \in \{0, \ldots, 2^{l-1}s - 1\}$ we define the coefficient $a_{l,j}^m$ by the

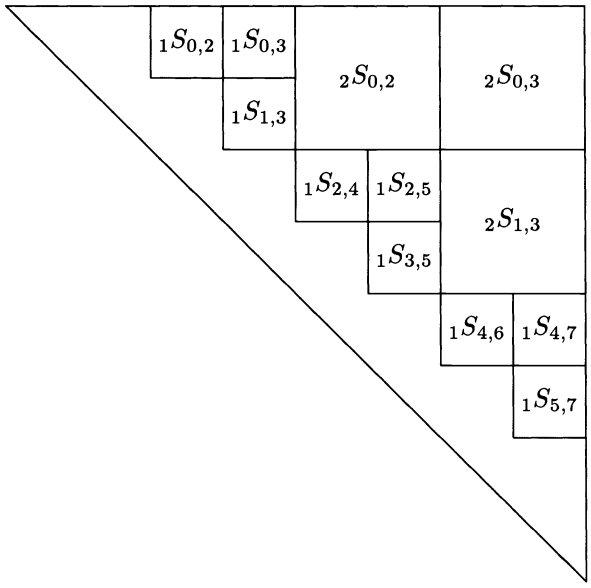


FIG. 4.2. *The upper triangle of the square* $[0,n]\times[0,n]$ *is subdivided into squares, each of which is separated from the diagonal* (*Definition* 3.1). *Here the number* h *of levels is equal to two.*

formula

$$a_{l,j}^m = \sum_{r=0}^{k-1} u_{r,m}^l \cdot c_{l,j}^r.$$

Finally, for $i \in \{0,\dots,n-1\}$, we define γ_i by the formula

$$\gamma_i = \sum_{l=1}^{\lceil \log_2((n-i)/(2s))\rceil} a_{l,j_{l,i}}^{m_{l,i}} + \sum_{r=i}^{\lfloor i/s\rfloor s+2s-1} M_{ir}^n \cdot \alpha_r, \tag{4.7}$$

where $j_{l,i} = \lfloor i/(2^{l-1}s)\rfloor$ and $m_{l,i} = i \bmod (2^{l-1}s)$.

The notation introduced so far allows us to extend the example of §4.1 to apply the entire matrix M^n to an arbitrary vector. The simplest algorithm, which is described in the next section, requires order $O(n\log n)$ operations. We have introduced, however, one change from the example of §4.1: the values of $\mathcal{M}$ are interpolated with respect to both the first and the second coordinates, rather than just the second coordinate. This change allows us to construct an algorithm requiring order $O(n)$ operations. This latter algorithm is described in §4.3, and will require some additional notation.

We note that for any integer $l \in \{2,\dots,h\}$, $m \in \{0,\dots,2^{l-1}s-1\}$, and $r \in \{0,\dots,k-1\}$, the Chebyshev interpolation coefficient $u_{r,m}^l$ defined in (4.3) can be equivalently given by the formula

$$u_{r,m}^l = \sum_{i=0}^{k-1} u_r\left(\frac{t_i}{2}\right) \cdot u_{i,m}^{l-1}. \tag{4.8}$$

Remark 4.2. Equation (4.8) is an instance of the more general formula

$$u_r(t) = \sum_{i=0}^{k-1} u_r\left(\frac{t_i}{2}\right) \cdot u_i(2t) \quad \text{for any } t \in \mathcal{R}, \tag{4.9}$$

which immediately follows from the observation that the summation in (4.9) is the $(k-1)$st-degree interpolating polynomial which agrees with the function u_r at points $t_0/2, t_1/2, \ldots, t_{k-1}/2$, and that u_r itself is a polynomial of degree $k-1$ (see (2.1)).

Equation (4.8) combined with (4.4) produces a recursive expression for $b^r_{l,j}$ given by the formulae

$$b^r_{1,j} = \sum_{i=0}^{s-1} u_r\left(\frac{i}{s}\right) \cdot \alpha_{i+js}, \tag{4.10}$$

$$b^r_{l,j} = \sum_{i=0}^{k-1} \left[u_r\left(\frac{t_i}{2}\right) \cdot b^i_{l-1,2j} + u_r\left(\frac{1+t_i}{2}\right) \cdot b^i_{l-1,2j+1}\right] \qquad (l \in \{2, \ldots, h\}). \tag{4.11}$$

Following a similar procedure, for each $I_{l,j}$ and $r \in \{0, \ldots, k-1\}$, we recursively define the coefficient $d^r_{l,j}$ by the formulae

$$\begin{aligned} d^r_{h,j} &= c^r_{h,j}, \\ d^r_{l,2j} &= c^r_{l,2j} + \sum_{i=0}^{k-1} u_i\left(\frac{t_r}{2}\right) \cdot d^l_{l+1,j} \qquad (l \in \{1, \ldots, h-1\}), \\ d^r_{l,2j+1} &= c^r_{l,2j+1} + \sum_{i=0}^{k-1} u_i\left(\frac{1+t_r}{2}\right) \cdot d^l_{l+1,j} \end{aligned} \tag{4.12}$$

where $c^r_{l,j}$ is given by (4.6). Finally, combining (4.10)–(4.12), we see that the vector $\vec{\gamma}$, defined by (4.7), can be equivalently given by the formula

$$\gamma_{m+js} = \sum_{i=0}^{k-1} u_i\left(\frac{m}{s}\right) \cdot d^i_{1,j} + \sum_{i=m}^{2s-1} M^n_{m+js,i+js} \cdot \alpha_{i+js}, \qquad \begin{matrix} m = 0, \ldots, s-1, \\ j = 0, \ldots, n/s - 1. \end{matrix} \tag{4.13}$$

4.3. Description of an $O(n \log n)$ algorithm. A simple algorithm for the application of the matrix M^n to an arbitrary vector $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$ can be constructed by combining the procedure of §4.1 and the subdivision of M^n shown in Figs. 4.1 and 4.2. The procedure of §4.1 works only for a submatrix which is separated from the diagonal. The matrix M^n can be divided, however, into a collection of submatrices, each of which is separated from the diagonal, plus an undivided part near the diagonal (see Fig. 4.1). By applying the scheme of §4.1 to each submatrix, we immediately obtain an order $O(n \log n)$ algorithm for the application of the matrix M^n (or L^n) to $\vec{\alpha}$.

4.4. Description of an $O(n)$ algorithm. Now we describe an order $O(n)$ algorithm for the application of the matrix M^n (or matrix L^n) to an arbitrary vector. The matrix M^n is divided into submatrices, each of which is separated from the diagonal. The scheme of such a subdivision is shown in Fig. 4.1. There are three squares of side length $n/4$, $3 \cdot 3$ of side length $n/8$, $3 \cdot 7$ of side length $n/16$, and so forth down to

$3 \cdot (2^h - 1)$ squares of side length $s = n/2^{1+h}$. The size $s \times s$ of the smallest squares in the subdivision is fixed, independent of n. For each square the separation condition of Theorem 3.2 holds.

The most direct application of Chebyshev expansions, to each square independently, leads to an order $O(n \log n)$ algorithm. This operation count is due to order $O(n)$ operations for all squares of one size, multiplied by $O(\log n)$ different sizes. The undivided part of the matrix near the diagonal is handled directly in order $O(n)$ operations.

We improve on this simple method by using in each square the Chebyshev expansion for M^n in both row and column directions. We then "gather" coefficients used in each row interval up from those for the next smaller intervals, compute $k \times k$ matrix-vector products, then "spread" the results down to the next smaller column intervals. To describe this procedure, we employ the notation introduced in §4.2. The coefficients $b^r_{l,j}$ are computed from level one to level h according to (4.10) and (4.11). Then the matrix-vector products ${}_lc^r_{i,j}$ are computed and summarized to the values $c^r_{l,j}$ (4.6). These values are used to compute the coefficients $d^r_{l,j}$ from level h to level one, as specified in (4.12). Finally, the vector $\vec{\gamma}$ is computed according to (4.13).

This incremental computation of the coefficients $b^r_{l,j}$ and the coefficients $d^r_{l,j}$ leads to the reduction in complexity from order $O(n \log n)$ to $O(n)$ operations. Now, instead of order $O(n)$ operations for all squares of one size, we expend order $O(n)$ operations for all squares of the smallest size, half as many for the next larger size, one fourth as many for the second larger size, and so forth. The sum of these operation counts remains order $O(n)$.

We divide the computation into an initialization phase, which is independent of $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$, depending only on n, and an evaluation phase, which does the rest of the work. This partitioning of the algorithm leads to substantial savings when one computes many Legendre transformations of the same size.

In the initialization phase, the values of ${}_l\mathcal{M}^{r,m}_{i,j}$ for each square, as defined in (4.5), are computed and stored; the near-diagonal values of $\mathcal{M}$, appearing in (4.13), are also computed. Details of an efficient scheme for the evaluation of $\mathcal{M}$ are contained in the Appendix. The values of u_r appearing in (4.10) and (4.11) are also computed during the initialization phase, and later used in the evaluation phase. The evaluation phase consists of using the stored values of $\mathcal{M}$ and u_r in computing, in succession, $b^r_{l,j}$, $c^r_{l,j}$, $d^r_{l,j}$, and γ_i, as described above.

5. Detailed description and complexity analysis of the algorithm.

5.1. Description of the algorithm.

Initialization Phase
Comment [Input to this phase is the number of values n].
Set the number of Chebyshev nodes per interval $k \approx \log(1/\epsilon)$,
where ϵ is the desired precision. Set the smallest interval
size $s \approx 4k$. (See §5.2, below.)

Step 1.
Comment [Construct Chebyshev nodes $t_0, t_1, \ldots, t_{k-1}$ on the interval $[0, 1]$, Chebyshev nodes $t'_0, \ldots, t'_{k-1}$ on the interval $[0, \frac{1}{2}]$, and Chebyshev nodes $t'_k, \ldots, t'_{2k-1}$ on the interval $[\frac{1}{2}, 1]$.]
do $r = 0, \ldots, k-1$
set $t_r = [1 - \cos((r + .5)\pi/k)]/2$

set $t'_r = t_r/2$ and $t'_{k+r} = (1 + t_r)/2$
enddo

Step 2.

Comment [Evaluate the denominators in the expressions for the Chebyshev interpolation coefficients $u_0, u_1, \ldots, u_{k-1}$.]
do $r = 0, \ldots, k-1$
set $den_r = \prod_{l=0, l\neq r}^{k-1}(t_r - t_l)$
enddo

Comment [Evaluate the Chebyshev interpolation coefficients $u_0, u_1, \ldots, u_{k-1}$ at the uniformly spaced nodes $0, 1/s, 2/s, \ldots, (s-1)/s$.]
do $l = 0, \ldots, s-1$
set $x = \prod_{r=0}^{k-1}(l/s - t_r)$
do $r = 0, \ldots, k-1$
set $u_r(l/s) = [x/(l/s - t_r)]/den_r$
enddo
enddo

Comment [Evaluate the Chebyshev interpolation coefficients $u_0, u_1, \ldots, u_{k-1}$ at the Chebyshev nodes $t'_0, \ldots, t'_{2k-1}$.]
do $l = 0, \ldots, 2k-1$
set $x = \prod_{r=0}^{k-1}(t'_l - t_r)$
do $r = 0, \ldots, k-1$
set $u_r(t'_l) = [x/(t'_l - t_r)]/den_r$
enddo
enddo

Step 3.

Comment [Evaluate the values ${}_l\mathcal{M}_{i,j}^{r,m}$ of the $k \times k$ matrix on each square ${}_lS_{i,j}$ of the subdivision.]
$h = \log_2(n/s) - 1$
do $l = 1, \ldots, h$
do $i = 0, \ldots, n/(2^{l-1}s) - 3$
do $j = i + 2, \ldots, i + 3 - i \bmod 2$
do $r = 0, \ldots, k-1$
do $m = 0, \ldots, k-1$
Calculate ${}_l\mathcal{M}_{i,j}^{r,m}$ according to (4.5).
enddo
enddo
enddo
enddo
enddo

Step 4.

Comment [Evaluate $\mathcal{M}$ in the undivided part of the matrix, near the diagonal.]
do $j = 0, \ldots, n/s - 1$
do $m = 0, \ldots, s-1$
set $p = 2s - 2 + m \bmod 2$
do $i = m, m+2, m+4, \ldots, p$
Calculate $\mathcal{M}(m + js, i + js)$ using (2.20).

enddo
enddo
enddo
End of Initialization Phase
Evaluation Phase
Comment [Input to this phase is $\vec{\alpha} = \langle \alpha_0, \ldots, \alpha_{n-1} \rangle$.]

Step 5.

Comment [Evaluate the coefficients $b^r_{1,j}$ from the input vector $\langle \alpha_0, \ldots, \alpha_{n-1} \rangle$ and the interpolation coefficients $u_r(i/s)$.]
do $j = 0, \ldots, n/s - 1$
  do $r = 0, \ldots, k-1$
    set $b^r_{1,j} = \sum_{i=0}^{s-1} u_r(i/s) \cdot \alpha_{i+js}$
  enddo
enddo

Step 6.

Comment [Evaluate the coefficients $b^r_{l,j}$ for $l \geq 2$, which correspond to larger interval sizes, from the coefficients for smaller interval sizes and the interpolation coefficients $u_r(t'_i)$.]
do $l = 2, \ldots, h$
  do $j = 0, \ldots, n/(2^{l-1}s) - 1$
    do $r = 0, \ldots, k-1$
      set $b^r_{l,j} = \sum_{i=0}^{k-1} \left[u_r(t'_i) \cdot b^i_{l-1,2j} + u_r(t'_{k+i}) \cdot b^i_{l-1,2j+1} \right]$
    enddo
  enddo
enddo

Step 7.

Comment [Evaluate the coefficients $c^r_{l,j}$ from the values ${}_l\mathcal{M}^{r,m}_{i,j}$ and the coefficients $b^r_{l,j}$.]
do $l = 1, \ldots, h$
  do $j = 0, \ldots, n/(2^l s) - 2$
    do $r = 0, \ldots, k-1$
      set $c^r_{l,2j} = \sum_{i=0}^{k-1} \left[{}_l\mathcal{M}^{r,i}_{2j,2j+2} \cdot b^i_{l,2j+2} + {}_l\mathcal{M}^{r,i}_{2j,2j+3} \cdot b^i_{l,2j+3} \right]$
      set $c^r_{l,2j+1} = \sum_{i=0}^{k-1} {}_l\mathcal{M}^{r,i}_{2j+1,2j+3} \cdot b^i_{l,2j+3}$
    enddo
  enddo
enddo

Step 8.

Comment [Evaluate the coefficients $d^r_{l,j}$ from the coefficents $c^r_{l,j}$ and the interpolation coefficients $u_r(t'_i)$.]
do $l = h, h-1, \ldots, 1$
  do $j = 0, \ldots, n/(2^{l-1}s) - 2$
    do $r = 0, \ldots, k-1$
      set $d^r_{l,2j} = c^r_{l,2j} + \sum_{i=0}^{k-1} u_i(t'_r) \cdot d^i_{l+1,j}$
      set $d^r_{l,2j+1} = c^r_{l,2j+1} + \sum_{i=0}^{k-1} u_i(t'_{k+r}) \cdot d^i_{l+1,j}$
    enddo
  enddo
enddo

Step 9.

Comment [Evaluate the final result $\vec{\gamma} = \langle \gamma_0, \ldots, \gamma_{n-1} \rangle$ from the coefficients $d^r_{1,j}$, the interpolation coefficients $u_r(m/s)$, the values of $\mathcal{M}$ near the diagonal, and the input vector $\vec{\alpha}$.]
do $j = 0, \ldots, n/s - 1$
  do $m = 0, \ldots, s - 1$
    set $\gamma_{m+js} = \sum_{i=0}^{k-1} u_i(m/s) \cdot d^i_{1,j}$
    set $p = 2s - 2 + m \bmod 2$
    do $i = m, m + 2, m + 4, \ldots, p$
      set $\gamma_{m+js} = \gamma_{m+js} + \mathcal{M}(m + js, i + js) \cdot \alpha_{i+js}$
    enddo
  enddo
enddo
End of Evaluation Phase
End of Algorithm

The algorithm for the opposite direction, namely, computing Legendre expansion coefficients from Chebyshev expansion coefficients, is identical to the algorithm given above, with $\mathcal{L}$ substituted for $\mathcal{M}$.

Remark 5.1. In Theorem 3.2, error bounds for Chebyshev expansions in a square separated from the diagonal are given for $\mathcal{L}(x, y)/(x + \frac{1}{2})$, rather than for $\mathcal{L}(x, y)$. In principle, then, the algorithm for computing Legendre coefficients from Chebyshev coefficients should apply the matrix corresponding to $\mathcal{L}(x, y)/(x+\frac{1}{2})$ using the method given above and then apply the diagonal matrix corresponding to $x + \frac{1}{2}$. Doing so would produce an algorithm of the same asymptotic time complexity as that for $\mathcal{M}$, but slightly more expensive. In practice, this alteration produces virtually no improvement in accuracy, so our implementation uses the same algorithm in both directions.

5.2. Complexity analysis. In the following table, we provide the operation count for each step of the algorithm.

Initialization Phase.

Step	Complexity	Explanation
1.	$O(k)$	Chebyshev nodes $t_0, \ldots, t_{k-1}$ and $t'_0, \ldots, t'_{2k-1}$ (a total of $3k$ nodes) are computed.
2.	$O(k^2 + ks)$	Interpolation coefficients $u_r(l/s)$, for $l = 0, 1, \ldots, s - 1$ and $u_r(t'_l)$, for $l = 0, \ldots, 2k - 1$ are computed ($r = 0, \ldots, k - 1$). The average cost per coefficient is constant.
3.	$O(nk^2/s)$	Matrix entries ${}_\iota\mathcal{M}^{r,m}_{i,j}$ are computed. There are k^2 of these values per square; there are $\frac{3}{2}(n/s - 2)$ squares of the smallest size, $\frac{3}{2}(n/(2s) - 2)$

Initialization Phase (continued).

Step	Complexity	Explanation
		squares of the next size, and so forth, up to three squares of the largest size. Thus the total number of squares is order $O(n/s)$, at a cost of order $O(k^2)$ per square.
4.	$O(ns)$	Matrix entries $\mathcal{M}(m+js, i+js)$ are computed. There are no more than $2s$ of these values per row of the matrix M^n, and M^n has n rows.
Total	$O(n(s+k^2/s))$	

Evaluation Phase.

Step	Complexity	Explanation
5.	$O(nk)$	Coefficients $b^r_{1,j}$, for $j = 0, 1, \ldots, n/s-1$ and $r = 0, \ldots, k-1$ are computed, at order $O(s)$ operations each.
6.	$O(nk^2/s)$	Coefficients $b^r_{l,j}$ for $l > 1$ are computed. There are $(n/s-4)\cdot k$ of these, and each requires $O(k)$ operations to compute.
7.	$O(nk^2/s)$	Coefficients $c^r_{l,j}$ are computed. There are $2[n/s - \log_2(n/s) - 1]\cdot k$ of these, and each is computed in order $O(k)$ operations.
8.	$O(nk^2/s)$	Coefficients $d^r_{l,j}$ are computed. Again there are $2[n/s - \log_2(n/s) - 1]\cdot k$ of these, and each is computed in order $O(k)$ operations.
9.	$O(nk+ns)$	Final results γ_i are computed. There are n of these and each requires order $O(k+s)$ operations to compute.
Total	$O(n(s+k+k^2/s))$	

The total computation time for the initialization phase is

$$t_{\text{init}} = a_1 k + a_2 k^2 + a_3 ks + a_4 nk^2/s + a_5 ns$$

and the time for the evaluation phase is

$$t_{\text{eval}} = a_6 ns + a_7 nk + a_8 nk^2/s,$$

where $a_1, a_2, \ldots, a_8$ depend on the implementation, language, and computer system. The length s of the smallest interval, however, is not determined by the problem, and can be chosen arbitrarily. Choosing s to minimize the evaluation time t_{eval} yields

$$s = k\sqrt{a_8/a_6}.$$

For this choice of s, the initialization time t_{init} and evaluation time t_{eval} each are order $O(nk)$. The actual values of k and s used in our implementation are given in the next section, which also includes running times and accuracies.

6. Numerical results. We have implemented the $O(n)$ algorithm just described. It has been combined with a cosine transform (order $n \log n$) to produce a code converting coefficients of Legendre expansions (of functions in the interval [-1,1]) into values of such functions at Chebyshev nodes, and vice versa. In the forward direction, the algorithm transforms function values at Chebyshev nodes to Legendre expansion coefficients, while in the backward direction, Legendre expansion coefficients are converted to function values at Chebyshev nodes. Here we present the results of applying the algorithm to inputs of varying size, giving accuracies and running times. These results are compared to a direct calculation.

The algorithm has been implemented both in single precision and double precision arithmetic. The number k of Chebyshev nodes per interval was chosen to retain maximum precision. The size s of the smallest interval was chosen to maximize efficiency. For the single precision version, we chose k to be 8 and s to be 32. For double precision, k is 18 and s is 64.

We ran our FORTRAN implementations on a Sun 3/50 equipped with an MC68881 floating-point coprocessor. For comparison, times required to compute a complex FFT on this hardware are also given. All accuracies were determined by comparing to results computed in quadruple precision (on a microVax), using the direct method. The measure of error was chosen to be the $\mathcal{L}^2$-norm:

$$E = \sqrt{\sum_{i=0}^{n-1} (\gamma_i - \lambda_i)^2 \Big/ \sum_{i=0}^{n-1} \lambda_i^2},$$

where $\vec{\gamma}$ is the result of the computation being tested and $\vec{\lambda}$ is the result of the quadruple precision computation. Input for the computations in the backward direction (converting from Legendre coefficients to function values) was a vector whose components were independent pseudorandom numbers, each drawn from a distribution uniformly distributed in the interval $[0, 1]$. Input for the forward direction was the function values from the backward quadruple precision computation.

The direct computation of $\sum \alpha_i \cdot P_i(t)$ was accomplished by applying the three-term recurrence relation (2.17) for Legendre polynomials. This procedure produces significant roundoff errors for t near -1 and 1, which impairs the accuracy of the direct computation. It is given here primarily for CPU time comparisons.

TABLE 6.1
Single precision computations.

Input	$\mathcal{L}^2$ *error*			*Time (seconds)*			
size	*Algorithm*		*Direct*	*Algorithm*		*Direct*	
(n)	*Forward*	*Backward*	*method*	*Initial.*	*Eval.*	*method*	FFT
64	0.335E−06	0.133E−06	0.112E−05	0.92	0.07	0.30	0.02
128	0.629E−06	0.107E−06	0.290E−05	2.34	0.17	1.16	0.06
256	0.888E−06	0.117E−06	0.762E−04	5.48	0.47	4.78	0.16
512	0.130E−05	0.167E−06	0.242E−03	11.60	1.07	18.26	0.32
1024	0.236E−05	0.125E−06	0.278E−03	24.24	2.31	73.96	0.72
2048	0.285E−05	0.171E−06	0.396E−02	48.76	4.90	297.56	1.66
4096	0.503E−05	0.262E−06	0.121E−01	100.18	9.84	1168.18	3.48

TABLE 6.2
Double precision computations.

Input size (n)	$\mathcal{L}^2$ *error*			*Time (seconds)*			
	Algorithm		*Direct*	*Algorithm*		*Direct*	
	Forward	*Backward*	*method*	*Initial.*	*Eval.*	*method*	FFT
64	0.152E−14	0.673E−15	0.756E−15	1.86	0.11	0.30	0.02
128	0.201E−14	0.695E−15	0.209E−14	4.56	0.28	1.18	0.08
256	0.312E−14	0.723E−15	0.134E−13	10.80	0.71	4.80	0.16
512	0.495E−14	0.725E−15	0.127E−12	24.32	1.96	19.22	0.36
1024	0.689E−14	0.768E−15	0.455E−12	52.30	4.62	76.92	0.78
2048	0.908E−14	0.787E−15	0.827E−12	108.50	10.12	307.40	1.76
4096	0.139E−13	0.840E−15	0.716E−11	223.00	21.38	1229.28	3.70

Several observations may be made from Tables 6.1 and 6.2.

1. The algorithm permits very high accuracy. In single precision the relative error is less than 6×10^{-6} for all input sizes up to 4096. For double precision, the relative error is less than 2×10^{-14}.
2. The error of the algorithm increases roughly as the square root of the length of the input vector.
3. The time required to make the single precision computation is less than three times that for an FFT of the same size. For double precision, the ratio is about 5.5 for n in the range we have tabulated. (For larger n, the ratio would be less than 5.5.) The initialization time is roughly ten times the computation time, for both single and double precision.
4. In single precision computations, for n as low as 64, the time required by the algorithm is less than one-fourth of the time required by the direct method. For $n = 4096$, the speedup is 120-fold. For double precision, the speedups are roughly threefold at $n = 64$ and 60-fold at $n = 4096$.

7. Generalizations and conclusions. An algorithm has been presented for the rapid conversion of the coefficients of a Legendre expansion of a function on the interval $[-1, 1]$ into its values at the Chebyshev nodes on that interval, and vice versa. The algorithm requires order $O(n \log n)$ operations to transform an n-term expansion into n function values, or n function values into an n-term expansion.

The method admits several straightforward generalizations.

1. As described in §4.4, the scheme requires that n (the number of Chebyshev nodes on the interval $[-1, 1]$, and also the length of the Legendre expansion) be a power of two. Clearly, this is not an essential requirement. A simple modification of the scheme will convert an n-term Chebyshev expansion into an n-term Legendre expansion, and vice versa, for any positive integer n. On the other hand, the FFT used to evaluate the Chebyshev series at the Chebyshev nodes does impose certain algebraic requirements on n. Thus, the scheme of this paper will perform efficiently whenever the FFT does.
2. Clearly, the problem solved by the algorithm of this paper is a particular case of the following problem, regularly encountered in the numerical treatment of equations of mathematical physics.

   PROBLEM 7.1. Given the coefficients α_{nm} for $n = 0, 1, \ldots, N-1$, and $m =$

$-n, -n+1, \ldots, n$, evaluate the expression

$$f(\theta, \varphi) = \sum_{n=0}^{N-1} \sum_{m=-n}^{n} \alpha_{nm} \cdot P_n^m(\cos\theta) \cdot e^{im\varphi}, \tag{7.1}$$

at the points (θ_i, φ_j), for $i, j = 0, 1, \ldots, N-1$, defined by the formulae

$$\theta_i = \frac{2i+1}{2N}\pi, \qquad \varphi_j = \frac{j}{N}2\pi.$$

(In (7.1), P_n^m denotes the associated Legendre function of degree n and order m, as defined in, e.g., [1].)

3. The heart of this paper is an order $O(n)$ algorithm converting an n-term Legendre expansion of a function into an n-term Chebyshev expansion of that function, and vice versa. The algorithm is based on the fact that elements of the matrices M^n, L^n connecting the two expansions are a smooth function of the indices, except near the diagonal. In this respect, the scheme of this paper is somewhat similar to those of [8], [11], and several others. A radical generalization of this approach is to observe that any matrix whose elements are a sufficiently smooth nonoscillatory function of their indices can be applied to an arbitrary vector for a cost proportional to n (with a fixed precision). As the matrix becomes less smooth, the procedure becomes less efficient. For matrices singular near the diagonal (such as the matrices M^n, L^n of this paper), the scheme is still of order $O(n)$. The same is true for matrices with singularities along a fixed number of bands of fixed width. It is easy to construct examples of matrices for which the method will be of order $O(n \log n)$, and other examples for which the method will fail completely. An investigation of these issues is in progress, and includes the work of several authors [2], [3], [4].

Appendix. Numerical evaluation of the function Λ. The definitions (2.20) and (2.21) of the functions $\mathcal{M}$, $\mathcal{L}$ involve the function Λ, defined as

$$\Lambda(z) = \frac{\Gamma(z+\frac{1}{2})}{\Gamma(z+1)},$$

for $z \in \mathcal{C}$. The algorithm, described in §§4 and 5, actually requires computation of $\Lambda(x)$ for $x \in \mathcal{R}^+$. The function Λ may of course be computed using the Sterling asymptotic expansion for Γ, but for an efficient computation we use the asymptotic formula

$$\Lambda(x) \sim \frac{1}{\sqrt{x}} \quad \text{as } x \to \infty. \tag{A.1}$$

For large values of x, therefore, the function $\Lambda(x)\sqrt{x}$ may be well approximated by a polynomial in $1/x$. Defining the change of variable $y = 1/x$, we obtain the approximate formula

$$\frac{\Lambda\left(\frac{1}{y}\right)}{\sqrt{y}} \approx a_0 + a_1 y + \cdots + a_5 y^5,$$

where the coefficients $a_0, a_1, \ldots, a_5$ have been determined by evaluating the function $\Lambda(1/y)/\sqrt{y}$ at Chebyshev nodes on each of four intervals. The values of the coefficients are shown in Table A.1.

TABLE A.1

Coefficients $a_0, a_1, \ldots, a_5$ in the formula $\Lambda(1/y) \approx \sqrt{y}\left[a_0 + \cdots + a_5 y^5\right]$, for four separate intervals.

Coefficient	$y \in (0, .02)$	$y \in (.02, .04)$
a_0	0.99999999999999999E+00	0.9999999999974725E+00
a_1	−0.12499999999996888E+00	−0.12499999994706490E+00
a_2	0.78124999819509772E−02	0.78124954632722315E−02
a_3	0.48828163023526451E−02	0.48830152125039076E−02
a_4	−0.64122689844951054E−03	−0.64579205161159155E−03
a_5	−0.15070098356496836E−02	−0.14628616278637035E−02

Coefficient	$y \in (.04, .058)$	$y \in (.058, .067)$
a_0	0.99999999999298844E+00	0.9999999996378690E+00
a_1	−0.12499999914033463E+00	−0.12499999657282661E+00
a_2	0.78124565447111342E−02	0.78123659464717666E−02
a_3	0.48839648427432678E−02	0.48855685911602214E−02
a_4	−0.65752249058053233E−03	−0.67176366234107532E−03
a_5	−0.14041419931494052E−02	−0.13533949520771154E−02

For small values of x, formula (A.1) is no longer useful. Fortunately, however, the evaluation of Λ at small values of x, which corresponds to the evaluation of $\mathcal{M}$ and $\mathcal{L}$ near the diagonal, is necessary only at half-integer values of x. In particular, it suffices to obtain Λ at values $x = 0, .5, 1.0, 1.5, \ldots, (s-3)/2$, and $x \in [s/2 - 1, \infty)$, where s is the size of the smallest interval, as defined in §4.2. For our implementation, $s \geq 32$ (see §6). Table A.1 shows the coefficients from the division of the interval $(0, 1/15)$ for y into four subintervals. The subintervals were chosen so as to limit the approximation's relative error to 2×10^{-15}. The coefficient values were computed using quadruple precision arithmetic.

Table A.2 contains the tabulation of Λ at the half-integer values up to 14.5.

TABLE A.2

Values of $\Lambda(x)$ for small x.

x	$\Lambda(x)$	x	$\Lambda(x)$
0	0.17724538509055159E+01	0.5	0.11283791670955126E+01
1	0.88622692545275794E+00	1.5	0.75225277806367508E+00
2	0.66467019408956851E+00	2.5	0.60180222245094006E+00
3	0.55389182840797380E+00	3.5	0.51583047638652002E+00
4	0.48465534985697706E+00	4.5	0.45851597901023999E+00
5	0.43618981487127934E+00	5.5	0.41683270819112728E+00
6	0.39984066363200604E+00	6.5	0.38476865371488672E+00
7	0.37128061622971992E+00	7.5	0.35911741013389425E+00
8	0.34807557771536241E+00	8.5	0.33799285659660638E+00
9	0.32873804562006448E+00	9.5	0.32020375888099550E+00
10	0.31230114333906123E+00	10.5	0.30495596083904336E+00
11	0.29810563682364938E+00	11.5	0.29169700601995452E+00
12	0.28568456862266400E+00	12.5	0.28002912577915634E+00
13	0.27469670059871537E+00	13.5	0.26965767667622464E+00
14	0.26488610414876124E+00	14.5	0.26035913610118244E+00

REFERENCES

[1] M. ABRAMOWITZ AND I. A. STEGUN, *Handbook of Mathematical Functions*, National Bureau of Standards, Washington, DC, 1972.

[2] B. ALPERT, *Sparse Representation of Smooth Linear Operators*, Ph.D. thesis, Yale University, New Haven, CT, December, 1990.

[3] B. ALPERT, G. BEYLKIN, R. COIFMAN, AND V. ROKHLIN, *Wavelets for the fast solution of second-kind integral equations*, Tech. Report, Department of Computer Science, Yale University, New Haven, CT, 1990, to appear.

[4] G. BEYLKIN, R. COIFMAN, AND V. ROKHLIN, *Fast wavelet transforms and numerical algorithms* I, Tech. Report, Department of Computer Science, Yale University, 1989; Comm. Pure Appl. Math., 1990, to appear.

[5] E. O. BRIGHAM, *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[6] G. DAHLQUIST AND A. BJÖRCK, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[7] I. S. GRADSHTEYN AND I. M. RYZHIK, *Table of Integrals, Series, and Products*, Academic Press, New York, 1980.

[8] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.

[9] N. N. LEBEDEV, *Special Functions and Their Applications*, Dover, New York, 1972.

[10] S. A. ORSZAG, *Fast eigenfunction transforms*, in Science and Computers, Advances in Mathematics Supplementary Studies, G. C. Rota, ed., Academic Press, New York, 1986, pp. 23–30.

[11] V. ROKHLIN, *A fast algorithm for the discrete Laplace transformation*, J. Complexity, 4 (1988), pp. 12–32.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 180–207, January 1991
© 1991 Society for Industrial and Applied Mathematics
010

THE RIEMANN PROBLEM FOR LONGITUDINAL MOTION IN AN ELASTIC-PLASTIC BAR*

JOHN A. TRANGENSTEIN† AND RICHARD B. PEMBER†

Abstract. In this paper the analytical solution to the Riemann problem for the Antman–Szymczak model of longitudinal motion in an elastic-plastic bar is constructed. The model involves two surfaces corresponding to plastic yield in tension and compression, and exhibits the appropriate limiting behavior for total compressions. The solution of the Riemann problem involves discontinuous changes in characteristic speeds due to transitions from elastic to plastic response. Illustrations are presented, in both state-space and self-similar coordinates, of the variety of possible solutions to the Riemann problem for possible use with numerical algorithms.

Key words. Riemann problem, plasticity

AMS(MOS) subject classifications. 35L65, 73E99, 73D05

1. Introduction.

The analysis of the wave structure of solid mechanics has been studied in a variety of ways for some time. On one hand, Hadamard's theory of propagating discontinuity surfaces [7] has been extended and applied to solid mechanics by a variety of authors. For more details and literature surveys, the interested reader can refer to Cristescu [2] and Kosiński [9]. The nature of characteristic speeds has also been studied; a comprehensive analysis is due to Truesdell and Noll [21]. Discontinuities in the characteristic structure due to plasticity have also been described (see, for example, [12]). These results have been used by a variety of authors to construct local solutions to initial value problems, using some variety of the method of characteristics. (See, for example, [2].) However, these studies by themselves do not give a complete determination of solutions to initial value problems. An important topic of current research involves the use of admissibility conditions to piece waves together into a *global* solution to one-dimensional initial value problems, particularly with piecewise constant initial data, where the discontinuity is confined to the origin. These initial value problems are known as Riemann problems, and their solution constitutes an essential step in modern shock-capturing methods (such as Godunov's method [6] and the random-choice method [4]), and front-tracking methods (see, for example, [5],[15]).

In the past decade, several important Riemann problems in solid mechanics have been solved. In 1980, Keyfitz and Kranzer [8] solved the Riemann problem for the longitudinal and transverse motion of an elastic string, as described by Cristescu [2]. This problem was of note because the local wave structure involved curves in state space along which the characteristic speeds coalesced. (A similar model, describing polymer flooding for enhanced oil recovery, has coalescing characteristic speeds *and* directions [14].) The authors showed that the standard Lax [10] and Oleinik [13] entropy conditions needed to be revised; they also showed that the wave curves could not necessarily be extended indefinitely. In 1982, Shearer [16] solved the Riemann problem for longitudinal motion in a strain-softening material, exhibiting a finite in-

* Received by the editors September 15, 1989; accepted for publication (in revised form) December 27, 1989. This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract W-7405-Eng-48. Partial support was provided by the Applied Mathematical Sciences Program of the Office of Energy Research, and by the Defense Nuclear Agency.

† Computing and Mathematics Research Division, Lawrence Livermore National Laboratory L-316, P.O. Box 808, Livermore, California 94550 (johnt@lll-crg.llnl.gov).

terval in strain within which the equations of motion are elliptic. Nevertheless, he showed that the Riemann problem could be uniquely solved with one of the states in the elliptic region, and discussed in later papers ([17],[18]) the connection between various entropy conditions and the resulting solutions to the Riemann problem. In 1987, Tang and Ting [19] solved the Goursat problem for three-dimensional displacement of an elastic bar, and found umbilic points in stress space, at which the wave speeds coalesced without any loss of eigenvectors.

The principle new aspect of this paper is the analysis of the effects of *plasticity* on the Riemann problem. Plasticity is an important physical phenomenon in the deformation of solids, and corresponds to the permanent dislocations of material particles. These dislocations remain even after any applied forces are removed. As a result, the response of the material to applied forces is a function of the past history. This history can complicate the description of the equations of motion for the material.

In this paper, we describe the analytical solution to the Riemann problem for a simple model of finite longitudinal deformation in an elastic-plastic bar, as first proposed by Antman and Szymczak [1]. The most notable complication in the solution to this Riemann problem is the discontinuous change in the wave structure between elastic and plastic behavior. This discontinuity in the characteristic structure leads to a fairly large number of structurally different solutions to the Riemann problem, and therefore a complicated description of its solution. Nevertheless, we hope that the solution to this problem can serve as an important test problem for numerical methods in solid mechanics, as well as an essential building block for high-resolution methods such as the random-choice method, front-tracking, and Godunov's method.

The outline of the paper is as follows. In §2, we describe the model, and in §3 we determine the centered rarefaction waves and shock waves. In §4, we connect these curves to form wave families, and show that they allow a complete parameterization of physical space. In §5, we use these wave curves to describe the Riemann problem solution in state space for all possible relative positions of the left and right states in the Riemann problem. We conclude the paper with some examples of the Riemann problem solutions in self-similar coordinates for a particular implementation of the model.

2. Mathematical description of the model. In order to describe the model proposed by Antman and Szymczak [1], we shall introduce the following notation. We denote the longitudinal component of the displacement gradient by ϵ, and the longitudinal component of the velocity by v. In addition, there is a history parameter π, representing the plastic displacement gradient. The longitudinal stress in the current configuration divided by the density in the reference configuration is denoted by σ. For simplicity, in the remainder of the paper we will refer to σ simply as the stress. If we ignore body forces, transverse displacements, and thermal effects, then the system of partial differential equations consists of conservation of momentum in the Lagrangian frame of reference and the equality of mixed partial derivatives of the displacement in time and space (measured in the reference configuration):

$$\begin{aligned} \frac{\partial \mathrm{v}}{\partial \tau} - \frac{\partial \sigma}{\partial x} &= 0, \\ \frac{\partial \epsilon}{\partial \tau} - \frac{\partial \mathrm{v}}{\partial x} &= 0. \end{aligned} \tag{1}$$

Here x is the Lagrangian spatial coordinate (i.e., original particle position) and τ is time. We are interested in solutions to the Riemann problem for (1), meaning that we specify the following piecewise constant initial data:

$$\mathrm{v}(x,0) = \begin{cases} \mathrm{v}_L, & x < 0, \\ \mathrm{v}_R, & x > 0, \end{cases}$$

$$\epsilon(x,0) = \begin{cases} \epsilon_L, & x < 0, \\ \epsilon_R, & x > 0, \end{cases}$$

$$\pi(x,0) = \begin{cases} \pi_L, & x < 0, \\ \pi_R, & x > 0. \end{cases}$$

The system of conservation laws can be closed once σ is described as a function of ϵ and π.

For any physically realistic displacement gradient ϵ, there are upper and lower bounds on the stress σ, given by the functions $c(\epsilon)$ and $t(\epsilon)$ representing the plastic compression and tension curves, respectively. In between these bounds, the stress is given by the elastic curves $\sigma = e(\epsilon, \pi)$. The history of the material is accounted for by the plastic displacement gradient π. It remains for us to place physical and mathematical constraints on these three functions $c(\epsilon)$, $t(\epsilon)$, and $e(\epsilon, \pi)$. (Please see Fig. 1.)

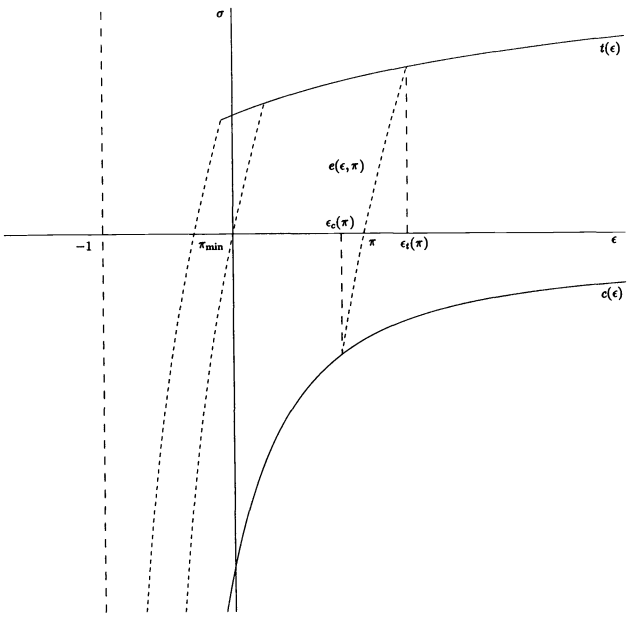


FIG. 1. *Stress-strain curves.*

Since the deformation gradient $1 + \epsilon$ is the ratio of the original mass density to the current mass density, it is required to be positive:

$$1 + \epsilon > 0.$$

We also require that an infinite force be required to totally compress the material:

$$c(\epsilon) \downarrow -\infty \text{ as } \epsilon \downarrow -1. \tag{2}$$

Further, we assume that plastic compression occurs only for negative stress,

$$c(\epsilon) < 0 \text{ for } -1 < \epsilon,$$

and that tension occurs only for positive stress,

$$t(\epsilon) > 0 \text{ for } \epsilon > -1.$$

Next, we need to describe the connections between the elastic curves $e(\epsilon, \pi)$ and the plastic curves $c(\epsilon)$ and $t(\epsilon)$. We assume that there is some lower limit on the plastic displacement gradient,

$$\pi_{\min} > -1,$$

corresponding to total compression:

$$e(\epsilon, \pi) \downarrow -\infty \text{ implies } \epsilon \downarrow -1 \text{ and } \pi \downarrow \pi_{\min}.$$

We assume that there are unique values $\epsilon_c(\pi) < \epsilon_t(\pi)$ of the longitudinal displacement gradient such that the elastic and plastic curves intersect:

$$c(\epsilon_c(\pi)) = e(\epsilon_c(\pi), \pi) \text{ for } \pi > \pi_{\min},$$

$$t(\epsilon_t(\pi)) = e(\epsilon_t(\pi), \pi) \text{ for } \pi > \pi_{\min}.$$

We assume that for a given value of the plastic displacement gradient, the elastic stress lies between the values of the plastic and tensile yield stresses:

$$c(\epsilon_c(\pi)) < e(\epsilon, \pi) < t(\epsilon_c(\pi)) \text{ for } \epsilon_c(\pi) < \epsilon < \epsilon_t(\pi).$$

We assume that

$$\frac{dc}{d\epsilon}, \frac{dt}{d\epsilon} \text{ and } \frac{\partial e}{\partial \epsilon} > 0 \text{ for } \epsilon > -1 \text{ and } \pi > \pi_{\min}, \tag{3}$$

so that the characteristic speeds are real, and that

$$\frac{d^2c}{d\epsilon^2}, \frac{d^2t}{d\epsilon^2} \text{ and } \frac{\partial^2 e}{\partial \epsilon^2} < 0 \text{ for } \epsilon > -1 \text{ and } \pi > \pi_{\min}, \tag{4}$$

so that the characteristic speeds are genuinely nonlinear.[1] In order to assure these conditions, we assume that the slopes of the plastic loading curves are less than the slopes of the elastic loading curve at the corresponding yield points:

$$\begin{aligned} &\frac{dc}{d\epsilon}(\epsilon_c(\pi)) < \frac{\partial e}{\partial \epsilon}(\epsilon_c(\pi), \pi) \qquad \text{for } \pi > \pi_{\min}, \\ &\frac{dt}{d\epsilon}(\epsilon_t(\pi)) < \frac{\partial e}{\partial \epsilon}(\epsilon_t(\pi), \pi) \qquad \text{for } \pi > \pi_{\min}. \end{aligned} \tag{5}$$

[1] Note that there is no compelling physical reason to assume genuine nonlinearity; indeed, many important models of nonlinear response in solids involve loss of genuine nonlinearity. We make this assumption in order to simplify the discussion, especially regarding the number of waves we have to categorize below.

We assume that the elastic curves are disjoint for distinct values of π:

$$\frac{\partial e}{\partial \pi}(\epsilon, \pi) < 0 \text{ for } \epsilon_c(\pi) < \epsilon < \epsilon_t(\pi) \text{ and } \pi > \pi_{\min}.$$

Together with (5), this assumption guarantees that $\epsilon_c' > 0$ and $\epsilon_t' > 0$; thus the elastic curves move to the right as π increases. Note that an additional assumption appears in (11) below.

During elastic response, π is fixed and $\sigma = e(\epsilon, \pi)$. During plastic compression, $\sigma = c(\epsilon)$ and the plastic displacement gradient π varies so that $c(\epsilon) = e(\epsilon, \pi)$. Similarly, during plastic tension, $\sigma = t(\epsilon) = e(\epsilon, \pi)$. Plastic compression occurs if and only if the material is at yield and the time rate of change of the deformation gradient is negative:

$$\epsilon = \epsilon_c(\pi) \text{ and } \left.\frac{\partial \epsilon}{\partial \tau}\right|_x < 0 \text{ during plastic compression.}$$

Similarly,

$$\epsilon = \epsilon_t(\pi) \text{ and } \left.\frac{\partial \epsilon}{\partial \tau}\right|_x > 0 \text{ during plastic tension.}$$

Otherwise, the material response is elastic.

3. Admissibility conditions and wave curves. In the previous section we established the governing equations for the model, and examined various physical and mathematical conditions on the model parameters. In this section, we shall analyze the characteristic structure of these equations, and determine admissible wave curves. We shall combine these wave curves into wave families in the next section.

During plastic tension or compression, the stress is a function of the displacement gradient ϵ only, and the plastic displacement gradient π does not affect the equations of motion.[2] On the other hand, during elastic response the plastic displacement gradient is constant and again does not affect the determination of the type of the system of conservation laws. In either case, it is straightforward to show from (3) that the system (1) is hyperbolic, with characteristic speeds $\pm\sqrt{\partial\sigma/\partial\epsilon}$. In fact, in centered rarefaction waves the conserved quantities v and ϵ are smooth functions of x/τ only, satisfying

$$0 = \frac{\partial \mathrm{v}}{\partial \tau} - \frac{\partial \sigma}{\partial x} = -\left[\mathrm{v}'\frac{x}{\tau} + \frac{\partial \sigma}{\partial \epsilon}\epsilon'\right]\frac{1}{\tau},$$

$$0 = \frac{\partial \epsilon}{\partial \tau} - \frac{\partial \mathrm{v}}{\partial x} = -\left[\epsilon'\frac{x}{\tau} + \mathrm{v}'\right]\frac{1}{\tau}.$$

By using the latter equation to eliminate v', the former equation becomes

$$0 = \left[\frac{\partial \sigma}{\partial \epsilon} - \left(\frac{x}{\tau}\right)^2\right]\epsilon'.$$

[2] This simplified dependence of the stress is due to the fact that the model allows only longitudinal displacements. In models involving both longitudinal and transverse displacements, response during yield usually depends on the history parameters.

This result determines the characteristic speeds, and shows that the rarefaction curves satisfy

$$\mathrm{v}' = \pm\sqrt{\frac{\partial\sigma}{\partial\epsilon}}\epsilon'. \tag{6}$$

The Rankine–Hugoniot equations show that the left and right states at traveling discontinuities are related by

$$-\begin{bmatrix} \sigma_R - \sigma_L \\ \mathrm{v}_R - \mathrm{v}_L \end{bmatrix} = \begin{bmatrix} \mathrm{v}_R - \mathrm{v}_L \\ \epsilon_R - \epsilon_L \end{bmatrix} s, \tag{7}$$

where s is the speed of the discontinuity. It is straightforward to solve these equations to find that the speed of the discontinuity is

$$s = \pm\sqrt{\frac{\sigma_R - \sigma_L}{\epsilon_R - \epsilon_L}}.$$

Again, inequalities (3) and the mean-value theorem show that the discontinuity speeds are always real for left and right states undergoing the same loading conditions (and having the same value of π for elastic response). Further, the Rankine–Hugoniot conditions (7) show that the traveling discontinuities satisfy

$$\mathrm{v}_R - \mathrm{v}_L = \pm\sqrt{\frac{\sigma_R - \sigma_L}{\epsilon_R - \epsilon_L}}(\epsilon_R - \epsilon_L). \tag{8}$$

The next step is to determine the conditions under which the centered rarefactions and traveling discontinuities are admissible solutions to the equations of motion. We will consider those states that can be connected to a given left state by centered rarefactions or shocks. First, we assume that the left state lies inside the elastic regime for the given value of π_L:

$$\epsilon_c(\pi_L) < \epsilon_L < \epsilon_t(\pi_L).$$

Recall that in this elastic regime we have $\sigma = e(\epsilon, \pi_L)$, and that $\pi = \pi_L$ is constant. During an admissible rarefaction, the characteristic speeds must increase from left to right. Thus inequality (4) implies that an elastic rarefaction moving to the left in physical space from a left state $(\mathrm{v}_L, \epsilon_L, \pi_L)$ to a right state $(\mathrm{v}_R, \epsilon_R, \pi_R)$ must satisfy

$$\mathbf{R}^{\mathbf{e}}_{-}: \quad \begin{aligned} \mathrm{v}_R &= \mathrm{v}_L + \textstyle\int_{\epsilon_L}^{\epsilon_R} \sqrt{\frac{\partial e}{\partial\epsilon}}\,d\epsilon, \\ \sigma_R &= e(\epsilon_R, \pi_R), \\ \pi_R &= \pi_L, \\ \epsilon_c(\pi_L) < \epsilon_L &< \epsilon_R \le \epsilon_t(\pi_L). \end{aligned}$$

Similarly, a rarefaction moving to the right satisfies

$$\mathbf{R}^{\mathbf{e}}_{+}: \quad \begin{aligned} \mathrm{v}_R &= \mathrm{v}_L + \textstyle\int_{\epsilon_R}^{\epsilon_L} \sqrt{\frac{\partial e}{\partial\epsilon}}\,d\epsilon, \\ \sigma_R &= e(\epsilon_R, \pi_R), \\ \pi_R &= \pi_L, \\ \epsilon_c(\pi_L) \le \epsilon_R &< \epsilon_L < \epsilon_t(\pi_L). \end{aligned}$$

During an admissible elastic shock with nonzero speed s, we must have [3],[11],[13],[22]

$$s\left[\frac{\sigma_R - \sigma_L}{\epsilon_R - \epsilon_L} - \frac{\sigma(\epsilon) - \sigma_L}{\epsilon - \epsilon_L}\right] \le 0 \text{ for all } \epsilon \text{ between } \epsilon_L \text{ and } \epsilon_R. \tag{9}$$

This inequality is equivalent to requiring that the wave speeds be determined by tangents to the convex hull of the stress curve for $s(\epsilon_R - \epsilon_L) > 0$, and by the tangents to the concave hull for $s(\epsilon_R - \epsilon_L) < 0$. (See Fig. 2 below.) It is straightforward to see that the admissible elastic shocks moving to the left satisfy

$$\mathbf{S}_-^{\mathbf{e}}: \quad \begin{aligned} v_R &= v_L - \sqrt{(\epsilon_R - \epsilon_L)(e(\epsilon_R, \pi_L) - e(\epsilon_L, \pi_L))}, \\ \sigma_R &= e(\epsilon_R, \pi_R), \\ \pi_R &= \pi_L, \\ \epsilon_c(\pi_L) \le \epsilon_R &< \epsilon_L < \epsilon_t(\pi_L). \end{aligned}$$

Similarly, admissible elastic shocks moving to the right satisfy

$$\mathbf{S}_+^{\mathbf{e}}: \quad \begin{aligned} v_R &= v_L - \sqrt{(\epsilon_R - \epsilon_L)(e(\epsilon_R, \pi_L) - e(\epsilon_L, \pi_L))}, \\ \sigma_R &= e(\epsilon_R, \pi_R), \\ \pi_R &= \pi_L, \\ \epsilon_c(\pi_L) < \epsilon_L &< \epsilon_R \le \epsilon_t(\pi_L). \end{aligned}$$

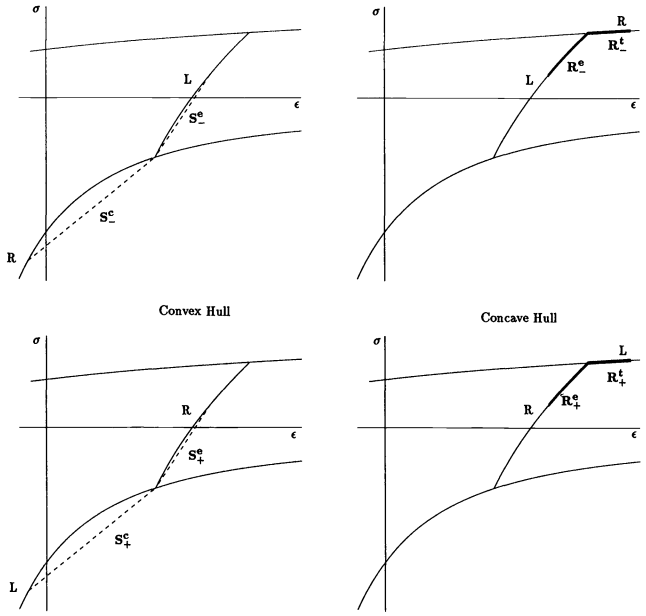


FIG. 2. *Admissibility conditions and stress functions.*

The determination of admissible centered rarefactions and shocks during plastic response is complicated by hysteresis. First, we will consider plastic compression; in other words, we assume that $\epsilon_L = \epsilon_c(\pi_L)$. Recall that the stress is given by $\sigma = c(\epsilon)$. Also recall that during plastic response the time derivative of the displacement gradient must be negative; otherwise, the material relaxes elastically. For this reason, centered rarefactions are impossible during plastic compression. (This is because $\epsilon_L < \epsilon_R$ corresponds to unloading during rarefactions moving to the left, and $\epsilon_L > \epsilon_R$ corresponds to unloading during rarefactions moving to the right.) We are left only

with shocks during plastic compression:

$$\mathbf{S}_{-}^{\mathbf{c}}: \quad \begin{array}{c} v_R = v_L - \sqrt{(\epsilon_R - \epsilon_L)(c(\epsilon_R) - c(\epsilon_L))}, \\ \sigma_R = c(\epsilon_R) = e(\epsilon_R, \pi_R), \\ \epsilon_s(\epsilon_L, \pi_L) < \epsilon_R < \epsilon_L = \epsilon_c(\pi_L), \end{array}$$

and

$$\mathbf{S}_{+}^{\mathbf{c}}: \quad \begin{array}{c} v_R = v_L - \sqrt{(\epsilon_R - \epsilon_L)(c(\epsilon_R) - c(\epsilon_L))}, \\ \sigma_R = c(\epsilon_R) = e(\epsilon_R, \pi_R), \\ \epsilon_L = \epsilon_c(\pi_L) < \epsilon_R. \end{array}$$

(We will discuss the lower bound ϵ_s on the extent of $\mathbf{S}_{-}^{\mathbf{c}}$ in §4 below.) Similarly, during plastic tension, shocks are unphysical, and only rarefactions are possible:

$$\mathbf{R}_{-}^{\mathbf{t}}: \quad \begin{array}{c} v_R = v_L + \int_{\epsilon_L}^{\epsilon_R} \sqrt{\frac{dt}{d\epsilon}} d\epsilon, \\ \sigma_R = t(\epsilon_R) = e(\epsilon_R, \pi_R), \\ \epsilon_L = \epsilon_t(\pi_L) < \epsilon_R, \end{array}$$

and

$$\mathbf{R}_{+}^{\mathbf{t}}: \quad \begin{array}{c} v_R = v_L + \int_{\epsilon_R}^{\epsilon_L} \sqrt{\frac{dt}{d\epsilon}} d\epsilon, \\ \sigma_R = t(\epsilon_R) = e(\epsilon_R, \pi_R), \\ -1 < \epsilon_R < \epsilon_L = \epsilon_t(\pi_L). \end{array}$$

4. Wave families. Let us summarize our results so far. For a given left state in the elastic region, there are four admissible and physically realistic wave curves. There is a rarefaction curve $\mathbf{R}_{-}^{\mathbf{e}}$ associated with negative characteristic speeds that moves through state space with increasing velocities v and displacement gradients ϵ until the tension boundary $\epsilon = \epsilon_t(\pi_L)$ of the elastic region is reached. There is also a rarefaction curve $\mathbf{R}_{+}^{\mathbf{e}}$ associated with positive characteristic speeds that moves through state space with increasing velocities and decreasing displacement gradients until the compression boundary $\epsilon = \epsilon_c(\pi_L)$ of the elastic region is reached. There is a shock curve $\mathbf{S}_{-}^{\mathbf{e}}$ associated with negative shock speeds that moves through state space with decreasing velocities and displacement gradients until the compression boundary of the elastic region is reached. Finally, there is a shock curve $\mathbf{S}_{+}^{\mathbf{e}}$ associated with positive shock speeds that moves through state space with decreasing velocities and increasing displacement gradients until the tension boundary of the elastic region is reached. Along each of these curves, the value of the plastic displacement gradient π is constant. (See Fig. 3.)

We also found wave curves for left states at yield in compression (i.e., $\epsilon_L = \epsilon_c(\pi_L)$). There is a plastic shock curve $\mathbf{S}_{-}^{\mathbf{c}}$ associated with negative shock speeds that moves through state space with decreasing velocities, decreasing displacement gradients, and decreasing plastic displacement gradients as $\epsilon \downarrow$. There is a plastic shock curve $\mathbf{S}_{+}^{\mathbf{c}}$ associated with positive shock speeds that moves through state space with decreasing velocities, increasing displacement gradients, and increasing plastic displacement gradients for arbitrarily large ϵ. It is also possible to construct an elastic rarefaction curve $\mathbf{R}_{-}^{\mathbf{e}}$ out of such a left state, associated with negative characteristic speeds and moving through state space with increasing velocities, increasing displacement gradient, and constant plastic displacement gradient until yield in tension $\epsilon = \epsilon_t(\pi_L)$ is reached. The elastic shock $\mathbf{S}_{+}^{\mathbf{e}}$, associated with positive shock speed and moving through state space with decreasing velocities, increasing displacement gradients, and

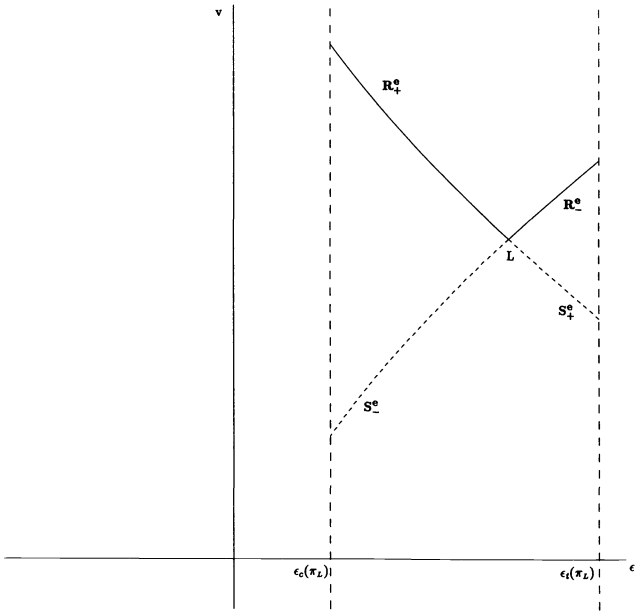


FIG. 3. *Wave curves for left state in the elastic regime.*

constant plastic displacement gradient until yield in tension is reached, is possible as well. It is not possible to construct rarefactions that correspond to plastic compression. It is also not possible to construct either an elastic rarefaction curve $\mathbf{R}^{\mathbf{e}}_+$ or an elastic shock $\mathbf{S}^{\mathbf{e}}_-$ out of this left state: such curves would require that the stress at the right state be simultaneously elastic with the same value of π and less than the stress at the left state. (See Fig. 4.)

Finally, we found wave curves for left states at yield in tension (i.e., $\epsilon_L = \epsilon_t(\pi_L)$). There is a plastic rarefaction curve $\mathbf{R}^{\mathbf{t}}_-$ associated with negative characteristic speeds that moves through state space with increasing velocities, increasing displacement gradients, and increasing plastic displacement gradients for arbitrarily large ϵ. There is a plastic rarefaction curve $\mathbf{R}^{\mathbf{t}}_+$ associated with positive characteristic speeds that moves through state space with increasing velocities, decreasing displacement gradients, and decreasing plastic displacement gradient as $\epsilon \downarrow -1$. Out of such a left state, it is also possible to construct an elastic shock curve $\mathbf{S}^{\mathbf{e}}_-$ associated with negative shock speeds and moving through state space with decreasing velocities, decreasing displacement gradient, and constant plastic displacement gradient; this curve can be continued until yield in compression $\epsilon = \epsilon_c(\pi_L)$ is reached. The elastic rarefaction $\mathbf{R}^{\mathbf{e}}_+$, associated with positive characteristic speeds and moving through state space with increasing velocities, decreasing displacement gradients, and constant plastic displacement gradient until yield in compression is reached, is possible as well. It is not possible to construct shocks that correspond to plastic tension. It is also not possible to construct either an elastic shock curve $\mathbf{S}^{\mathbf{e}}_+$ or an elastic rarefaction $\mathbf{R}^{\mathbf{e}}_-$ out this left state: such curves would require that the stress at the right state be simultaneously elastic with the same value of π and greater than the stress at the left. (See Fig. 5.)

Our next task is to assemble the centered rarefactions and shocks to form wave

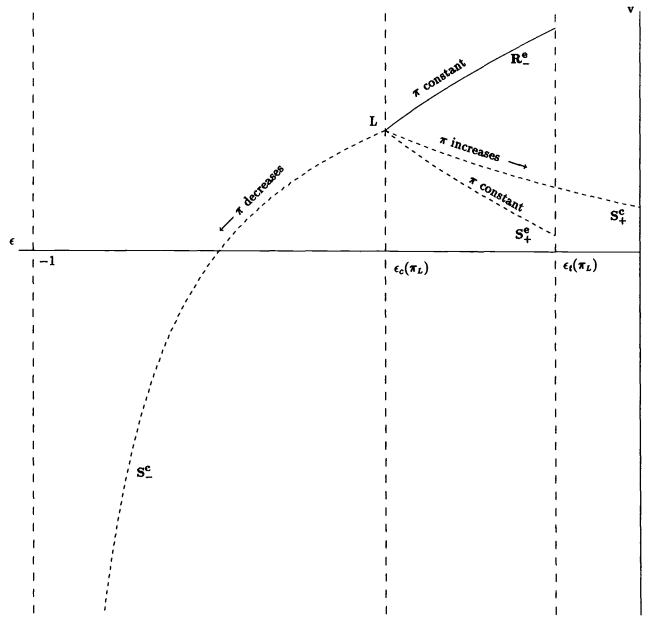


FIG. 4. *Wave curves for left state at compressive yield.*

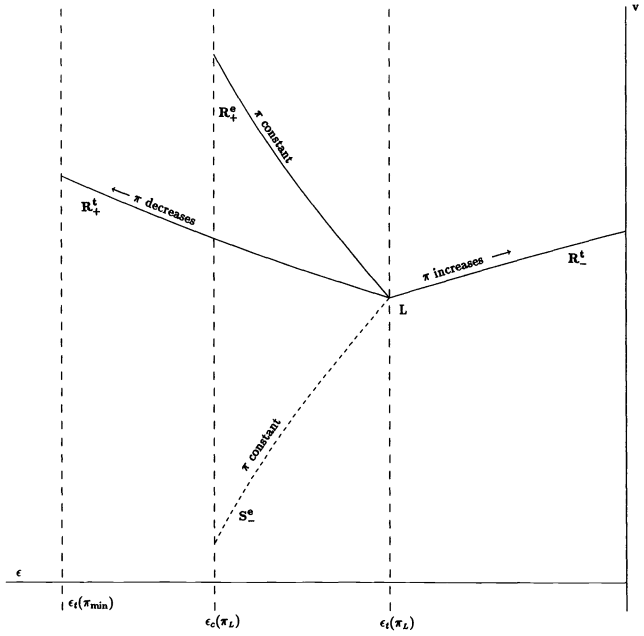


FIG. 5. *Wave curves for left state at tensile yield.*

families. At this point, it is useful to draw the wave curves in terms of v and σ, instead of ϵ. We shall also draw only the waves moving with negative speeds out of a given left state, and only waves moving with positive speeds out of a given right state. This gives us the curves shown in Fig. 6. Given values for π_L, v_L, and ϵ_L satisfying

$$\epsilon_c(\pi_L) \le \epsilon_L \le \epsilon_t(\pi_L),$$

we construct the elastic rarefaction curve $\mathbf{R}_-^{\mathbf{e}}$ and shock curve $\mathbf{S}_-^{\mathbf{e}}$ associated with negative speeds. (Either curve is possibly of zero extent if the left state lies on a yield curve.) Along these curves the plastic displacement gradient π is constant. At the point where the elastic rarefaction reaches yield in tension, the only possibility associated with negative speeds is to continue the curve into the plastic regime using the equation for the plastic rarefaction $\mathbf{R}_-^{\mathbf{t}}$. (See Fig. 5.) On the other hand, at the point where the elastic shock reaches yield in compression, the only possibility is to continue the curve into the plastic regime using the equation for the plastic shock $\mathbf{S}_-^{\mathbf{c}}$. (See Fig. 4.) The chord condition (9) implies that this shock curve is admissible for $\epsilon > \epsilon_s(\epsilon_L, \pi_L)$, where the transition point ϵ_s is defined for $\epsilon_t(\pi_L) > \epsilon_L > \epsilon_c(\pi_L)$ by

$$(10)\ \epsilon_s(\epsilon_L, \pi_L) \equiv \sup\left\{-1 < \epsilon < \epsilon_c(\pi_L) : \frac{e(\epsilon_L, \pi_L) - c(\epsilon_c(\pi_L))}{\epsilon_L - \epsilon_c(\pi_L)} \le \frac{e(\epsilon_L, \pi_L) - c(\epsilon)}{\epsilon_L - \epsilon}\right\}.$$

Note that condition (2) implies that $\epsilon_s > -1$. Also note that ϵ_s is a nondecreasing function of ϵ for fixed π; this can be seen from Fig. 7, or by differentiating (10) and applying (4) and the mean-value theorem. At any rate, points to the left of ϵ_s in Fig. 6 are connected directly to the left state by a single elastic-plastic shock, defined by

$$\mathbf{S}_-^{\mathbf{s}}: \quad \begin{aligned} \mathrm{v}_R &= \mathrm{v}_L - \sqrt{(\epsilon_R - \epsilon_L)(c(\epsilon_R) - e(\epsilon_L, \pi_L))}, \\ \sigma_R &= c(\epsilon_R) = e(\epsilon_R, \pi_R), \\ -1 &< \epsilon_R < \epsilon_s(\epsilon_L, \pi_L). \end{aligned}$$

Note that the waves in this wave family do not permit any reversals in loading direction; in other words, this wave family behaves as if the stress curve were given by

$$\sigma_\pi(\epsilon) \equiv \begin{cases} c(\epsilon) & \text{for} \quad \epsilon < \epsilon_c(\pi), \\ e(\epsilon, \pi) & \text{for} \quad \epsilon_c(\pi) < \epsilon < \epsilon_t(\pi), \\ t(\epsilon) & \text{for} \quad \epsilon_t(\pi) < \epsilon, \end{cases}$$

with $\pi = \pi_L$. This function σ_π is strictly increasing; thus its chord slopes are always positive, leading to real shock speeds. Also note that at the yield curves where the elastic wave curves join the plastic wave curves, there are discontinuous changes in the characteristic speeds and shock speeds.

Next, we will construct the wave family associated with positive wave speeds. Given values for π_R, v_R, and ϵ_R satisfying

$$\epsilon_c(\pi_R) \le \epsilon_R \le \epsilon_t(\pi_R),$$

we construct the elastic rarefaction curve $\mathbf{R}_+^{\mathbf{e}}$ and shock curve $\mathbf{S}_+^{\mathbf{e}}$. Along these curves the plastic displacement gradient π is constant. At the points where these curves reach the yield curves, we again continue the curves further into the plastic regimes, using the equations for plastic rarefactions $\mathbf{R}_+^{\mathbf{t}}$ and shocks $\mathbf{S}_+^{\mathbf{c}}$. At $\epsilon = \epsilon_s(\epsilon_R, \pi_R)$, the plastic shock $\mathbf{S}_+^{\mathbf{c}}$ undergoes a transition into an elastic-plastic shock $\mathbf{S}_+^{\mathbf{s}}$ defined by

$$\mathbf{S}_+^{\mathbf{s}}: \quad \begin{aligned} \mathrm{v}_R &= \mathrm{v}_L - \sqrt{(\epsilon_R - \epsilon_L)(c(\epsilon_R) - e(\epsilon_L, \pi_L))}, \\ \sigma_R &= c(\epsilon_R) = e(\epsilon_R, \pi_R), \\ \epsilon_L &< \epsilon_s(\epsilon_R, \pi_R). \end{aligned}$$

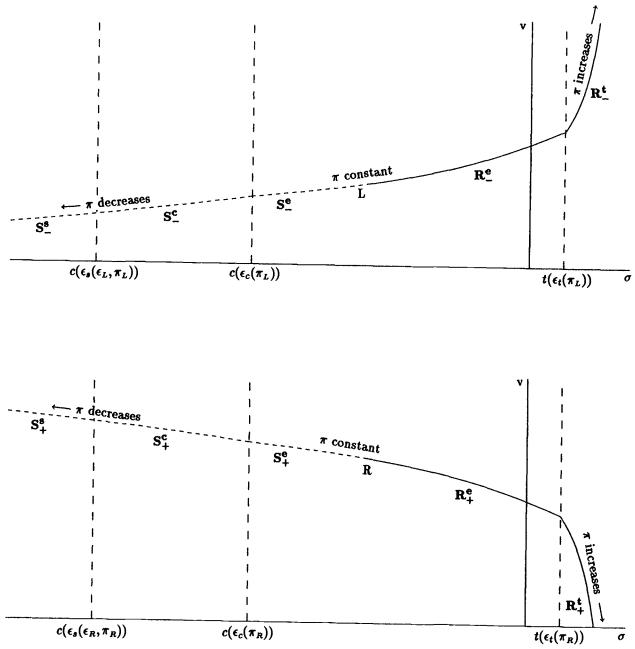


FIG. 6. *Wave curves in stress-velocity space.*

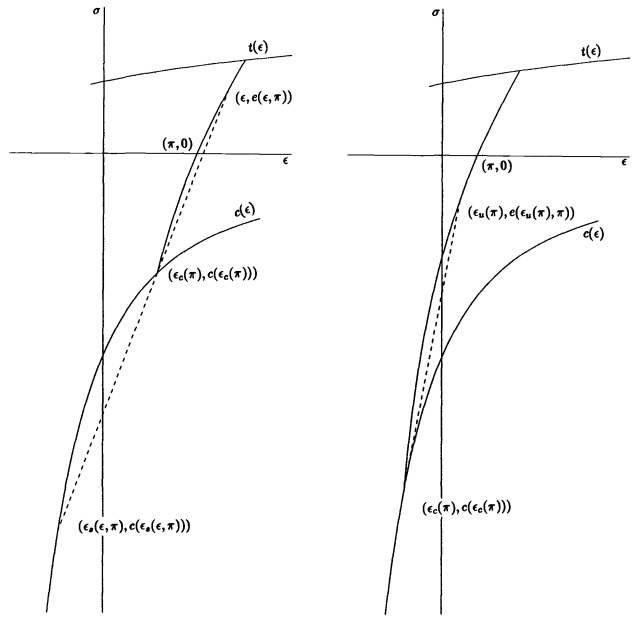


FIG. 7. *Transition from elastic and plastic shocks to single elastic-plastic shocks.*

(See the lower half of Fig. 6.)

Note that the elastic-plastic shock curve $\mathbf{S}^{\mathbf{s}}_{-}$ can be continued indefinitely from a given left state to arbitrarily negative values of v and σ. This is because (2) and (8) show that both the velocity and the stress approach $-\infty$ as $\epsilon \downarrow -1$. Similarly, the elastic-plastic shock curve $\mathbf{S}^{\mathbf{s}}_{+}$ can be continued from a given right state to arbitrarily large values of v and arbitrarily negative values of σ.

The analysis of the asymptotic behavior of the rarefaction curves is more problematical. If

$$\int_0^\infty \sqrt{\frac{dt}{d\epsilon}}\,d\epsilon = \infty,$$

then the expressions for $\mathbf{R}^{\mathbf{t}}_{\pm}$ show that the velocity becomes infinite as $\epsilon \uparrow \infty$. In such a case, the wave curves out of any given left state must always intersect the wave curves out of any given right state at one and only one point in (σ, v)-space. This gives us a unique solution to the Riemann problem. If this intersection point occurs in the plastic regime for both the left and right state, then the value of π at the point of intersection is the same along both curves. On the other hand, if the intersection occurs inside the elastic regime for either state, then the value of π is not necessarily the same along each curve at the point of intersection. In this case, a contact discontinuity in ϵ and π occurs, since the Rankine–Hugoniot conditions (7) are satisfied with $s = 0$. Note that the speed of this contact discontinuity is properly nested in the assortment of speeds associated with the other wave curves.[3]

On the other hand, if

$$\int_0^\infty \sqrt{\frac{dt}{d\epsilon}}\,d\epsilon < \infty, \tag{11}$$

then it is possible that for v_R sufficiently larger than v_L the wave curves drawn out of the left and right states will not intersect. In this case, the material will develop an infinite displacement gradient around the point of the initial discontinuity; in other words, the material will experience *tensile failure* and will pull apart. In such a case, the stress associated with the state at tensile failure must be finite; this can be seen by applying Schwarz's inequality:

$$t_\infty - t_0 = \int_0^\infty \frac{dt}{d\epsilon}\,d\epsilon \le \left[\int_0^\infty \sqrt{\frac{dt}{d\epsilon}}\,d\epsilon\right]^2 < \infty.$$

5. Riemann problem solution. In this section we will combine the results of the previous sections to construct all possible solutions to the Riemann problem. Because the analysis of the wave structure during tensile failure greatly complicates the discussion, *we assume that tensile failure is impossible.* In other words, we assume that inequality (11) does not hold. For our illustrations and examples, we will fix the material model. In a fashion similar to Antman and Szymczak, we take

$$c(\epsilon) = -0.1 - (1+\epsilon)^{-2},$$

[3] The reader might view this contact discontinuity as arising physically from the juxtaposition of two similar materials that have experienced different histories. Another viewpoint sees the contact discontinuity as arising mathematically from the equation $d\pi/d\tau = 0$ during elastic response. In this view, there would be an additional equation in the system, and thus an additional characteristic speed. This characteristic speed would be identically zero, justifying the labeling of the stationary discontinuity as a contact discontinuity.

$$t(\epsilon) = 1.1 - (2+\epsilon)^{-0.5},$$

$$e(\epsilon, \pi) = -0.49(1 + \epsilon + \pi_{\min} - \pi)^{-1} + (1 + \epsilon + \pi_{\min} - \pi),$$

$$\pi_{\min} = -0.3.$$

We have chosen these model functions in order to satisfy the conditions in §2 and to violate (11).

In the discussion below, we will label waves in terms of their type (e.g., $\mathbf{R}^{\mathbf{e}}_{-}$ or $\mathbf{S}^{\mathbf{c}}_{+}$), and by the "left state" from which the individual wave originates. This "left state" may be an intermediate state in the solution of the Riemann problem. In order to simplify the discussion, we have included a table of the wave families in the different regions of the solution to the Riemann problem (Table 1). In the figures below, we use solid curves to denote rarefactions, and dashed curves to denote shocks and contact discontinuities. We always draw the wave family between two points in state space, instead of connecting two points on either side of a shock by a straight line, as might be found by plotting numerical results for a front-tracking method.

In order to make sure that we account for all possible cases, let us review the various points that determine the different waves in the Riemann problem solution. From Fig. 6, we see that the kinds of waves in the wave family for negative speeds depends on the location of four different values of stress,

$$c(\epsilon_s(\epsilon_L, \pi_L)) \le c(\epsilon_c(\pi_L)) \le \sigma_L \le t(\epsilon_t(\pi_L)), \tag{12}$$

relative to the point of intersection with the positive wave family. Since our goal is to determine how to connect to all right states with a given value of π_R, the four values

$$c(\epsilon_s(\epsilon_c(\pi_R), \pi_R)) < c(\epsilon_s(\epsilon_t(\pi_R), \pi_R)) \le c(\epsilon_c(\pi_R)) < t(\epsilon_t(\pi_R)) \tag{13}$$

must also be positioned relative to the intersection point in order to determine the form of the solution to the Riemann problem. (The case when $\epsilon_s(\epsilon_t(\pi_R), \pi_R) = \epsilon_c(\pi_R)$, which occurs for π_R sufficiently close to $\pi_{\min}$, will be handled separately.) These two lists of ordered numbers must be merged in all possible ways in order to produce all possible solutions to the Riemann problem.

In order to determine the number of possible waves that can connect left and right states, we note that there are five different kinds of waves in the negative wave family, corresponding to the left-hand column of Table 1. These five different kinds of waves appear in regions that are bounded by the four special values of the stress in inequalities (12) above. Similarly, there are five different kinds of waves in the positive wave family, corresponding to the top row of Table 1. Thus, it would appear that there should be 25 different ways to connect left and right states in the solution of the Riemann problem. However, it is impossible for the left and right wave families to intersect in tension for one family and in compression for the other, since $\sigma = 0$ would have to lie to the left of the intersection point on one of the wave families, and to the right on the other. Thus there are four blank entries in Table 1, corresponding to these impossible wave connections.

Also note that contact discontinuities are possible in all solutions to the Riemann problem for which the left and right wave families intersect in the elastic regime for either the left or right state. Thus the only cases for which contact discontinuities are impossible are regions 13, 14, 19, 20, and 21. We omit references to these contact

discontinuities in the discussion below for sake of brevity. Finally, note that we make the generic assumption that the wave family out of the left state involves a plastic shock S_-^c. There are cases, however, in which this shock may be of zero extent in the wave family.

TABLE 1
Regions in Riemann problem solution and their wave families.

	$R_+^t R_+^e$	R_+^e	S_+^e	$S_+^c S_+^e$	S_+^s
$R_-^e R_-^t$	13	9	10		
R_-^e	5	1	2	12	17
S_-^e	8	3	4	11	18
$S_-^e S_-^c$		6	7	14	20
S_-^s		15	16	19	21

6. Examples. In this section we present specific examples of each of the 21 different wave patterns found in the solution to the Riemann problem above. Our purpose in doing so is to provide useful test problems for researchers interested in developing and testing numerical methods.

Figures 8–28 show the pictures of the Riemann problem solutions for each of the 21 possible kinds of waves. In each figure, we show the path between the left and right states in terms of velocity and stress, and show the chords on the stress functions in order to verify the admissibility conditions. We also show profiles of velocity, stress, displacement gradient, and characteristic speed as functions of the similarity variable x/τ. Whenever possible, we have also selected the example so that a contact discontinuity is included.

Note that the characteristic speeds are not uniquely defined at points on the yield curves. When this multiple value must be displayed in the figures, we picture the elastic value for half of the constant state, and the plastic value for the other half. (See, for example, Fig. 13.) Numerical results with a shock-capturing scheme may show oscillations between these two values, due to small numerical noise in the computations, or may choose to draw the jump in wavespeed at some point other than the center of the constant state.

7. Summary. In this paper we have presented the solution to a Riemann problem for longitudinal deformation of an elastic-plastic bar. The solution to this problem was made possible because of several features of the model. First, the system of partial differential equations could be formulated as a system of two first-order equations. Thus, in order to prove the global solution to the Riemann problem we only needed to show that a two-parameter family of curves covered all of state space. This argument becomes more difficult in higher dimensions, so discussions of combined compression and shear is more difficult. We also assumed that the material model did not allow for tensile failure. If inequality (11) is satisfied, then the discussion of the solution to the Riemann problem also becomes considerably more complicated. One approach to this problem might be to replace the stress-strain curves beyond some given level of strain with curves that smoothly and rapidly undergo a transition to zero stress (a kind of strain-softening). While such a model might be interesting to study for its mathematical structure, it is lacking many of the important physical effects that occur in material failure (e.g., strain-rate effects and thermal effects).

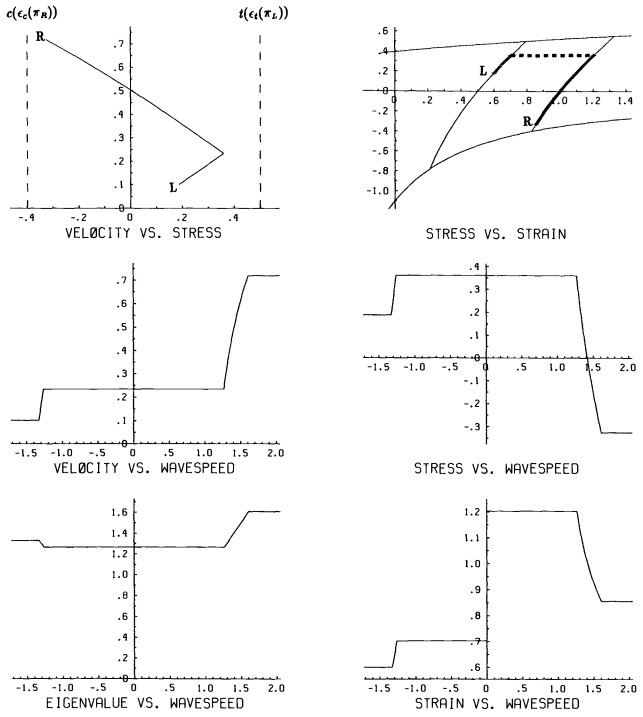


FIG. 8. *Region* 1 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$) *example solution.*

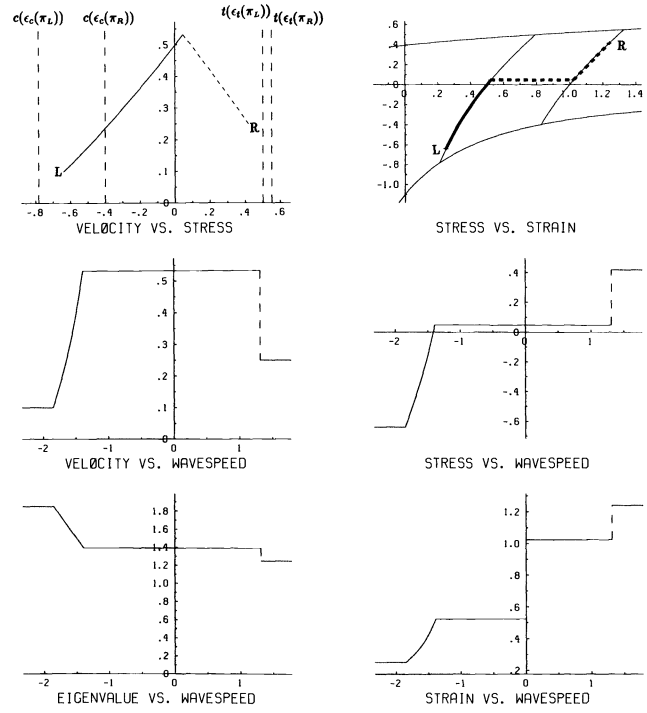


FIG. 9. *Region* 2 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$) *example solution.*

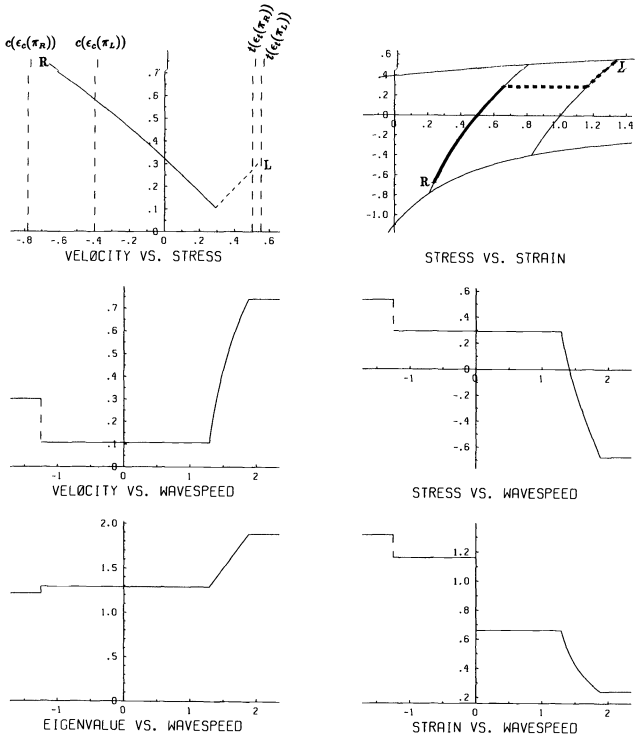


FIG. 10. *Region* 3 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$) *example solution.*

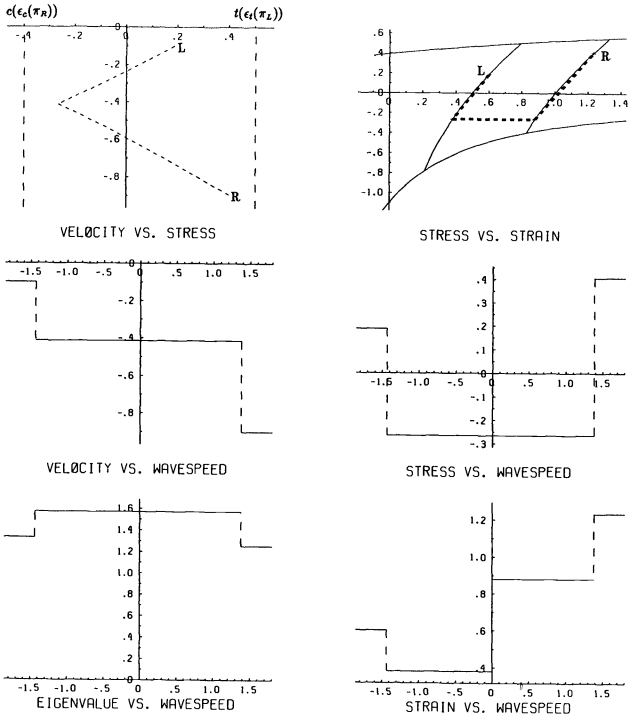


FIG. 11. *Region* 4 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$) *example solution.*

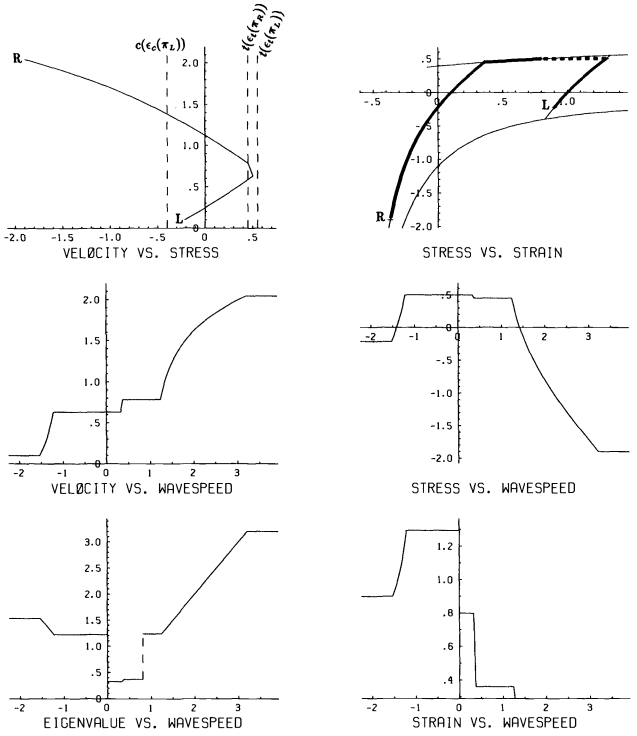


FIG. 12. *Region* 5 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{t}}_{+}\mathbf{R}^{\mathbf{e}}_{+}$) *example solution.*

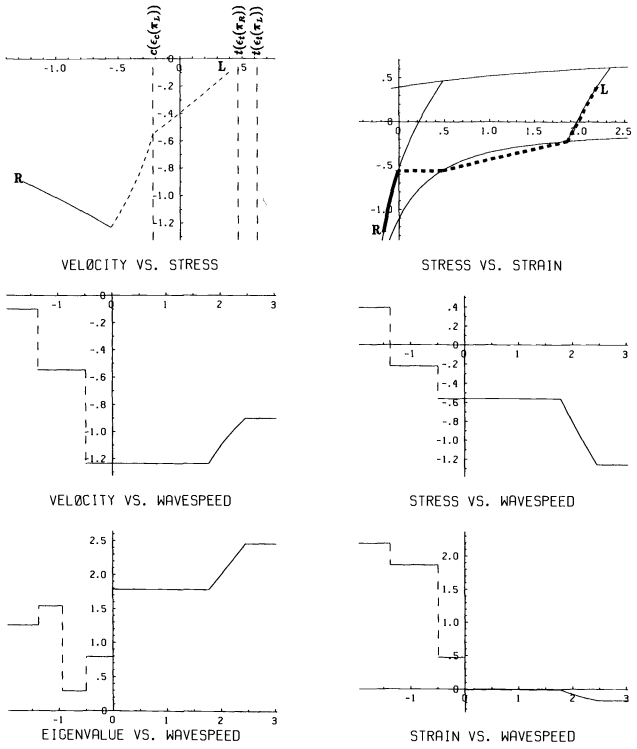


FIG. 13. *Region* 6 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{c}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$) *example solution.*

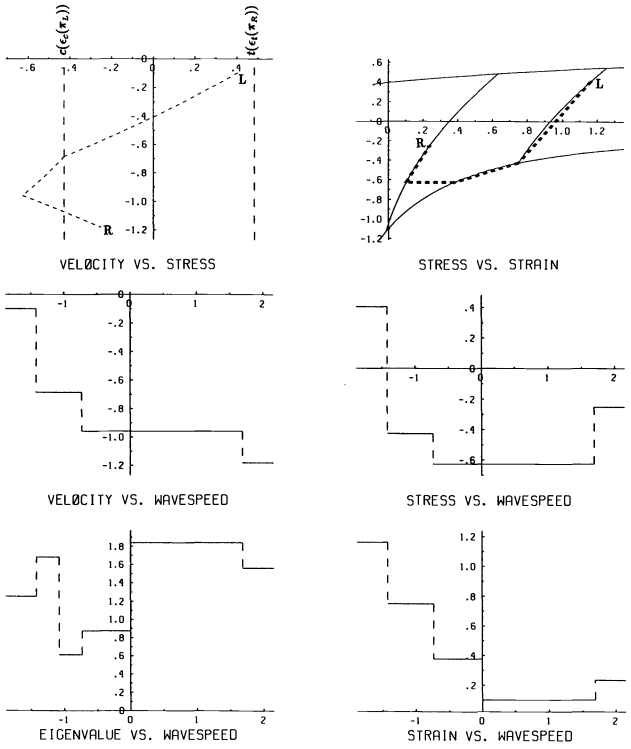


FIG. 14. *Region* 7 ($\mathbf{S^e_- S^c_- S^e_+}$) *example solution.*

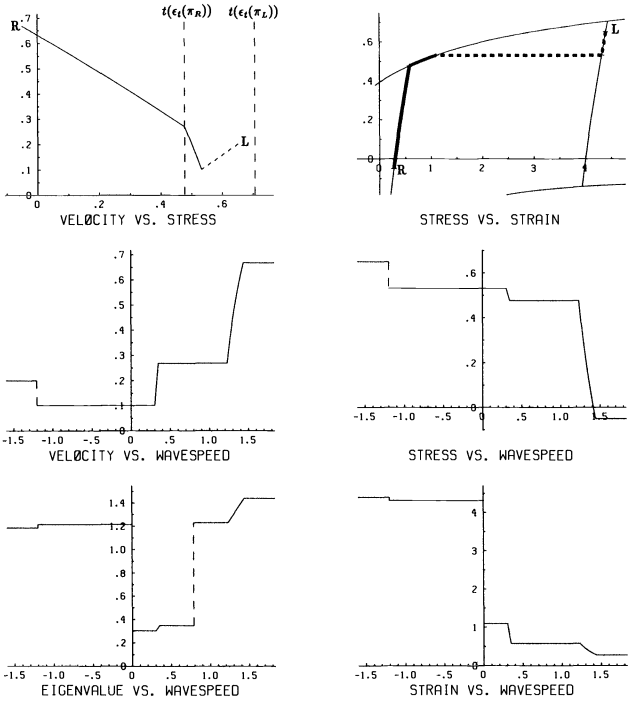


FIG. 15. *Region* 8 ($\mathbf{S^e_- R^t_+ R^e_+}$) *example solution.*

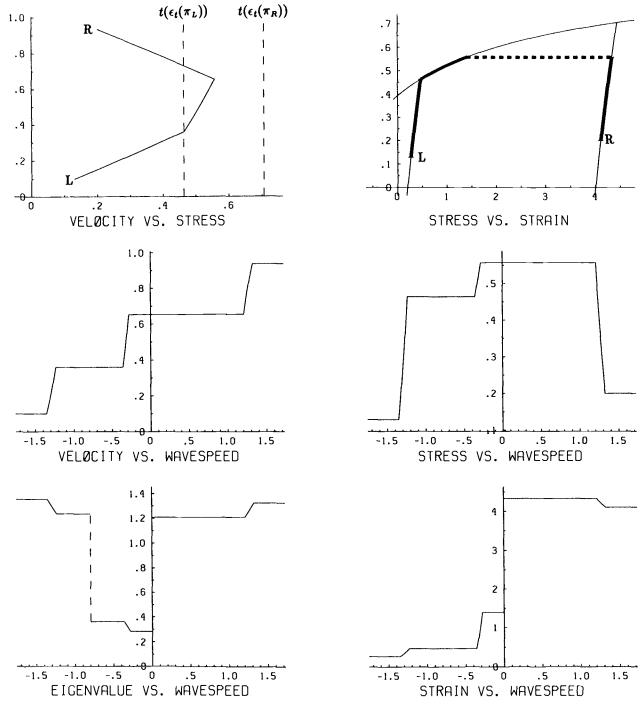


FIG. 16. *Region* 9 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{t}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$) *example solution.*

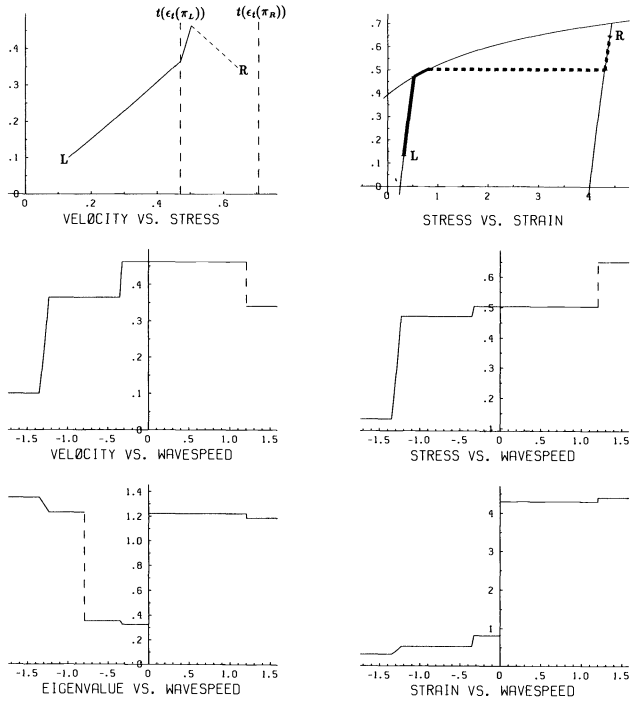


FIG. 17. *Region* 10 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{t}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$) *example solution.*

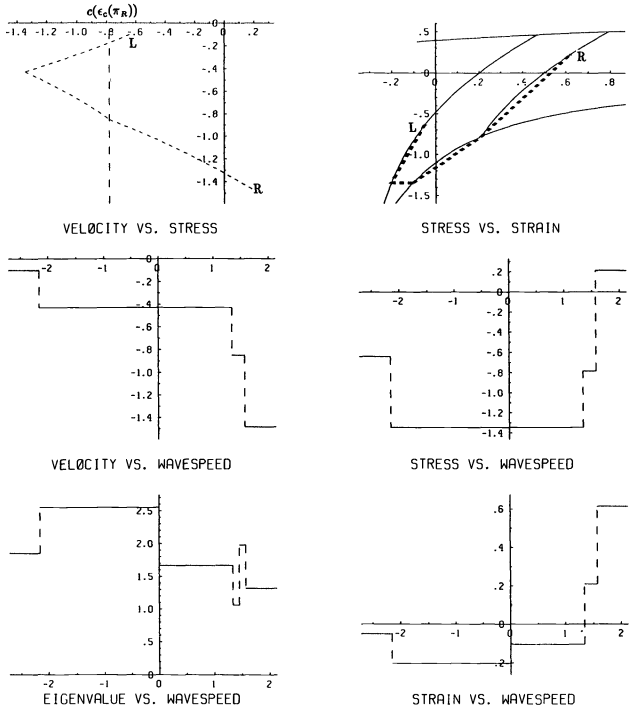


FIG. 18. *Region* 11 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{c}}_{+}\mathbf{S}^{\mathbf{e}}_{+}$) *example solution.*

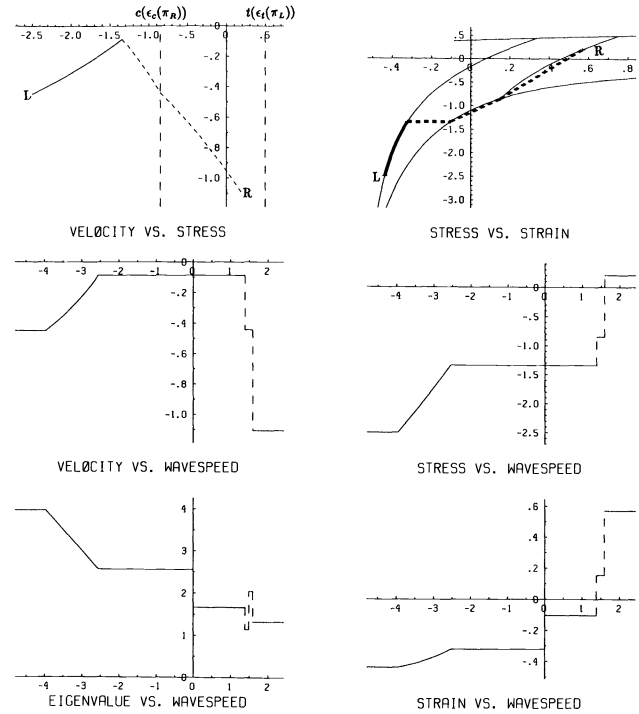


FIG. 19. *Region* 12 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{c}}_{+}\mathbf{S}^{\mathbf{e}}_{+}$) *example solution.*

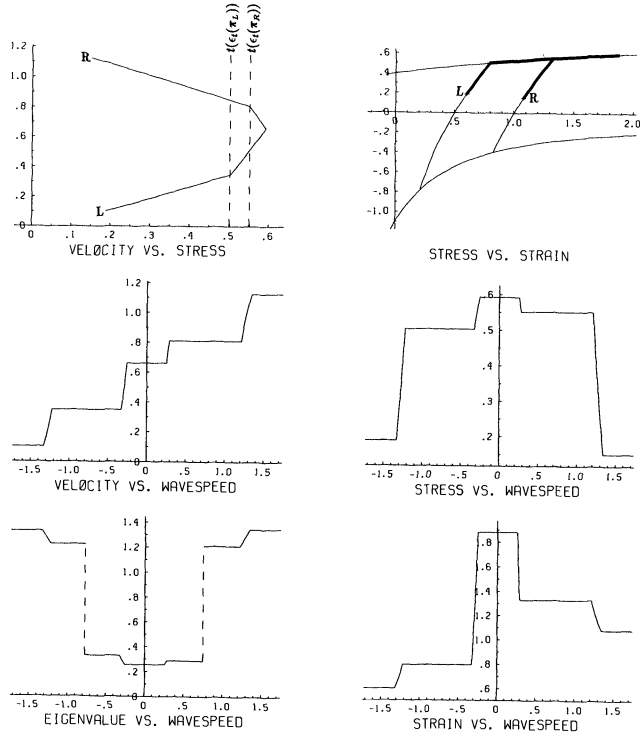


FIG. 20. *Region* 13 ($\mathbf{R}^{e}_{-}\mathbf{R}^{t}_{-}\mathbf{R}^{t}_{+}\mathbf{R}^{e}_{+}$) *example solution.*

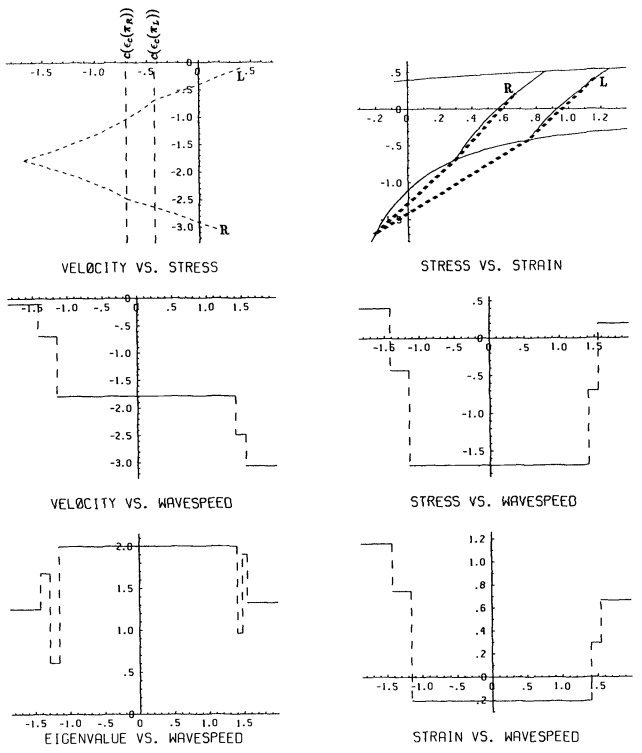


FIG. 21. *Region* 14 ($\mathbf{S}^{e}_{-}\mathbf{S}^{c}_{-}\mathbf{S}^{c}_{+}\mathbf{S}^{e}_{+}$) *example solution.*

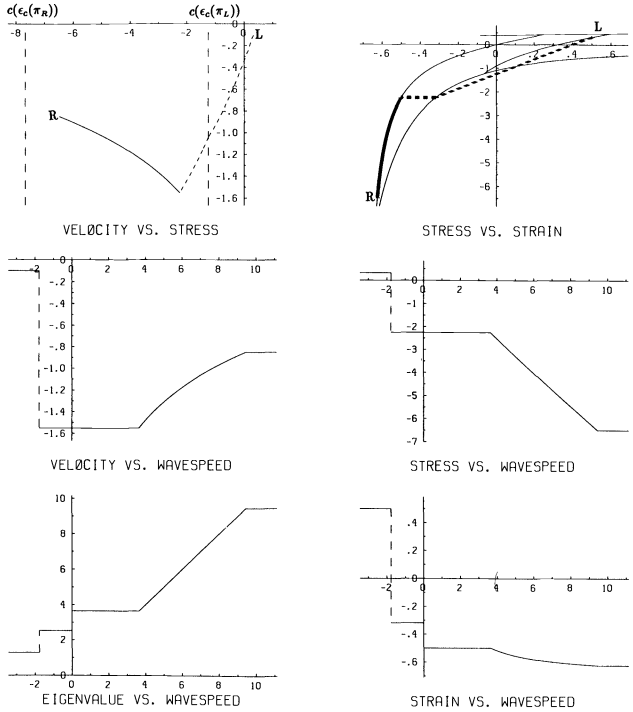


FIG. 22. *Region* 15 ($\mathbf{S}^{\mathbf{s}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$) *example solution.*

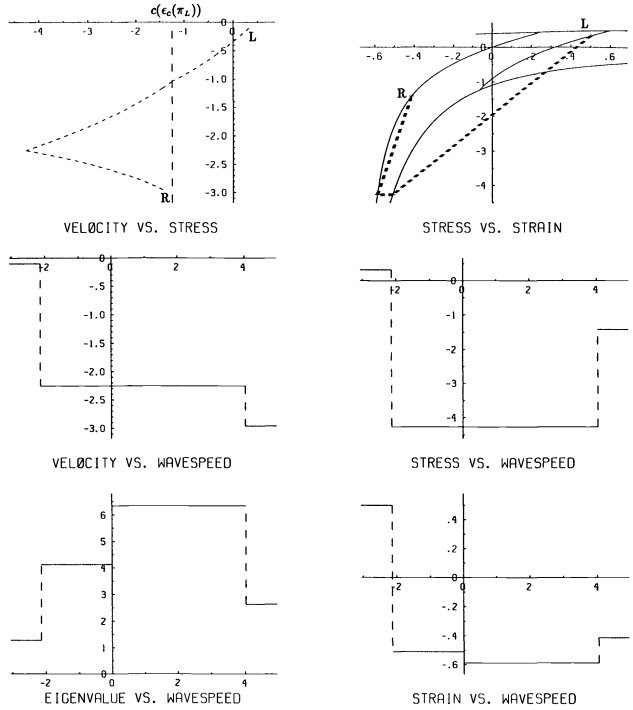


FIG. 23. *Region* 16 ($\mathbf{S}^{\mathbf{s}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$) *example solution.*

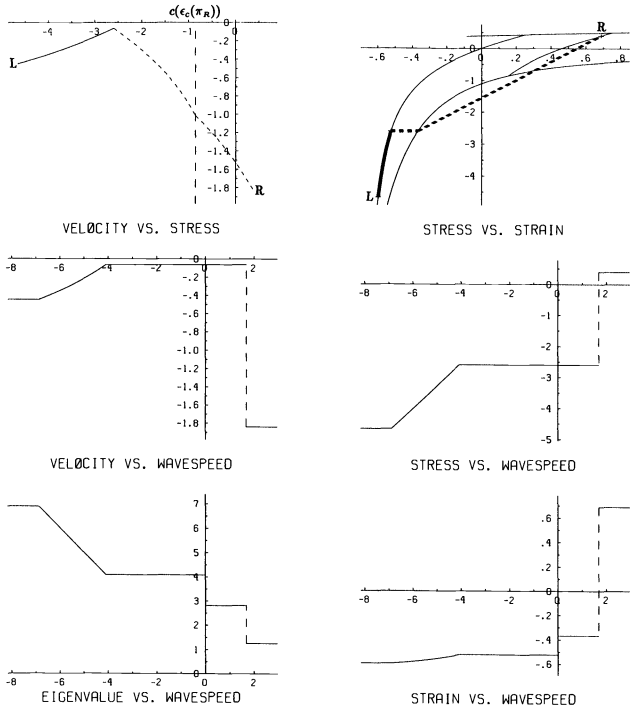


FIG. 24. *Region* 17 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{s}}_{+}$) *example solution.*

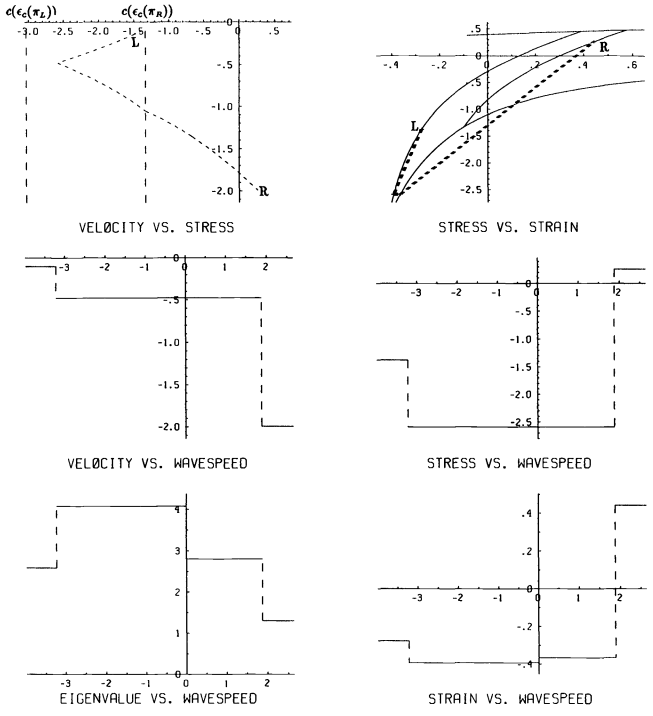


FIG. 25. *Region* 18 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{s}}_{+}$) *example solution.*

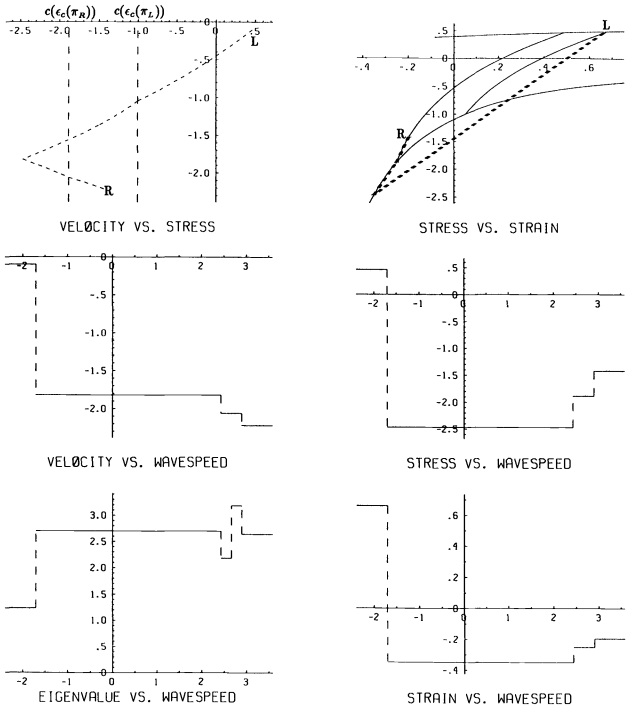


FIG. 26. *Region* 19 ($\mathbf{S^s_- S^c_+ S^e_+}$) *example solution.*

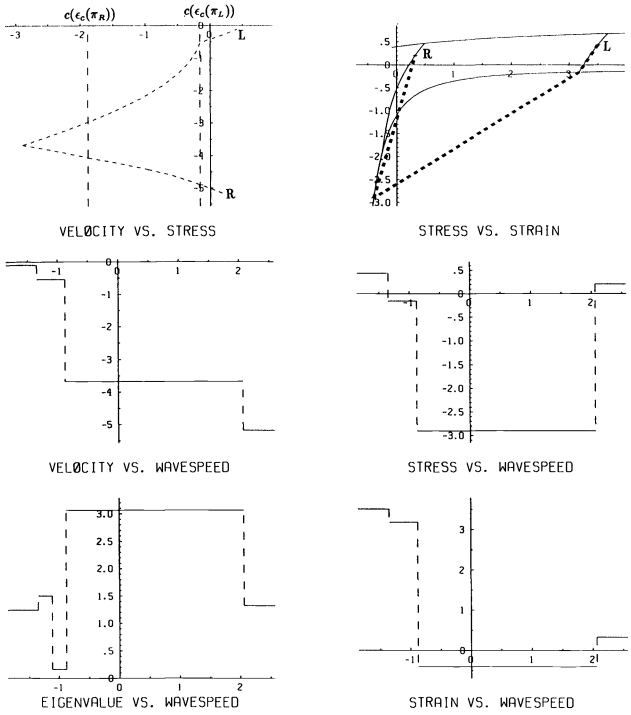


FIG. 27. *Region* 20 ($\mathbf{S^e_- S^c_- S^s_+}$) *example solution.*

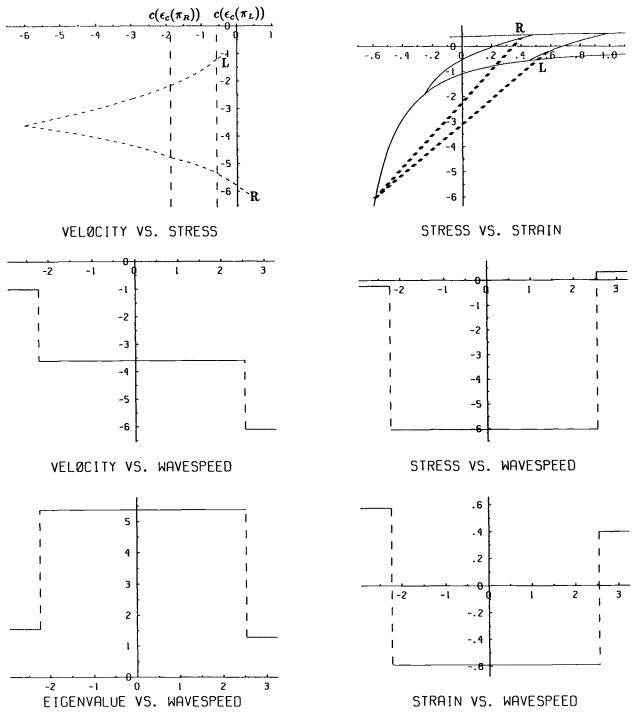


FIG. 28. *Region* 21 ($\mathbf{S}^{\mathbf{s}}_{-}\mathbf{S}^{\mathbf{s}}_{+}$) *example solution.*

TABLE 2
Parameters for Figs. 8 *to* 28.

Region	π_L	π_R	ϵ_L	v_L	σ_R	v_R
1 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$)	0.5	1.	0.6	0.1	-0.3259	0.72
2 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$)	0.5	1.	0.25	0.1	0.418	0.25
3 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$)	1.	0.5	1.315	0.3	-0.6762	0.7405
4 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$)	0.5	1.	0.6	-0.1	0.406	-0.902
5 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{t}}_{+}\mathbf{R}^{\mathbf{e}}_{+}$)	1.	0.1	0.9	0.1	-1.9	2.044
6 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{c}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$)	1.967	0.217	2.19	-0.1	-1.26	-0.9
7 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{c}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$)	0.93	0.35	1.161	-0.1	-0.252	-1.181
8 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{t}}_{+}\mathbf{R}^{\mathbf{e}}_{+}$)	4.	0.3	4.397	0.2	-0.05	0.668
9 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{t}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$)	0.2	4.	0.2673	0.1	0.2	0.937
10 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{t}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$)	0.25	4.	0.3184	0.1	0.65	0.34
11 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{c}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$)	0.2	0.5	-0.05	-0.1	0.214	-1.484
12 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{c}}_{+}\mathbf{S}^{\mathbf{e}}_{+}$)	0.08	0.46	-0.43735	-0.45	0.205	-1.106
13 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{R}^{\mathbf{t}}_{-}\mathbf{R}^{\mathbf{t}}_{+}\mathbf{R}^{\mathbf{e}}_{+}$)	0.5	1.	0.6	0.1	0.151	1.125
14 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{c}}_{-}\mathbf{S}^{\mathbf{c}}_{+}\mathbf{S}^{\mathbf{e}}_{+}$)	0.93	0.56	1.16	-0.1	0.196	-3.063
15 ($\mathbf{S}^{\mathbf{s}}_{-}\mathbf{R}^{\mathbf{e}}_{+}$)	0.35	0.	0.5	-0.1	-6.5	-0.85
16 ($\mathbf{S}^{\mathbf{s}}_{-}\mathbf{S}^{\mathbf{e}}_{+}$)	0.32	0.	0.5	-0.1	-1.422	-2.965
17 ($\mathbf{R}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{s}}_{+}$)	0.	0.46	-0.5972	-0.45	0.4	-1.841
18 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{s}}_{+}$)	0.13	0.3	-0.2764	-0.1	0.26	-1.993
19 ($\mathbf{S}^{\mathbf{s}}_{-}\mathbf{S}^{\mathbf{c}}_{+}\mathbf{S}^{\mathbf{e}}_{+}$)	0.392	0.217	0.66	-0.1	-1.423	-2.221
20 ($\mathbf{S}^{\mathbf{e}}_{-}\mathbf{S}^{\mathbf{c}}_{-}\mathbf{S}^{\mathbf{s}}_{+}$)	3.25	0.217	3.5	-0.1	0.207	-5.181
21 ($\mathbf{S}^{\mathbf{s}}_{-}\mathbf{S}^{\mathbf{s}}_{+}$)	0.683	0.217	0.575	-1.	0.325	-6.106

Our assumption of genuine nonlinearity was principally a matter of convenience. The techniques for handling loss of genuine nonlinearity are discussed in [11],[22]; the interested reader can see from those papers that the discussions regarding the global solution of Riemann problems are complicated. However, genuine nonlinearity does not greatly complicate the description of the solution to a specific Riemann problem. In fact, the chord condition (9) is sufficient to describe the solution of Riemann problems for generalizations of this model involving a loss of genuine nonlinearity.

This discussion points out the fact that the analysis in this paper can be extended to uniaxial displacements described by more general material models. For example, it is possible to determine curves similar to those in Fig. 1 by integrating hypoelastic material models involving plasticity and work-hardening. Typically, these models are not intended to describe the full spectrum of strains, and so it is not possible to provide them with a complete global solution to Riemann problems. However, for impact velocities and initial strains well within the realm of intended application of these models, it is possible to use the techniques in this paper to derive theoretically exact solutions to Riemann problems. This can be of use in validating finite difference methods for materials described by these models. As an example, we suggest that the reader see [20].

Finally, we note that Goursat problems are a simple subcase of Riemann problems. Thus, it is possible to understand the response of materials to which normal stresses or velocities are applied continuously from initial time at one end, by taking the appropriate forward- or backward-moving wave family from the analysis above.

REFERENCES

[1] S. S. ANTMAN AND W. G. SZYMCZAK, *Nonlinear elastoplastic waves*, Contemp. Math., 100 (1989), pp. 27–54.
[2] N. CRISTESCU, *Dynamic Plasticity*, North-Holland, Amsterdam, the Netherlands, 1967.
[3] C. M. DAFERMOS, *The entropy rate admissibility criterion for solutions of hyperbolic conservation laws*, J. Differential Equations, 14 (1973), pp. 202–212.
[4] J. GLIMM, *Solutions in the large for nonlinear hyperbolic systems of equations*, Comm. Pure Appl. Math., 18 (1965), pp. 697–715.
[5] J. GLIMM, E. ISAACSON, D. MARCHESIN, AND O. MCBRYAN, *Front tracking for hyperbolic systems*, Adv. in Appl. Math., 2 (1981), pp. 91–119.
[6] S. GODUNOV, *Difference methods for the numerical calculation of the equations of fluid dynamics*, Mat. Sb., 47 (1959), pp. 271–306.
[7] J. HADAMARD, *Leçons sur la propagation des ondes et les equations de l'hydrodynamique*, Hermann, Paris, 1903; reprinted by Chelsea Publishing Company, New York, 1949.
[8] B. L. KEYFITZ AND H. C. KRANZER, *A system of non-strictly hyperbolic conservation laws arising in elasticity theory*, Arch. Rat. Mech. Anal., 72 (1980), pp. 220–241.
[9] W. KOSIŃSKI, *Field Singularities and Wave Analysis in Continuum Mechanics*, Ellis Horwood Limited, Chichester, U.K., 1986.
[10] P. LAX, *Hyperbolic systems of conservation laws* II, Comm. Pure Appl. Math., 10 (1957), pp. 537–566.
[11] T. P. LIU, *The Riemann problem for general systems of conservation laws*, J. Differential Equations, 18 (1975), pp. 218–234.
[12] J. MANDEL, *Ondes plastiques dans un milieu indéfini à trois dimensions*, J. Méc., 1 (1962), pp. 13–30.
[13] O. A. OLEINIK, *On the uniqueness of the generalized solution of Cauchy problem for nonlinear system of equations occurring in mechanics*, Uspekhi Mat. Nauk. (N.S.), 12 (1957), pp. 3–73.
[14] G. A. POPE, *The application of fractional flow theory to enhanced oil recovery*, Society of Petroleum Engineers J., 20 (1980), pp. 191–205.
[15] R. RICHTMYER AND K. MORTON, *Difference Methods for Initial Value Problems*, Interscience, New York, 1967.

[16] M. SHEARER, *The Riemann problem for a class of conservation laws of mixed type*, J. Differential Equations, 46 (1982), pp. 426–443.

[17] ———, *Admissibility criteria for shock wave solutions of a system of conservation laws of mixed type*, Proc. Royal Soc. Edinburgh, Sect. A, 93 (1983), pp. 233–244.

[18] ———, *Nonuniqueness of admissible solutions of Riemann initial value problems for a system of conservation laws of mixed type*, Arch. Rational Mech. Anal., 93 (1986), pp. 45–59.

[19] A. TANG AND T. C. T. TING, *Wave curves for the Riemann problem of plane waves in isotropic elastic solids*, Internat. J. Engrg. Sci., 25 (1987), pp. 1343–1381.

[20] J. TRANGENSTEIN, *A second-order algorithm for the dynamic response of soils*, Impact of Computing in Science and Engineering, 2 (1990), pp. 1–39.

[21] C. TRUESDELL AND W. NOLL, *The nonlinear field theories of mechanics*, in Handbuch der Physik, Vol. III/3, Springer-Verlag, Berlin, New York, 1965.

[22] B. WENDROFF, *The Riemann problem for materials with nonconvex equations of state. I. Isentropic flow*, J. Math. Anal. Appl., 38 (1972), pp. 454–466.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 208–220, January 1991
© 1991 Society for Industrial and Applied Mathematics
011

TIMELY COMMUNICATIONS

Under the "timely communications" policy for the SIAM Journal on Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of the journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

NOTE ON THE PARALLEL EFFICIENCY OF THE FREDERICKSON–MCBRYAN MULTIGRID ALGORITHM*

NAOMI H. DECKER†

Abstract. Standard multigrid algorithms must lead to processor idle time on large-scale parallel computers because the coarsest grids have fewer points than processors. In some cases, this may be considered to be a disadvantage. Frederickson and McBryan [*Multigrid Methods*, Marcel Dekker, New York, 1988] show that retaining all points on all grid levels (using all processors) can lead to a "superconvergent" algorithm in that a very good convergence rate is obtained. Has the "parallel superconvergent" multigrid algorithm (PSMG) of Frederickson and McBryan solved the problem of implementing multigrid on a massively parallel single-instruction-multiple-data (SIMD) architecture? How much can be gained by retaining all points on all grid levels, keeping all processors busy?

The purpose of this note is to compare the parallel efficiency of the PSMG algorithm to a standard multigrid algorithm. It is shown that the perfect processor utilization and the good convergence rates of the PSMG algorithm do lead to a more efficient algorithm for the special case of one (or fewer) grid points per processor. Normalized computation and communication requirements are given, so that the two types of algorithms can be compared directly.

Key words. parallel multigrid, parallel multilevel algorithms, multigrid

AMS(MOS) subject classifications. 65F05, 65N05, 68Q25

1. Introduction.

The parallel multigrid algorithm of Frederickson and McBryan [4] is frequently mentioned as an efficient method for implementing multigrid on a fine-grained single-instruction-multiple-data (SIMD) architecture, specifically the connection machine. At first glance, their use of multiple coarse grids (using the same number of grid points on all grid levels) looks very promising since

1. The processor utilization rate is high;
2. The convergence rate is very good, at least for simple model problems;
3. It eliminates the aliasing between modes, so that a lower order restriction may be used.

Just how much more efficient is this parallel "superconvergent" multigrid algorithm (PSMG) than the standard sequential algorithms implemented efficiently in parallel?

There is a limit to the available parallelism in a standard multigrid method. In the correction scheme, which is used for solving linear problems, the coarse grids are used to find error corrections to the current fine grid approximations of the solution. It is therefore inconsistent to do any additional work on a fine grid until the correction has been found on the coarse grid. This imposes a fixed grid visitation schedule. To take full advantage of a multigrid approach, we must use $\log(N)$ grid levels where N is the number of fine grid points. Since it takes a number of cycles proportional to $\log(N)$ to solve to discretization error (assuming a discretization with accuracy $O(h^q)$ for some q), the minimal total time for a multigrid solve by successive V-cycles is proportional to $\log^2 N$. For a full multigrid cycle (FMG), the minimal complexity

* Received by the editors December 6, 1989; accepted for publication (in revised form) October 6, 1990. This research was supported by the National Aeronautics and Space Administration (NASA) contract NAS1-18605.

† Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia 23665-5225.

is the same: an FMG cycle will take at least as many parallel computation steps as the number of grids visited, which is proportional to $\log^2 N$.

In most practical situations, the computation on the grid levels where there are more grid points than processors dominates the total computation cost. As long as the communication costs can be overlapped, or are small relative to the computation time, idle processors on the coarsest grids are not an issue. In fact, as observed in [2], for very large scale problems solved on machines with powerful nodes, the time spent on the coarsest grids is relatively insignificant. In the extreme case of a very fine grained architecture such as the connection machine, where the number of processors could be as large as the number of points on the fine grid, the total number of grids visited, i.e., $\log^2 N$, is a lower bound on the parallel computation time. For this special situation, most of these processors will be idle on the coarser grids. Using N processors, this polylog complexity in computation time can be achieved, though communication time becomes a major factor on most architectures as the number of grid levels becomes larger than four or five, as observed in [8].

For many model problems, the $\log^2 N$ time is not optimal. For example, the Poisson equation on a rectangular domain, with Dirichlet boundary conditions, can be solved in $C \log N$ time using N processors. Novel multigrid methods have been proposed by Gannon and Van Rosendale [6] and Greenbaum [7], which attempt to gain more parallelism by altering the standard grid visitation schedule. The PSMG algorithm keeps the standard grid scheduling, and thus will have, at best, $\log^2 N$ complexity.

Nevertheless, the hope of PSMG is that there is *useful* work to be done by the otherwise idle processors on the coarse levels and that the total solution times can be reduced by a constant factor. It is claimed that the payoff is much faster convergence at no added computational cost [5]. Although it is correct to say that the extra coarse grid problems can be solved at no extra cost, it is not clear whether this extra work is itself responsible for the faster convergence. The other components of the PSMG algorithm, such as the interpolation and relaxation, do affect both the convergence rate and the overall computational cost of the algorithm.

In this note, it is shown that the better convergence rate of PSMG is achieved using relatively expensive interpolation and relaxation methods. A careful calculation of computation and communication costs, with a normalization based on the convergence rates, shows that, for the Poisson equation model problem in two dimensions, for which the PSMG algorithm was originally devised, PSMG is more efficient in computation than a standard sequential algorithm implemented efficiently in parallel. The standard algorithm is a simple red-black Gauss–Seidel with the usual restriction and interpolation operators.

It should also be noted that reducing the convergence rate at the expense of additional computation and communication cost is usually not worthwhile except in this special case where we are assuming there are as many processors as grid points. In particular, using V cycles instead of an FMG algorithm can be more efficient if the fine grid computation costs do not dominate the intermediate and coarse grid computations. To see how this can happen, consider an FMV cycle (an FMG using V cycles). Under the assumptions of one processor per grid point on the finest grid, the number of grids visited in an FMV cycle, prior to the very last V-cycle, is

$$\log^2(N) + \text{lower order terms}$$

and the number of grids visited in the last V-cycle is

$$2\log(N) - 1.$$

Thus the minimal complexity of one FMV cycle is $\log^2(N)$ plus lower order terms (l.o.t).

If we are just using V-cycles, it will take $C \log(N) + (\text{l.o.t.})$ V-cycles to reduce the error down to the truncation error, where the constant, C, depends on the convergence rate per cycle. Thus the overall minimal complexity is $2C \log^2(N) + (\text{l.o.t.})$, and can be reduced by a constant factor by improving the convergence rate.

2. Assumptions. In order to compare parallel superconvergent multigrid (PSMG) to a standard multigrid cycle implemented in parallel, it is clearly insufficient to give processor utilizations or megaflop rates. We must have a measure of processor utilization which reflects the amount of useful work being done.

For direct methods we can estimate the total number of computation and communication steps required to solve the problem completely. Similarly, for the FMG algorithm (full multigrid algorithm), we can count the total number of steps required to solve the problem to within truncation (discretization) error. When using an iterative technique, like a V-cycle multigrid algorithm, the efficiency of the algorithm is usually expressed as convergence rate per iteration, combined with the cost per iteration. In the case of parallel algorithms, the cost per iteration is the parallel cost, i.e., the cost per node. This information generally suffices for comparing similar multigrid algorithms, since the error reduction tends to be fairly uniform throughout the iteration process.

The comparison analysis also depends on the model of computation. We wish to compare standard multigrid to PSMG on its home turf, a massively parallel machine in which there are at least as many processors as there are grid points (unknowns) in the discrete equations. If there were fewer processors than grid points, then the efficiency of PSMG would drop relative to standard red-black algorithms.

As well as being dependent on the machine architecture and the data dependencies dictated by the underlying partial differential equation, the computation and communication counts are very sensitive to algorithmic details. There are the obvious choices to be made about interpolation and restriction operators and the order of the red-black sweeps. There is also the usual trade-off between the computation cost and the communication cost. In presenting the standard methods, where there are a number of variations on the basic method, we consider only those algorithms which are relatively simple and can be easily compared to the PSMG algorithm. Although there are ways to further optimize the standard multigrid algorithm, either reducing communication at the expense of computation or vice versa, we have sacrificed a little efficiency in the interest of simplicity and practicality.

2.1. Machine assumptions. For determining the parallel computation and communication costs we have made the following assumptions:

1. Each processor can fetch only one value at a time;
2. Each processor can do only one add/multiply at a time;
3. At any given time, every processor must either execute the same instruction as all other processors or do nothing;
4. Each processor can use locally stored constants, which may differ from processor to processor, to be used in the computational steps;
5. There is no overlap of computation and communication.

We also assume, as on the connection machine, that the cross data communication traffic from the simultaneous relaxations on the multiple coarse grids does not degrade the overall performance of PSMG relative to the standard RB algorithms. The cost of storage in a register is included in the cost of fetching a value or in the floating point operation. This is a valid assumption on, for example, the connection machine

(see [12]).

2.2. Work measures. Using the above assumptions, we define our work units as follows.

Computation. We define one (parallel) computation step to be one instruction executed by a subset of the processors involving at most one add and one multiply.

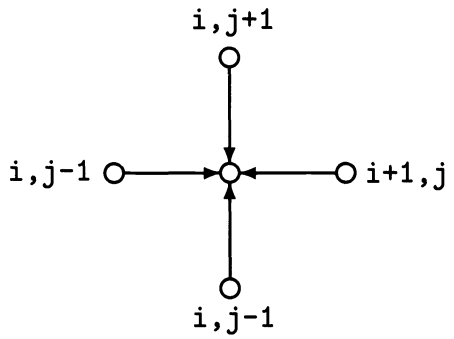
Communication. We define one (parallel) communication step on level k to be one instruction executed by a subset of the processors involving at most one fetch of a value from a nearest neighbor (on level k) processor.

To illustrate the method of counting computation and communication steps and to give an example of the type of optimizing which has been assumed in the comparisons of the two algorithms, consider the cost of finding the average of values of nearest neighbors at every grid point,

$$\bar{u}_{ij} = (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1})/4.$$

We assume that there are exactly as many grid points as processors. Suppose that each grid point, (i, j), has been assigned to a processor, p_{ij}, and that p_{ij} has u_{ij} in its local memory. At each processor, p_{ij}, the following four operations can be performed simultaneously, the average being stored in $\bar{u}$:

1. fetch $u_{i,j+1}$; store in t_{ij}
2. fetch $u_{i-1,j}$; add to t_{ij}; store in t_{ij}
3. fetch $u_{i,j-1}$; add to t_{ij}; store in t_{ij}
4. fetch $u_{i+1,j}$; add to t_{ij} and divide by 4; store in $\bar{u}_{ij}$



Using the above definitions, this algorithm takes three computation and four communication steps.

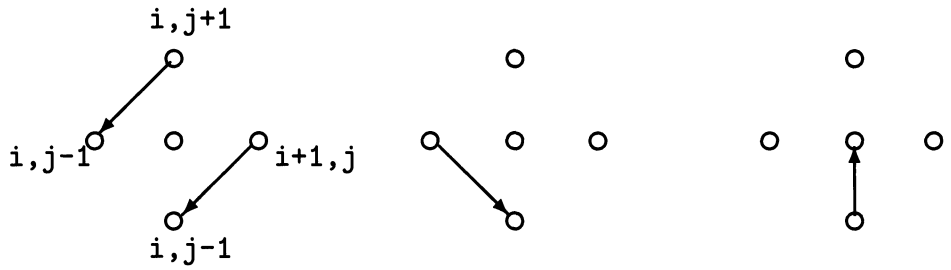
One of the addition steps can be done in parallel. If the third and fourth steps are replaced by one step, such as:

1. fetch $u_{i,j+1}$; store in t_{ij}
2. fetch $u_{i-1,j}$; add to t_{ij}; store in t_{ij}
3. fetch $t_{i+1,j-1}$; add to t_{ij} and divide by 4; store in $\bar{u}_{ij}$

then the number of computation steps is reduced to two, and, assuming the fetch of a diagonal neighbor must be done in two communication steps, the communication is still done in four steps. Note that, if the cost of a diagonal neighbor communication is the same as the cost of a nearest neighbor communication, then the same calculation can be done in two computation and three communication steps. A more straightforward way to implement this is as follows:

1. fetch $u_{i+1,j+1}$; add to u_{ij}; store in t_{ij}
2. fetch $t_{i-1,j+1}$; add to t_{ij} and divide sum by 4; store in t_{ij}

3. fetch $t_{i,j-1}$; store in $\bar{u}_{ij}$



Thus, depending on the machine architecture and communication software, the cost of the standard algorithm could be further reduced.

The nine-point average given by

$$a_{ij} = \frac{1}{16}(4u_{ij} + 2u_{i+1,j} + 2u_{i-1,j} + 2u_{i,j+1} + 2u_{i,j-1} + u_{i+1,j+1} + u_{i-1,j+1} + u_{i+1,j+1} + u_{i+1,j-1})$$

can be computed as follows. At each processor, p_{ij}, the following four operations can be performed simultaneously, the average being stored in a_{ij}:

1. fetch $u_{i,j+1}$; add to u_{ij}; store in a_{ij}
2. fetch $a_{i-1,j}$; add to a_{ij}; store in a_{ij}
3. fetch $a_{i+1,j}$; add to a_{ij}; store in a_{ij}
4. fetch $a_{i,j-1}$; add to a_{ij} and divide sum by 16; store in a_{ij}

This particular nine-point formula is relatively inexpensive, taking only four computation steps and four communication steps. Any other nonzero choice of the weights would be more expensive.

We also note the following:

- The algorithm we consider involves a fixed number of iterations of the relaxation to be performed on each grid level. Thus approximately the same amount of time is spent on each grid.
- Because of the varying length scales, the communication costs are grid dependent. It is crucial to the arguments of this paper that *all comparisons of communication costs are made between corresponding levels.*

2.3. Efficiency measures. A usual measure of efficiency [10, p. 53] is

$$op(\epsilon) := w\frac{\log \epsilon}{\log \rho} \tag{1}$$

where ρ is an estimate of either the asymptotic convergence factor or some norm convergence factor and w is some measure of the work per iteration. In our case, w is either the number of communication steps or computation steps to be performed per V cycle at each grid level.

Using common logarithms, this normalizes the number of computation or communication steps giving us the number of steps required per factor of ten reduction in the error. We take ρ to be the multigrid asymptotic convergence rate.

3. The algorithms. Our comparison is made for the same model problem for which the original PSMG algorithm was proposed: Poisson's equation in the unit square with periodic boundary conditions.

We assume that on each intermediate grid the initial iterate is set to zero. Since the same strategy can be used for both methods to solve, or approximately solve, the equations on the coarsest grid(s), it is enough to compare the computation and communication requirements on all other grids. The comparison will be based on a count of the number of computation and communication steps required for any two intermediate grids: for prerelaxation (if used), coarse grid correction (residual calculation, residual restriction, the interpolation of the coarse grid correction to the fine grid and update of u^h) and postrelaxation. These computation and communication counts per grid level, per cycle, are then normalized to give the work units per factor of ten reduction of the error.

3.1. PSMG. The "Z" relaxation scheme used by Frederickson and McBryan in PSMG is given by

$$u^h \leftarrow u^h + \begin{bmatrix} z_2 & z_1 & z_2 \\ z_1 & z_0 & z_1 \\ z_2 & z_1 & z_2 \end{bmatrix} r^h$$

where

$$r^h = f^h - L^h u^h,$$

for some parameters $z_i, i = 0, 1, 2$ and L^h is a discretization of the Laplacian.

In the PSMG algorithm, there is no smoothing in the fine to coarse grid portion of the cycle. In fact, assuming an initial guess of zero on each grid, the residuals need only be computed on the finest grid. In addition, the residual transfer is straight injection, and hence the residual on any coarse grid is precisely the residual at that point on the fine grid. In [4], two coarse-to-fine grid transfer operators were considered. The nine point (3×3) prolongation, when combined with the averaging of the four coarse corrections, is given by

$$v^h \leftarrow \bar{q}_0 \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} v^{2h}$$

where the stencil is evaluated on the appropriate coarse grid. A 25-point (5×5) prolongation/correction operator, Q, is also considered, given by

$$v^h \leftarrow \begin{bmatrix} q_5 & q_4 & q_3 & q_4 & q_5 \\ q_4 & q_2 & q_1 & q_2 & q_4 \\ q_3 & q_1 & q_0 & q_1 & q_3 \\ q_4 & q_2 & q_1 & q_2 & q_4 \\ q_5 & q_4 & q_3 & q_4 & q_5 \end{bmatrix} v^{2h}.$$

In counting the computation and communication requirements of these algorithms, we assume that the optimal values of the z_i, $\bar{q}_0$, and q_i are known.

We consider the four different versions of PSMG presented in [4]. The first two are used to solve the equations resulting from the five-point discretization of the Laplacian. The algorithms PSMG5-9 and PSMG5-25 will refer to the methods using the nine- and the 25-point "Q," respectively.

The other two methods apply to the "Mehrstellen" discretization of the Laplacian

$$\frac{-1}{6h^2} \begin{bmatrix} -1 & -4 & -1 \\ -4 & 20 & -4 \\ -1 & -4 & -1 \end{bmatrix}.$$

TABLE 1
Comparisons on intermediate grid levels.

	Conv. rate	Comp.	Comm.	Normalized Comp.	Normalized Comm.
PSMG					
PSMG5-9	.2115	15	13	22.3	19.3
PSMG5-25	.0831	26	17	24.1	15.7
PSMG9-9	.0275	18	16	11.6	10.3
PSMG9-25	.0045	29	20	12.3	8.5
Standard					
RBTRB	.19	13	21	18.0	29.1

This particular nine-point discretization is of interest because it can lead to a fourth-order accurate discretization of Laplace's equation (and Poisson's equation, if the right-hand side is treated properly). As before, the two methods differ by the size of the stencil for Q: nine- or 25-point. We refer to these as PSMG9-9 and PSMG9-25. It is the PSMG9-25 that was shown in [4] to have the "superconvergent" convergence rate. The other, less expensive, PSMG methods have less impressive convergence rates.

The number of computation steps and communication steps is different for each of the methods. For example, PSMG5-9 requires a total of 15 computation and 13 communication steps on intermediate grids and 19 computation and 16 communication steps on the finest grid. See the Appendix for details. Because the nine-point "Q" is exactly the nine-point average given in the previous section, it can be evaluated at every grid point in only four computation and four communication steps. However, the costs of using a 25-point "Q," with the five arbitrary parameters, depend a great deal on implementation details. A naive algorithm would be very expensive, but there is an algorithm which reduces the costs to only 15 computation and eight communication steps. It is not clear whether further optimization is possible. For the operation counts of the four PSMG methods, see columns 3 and 4 of Table 1.

3.2. Standard multigrid. It is well known that Jacobi-like relaxations are not optimal when using standard multigrid as a fast Poisson solver for the five-point discretization on a sequential machine. One of the most efficient sequential algorithms is obtained by using red-black Gauss–Seidel (RB) sweeps as the relaxation scheme. The operation count of RB multigrid depends on the order in which the sweeps are performed. For example, if a black sweep precedes a residual transfer, the residual,which is then zero at black points, needs to be computed only at red points. The restriction operator can also ignore black point residual values. Moreover, interpolation and correction need only be done at the fine grid red points because the black point values will not influence the first black point sweep. As in Stüben and Trottenberg [11] for the sequential algorithms, we have considered various sweep order strategies, and invite the reader to try to find particular combinations which further reduce the parallel costs. We consider only the standard restriction and projection operators, namely, full weighting (FW), half weighting (HW) and their adjoints relative to the discrete L^2 inner product, FW* (bilinear interpolation), and HW*. The fine-to-coarse grid stencil for full weighting restriction is

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_k^{k+1}$$

and for half weighting is

$$\frac{1}{8}\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix}_k^{k+1}.$$

For example, consider using a black-red Gauss–Seidel iteration for the pre- and post-smoothing steps, FW restriction, and HW* projection. We denote the coarse grid correction operator by T and represent this particular algorithm (in "reverse" order, operator-style) by the notation $RBTRB$. The red sweep prior to the coarse grid correction ensures that the residual is zero at all red points, thus simplifying the restriction to a four point formula. The black sweep immediately following the coarse grid correction eliminates the need to compute the projected correction at black points. Thus the projection of the coarse grid correction at all red points which correspond to coarse grid points is simply its value on the coarse grid, and at the other red points it is zero. The RB T RB can be implemented with 13 parallel computation and 21 parallel communication steps per intermediate grid level and 17 computation and 24 communication steps on the finest level. See the Appendix for a detailed description of the algorithm. As observed earlier in this note, if a diagonal neighbor communication is the same cost as a nearest neighbor communication, then the intermediate grid level counts can be reduced to 13 parallel computation and 12 parallel communication steps. For our comparisons with PSMG, we assume no diagonal communication.

An RB algorithm is not as efficient for nine-point discretizations. Most of the simplifications obtained above for the five-point discretization disappear. Because we are assuming only one grid point per processor, the multicolor relaxation schemes are also more expensive. We do not give operation counts for these methods because, as explained in the following, there is no advantage to using nine-point stencils for the Laplacian on intermediate grid levels.

3.3. Comparisons. In order to compare the methods, we must normalize the computation and communication costs relative to the convergence properties of the methods.

For the five-point discretization, the optimal parameters for the "Z" relaxation when using a "smoother" in PSMG, are given in [4] and, from [4], we see that the multigrid convergence rate bound for PSMG5-9 is .2115. For the 25-point stencil, the multigrid convergence rate bound for PSMG5-25 is given as .0831. Similarly, for the Mehrstellen discretization, the multigrid convergence rate bounds are given in [4] as .02754 for PSMG9-9 and .00446 for PSMG9-25 . The convergence rates are listed in column 2 of Table 1. Table 1 also lists the asymptotic convergence rate bound of the RB standard multigrid algorithm, independent of the number of grid levels.

Using these convergence rates, we can normalize the costs (i.e., find the computation and communication costs per factor of ten reduction in error; see (1)). For example, the multigrid rate for the standard multigrid algorithm is .19. Thus, the

normalized costs will be scaled by the factor

$$\frac{\log_{10}(.100)}{\log_{10}(.19)} = \frac{-1}{\log_{10}(.19)} \approx 1.39.$$

Therefore, the 13 computation steps required to change the error by a factor of .19 is equivalent to approximately 18.0 computation steps required to change the error by a factor of .100, since

$$13(1.39) \approx 18.0.$$

It is very important to realize that the improvements in convergence rates can be deceptive. Consider two algorithms with identical computation and communication costs, the first with a convergence rate (i.e., error reduction rate) of .063 and the second with a convergence rate of .004 ($= .063^2$). Even though the convergence rate of the second appears to be more than a factor of ten better, it is only the ratio of their logarithms which is a valid comparison. Thus two iterations, each reducing the error by .063 is equivalent to one iteration reducing the error by .004. Thus the second algorithm is only twice as efficient as the first.

Finally, we note that the higher order accuracy of the Mehrstellen discretization is an advantage only on the finest grid. The coarse grid correction operators on all other levels should be chosen to optimize the reduction of the algebraic error. The use of high-order fine grid discretizations with lower order coarse grid discretizations is an example of the so-called "double-discretization" method (see [1]). A complete analysis of the particular case of using the five-point coarse grid discretizations with the Mehrstellen discretization on the finest grid was given in [9], in the algorithm called "MGH." Since we are assuming that there are as many processors as grid points on the finest level, approximately the same amount of time is spent on all grid levels. It therefore pays to use the five-point discretization on all of the intermediate and coarsest levels, using the more expensive, higher order stencil on only the finest grid. This must be done in such a way as to preserve the convergence rate while switching from one discretization to the other. Using, for example, two "Z" relaxation sweeps on the finest grid and the five-point Laplacian on the next grid, we can easily match the convergence rates of the five-point RB T RB method on the intermediate grid levels. Therefore, the RB T RB method can be used just as effectively for either the five-point or the Mehrstellen discretizations, incurring additional costs on only the finest grid level.

4. Conclusions. Given certain machine assumptions, a comparison has been made between the PSMG algorithm and a standard multigrid algorithm. In the special case of one grid point per processor, the PSMG algorithms can significantly reduce the communication costs relative to a standard red-black Gauss-Seidel multigrid algorithm.

We note that RB T RB is only one of many variants of the standard multigrid algorithm. Improved multigrid convergence rates result from changing the number of red and black relaxations or the order in which the red and black sweeps are done. The use of damping either in the relaxation or in the coarse grid correction can also be used to improve the convergence rate. We have not optimized over all possible algorithms. Our comparisons in this paper are intended only to give a rough guide of the approximate savings which can be expected when using a PSMG algorithm instead of standard multigrid.

Appendix. The operation counts for the various methods are sensitive to algorithmic details. The following algorithm counts assume that there is one point assigned

to each processor. The computation and communication counts use the machine assumptions of § 2.

PSMG5-9.

On level $k > 1$, do: [1]

1. If not at coarsest level, repeat steps 1–3 with $k \leftarrow k+1$, $h \leftarrow 2h$
2. At all points, compute nine point interpolation of u^{k+1}'s and store in u_{ij}^k:

$$u_{ij}^k \leftarrow \bar{q}_0 \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} u^{k+1}$$

number of computation steps = 4, number of communication steps = 4

3. Relax on all points:

$$u_{ij}^k \leftarrow u_{ij}^k + \begin{bmatrix} z_2 & z_1 & z_2 \\ z_1 & z_0 & z_1 \\ z_2 & z_1 & z_2 \end{bmatrix} r$$

where

$$r_{ij} \leftarrow f_{ij} - \frac{1}{h^2} \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix} u^k$$

number of computation steps = 11, number of communication steps = 9

Standard RB.

Color the grid points on every grid in a checkerboard manner, with red (R) and black (B) points, where the coarse grid points are a subset of the red points. For clarity, we present the analysis of operation counts in two phases. First we give a detailed version of the algorithm, followed by a possible parallel implementation of the algorithm.

On each grid level, $k > 1$:[2]

1. Do a black sweep, followed by a red sweep, starting with an initial guess = 0.

at B points:

$$u_{ij}^k \leftarrow \frac{h_k^2}{4} f_{ij}^k$$

at R points:

[1] N.B. On the finest grid, level one, there are four additional computation and three additional communication steps needed to compute the residual, since the initial iterate is not assumed to be zero.

[2] On the finest grid level, $k = 1$, the relaxation does not start with an initial guess of zero, so the relaxation and residual calculation do not simplify as on the coarse grids.

$$u_{ij}^k \leftarrow \frac{h_k^2}{4} f_{ij}^k + \frac{1}{4} \begin{bmatrix} & 1 & \\ 1 & & 1 \\ & 1 & \end{bmatrix} u^k = \frac{1}{4} \begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix} u^k$$

2. Compute the residual at black points (having just performed a red sweep, we know that it is zero at red points).

at B points:

$$r_{ij}^k \leftarrow f_{ij}^k - \frac{1}{h_k^2} \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix} u^k = \frac{1}{h_k^2} \begin{bmatrix} & 1 & \\ 1 & & 1 \\ & 1 & \end{bmatrix} u^k$$

3. Restrict (FW) the residual to the coarse grid. Notice that the restriction simplifies considerably, since the residual is zero at red points.

at R points which correspond to $k+1$ level points:

$$f_{ij}^{k+1} \leftarrow \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_k^{k+1} r^k = \frac{1}{8} \begin{bmatrix} & 1 & \\ 1 & & 1 \\ & 1 & \end{bmatrix} r^k$$

4. If not at coarsest level, repeat steps 1–6 with $k \leftarrow k+1$
5. Interpolate (HW)* and correct. This only needs to be done at red points, since this step is immediately followed by a black sweep.

at R points which correspond to $k+1$ level points:

$$I_{k+1}^k u_{ij}^{k+1} = u_{ij}^{k+1}$$

at R points which do not correspond to $k+1$ level points:

$$I_{k+1}^k u_{ij}^{k+1} = 0$$

at all R points:

$$u_{ij}^k \leftarrow u_{ij}^k + I_{k+1}^k u_{ij}^{k+1}$$

6. Do a black sweep, followed by a red sweep.

at B points:

$$u_{ij}^k \leftarrow \frac{h_k^2}{4} f_{ij}^k + \frac{1}{4} \begin{bmatrix} & 1 & \\ 1 & & 1 \\ & 1 & \end{bmatrix} u^k$$

at R points:

$$u_{ij}^k \leftarrow \frac{h_k^2}{4} f_{ij}^k + \frac{1}{4} \begin{bmatrix} & 1 & \\ 1 & & 1 \\ & 1 & \end{bmatrix} u^k$$

Parallel Algorithm.

Because of the one-to-one correspondence of grid points to processors, "at all red points" means the same as "on all processors who have been assigned a red point."

on level $k > 1$, do: [3]

1. finish the R sweep:

   at R points: average nearest r_{ij}^k's, add to r_{ij}^k, store in r_{ij}^k
   at B points: do nothing
   number of computation steps = 3, number of communication steps = 4

2. compute residual at B points, storing at R point processors:

   at R points: compute $(r_{ij}^k + r_{i+1,j+1}^k + r_{i,j+2}^k + r_{i-1,j+1}^k)/4$ and store in s_{ij}
   at B points: do nothing
   number of computation steps = 2, number of communication steps = 4

3. restrict residual to initialize r^{k+1}:

   at level $k+1$ points: compute $(s_{ij} + s_{i+1,j-1} + s_{i,j-2} + s_{i-1,j-1})/2$ and store in r_{ij}^{k+1}
   at all other points: do nothing
   number of computation steps = 2, number of communication steps = 4

4. If not at coarsest level, repeat steps 1–6 with $k \leftarrow k+1$, $h \leftarrow 2h$

5. interpolate and add:

   at level $k+1$ points: compute $h^2 u_{ij}^{k+1}/4 + r_{ij}^k$, store in s_{ij}
   at other R points: store r_{ij}^k in s_{ij}
   at B points: do nothing
   number of computation steps = 1, number of communication steps = 0

[3] N.B. On the finest grid, level one, there are four additional computation and three additional communication steps needed since the initial iterate is not assumed to be zero.

6. do a full black then red relaxation:

at R points:	do nothing
at B points:	get $s_{i,j-1}$ and store in s_{ij}
at all points:	compute $(s_{ij} + s_{i+1,j+1} + s_{i,j+2} + s_{i-1,j+1})/4$, store in s_{ij}
at R points:	compute $(s_{ij} + s_{i+1,j-1} + s_{i,j-2} + s_{i-1,j-1})/4$, store in s_{ij}
at B points:	do nothing
at all points:	compute $u_{ij}^k + 4s_{ij}/h^2$ and store in u_{ij}^k

number of computation steps = 5, number of communication steps = 9

REFERENCES

[1] A. BRANDT, *Multigrid techniques:* 1984 *guide with applications to fluid dynamics*, Tech. Report GMD-Studien Nr. 88, Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin, Federal Republic of Germany, 1984.

[2] B. BRIGGS, L. HART, S. MCCORMICK, AND D. QUINLAN, *Multigrid methods on a hypercube*, in Multigrid Methods: Theory, Applications and Supercomputing, S. F. McCormick, ed., Lecture Notes in Pure and Applied Mathematics, Vol. 110, New York, 1987, Third Copper Mountain Conference on Multigrid Methods, Marcel Dekker, New York, 1987, pp. 63–83.

[3] T. CHAN AND R. TUMINARO, *Analysis of a parallel multigrid algorithm*, in Proc. Fourth Copper Mountain Conference on Multigrid Methods, J. Mandel, S. McCormick, J. E. Dendy, C. Farhat, G. Lonsdale, S. Parter, J. Ruge, and K. Stüben, eds., Society for Industrial and Applied Mathematics, Philadelphia, 1989, pp. 66–86

[4] P. FREDERICKSON AND O. MCBRYAN, *Parallel superconvergent multigrid*, in Multigrid Methods, S. McCormick, ed., Marcel Dekker, New York, 1988.

[5] P. O. FREDERICKSON, *Totally parallel multilevel algorithms for sparse elliptic systems*, RIACS Report 89.10, Research Institute for Computer Science, NASA–Ames Research Center, Moffett Field, CA, 1989.

[6] D. GANNON AND J. VAN ROSENDALE, *Highly parallel multigrid solvers for elliptic* PDE*'s: An experimental analysis*, ICASE Report 82-36, Institute for Computer Applications in Science and Engineering, NASA–Langley Research Center, Hampton, VA, 1982.

[7] A. GREENBAUM, *A multigrid method for multiprocessors*, in Proc. Second Copper Mountain Conference on Multigrid Methods, S. McCormick, ed., North Holland, Amsterdam, 1986, pp. 75–88.

[8] O. A. MCBRYAN, *The Connection Machine:* PDE *solution on* 65 536 *processors*, Parallel Comput., 9 (1988/89), pp. 1–24.

[9] S. SCHAFFER, *Higher order multi-grid methods*, Math. Comp., 43 (1984), pp. 89–115.

[10] K. STÜBEN AND U. TROTTENBERG, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, 1981, pp. 1–176.

[11] ———, *On the construction of fast solvers for elliptic equations*, IMA-Report Mr. 82.0201, Gesellschaft für Mathematik und Datenverarbeitung, Bonn, Federal Republic of Germany, 1982.

[12] THINKING MACHINES CORPORATION, *Paris Reference Manual*, TMC, Cambridge, MA, 1989.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 1, pp. 221–229, January 1991
© 1991 Society for Industrial and Applied Mathematics
012

NORMALIZED CONVERGENCE RATES FOR THE PSMG METHOD*

PAUL O. FREDERICKSON† AND OLIVER A. MCBRYAN‡

Abstract. In a previous paper ["Parallel Superconvergent Multigrid," in *Multigrid Methods*, Marcel Dekker, New York, 1988] the authors introduced an efficient multiscale PDE solver for massively parallel architectures, which was called Parallel Superconvergent Multigrid, or PSMG. In this paper, sharp estimates are derived for the normalized work involved in PSMG solution—the number of parallel arithmetic and communication operations required per digit of error reduction. PSMG is shown to provide fourth-order accurate solutions of Poisson-type equations at convergence rates of .00165 per single relaxation iteration, and with parallel operation counts per grid level of 5.75 communications and 8.62 computations for each digit of error reduction. The authors show that PSMG requires less than one-half as many arithmetic and one-fifth as many communication operations, per digit of error reduction, as a parallel standard multigrid algorithm (RBTRB) presented recently by Decker [*SIAM J. Sci. Statist. Comput.*, 12 (1991), pp. 208–220].

Key words. parallel multigrid, parallel multilevel, multigrid, parallel, fast solver

AMS(MOS) subject classifications. 65F10, 65N20, 65N05, 68Q25

1. Introduction.

The PSMG totally parallel multiscale algorithm was introduced in [1], where rigorous upper bounds on the attainable convergence rates were presented. In this paper we present actual PSMG multigrid convergence rates, verified on grids of up to 4 million points. We also provide details of the methods we use for implementing the various PSMG linear operators in order to minimize parallel operation counts. Based on these, we then quantify PSMG performance in terms of the number of parallel operations required per digit of error reduction.

Most high performance (floating point) massively parallel computers require substantially longer to communicate a single number than to perform an arithmetic operation. Consequently, we regard normalized communication performance as the most relevant parameter. PSMG requires only 5.75 parallel communication operations per grid level for each digit of convergence. Even if computation speed was much slower than communication, PSMG uses only 8.62 arithmetic operations on each grid level per digit of convergence.

These normalized rates assume that there is one processor per fine grid point. As remarked in [6], PSMG is an example of an *intrinsically parallel algorithm.* It is highly efficient if sufficient processors are available, but is extremely inefficient on serial or low-parallelism computers. In situations where there are substantially more fine grid points than processors, an efficient approach might use a hybrid algorithm—using standard multigrid on the finest grids, but switching to PSMG on grid levels where the number of processors approximates or exceeds the number of grid points. The most interesting feature of the PSMG method is not so much the computational efficiency as the ease with which it can be analyzed theoretically (see, for example, [8]).

* Received by the editors June 5, 1990; accepted for publication (in revised form) October 6, 1990.

† Research Institute for Advanced Computer Science (RIACS), NASA Ames Research Center, Moffett Field, California 94035. The work of this author was supported by the NAS Systems Division of the National Aeronautics and Space Administration (NASA) and the Defense Advanced Research Project Agency (DARPA) via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Association (USRA).

‡ Department of Computer Science, University of Colorado, Boulder, Colorado 80309. The work of this author was supported by Air Force Office of Scientific Research grant AFOSR-89-0422.

Recent papers by Decker [2], [3] describe a parallel version of a variant (RBTRB) of the standard red-black multigrid algorithm of Stüben and Trottenberg [4]. We compare that algorithm with PSMG in the Appendix, and show that it requires over twice the arithmetic and over five times the communication for the same level of error reduction. We also briefly analyze in the Appendix the parallelization of the conventionally accepted fastest standard multigrid method for Poisson's equation [4]. While faster than RBTRB, it is still up to 4 times slower than PSMG.

A brief but complete description of the PSMG algorithm is presented in §2 below, while §3 summarizes the results for convergence rates and normalized operation counts. Section 4 describes the methods used to verify the convergence rates, and §5 provides the algorithms used to minimize the operation counts.

2. The PSMG algorithm. The PSMG algorithm works with a single grid of points $G^{(L)}$ of size 2^L in each dimension (called the level L grid, or the *fine* grid), but utilizes operators with different scales $l \leq L$ on that grid. Thus the algorithm is, strictly speaking, multiscale rather than multigrid. There are three basic operators: a finite difference operator A, an interpolation operator Q, and a smoothing operator Z. All operators are periodic on the grid in each coordinate direction. The PSMG algorithm extends naturally to both Neumann and Dirichlet boundary conditions, with no increase in convergence rate. The simplest approach to implementing Neumann or Dirichlet boundary conditions is to use reflection or anti-reflection boundary conditions and an extended grid. However, we will discuss only the periodic case here for simplicity.

The operators at scale level l, denoted $A^{(l)}$, $Q^{(l)}$, and $Z^{(l)}$, couple points at a distance $d_l \equiv 2^{L-l}$. Each level l operator is defined at all points of the grid $G^{(L)}$. The basic steps involved at level l, $0 < l \leq L$, for the solution of $A^{(L)}U = f$, starting with an initial guess u, are described by:

Algorithm PSMG(l,u,f):

1. Compute residual: $r = f - A^{(l)}u$.
2. Project residual to coarse grid: $r = r$ (trivial injection).
3. Solve coarse grid residual equation using PSMG: $e = \text{PSMG}(l-1, 0, r)$.
4. Interpolate to fine grid: $e' = Q^{(l)}e$.
5. Apply a relaxation: $e'' = (I - Z^{(l)}A^{(l)})e' + Z^{(l)}r$.
6. Compute and return the new solution: $u'' = u + e''$.

An exact solver is utilized on the coarsest grid. The PSMG strategy is to choose $Q^{(l)}$ and $Z^{(l)}$ as functions of $A^{(l)}$ in such a way as to optimize the convergence rate of the above algorithm. Explicit choices for $Q^{(l)}$ and $Z^{(l)}$ are given in [1] for the cases where $A^{(l)}$ represents either the standard 5-point or Mehrstellen discretizations of the Laplacian. In each case we provided upper bounds on the convergence rate for the procedure, which are uniform in l.

3. PSMG performance. In [1] we did not indicate good performance for PSMG on a 5-point operator. The 5-point case was presented in [1] only because the PSMG formalism was easiest to explain in that setting, while the details extended naturally to more complex operators. Indeed, the stated multigrid convergence rate upper bound of .2115 per iteration given in [1] for PSMG5-9 was very uninspiring, although we show here that actual 5-point convergence rates are substantially better. In this note, as in [1], we will focus on multigrid convergence rates. For V-cycles, as considered here, it is well known that two-grid rates are often a poor indicator of multigrid convergence rates [4].

We indicated much-improved performance in [1] for the Mehrstellen discretization of the Laplacian, where convergence rate bounds as good as .0045 were shown. Thus we had the interesting conclusion that in most cases someone wishing to solve Poisson's equation would be better off using the Mehrstellen discretization, which would yield a solution in much shorter time, than if a 5-point discretization were used, while providing for free a better (isotropic to fourth order) approximation to the Laplacian. Indeed, a fourth order accurate solution is obtained if the right-hand side is preprocessed suitably [5].

We have analyzed both 5-point and 9-point discretizations using the definition of computation model given in [2], [3]. For several PSMG methods we present asymptotic convergence rates, the number of parallel arithmetic and communication operations required on each grid per iteration, and also the normalized operation count for arithmetic and communication. If the asymptotic convergence rate of a method is ρ and the method requires w operations per iteration, the normalized operation count is defined as $w/\log_{10}\rho$ and measures the parallel work required per grid level to reduce the error by a factor of 10 (see [4]). We summarize the results for several simple cases in Table 1.

TABLE 1
PSMG *convergence rates.*

Method	Convergence Rate	Steps per Level		Normalized Steps	
		Comp.	Comm.	Comp.	Comm.
PSMG 5-9	.08867	14	12	13.31	11.40
PSMG 5-25	.02504	22	16	13.74	9.99
PSMG 9-9	.02165	16	12	9.61	7.21
PSMG 9-25	.00165	24	16	8.62	5.75

The corresponding coefficients for the interpolation operator Q and the smoothing operator Z are (in the notation of [1]):

PSMG5-9:	q_0=.25	q_1=.125	q_{11}=.0625
	z_0=.278079	z_1=.0534577	z_{11}=.0125615
PSMG5-25:	q_0=.361017	q_1=.11458	q_{11}=.0625
	q_2=-.0309162	q_{12}=.00521024	q_{22}=.00316188
	z_0=.361452	z_1=.0891718	z_{11}=.0293793
PSMG9-9:	q_0=.25	q_1=.125	q_{11}=.0625
	z_0=.300589	z_1=.0432465	z_{11}=.0139994
PSMG9-25:	q_0=.34152	q_1=.0995677	q_{11}=.0625
	q_2=-.0199225	q_{12}=.0127161	q_{22}=-.00295755
	z_0=.283286	z_1=.0323815	z_{11}=.00835795

Using these coefficients and the definition of the PSMG algorithm given in §2, one can verify the stated convergence rates. We have, in fact, verified all of the rates for systems of up to 4 million processors. The method used to verify these rates is described in §4. The methods used to optimize and compute the numbers of communication and computation steps presented in Table 1 are described in §5.

4. Verification of convergence rates. To verify the rates in Table 1, we have *exactly* computed the spectral radius of the self-adjoint PSMG multigrid error reduction operator $\mathbf{M}^{(L)}$ for all grids $G^{(L)}$ with L ranging from 0 to 11. Here $G^{(L)}$ is a

square grid with $n \equiv 2^L$ points on a side, as in [1]. The verification is based on the iterative formula

$$M^{(l)} = Z^{(l)} + (I - Z^{(l)}A^{(l)})\, Q^{(l)}\, M^{(l-1)}, \qquad 1 \le l \le L, \tag{1}$$

for the multigrid iteration operator $M^{(l)}$, presented in [1, §2.4]. The recursion begins on level 0 with

$$M^{(0)} = {A^{(0)}}^{-1}.$$

The operator $M^{(l)}$ is exactly the same as the operator $\mathbf{PSMG}(l, 0, f)$ defined in §2 above. The multigrid error-correction contraction $\mathbf{M}^{(l)}$ and the multigrid convergence rate μ are defined in [1] as:

$$\mathbf{M}^{(l)} = I - M^{(l)}A^{(l)}, \qquad \mu = \sup_L ||\mathbf{M}^{(L)}||. \tag{2}$$

As in [1], we use italics for the multigrid iteration operator $M^{(l)}$ which attempts to approximate the inverse of A, while the multigrid error reduction operator $\mathbf{M}^{(l)}$ is distinguished with bold typeface.

In a translation-invariant problem, (1) is most easily studied by Fourier transformation, in which case all quantities in (1) become multiplication operators by functions ${A^{(l)}}_k$, ${Q^{(l)}}_k$, and ${Z^{(l)}}_k$, where $k \equiv (k_1, k_2)$ are the relevant frequencies for grid $G^{(L)}$: $0 \le k_i < n$. Explicit formulae for these functions ("kernels") are given in [1] for the various 5-, 9-, and 25-point operators used in the PSMG procedures in Table 1. One final issue relates to the fact that the Poisson equation with periodic boundary data is singular. The correct domain for the translation invariant Poisson equation is the set of grid functions orthogonal to constants. In Fourier space this means that points aliased to $(0, 0)$ are omitted from the computation. Equivalently, one defines $M^{(0)} \equiv 0$ in (1).

In [1] we introduced a quantity μ^*, which is a rigorous upper bound for μ (see [1, §4.2]), and then proceeded to give numerically derived estimates for μ^*. The bound μ^*, evaluated in frequency space, involves the supremum over all possible grid levels of a functional of the kernels A, Q, Z, which in turn are simple trigonometric polynomials of the allowed frequencies on a grid—indeed, polynomials in the quantities $x_i = \cos(2\pi k_i d_l / n)$, where k_1 and k_2 are the frequencies. The numerical approximation used in [1] was to evaluate μ^* by choosing 1000 values for each x_i, uniformly distributed in [-1,1]; i.e., we searched for the maximum on a 1000×1000 grid embedded in the domain $[-1, 1] \times [-1, 1]$ of the (trigonometric) polynomials. Thus there were two sources of inaccuracy in the convergence rates: (1) the fact that μ^* was only an upper bound and (2) the fact that μ^* was itself approximated numerically.

In order to get a more realistic view of PSMG convergence rates, as needed for an effective comparison to [2], [3], we have attempted to compute μ directly, rather than through upper bounds. To be specific, we exactly compute the quantity

$$\mu^{(L)} = ||\mathbf{M}^{(L)}|| \tag{3}$$

on grid $G^{(L)}$. We compute $\mu^{(L)}$ by evaluating the recurrence (1) from $l = 0$ to $l = L$ in Fourier space for every frequency pair k_1,k_2 appropriate to $G^{(L)}$—i.e., for $0 \le k_i < 2^L$. The only approximation in this procedure is that the kernels are evaluated in double precision rather than infinite precision arithmetic.

The convergence rates given in Table 1 are the maximum values of $\mu^{(L)}$ for $0 \leq L \leq 11$, and therefore they bound the exact convergence rates for all grids up to size 4 million points. In practice we find that the convergence rates $\mu^{(L)}$ are unchanged to several digits of precision beyond about level $L = 6$. We therefore are confident, although this does not constitute a proof, that the convergence rates in Table 1 extend to arbitrary numbers of grid levels. As a final check we have solved the Poisson equation using PSMG, with zero right-hand side and a random initial guess, and in each case we have verified the convergence rates of Table 1.

We have used the same nonlinear optimizer in these measurements as was used in [1]. The only difference from [1] is that here we are optimizing $\mu^{(L)}$ rather than the substantially less expensive bound μ^*. Indeed, we introduced μ^* in [1] solely to provide a cheap evaluation function for the optimization. The fact that actual convergence rates were substantially faster than indicated by μ^* was known from numerical tests of the PSMG algorithm. The parameters given above, which yield the convergence rate of .00165, yield a value of .00365 for μ^* in close agreement with the value given in [1]. We have computed μ^* exactly for all grids up to 4 million points, thereby removing the error (2) discussed above, which was itself introduced by numerical approximation in [1].

5. Operation counts. We have followed Decker [2], [3] in our definition of the model computational problem and operation counts. In particular, a single computation is defined as a parallel add, a multiply, or an add/multiply pair [2], [3]. Furthermore, direct communication is allowed only with four nearest neighbors. All arithmetic and communication operations are required to be SIMD. As in [2], [3] the operation counts will be those for intermediate grids and the communication unit is grid level dependent—the cost for communication between "nearest neighbors" at that grid level. Notation in this section for the A, Q, and Z operators and their parameters follows [1].

Lemma 1 relates the cost of a PSMG intermediate level to the costs of the individual A (difference), Q (interpolation), and Z (smoothing) operators that are utilized. In Table 2 below, we present the operation counts for each of the individual operators encountered in the four PSMG algorithms of Table 1. Thus the operation counts in Table 1 may be verified by combining Lemma 1 with Table 2.

Some of the operation counts in Table 2 may be found in [2], [3], or are slightly sharper than counts given in [2], [3]. To maintain brevity, we will present, in Lemmas 2 and 3, a complete accounting of all of the operators in Table 2 that are required to verify the costs for the two best algorithms in Table 1: PSMG9-9 and PSMG9-25. The counts for the 5-point cases are derived from similar but simpler lemmas.

LEMMA 1. *At intermediate grid levels* $(0 < l < L)$, *the parallel communication and computation costs for the* PSMG *algorithm are given by:*

$$comm(\text{PSMG}(l)) = comm(Q) + comm(A) + comm(Z).$$
$$comp(\text{PSMG}(l)) = comp(Q) + comp(A) + comp(Z) + 2.$$

Proof. The PSMG algorithm was described as 6 separate steps in §2 above, to which we now refer. For intermediate grids $(0 < l < L)$ the initial guess is taken to be 0 so that step 1 is not needed. Similarly, step 6 is relevant only to the top level grid L. Step 2 is free because PSMG uses injection, while step 3 is counted at level $l - 1$ or lower. Thus only steps 4–5 are to be counted at level l. Apart from the 3 operators involved, we note that no communication is required in these steps while two computations are required in step 5. This completes the proof. □

TABLE 2
Operation counts for A, Q, and Z.

Operator	Comp. Steps	Comm. Steps
5-pt A (Laplacian)	3	4
9-pt A (Laplacian)	5	4
9-pt Q (Interpolation)	4	4
25-pt Q (Interpolation)	12	8
9-pt Z (Relaxation)	5	4

Based on Lemma 1, the critical issue is therefore the count of computation and communication costs for the application of the A, Q, and Z operators. A careful analysis provides the following values for the various cases of A, Q, and Z operators used in our algorithms.

We present here all of the algorithms needed to verify the counts in Table 2 for the case of the optimal PSMG9-9 and PSMG9-25 algorithms. We will prove the counts of Table 2 for the cases 9-point A, Q, Z in Lemma 2 and for the 25-point Q in Lemma 3. Decker [2], [3] has also presented an optimal algorithm for the case of bilinear interpolation.

Each of $A^{(l)}$, $Q^{(l)}$, $Z^{(l)}$ is translation invariant and is represented as a *difference star* in the notation of [4]. We assume a symmetric form for Q and Z, given by the following representations of the upper right quadrants of the stars:

$$Q_{25}{}^{(l)} = \begin{bmatrix} q_2 & q_{12} & q_{22} \\ q_1 & q_{11} & q_{12} \\ q_0 & q_1 & q_2 \end{bmatrix}, \qquad Z_9{}^{(l)} = \begin{bmatrix} z_1 & z_{11} \\ z_0 & z_1 \end{bmatrix}.$$

LEMMA 2. *Evaluation of $w = Zu$, where Z is the symmetric 9-point operator with three parameters z_0, z_1, z_{11}, may be accomplished in parallel with 5 computational and 4 communication steps. The computations reduce to 4 if $z_1{}^2 = z_0 z_{11}$, as is the case for the 9-point Q.*

Proof. If $z_{11} \neq 0$, we precompute $a = z_1/z_{11}$ and $b = z_0 - z_1{}^2/z_{11}$. The algorithm then consists of the following 3 steps, beginning with a function u on the grid and ending with the function $w = Zu$. Each step is executed in parallel on all processors (points i, j). In case $z_1{}^2 = z_0 z_{11}$, the third step is not needed. (The case $z_{11} = 0$, not relevant here, is even simpler, and may be accomplished with 3 computations and 4 communications).

Operation	Comp.	Comm.
$v(i,j) = z_{11} * (u(i-1,j) + u(i+1,j)) + z_1 * u(i,j)$	2	2
$w(i,j) = v(i,j-1) + v(i,j+1) + a * v(i,j)$	2	2
$w(i,j) = w(i,j) + b * u(i,j)$	1	0
Total cost for all steps:	5	4

LEMMA 3. *Evaluation of $w = Qu$, where Q is the symmetric 25-point operator with six parameters q_0, q_1, q_{11}, q_2, q_{12}, q_{22}, may be accomplished in parallel with 12 computational and 8 communication steps.*

Proof. If $q_{12} \neq 0$, we precompute $a = q_{22}/q_{12}$, $b = q_{11}/q_{12}$, $c = q_2/q_{12}$, and $d = q_1/q_{12}$. The algorithm then consists of the following 9 steps, beginning with a function u on the grid and ending with the function $w = Qu$. Each step is executed

in parallel on all processors (points i, j). (The case $q_{12} = 0$, not relevant to this paper, is even simpler, involving only a 17-point operator, and may be accomplished with 10 computations and 8 communications.)

Operation	Comp.	Comm.
$f(i,j) = u(i+1,j)$	0	1
$g(i,j) = u(i-1,j)$	0	1
$r(i,j) = z_{12} * (f(i,j) + g(i,j))$	1	0
$s(i,j) = z_{12} * (f(i+1,j) + g(i-1,j))$	1	2
$t(i,j) = a * s(i,j) + r(i,j) + z_2 * u(i,j)$	2	0
$x(i,j) = s(i,j) + b * r(i,j) + z_1 * u(i,j)$	2	0
$y(i,j) = x(i,j) + t(i,j+1)$	1	1
$z(i,j) = x(i,j) + t(i,j-1)$	1	1
$w(i,j) = y(i,j+1) + z(i,j-1) + c * s(i,j) + d * r(i,j) + z_0 * u(i,j)$	4	2
Total cost for all steps:	12	8

6. Conclusions. PSMG is shown to provide solutions of Poisson-type equations at convergence rates of .00165 per iteration, and with parallel operation counts per digit of error reduction as low as 5.75 communications and 8.62 computations on each grid level.

7. Appendix: Comparisons to standard multigrid. Recent papers by Decker [2], [3] describe a parallel version of a variant (RBTRB) of the standard red-black multigrid algorithm of Stüben and Trottenberg [4]. The papers [2], [3] provide an extremely efficient parallel RBTRB implementation, using a very clever rolling together of several phases of the multigrid procedure for RBTRB. Furthermore, those papers introduce the correct method of comparing PSMG and standard multigrid—by comparison of normalized convergence rates for communication and computation, rather than comparing absolute convergence rates. We differ, however, with [3] and [2] on several key points, as outlined, respectively, in the following two paragraphs. The effect is to produce a qualitative difference from [2] and a smaller quantitative but still significant difference from the more recent paper [3].

While [3] used the convergence rate upper bounds from [1], PSMG actually converges faster than those theoretical bounds, as shown here in Table 1. Furthermore, optimal parallel algorithms for various PSMG steps (relaxation, prolongation) were not used in [3], but are provided here in Lemmas 2 and 3. The effect of these differences is to reduce the normalized PSMG9-25 costs from 12.3 computations and 8.5 communications as reported in [3] to 8.62 computations and 5.75 communications as reported here. The differences for the 5-point operators are even larger, approaching a factor of two.

We note in passing that several further differences existed between the original technical report [2] and the published paper [3]. These resulted in the report's conclusion that RBTRB is as fast as PSMG even when there are as many processors as fine grid points. In part this was because [2] omitted from consideration the PSMG9-9 and PSMG9-25 methods, known from [1] to have the fastest convergence rates, and shown here to be the most efficient. Additionally, [2] used a two-grid RBTRB rate of .074 in the comparisons rather than the multigrid rate of .19 used here and in [3]. Finally, the report assigned the same cost to diagonal communications as to nearest neighbor communications in analyzing the RBTRB method. While these differences

qualitatively affected the results of [2], the basic approach to comparing the PSMG and standard multigrid methods outlined in [2] is not in dispute, and the RBTRB algorithm as presented in [2] stands as the most efficient standard MG implementation that we have seen.

A comparison of Table 1 in [2], [3] and in this paper shows the effects of these differences on PSMG performance relative to RBTRB. As shown in [3], RBTRB requires 13 arithmetic and 21 communication operations for a multigrid convergence rate of .19, yielding normalized values of 18.02 parallel arithmetic and 29.12 parallel communication operations per digit of error reduction. Comparing with the PSMG figures presented here in Table 1, we see that RBTRB requires 2.09 times as much arithmetic and 5.15 times as much communication as PSMG. The algorithm BRTBR has a multigrid convergence rate of .12, and would appear to provide a slightly better method than RBTRB, although communication costs differ somewhat.

One possibility for a faster parallel standard multigrid would be to use what is conventionally regarded as the fastest serial multigrid solver for the Poisson equation [4]. The algorithm, which we will call BR3, uses two BR sweeps, half-weighting restriction, bilinear interpolation, and one final BR sweep (we are using reverse operator notation here as in [3]). This algorithm has a multigrid convergence rate of .059 [4]. For the first red (R) sweep the initial guess is 0 on intermediate grids, allowing the red sweep to be folded in with the black sweep ($u = h^2 f/4$) as in [3], which requires 3 arithmetic and 4 communication steps, abbreviated as $3 + 4$ below. Each of the following red and black steps is a 4-point average, which also requires $3+4$ operations (similar to application of the 5-point A). The residual computation, $r = f - Au$, requires $4 + 4$ operations. Because the black residuals are zero, the half-weighting restriction reduces to injection of red values, multiplied by .5. This requires zero operations, as the multiple can be absorbed in the previous residual step. The bilinear interpolation is an application of the 3×3 Q and therefore costs $4 + 4$ operations. However, this may be reduced to $2 + 4$ by observing that only the values at black points are actually required for the following red sweep. The final BR sweep consumes $3 + 4$ operations for each phase. The total operation count for intermediate grids is therefore $21+28$, with corresponding normalized rates of 17.09 arithmetic and 22.78 communications per digit of error reduction—a substantial improvement over RBTRB, especially if communication dominates. It is likely that with care these numbers may be optimized somewhat. However, it is doubtful that they could approach the 5.75 communication rate of PSMG 9-25.

We do not address in this paper the question of whether other standard multigrid cycles may give better parallel performance than BR3. We refer to [7] for a detailed comparison of several hundred red-black standard multigrid methods, including several that are somewhat more efficient than the BR3 method.

REFERENCES

[1] P. Frederickson and O. McBryan, *Parallel superconvergent multigrid*, in Multigrid Methods, S. McCormick, ed., Marcel Dekker, New York, 1988.

[2] N. Decker, *On the parallel efficiency of the Frederickson–McBryan multigrid*, ICASE Report No. 90-17, Institute for Computer Applications in Science and Engineering, Hampton, VA, February 1990.

[3] ———, *A note on the parallel efficiency of the Frederickson-McBryan multigrid algorithm*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 208–220.

[4] K. STÜBEN AND U. TROTTENBERG, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, in Multigrid Methods, W. Hackbusch and U. Trottenberg, eds., Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, New York, 1981, pp. 1–176.

[5] L. COLLATZ, *The numerical treatment of differential equations*, Springer-Verlag, Berlin, New York, 1966, p. 542.

[6] O. MCBRYAN, *New architectures: Performance highlights and new algorithms*, Parallel Computing, 7 (1988), pp. 477–499.

[7] ———, *Sequential and Parallel Efficiency of Multigrid Fast Solvers*, Tech. Report, Computer Science Department, University of Colorado, Boulder, CO, September 1990.

[8] P. FREDERICKSON AND O. MCBRYAN, *Recent Developments for Parallel Multigrid*, in Proc. Third European Conference on Multigrid Methods, October 1990, to appear.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 231-246, March 1991
© 1991 Society for Industrial and Applied Mathematics

A CONSTRUCTIVE METHOD FOR NUMERICALLY COMPUTING CONFORMAL MAPPINGS FOR GEARLIKE DOMAINS*

KENT PEARCE†

Abstract. An algorithm and program are described for computing the conformal mapping function from the unit disk to a gearlike domain, i.e., a domain containing the origin whose boundary consists of arcs of circles centered at the origin and segments of radial lines emanating from the origin. The gearlike domain may be bounded or unbounded and may have slits. Both the forward and the inverse mapping functions are computable. Several examples are illustrated.

An application of the program is given which shows that a coefficient conjecture of R. W. Barnard for bounded univalent starlike functions fails.

Key words. numerical conformal mapping, gearlike domains, bounded starlike functions, coefficient conjecture

AMS(MOS) subject classifications. 30C30, 30C20, 65E05

1. Introduction. The Riemann mapping theorem asserts that the open unit disk $\mathcal{D} = \{z \,|\, |z| < 1\}$ is conformally equivalent to each simply connected domain $\mathcal{G}$ in the complex plane, whose boundary consists of at least two points, i.e., there exists a function f, analytic and univalent on $\mathcal{D}$, such that f maps $\mathcal{D}$ onto $\mathcal{G}$. More precisely, if d_o is an arbitrary point in $\mathcal{D}$ and g_0 is an arbitrary point in $\mathcal{G}$, then the Riemann mapping theorem asserts that there exists a unique conformal mapping f of $\mathcal{D}$ onto $\mathcal{G}$ such that $f(d_o) = g_o$ and $f'(d_o) > 0$. If the boundary of $\mathcal{G}$ is piecewise analytic and g_1 is a point on the boundary of $\mathcal{G}$, then the uniqueness assertion of the Riemann mapping theorem can be reformulated alternately as the statement that there exists a unique conformal mapping f of $\mathcal{D}$ onto $\mathcal{G}$ such that $f(d_o) = g_o$ and $f(1) = g_1$.

The problem of constructing the explicit conformal mapping guaranteed by the Riemann mapping theorem is usually difficult, even numerically. There are constructive proofs [He] of the Riemann mapping theorem, but, because of their general nature, they often converge only slowly to the desired solution.

For polygonal domains, i.e., simply connected domains bounded by segments of straight lines, there is a well-known representation formula, called the Schwarz-Christoffel formula (see [Ne, p. 180]), which the conformal mapping functions must satisfy. The difficulty with using the Schwarz-Christoffel formula to construct the conformal mapping for an explicit polygonal domain has always been that the formula contains unknown parameters, called the accessory parameters, which must be determined before the mapping function can be computed. Gaier, in his monograph [Ga] on conformal mapping, surveyed several methods which had been proposed for solving the accessory parameter problem and proposed a method for reducing the problem to a system of nonlinear equations. In 1980, Trefethen [Tr] devised an effective procedure for solving the accessory parameter problem and, hence, constructing the conformal mapping function for a polygonal domain. Trefethen used the geometry of the polygonal domain to construct a constrained system of nonlinear equations, the solution set of which would be the accessory parameter set.

In [BaPe], following the work of Goodman [Go], necessary and sufficient conditions were given for representing the conformal mapping function which would map $\mathcal{D}$ onto an image domain $\mathcal{G}$ which is gearlike. A gearlike domain is a simply connected

* Received by the editors March 13, 1989; accepted for publication (in revised form) November 2, 1989.
† Texas Tech University, Department of Mathematics, Box 4319, Lubbock, Texas 79409.

domain which contains the origin and is bounded by arcs of circles centered at the origin and by segments of straight lines through the origin. (See Fig. 1.) We note that the logarithmic image of a gearlike domain is a periodic polygonal domain. Hence, one possible method for solving the conformal mapping problem for a gearlike domain would be to apply an adaption of Trefethen's method to the logarithmic image. In this paper we propose to directly use the representation formula for gearlike functions and a development motivated by the work of Trefethen to construct a procedure for numerically computing the conformal mapping function for a gearlike domain. The kernel of that procedure is an algorithm which prescribes a system of nonlinear equations whose solution set is the unknown accessory parameter set for the gearlike domain being considered.

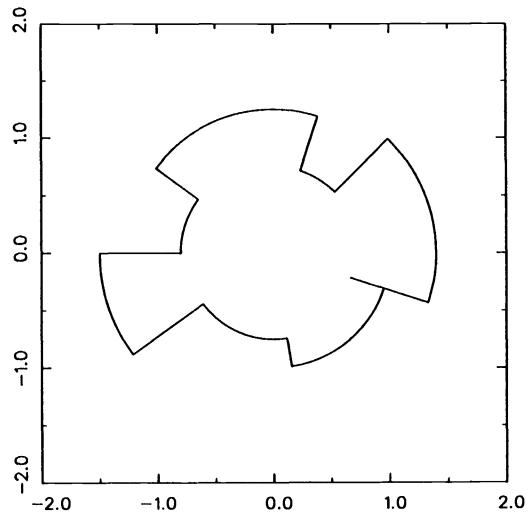


FIG. 1

We will give several examples using this procedure for constructing both the forward mappings from $\mathcal{D}$ to explicit gearlike domains $\mathcal{G}_i$ and the inverse mapping from the domains $\mathcal{G}_i$ back to $\mathcal{D}$. We will conclude with an application which shows that a coefficient conjecture of Barnard [Ba] fails.

There are several recent references which give alternate constructions for related conformal mapping problems. Jackson and Mason [JaMa] treated a problem of crack stress around holes in two-dimensional plates by mapping the crack region conformally to the exterior of the unit circle. They used an adaption, made by Trefethen [Tr83], of the SCPACK programs Trefethen developed for solving Schwarz-Christoffel mapping problems. Recently, Bjørstad and Grosse [BjGr] devised a method for constructing the conformal mapping functions (from the unit disk) to circular arc polygons. A circular arc polygon is a polygon where the sides of the polygon are allowed to be general arcs of circles. Their method applies an ordinary differential equation solver to a specific second-order differential equation which represents the circular arc polygon mapping function.

2. Gearlike domains. Let $\mathcal{G}$ be a gearlike domain in the complex plane with n sides. Let w_k, $1 \leqq k \leqq n$, denote the vertices of $\mathcal{G}$, some of which may lie at infinity. For convenience, we will also denote w_n as w_0. For each finite vertex w_k let $\pi\alpha_k$ denote the interior angle at w_k and let $\pi\beta_k$ denote the exterior angle at w_k. By definition, α_k

and β_k satisfy the relation $\alpha_k + \beta_k = 1$. For each infinite vertex w_k set $\beta_k = 1$. We note, in this latter case, that $\pi\beta_k$ is generally not the exterior angle at w_k.

For example, in Fig. 2, at the corner vertices w_3 and w_6 we have $\beta_3 = \frac{1}{2}$ and $\beta_6 = -\frac{1}{2}$. At w_2, the tip of a slit, the external angle is $-\pi$, so $\beta_2 = -1$.

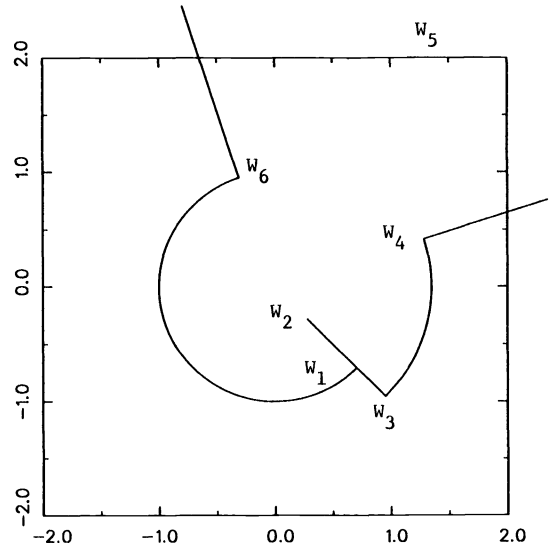


FIG. 2

In [BaPe] we have shown that the following representation formula describes the conformal function f mapping $\mathcal{D}$ onto $\mathcal{G}$ with $f(0) = 0$:

$$\frac{zf'(z)}{f(z)} = \prod_{k=1}^{n} (1 - \bar{z}_k z)^{-\beta_k}, \tag{1}$$

where the points z_k are the preimages on $|z| = 1$ of the vertices w_k and where the following constraint conditions must be satisfied:

(i) $\beta_k \in \{-1, -\frac{1}{2}, 0, \frac{1}{2}, 1\}, \qquad 1 \leqq k \leqq n,$

(ii) $\sum_{k=1}^{n} \beta_k = 0,$

and

(iii) $\sum_{k=1}^{n} \beta_k \operatorname{Arg}(z_k) \equiv 0 \pmod{\pi}.$

We note that if $\mathcal{G}$ is gearlike, then it is easily seen for the mapping function f that we must have that $zf'(z)/f(z)$ is either pure real or pure imaginary on the boundary of $\mathcal{D}$. If a function f satisfies (i) and (ii), but not (iii), then $zf'(z)/f(z)$ will map $\mathcal{D}$ to a domain bounded by radial segments which emanate from the origin, but which are twisted away from the coordinate axes and, hence, f will map $\mathcal{D}$ to a domain which is not gearlike.

Equation (1) can be solved to explicitly represent the conformal mapping f on $\mathcal{D}$ as

$$f(z) = cz \exp\left\{\int_0^z \frac{[\prod_{k=1}^{n} (1 - \bar{z}_k w)^{-\beta_k} - 1]}{w}\, dw\right\}. \tag{2}$$

In the representation formula (2) the parameter c and the prevertices z_k, $1 \leqq k \leqq n$, are accessory parameters which must be determined before the conformal mapping f can be computed. For each accessory parameter set satisfying (i), (ii), and (iii) a function f given by (2) will map $\mathcal{D}$ onto a gearlike domain $f(\mathcal{D})$ whose local boundary at each vertex $w(k) = f(z_k)$ will be conformally homotopic to the local boundary of $\mathcal{G}$ at w_k. However, corresponding sides of $f(\mathcal{D})$ and $\mathcal{G}$ which are arcs of circles will generally not have the same modulus and corresponding sides of $f(\mathcal{D})$ and $\mathcal{G}$ which are segments of straight lines will generally not have the same argument.

Using the second formulation for uniqueness in the statement of the Riemann mapping theorem we may suppose that the prevertex $z_n = 1$, i.e., that $f(1) = w_n$, and that w_n is finite. The latter assumption can always be made via a suitable reindexing of the vertices of $\mathcal{G}$. The parameter c can be computed directly. If we let

$$w^* = \exp\left\{\int_0^1 \frac{[\prod_{k=1}^n (1-\bar{z}_k w)^{-\beta_k} - 1]}{w}\, dw\right\},$$

then

$$c = w_n / w^*.$$

Thus, there remain $n-1$ unknown accessory parameters to be calculated. To determine the accessory parameters we will construct a system of $n-1$ real equations, dependent on the parameters $z_1, \cdots, z_{n-1}$, whose solution set will be the correct accessory parameter set.

If the domain $\mathcal{G}$ is starlike, then there exists a unique function f which will map $\mathcal{D}$ onto $\mathcal{G}$ as determined by the vertices $\{w_k\}$ and the exterior angles $\{\beta_k \pi\}$. However, if the domain $\mathcal{G}$ is nonstarlike, there exist several functions which will map $\mathcal{D}$ onto a domain determined only by specifying the vertices $\{w_k\}$ and the exterior angles $\{\beta_k \pi\}$—precisely because $\arg(w_k)$ is not uniquely determined. However, only one of these functions will be globally univalent on $\mathcal{D}$. In order to construct the univalent mapping f from $\mathcal{D}$ to $\mathcal{G}$, i.e., in order to solve the accessory parameter problem for the univalent mapping, we will need to measure the total changes in argument over the circular sides of $\mathcal{G}$ rather than measuring the absolute arguments at the vertices of $\mathcal{G}$. We will denote the change in argument over a circular side of $\mathcal{G}$ from vertex w_{k-1} to w_k by $\Delta \arg(w_k)$ and we will denote the total change in argument for the mapping function f over the arc (on the boundary of $\mathcal{D}$) from z_{k-1} to z_k by $\Delta \arg(w(k))$, where, again, $w(k) = f(z_k)$.

We will require, in the presentation of vertices of $\mathcal{G}$, that the interior angle at w_{n-1} not be π.

For each vertex w_k, $1 \leqq k \leqq n-2$, there are four (mutually exclusive) possibilities that can arise:

(a) The vertex w_k lies at infinity;

(b) The vertex w_k is finite and lies on a radial side of $\mathcal{G}$ which comes in from infinity (as the boundary of $\mathcal{G}$ is traversed positively), but not at the end of the radial side, i.e., the interior angle at w_k is π:

(c) The vertex w_k is finite and lies at the end of a radial side of $\mathcal{G}$, which comes in from infinity;

(d) The vertex w_k is otherwise finite.

If (a) holds, then we will not impose any condition on $w(k)$; however, the geometry of $\mathcal{G}$ will force the next vertex to be finite.

If (b) holds, then we will impose a modulus condition on $w(k)$ and require that

$$|w(k)| - |w_k| = 0. \tag{3a}$$

If (c) holds, then we will impose two conditions on $w(k)$. We will require that the modulus condition (3a) holds and we will require that

$$\text{Arg}\,(w(k)) - \text{Arg}\,(w_k) = 0. \tag{3b}$$

In the last case, then either $|w_k| = |w_{k-1}|$ or $\text{Arg}\,(w_k) = \text{Arg}\,(w_{k-1})$. If $|w_k| = |w_{k-1}|$, then we will require that

$$\Delta \arg\,(w(k)) - \Delta \arg\,(w_k) = 0, \tag{3c}$$

otherwise we will require (3a) to hold.

The above construction will generate $n-2$ equations if the vertex w_{n-2} does not lie at infinity. If, however, w_{n-2} does lie at infinity, then one additional equation will need to be generated. The vertex w_{n-1} must be finite and satisfy condition (c). Thus we can add one equation to the system by adding the modulus equation (3a) with $k = n-1$.

The system of equations generated at this point may not be sufficient to characterize the accessory parameter set for the correct mapping function, especially in the case that $\mathcal{G}$ is unbounded and nonstarlike. We may need to replace one of the equations in the system with a secondary equation to control the total argument change for the mapping function or to control the location of the tip of a slit. There are three cases which need to be considered.

Case 1. The vertex w_{n-1} or w_n lies at the end of a radial side of $\mathcal{G}$ going out to infinity and the interior angle at that end is $\pi/2$. If w_{n-1} lies at the end of the radial side, then the last equation in the system of type (3b) must be replaced by an argument condition (3c) with $k = n-1$. Alternately, if w_n lies at the end of the radial side, then either w_{n-1} is the tip of a circular slit or the boundary of $\mathcal{G}$ has a right angle corner at w_{n-1}. If w_{n-1} is the tip of a circular slit, then the last equation in the system of type (3a) must be replaced by an argument condition (3c) with $k = n$; otherwise, the last equation in the system of type (3b) must be replaced by an argument condition (3c) with $k = n$.

Case 2. Case 1 does not apply and condition (c) holds for some vertex w_{k_0}, $1 \leqq k_0 \leqq n-1$, such that w_{k_0} lies on the component of the boundary of $\mathcal{G}$ which contains w_n and the boundary of $\mathcal{G}$ has a right angle corner at w_k. If w_{n-1} is the tip of a radial slit, then the argument equation (3b) generated by condition (c) at $k = k_0$ must be replaced by a modulus condition (3a) with $k = n-1$. Alternatively, if $k_0 = n-1$, then the modulus equation (3a) generated by condition (c) at $k = k_0$ must be replaced by an argument condition (3c) with $k = n$. Alternately, if w_{n-1} lies at the tip of a circular slit, then the last equation in the system of type (3a) must be replaced by an argument condition (3c) with $k = n-1$. Otherwise, if the segment on the boundary of $\mathcal{G}$ between w_{n-2} and w_{n-1} is a circular arc, then the argument equation (3b) generated by condition (c) at $k = k_0$ must be replaced by an argument condition (3c) with $k = n-1$.

Case 3. The domain $\mathcal{G}$ is bounded and the vertex w_{n-1} lies at the tip of a slit. The above construction does not completely characterize the geometry of $\mathcal{G}$, i.e., the generated mapping function f may not satisfy $f(z_{n-1}) = w_{n-1}$. The last equation of the system, that is, the equation representing the vertex w_{n-2}, must be replaced by a condition determined by the geometry at the vertex w_{n-1}. If w_{n-1} lies at the tip of a radial slit, then we will require that (3a) holds for $k = n-1$, otherwise we will require that (3c) holds for $k = n-1$.

To determine the full set of $n-1$ accessory parameters we need to add one additional equation to the above system of equations. We have not yet imposed the constraint condition (iii) in the representation formula for gearlike mappings. While

we could solve (iii) for one of the prevertices (in terms of the others) and thus reduce the number of unknown accessory parameters from $n-1$ to $n-2$, our computational experience has been that we have been able to treat a variety of problems (i.e., solve the accessory parameter problem) using a full set of $n-1$ equations, with its extra degree of freedom, which we could not successfully treat using a reduced set of $n-2$ equations. We will, therefore, impose (iii) as the $(n-1)$st system equation.

Let us note that if the interior angle at w_{n-1} were allowed to be π, then the generated mapping f might not satisfy $f(z_{n-1}) = w_{n-1}$. No auxiliary condition could be added to the above construction without overdetermining the problem. On the other hand, the point w_{n-1} could be removed from the set of vertices of $\mathcal{G}$ without changing the geometry of $\mathcal{G}$.

3. Computation. In the actual practice of solving the nonlinear system of equations (3) and computing the conformal mapping f, we have generally employed the techniques described by Trefethen (see [Tr, pp. 86-90]). The given form (3) requires solving for the unknown complex prevertices z_k on the unit circle $|z| = 1$. (As with the vertices, we will also denote z_n as z_0.) It is numerically more convenient to transform the points z_k to their arguments θ_k by

$$z_k = e^{i\theta_k}, \qquad 0 \leqq \theta_k \leqq 2\pi.$$

The arguments θ_k are, however, severely constrained by a set of linear inequalities because the points z_k are ordered around the unit circle. The ordering constraints on (3) can be removed by transforming the arguments θ_k to a set of unconstrained variables y_k via the equations

$$y_k = \log \frac{\theta_k - \theta_{k-1}}{\theta_{k+1} - \theta_k}, \qquad 1 \leqq k \leqq n-1, \tag{4}$$

where we take θ_0 to be $\operatorname{Arg}(z_0) = 0$. Because the arguments θ_k in (4) are coupled, the transformation can be inverted to recover the arguments θ_k from the images y_k.

At each step in the iteration we will calculate from the unconstrained variables y_k, $1 \leqq k \leqq n-1$, a set of arguments θ_k and then a set of prevertices z_k. (The prevertex z_n is fixed at 1.) Finally, we will calculate the values of the $n-1$ nonlinear equations (3) for the current set of prevertices.

In the computation of the values $f(z_k)$, we generally choose a path for the integration in (2) which is the straight line segment $[0, z_k]$ in $\mathcal{D}$. The integrand in (2) will have a singularity at both 0 and at z_k. However, the singularity at 0 is removable and can be controlled directly. The singularity at z_k is of the form $(1-\bar{z}_k z)^{-\beta_k}$ where β_k can take on only one of five possible values. Normally, such a singularity in an integral could be easily, and highly accurately, treated by Gauss-Jacobi quadrature. However, there may be, and typically there are, other prevertices z_j clustered near z_k, which could affect the accuracy of the quadrature result. Trefethen described a type of compound Gauss-Jacobi quadrature (see [Tr, p. 87]) which divides the path of integration into subpaths, where the length of each subpath is dependent on how closely other prevertices z_j are clustered to z_k. This type of compound Gauss-Jacobi quadrature has produced both highly accurate and efficient quadrature results.

We have used the library subroutine GAUSSQ by Golub and Welsch [GoWe] to calculate the nodes and weights for the Gauss-Jacobi quadrature. We have usually set the number of nodes NPTSQ to be computed by GAUSSQ at 8. Since the compound Gauss-Jacobi quadrature always divides the path of integrations into two halves and one of the halves may be further subdivided, dependent on the distribution of the current set of prevertices z_k, we achieve, in practice, at least 16 nodes for integration on each path.

To solve the system of unconstrained nonlinear equations we have used the library subroutine NS01A by Powell [Po], which employs a hybrid between the method of steepest descent (initially) and Newton's method (terminally).

While the solution driver which calls NS01A can pass specialized initial sets of prevertices, should such information be available, the general initialization does not take into account any information about the geometry of the particular gearlike conformal mapping problem. In general, the initialization allocates a set of initial prevertices uniformly distributed around the unit circle.

Two of the control parameters required by NS011 are: DSTEP, the stepsize used by NS01A to calculate the Jacobian of (3) by finite-difference methods—generally fixed at 10^{-8}; and TOL, the convergence criterion for returning from NS01A to the calling program after having successfully solved (3). The value of TOL is closely related to the number of nodes NPTSQ selected for the compound Gauss-Jacobi quadrature. For a problem with vertices close to unit norm TOL is typically set at $10^{-(\mathrm{NPTSQ}+1)}$.

Once the accessory parameters have been determined, values of the mapping function f can be computed for given initial points z in $\mathcal{D}$ or on the boundary of $\mathcal{D}$. In either case, the values are computed by using compound Gauss-Jacobi quadrature. While the value of f at a point z can always be computed using (2) directly, if z is near a point z^*, $z^* \neq 0$, where the value of f is known, then the value $f(z)$ can be alternately computed as

$$f(z) = \frac{f(z^*)}{z^*}\, z \exp\left\{ \int_{z^*}^{z} \frac{\left[\prod_{k=1}^{n} (1-\bar{z}_k w)^{-\beta_k} - 1\right]}{w}\, dw \right\}.$$

The point z^* can be one of the prevertices corresponding to a known finite vertex of $\mathcal{G}$ or the point z^* can be the point where the value of f has been previously computed.

Values for the inverse mapping function from the gearlike domain $\mathcal{G}$ to the unit disk $\mathcal{D}$ can also be generated. Let w be in $\mathcal{G}$ or on the boundary of $\mathcal{G}$. If, for $z = f^{-1}(w)$, a nearby initial estimate z^* can be given, then Newton's method can be employed to solve for z. (The value of the derivative of f can be calculated from equation (1).) On the other hand, when no initial estimate z^* is known, we can rewrite equation (1) as

$$\frac{dw}{dz} = \frac{w}{z} \prod_{k=1}^{n} (1-\bar{z}_k z)^{-\beta_k}, \tag{5}$$

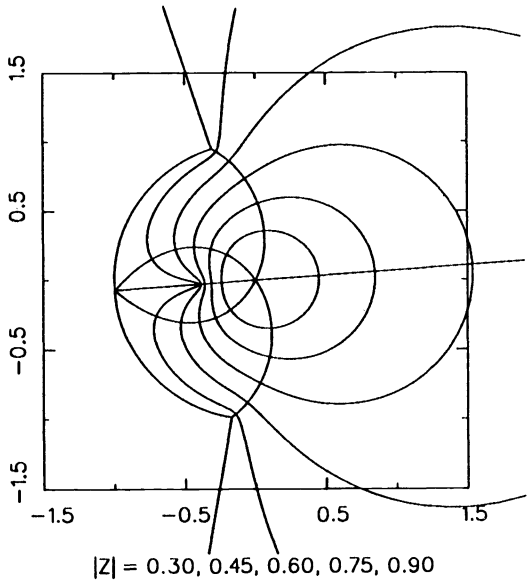
where $w = f(z)$. The univalence of the mapping function implies that equation (5) can be inverted to obtain

$$\frac{dz}{dw} = \frac{z}{w} \prod_{k=1}^{n} (1-\bar{z}_k z)^{+\beta_k}. \tag{6}$$

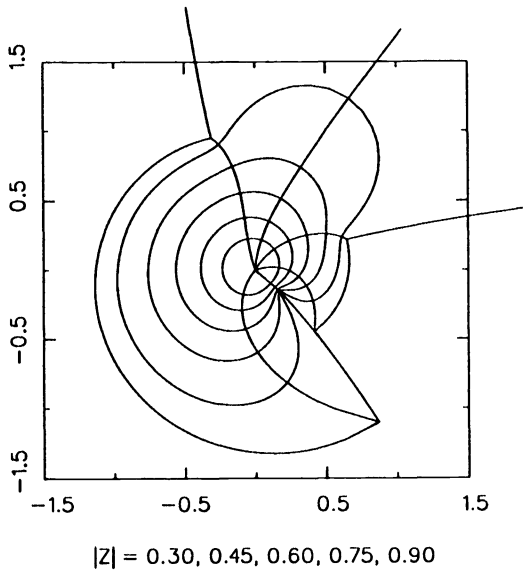
Equation (6) can be viewed as an ordinary differential equation which lifts a straight line segment $[\tilde{w}, w]$ in $\mathcal{G}$ to a solution curve in $\mathcal{D}$ from a point $\tilde{z} = f^{-1}(\tilde{w})$ to the point z. The point $\tilde{w}$ can be any point in $\mathcal{G}$ provided that the straight line segment $[\tilde{w}, w]$ lies entirely in $\mathcal{G}$. To solve the ordinary differential equation we have used the library subroutine ODE by Shampine and Gordon [ShGo]. The code in ODE treats real differential equations, which required that we transform (6) to a coupled pair of real differential equations.

We have, following Trefethen, combined the above two approaches, unless an initial estimate is known for which Newton's method can be applied directly. We will first solve (6), starting typically at $\tilde{w} = 0$, to obtain a low-order approximation z^* to z. We will then follow up using Newton's method to move from z^* to a final approximation of z with high-order accuracy.

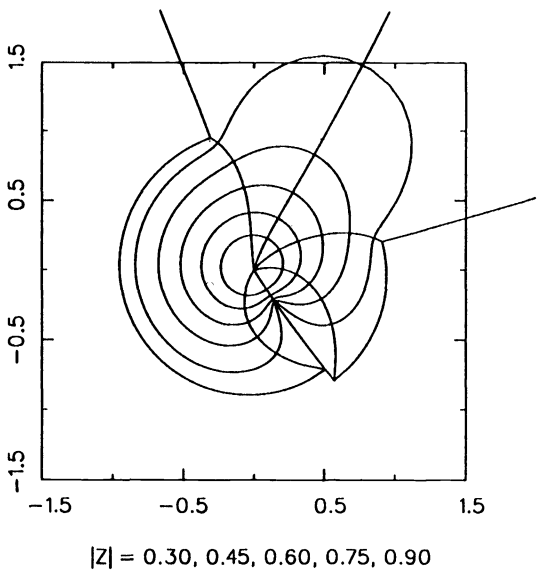
It is important that not only are we able to construct the approximate numerical mapping by solving the accessory parameter problem, but that we are able to estimate the accuracy of the approximation we have generated. After the nonlinear solve NS01A returns a solution for the accessory parameter problem we check the accuracy of the computed solution by checking at each finite node w_k the difference $|w_k - w(k)|$. At each infinite node w_k we check the difference $\|w_{k-1} - w_{k+1}| - |w(k-1) - w(k+1)\|$. An error routine returns the maximum of the above tested differences as an error estimate for the accuracy of the generated mapping function.



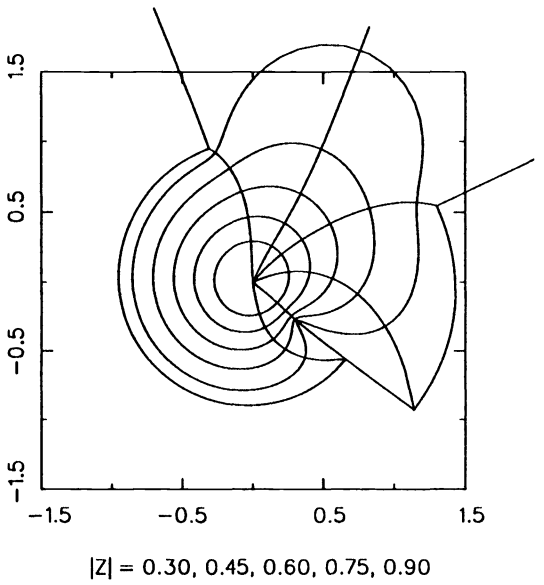
I5



I7



I8



I9

FIG. 3

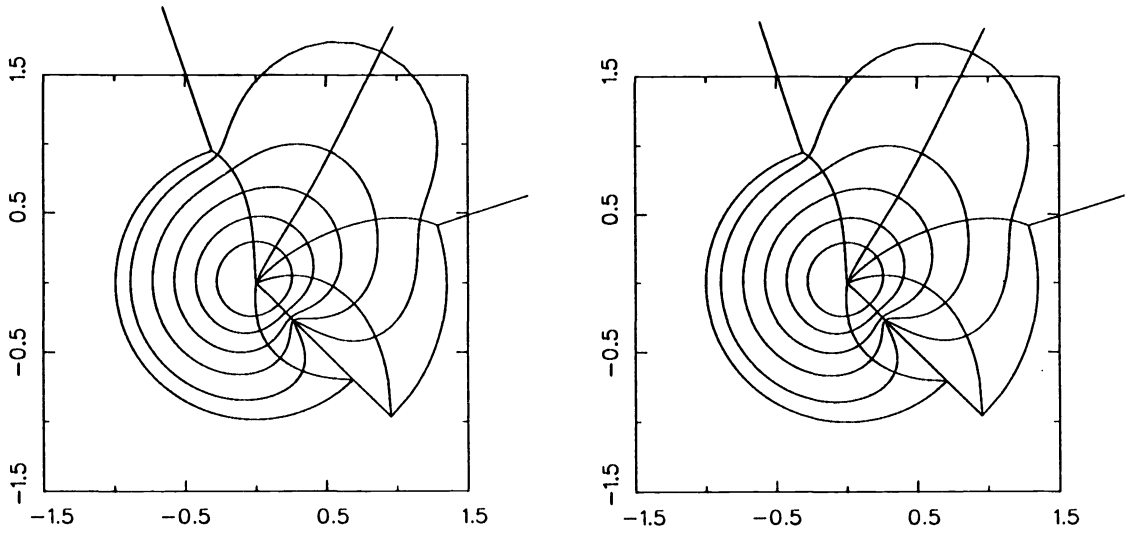


FIG. 3—continued

4. Examples. In Fig. 3 we show conformal mappings generated at several of the iterative steps which arise in constructing the conformal mapping in Fig. 2. We start the process with no assumption about the geometry of the prevertices for the image domain $\mathcal{G}$ in Fig. 2, i.e., we begin with an initial uniform distribution about the unit circle for the prevertices. Also, we have set the number of nodes for the compound Gauss-Jacobi quadrature at NPTSQ = 8 and have set the level of desired accuracy at TOL = 10^{-9}. The plots of Fig. 3 show several contours (images of circles centered at the origin) and the images of the six radial segments which join the origin to the prevertices. As Trefethen noted, the conformality of the mappings at the origin can be used to interpolate on the plots in Fig. 3 the arguments of the prevertices via the arguments of the images of the radial segments as they approach the origin. The example in Fig. 3 is typical in that it shows some of the prevertices to be clustering as the iterations move towards the mapping solution at stage I20.

In Figs. 4 and 5 we show examples which illustrate the dispersion of the contours for several sets of bounded and unbounded gearlike domains. In each case the contours are the images of the circles $|z| = r$, $0 < r < 1$.

In Figs. 6 and 7 we show examples which illustrate the dispersion of the level sets for inverse mappings for the gearlike domains depicted in Figs. 4 and 5. In each case the level sets are the images under f^{-1} of $|w| = R$, $R > 0$. The ticks shown on the boundary of $\mathcal{D}$ mark the location of the prevertices z_k.

Finally, let us note one difficulty which we have encountered. The only requirement for applying the above algorithm to a gearlike domain mapping problem is that one of the finite vertices must be distinguished as the last vertex w_n (and, of course, then w_{n-1} must not be removable). There may be, for a given gearlike domain, several vertices which could be so distinguished. Our experience, however, in implementing the algorithm, i.e., in solving the nonlinear system of parameter equations via NS01A, has been that frequently only one of the several choices (for the distinguished vertex) has led to a tractable system of equations.

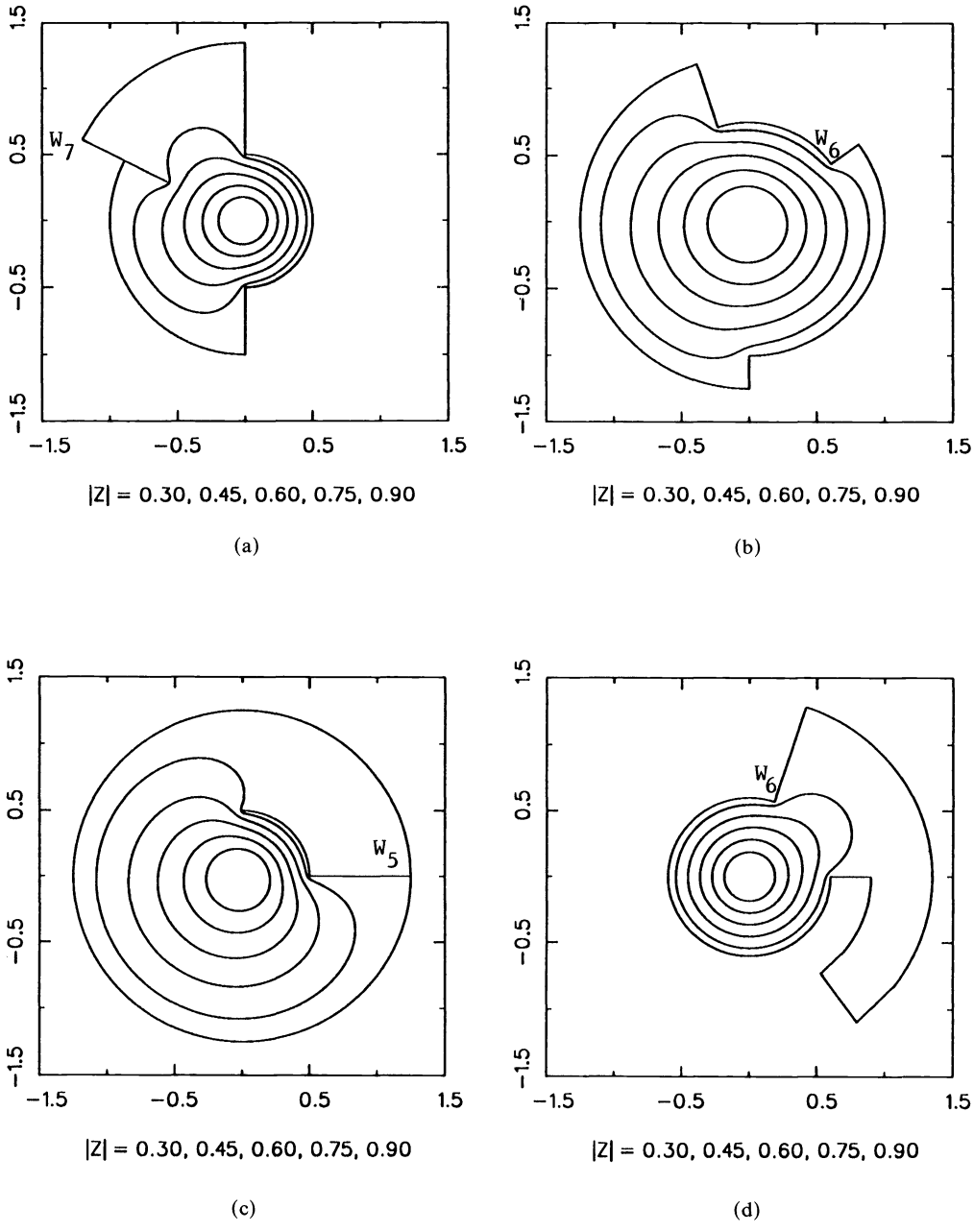
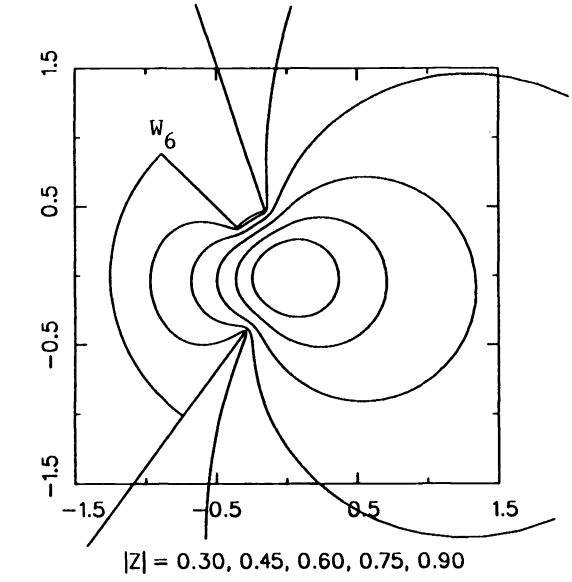
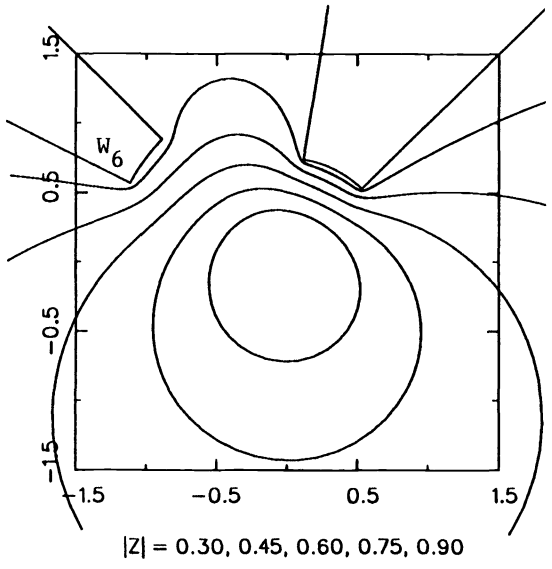

1.5
0.5
−0.5
−1.5
−1.5
−0.5
0.5
1.5
W7
|Z| = 0.30, 0.45, 0.60, 0.75, 0.90
(a)
W6
|Z| = 0.30, 0.45, 0.60, 0.75, 0.90
(b)
W5
|Z| = 0.30, 0.45, 0.60, 0.75, 0.90
(c)
W6
|Z| = 0.30, 0.45, 0.60, 0.75, 0.90
(d)

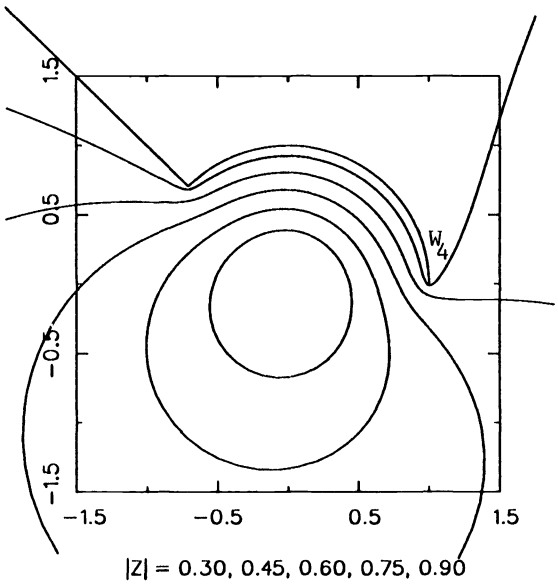
FIG. 4


W6
1.5
0.5
-0.5
-1.5
-1.5
-0.5
0.5
1.5
|Z| = 0.30, 0.45, 0.60, 0.75, 0.90

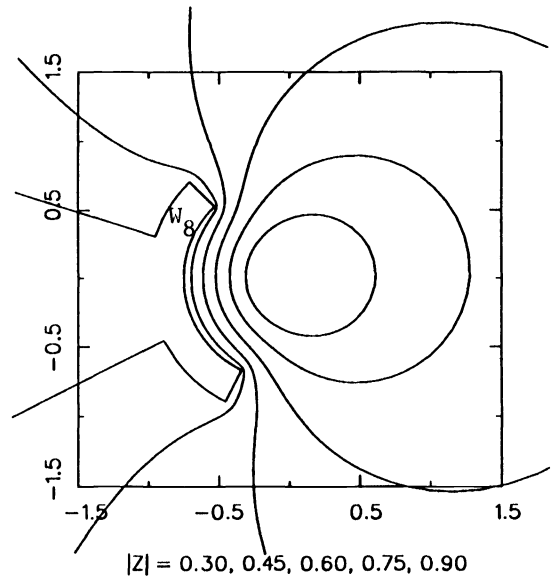
(a)


W6
1.5
0.5
-0.5
-1.5
-1.5
-0.5
0.5
1.5
|Z| = 0.30, 0.45, 0.60, 0.75, 0.90

(b)

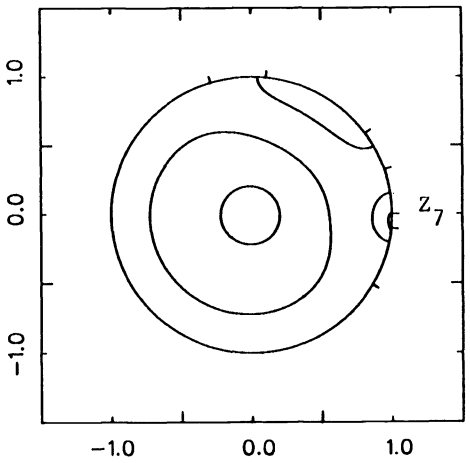

W4
1.5
0.5
-0.5
-1.5
-1.5
-0.5
0.5
1.5
|Z| = 0.30, 0.45, 0.60, 0.75, 0.90

(c)


W8
1.5
0.5
-0.5
-1.5
-1.5
-0.5
0.5
1.5
|Z| = 0.30, 0.45, 0.60, 0.75, 0.90

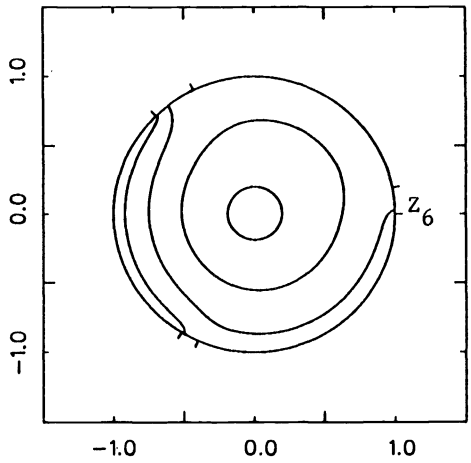
(d)

FIG. 5



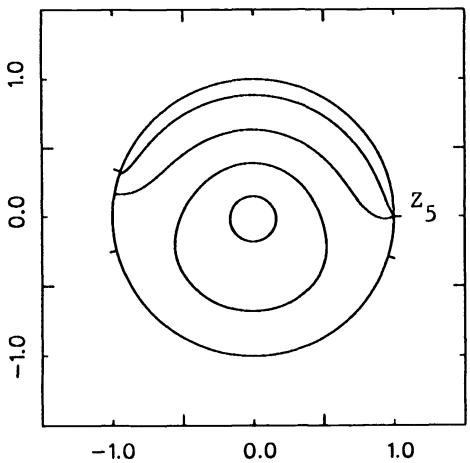
|w| = 0.13, 0.38, 0.71, 1.14

(a)



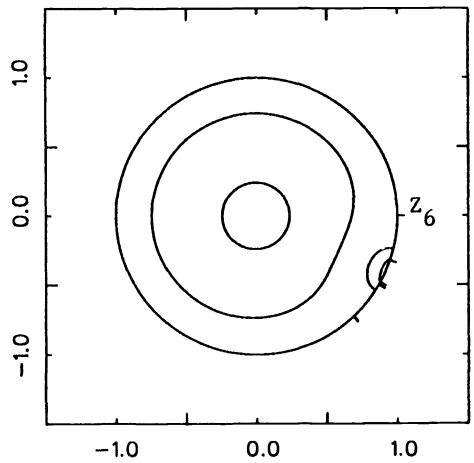
|w| = 0.19, 0.56, 0.88, 1.13

(b)



|w| = 0.13, 0.38, 0.69, 1.06

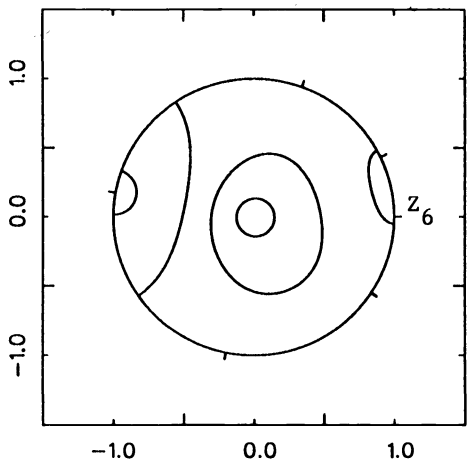
(c)



|w| = 0.15, 0.45, 0.79, 1.16

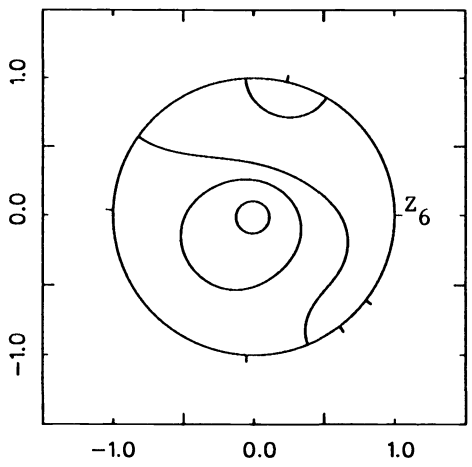
(d)

FIG. 6



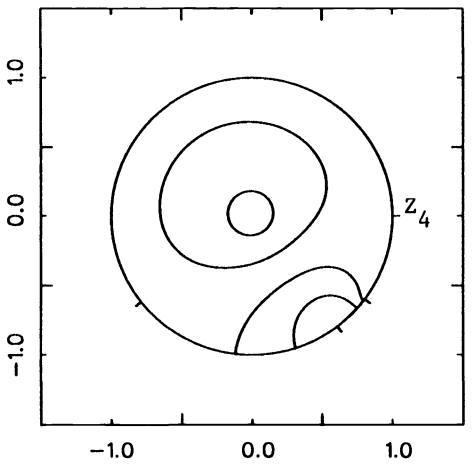
|w| = 0.13, 0.38, 0.88, 6.25

(a)



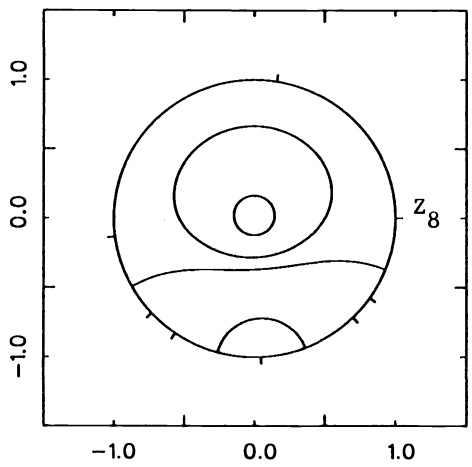
|w| = 0.19, 0.56, 1.00, 6.25

(b)



|w| = 0.25, 0.75, 2.00, 7.00

(c)



|w| = 0.19, 0.56, 0.88, 6.00

(d)

FIG. 7

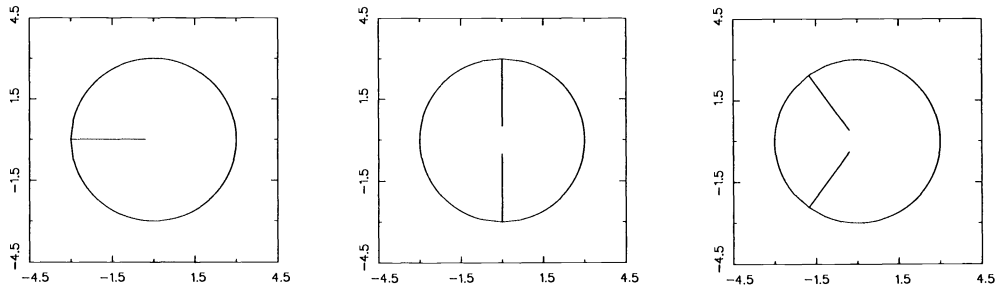
5. Application. Let St be the class of analytic univalent functions f on $\mathcal{D}$, normalized by $f(0)=0$ and $f'(0)=1$, which map $\mathcal{D}$ to an image domain which is starlike with respect to the origin. For $M \geqq 1$ let $\mathcal{D}(M)=\{z \mid |z|<M\}$ and let $St(M)$ be the subclass of St of functions which satisfy the property that $f(\mathcal{D}) \subset \mathcal{D}(M)$. Let $f \in St(M)$ and have a power series representation

$$f(z)=z+\sum_{n=2}^{\infty} a_n z^n.$$

Barnard and Lewis [Ba], [BaLe] considered the problem of finding

$$\max_{f \in St(M)} \operatorname{Re} a_3. \tag{7}$$

They showed that if $M \geqq 5$ and if f is extremal for the coefficient problem (7), then f is the Pick function in $St(M)$, i.e., f maps $\mathcal{D}$ onto the disk $\mathcal{D}(M)$ minus a radial slit on the negative real axis. (See Fig. 8(a).) If $1<M \leqq e$, they showed that if f is extremal for (7), then f is the SRT (square root transform) Pick function in $St(M)$, i.e., f maps $\mathcal{D}$ onto the disk $\mathcal{D}(M)$ minus a pair of symmetric radial slits on the imaginary axis. (See Fig. 8(b).)



(a) (b) (c)

FIG. 8

While they were not able to solve the extremal problem for $e<M<5$, they were able to show that the extremal function must map $\mathcal{D}$ onto $\mathcal{D}(M)$ minus a pair of real-axis symmetric radial slits. (See Fig. 8(c).) They noted that if $M=3$, then the third coefficient for the Pick function in $St(M)$ and the third coefficient for the SRT Pick function in $St(M)$ are the same. Lewis posed the coefficient problem (7) for $e<M<5$ as Problem 6.65 in [CaClHa] and Barnard conjectured [Ba] for $M \geqq 3$ that (7) was maximized by the Pick function in $St(M)$ and for $1<M \leqq 3$ that (7) was maximized by the SRT Pick function in $St(M)$.

To support that conjecture, the problem was posed to show numerically for $e<M<5$, M fixed, and for the real-axis symmetric double radial slit mappings in $St(M)$ that $\operatorname{Re} a_3$ was concave as a function of the argument α of the omitted slit, $\pi/2 \leqq \alpha \leqq \pi$. Each such function f is a bounded gearlike mapping and can be represented by (from the geometry of the image domain)

$$\frac{zf'(z)}{f(z)}=\sqrt{\frac{p_2^2(z)}{p_1(z)p_3(z)}}=Q(z),$$

where $p_j(z) = 1 - 2x_j z + z^2$, $x_j = \operatorname{Re} z_j = \cos \theta_j$, $z_j = e^{i\theta_j}$, and $0 \leqq \theta_1 \leqq \theta_2 \leqq \theta_3 \leqq \pi$. If we let

$$Q(z) = 1 + \sum_{k=1}^{\infty} q_k z^k,$$

then

$$a_3 = \frac{q_1^2 + q_2}{2},$$

where

$$q_1 = x_1 + x_3 - 2x_2,$$

$$q_2 = \frac{3(x_1 + x_3)^2 - 4(x_1 x_2 + x_2 x_3 + x_3 x_1)}{2}.$$

We specifically considered the case where $M = 3$. As we constructed the values of Re a_3, for incremental values of argument α of the omitted slit, we found (using a value for TOL $= 10^{-9}$), however (see Table 1), that the conjecture was not correct. Indeed, Table 1 suggests for $M = 3$ that Re a_3 is unimodal as a function of α over the interval $[\pi/2, \pi]$ and that the maximum value occurs for some α in $[0.80\pi, 0.90\pi]$.

TABLE 1

α/π	Re a_3
0.500	0.88888889
0.550	0.89011287
0.600	0.89358764
0.650	0.89873157
0.700	0.90461073
0.750	0.91000424
0.800	0.91353195
0.850	0.91388099
0.900	0.91009490
0.950	0.90174616
1.000	0.88888889

We considered various values for M and found evidence to suggest that the argument α of the omitted slit for the extremal function for (7) appears to vary monotonically with M from $\pi/2$ to π as M varies from e to 5.

Barnard remarked, after seeing the numerical results in Table 1, that it should be possible to show analytically that the conjecture was false. Indeed, a long, extensive calculus argument can be given which shows that the conjecture fails for $2.83912 < M \leqq 3$. Specifically, the argument shows for $2.83912 < M$ that Re a_3 takes on a local minimum value at the SRT Pick function over the set of real-axis symmetric double radial slit mappings in $St(M)$.

Acknowledgment. We thank Professor R. W. Barnard for his helpful comments and suggestions during the preparation of this paper.

REFERENCES

[Ba] R. W. BARNARD, *A variational technique for bounded starlike functions*, Canad. J. Math., 2 (1975), pp. 337-347.

[BaLe] R. W. BARNARD AND J. L. LEWIS, *Coefficient bounds for some classes of starlike functions*, Pacific J. Math., 75 (1975), pp. 325–331.

[BaPe] R. W. BARNARD AND K. PEARCE, *Rounding corners of gearlike domains and the omitted area problem*, J. Comput. Appl. Math., 14 (1986), pp. 217–226.

[BjGr] P. BJØRSTAD AND E. GROSSE, *Conformal mapping of circular arc polygons*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 19–32.

[CaClHa] D. M. CAMPBELL, J. G. CLUNIE, AND W. K. HAYMAN, *Research problems in complex analysis*, in Aspects of Contemporary Complex Analysis, Proc. NATO Advanced Study Institute, University of Durham, 1979, Academic Press, London, 1980, pp. 527–572.

[Ga] D. GAIER, *Konstruktive Methoden der konformen Abbildung*, Springer-Verlag, Berlin, 1964.

[GoWe] G. H. GOLUB AND J. H. WELSCH, *Calculation of Gaussian quadrature rules*, Math. Comp., 23 (1969), pp. 221–230.

[Go] A. W. GOODMAN, *Conformal mapping onto certain curvilinear polygons*, Univ. Nac. Tucumán Rev. Ser. A, 13 (1960), pp. 20–26.

[He] P. HENRICI, *Applied and Computational Complex Analysis*, Vol. III, John Wiley, New York, 1986.

[JaMa] K. P. JACKSON AND J. C. MASON, *The approximation by complex functions of stresses in cracked domains*, in Algorithms for Approximation, Oxford University Press, London, 1987.

[Ne] Z. NEHARI, *Conformal Mapping*, McGraw-Hill, New York, 1952.

[Po] M. J. D. POWELL, *A* FORTRAN *subroutine for solving systems of nonlinear algebraic equations*, Tech. Report AERE-R.5947, Her Majesty's Stationery Office, London, England, 1968.

[ShGo] L. SHAMPINE AND M. GORDON, *Computer solution of ordinary differential equations: The initial value problem*, W. H. Freeman, San Francisco, CA, 1975.

[Tr] L. N. TREFETHEN, *Numerical computation of the Schwarz–Christoffel transformation*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 82–102.

[Tr83] ———, *Radial and gearlike domains*, unpublished memo, 1983.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 247-259, March 1991

© 1991 Society for Industrial and Applied Mathematics
002

PARALLEL ALGORITHMS FOR THE NUMERICAL SOLUTION OF INCOMPRESSIBLE FINITE ELASTICITY PROBLEMS*

MICHAEL SCHÄFER†

Abstract. This paper deals with the numerical solution of three-dimensional equilibrium problems in incompressible finite elasticity on parallel computers. The approximation of the underlying variational problems by appropriate finite-element methods is discussed and an augmented Lagrangian formulation of the resulting discrete systems is given. Starting from this formulation, a combination of a simple gradient method and a block relaxation process is used to obtain an algorithm for the complicated nonlinear problems, which can be implemented on parallel computers in a very efficient way. Several results of numerical experiments on an array processor illustrate the capabilities of the considered methods.

Key words. finite elasticity, parallel algorithms, augmented Lagrangian methods, preconditioned conjugate gradients

AMS(MOS) subject classifications. 73G05, 65N30

1. Introduction.

We consider the physical problem of the deformation of an incompressible hyperelastic body subjected to external forces. In the setting of nonlinear elasticity theory (e.g., Truesdell and Noll [13]) the problem can be formulated mathematically as a saddle point problem for a highly nonlinear Lagrangian functional in suitable Sobolev spaces.

The discretization of this problem by appropriate mixed finite-element methods results in a very large system of nonlinear equations. For the numerical solution of this system we use an augmented Lagrangian technique, which was proposed by Glowinski and Le Tallec [5]. Numerical results in Glowinski and Le Tallec [5] and Ruas [11] demonstrate the capabilities of these methods on sequential computers. We show that the underlying ideas can be used to obtain very efficient algorithms for the solution of incompressible finite elasticity problems on parallel computers.

Each step of the resulting method requires the solution of three linear systems with a symmetric, positive-definite matrix, which remains unchanged throughout the whole algorithm, and a large number of nonlinear systems (one for each finite element) of size 10×10. The small nonlinear systems are completely independent from each other, and therefore, their numerical solution, for instance by the Newton-Raphson method, can be parallelized in an optimal way. The linear systems can be solved by powerful preconditioned conjugate gradient methods. We discuss incomplete Cholesky factorization preconditioning methods and their parallelization by means of a multicolored ordering of the unknowns.

By a number of numerical experiments, which have been carried out on an ICL Distributed Array Processor with a 64×64 array of processing elements, we study convergence properties and computational efficiency of the proposed algorithms. The results illustrate that the methods are very well suited for solving problems of the considered type on parallel computers.

The underlying ideas can also be used to obtain efficient parallel algorithms for other types of problems, for example, from the area of compressible, inextensible, and dynamical nonlinear elasticity or from fluid mechanics.

* Received by the editors February 1, 1989; accepted for publication (in revised form) October 10, 1989.
† Institut für Mathematische Maschinen und Datenverarbeitung VII, Universität Erlangen-Nürnberg, Martensstrasse 3, D-8520 Erlangen, Federal Republic of Germany.

2. Statement of the problem. We consider the large deformation of an incompressible hyperelastic body under the action of external forces. Let $\Omega \subset R^3$ denote the open set occupied by the interior of the body in a fixed reference configuration. The deformation of the body is characterized by a vector field $u = u(x)$, where x denotes a particle $(x_1, x_2, x_3) \in \Omega$ labelled according to its position relative to a fixed Cartesian coordinate system in the reference configuration.

The body is subjected to body forces of intensity f per unit mass in Ω (i.e., gravitational or buoyancy forces) and to surface tractions t measured per unit area in the reference configuration. We consider the case of dead loading, where f and t do not depend on u. t is prescribed on a portion $\partial\Omega_2$ of the boundary $\partial\Omega$ of Ω; and on another part $\partial\Omega_1$ of $\partial\Omega$ the deformation takes the prescribed value u_0. We assume that

$$\partial\Omega = \overline{\partial\Omega}_1 \cup \overline{\partial\Omega}_2, \quad \partial\Omega_1 \cap \partial\Omega_2 = \varnothing, \quad \text{mes}\,(\partial\Omega_i) > 0 \quad (i = 1, 2).$$

The potential energy of the body is given by the functional

$$P(u) = \int_\Omega \sigma(x, \nabla u)\, dx + F(u),$$

where

$$F(u) := -\int_\Omega f \cdot u\, dx - \int_{\partial\Omega_2} t \cdot u\, ds$$

denotes the functional of the external forces and σ the stored energy function, which describes the mechanical response of the material under consideration. We suppose that σ has the form

$$\sigma = \frac{C}{2}(|\nabla u|^2 - 3) + g(|\text{adj}\,\nabla u|^2 - 3)$$

with $C > 0$ and g a polynomial. Here adj ∇u denotes the adjugate of ∇u and $|\cdot|$ is the usual norm of the space of 3×3 matrices. This choice of σ yields a good characterization of the mechanical behaviour of rubberlike solids (Rivlin and Saunders [10]).

Since we are considering incompressible materials the equilibrium positions of the body are characterized as stationary points of the Lagrange functional

$$L(u, p) = P(u) - \int_\Omega p(\det \nabla u - 1)\, dx, \tag{1}$$

where the scalar function p, which is called the hydrostatic pressure, constitutes the Lagrange multiplier associated to the incompressibility constraint $\det \nabla u = 1$.

In order to give an exact mathematical formulation of our problem, we define the Sobolev spaces

$$X := \{v \in W^{1,p}(\Omega)^3 : v = 0 \quad \text{on } \partial\Omega_1\}, \qquad Y := L^r(\Omega)$$

with p, $r \in (3, \infty)$ such that for $(u, p) \in X \times Y$ the integrals in (1) exist, and we make the assumptions

$$f \in L^q(\Omega)^3, \qquad t \in L^q(\partial\Omega_2)^3, \qquad u_0 \in W^{1,p}(\Omega)^3$$

with $q = p/(p-1)$.

Setting $v := u - u_0$, we obtain the following weak formulation of our problem (e.g., Oden [8]):

(2) Find $(v, p) \in X \times Y$ with

$$[L_u(v+u_0, p), w]_X = 0 \quad \forall w \in X,$$
$$[L_p(v+u_0, p), q]_Y = 0 \quad \forall q \in Y,$$

where

$$[L_u(v+u_0, p), w]_X := \int_\Omega \left[\frac{\partial \sigma(\nabla v + \nabla u_0)}{\partial \nabla u} - p \operatorname{adj} (\nabla v + \nabla u_0)^T\right] \cdot \nabla w \, dx + F(w),$$

$$[L_p(v+u_0, p), q]_Y := \int_\Omega q[\det (\nabla v + \nabla u_0) - 1] \, dx.$$

3. Discrete problems. To solve (2) numerically, we approximate X and Y by finite-element spaces $X_h \subset X$ and $Y_h \subset Y$, where $h > 0$ is a discretization parameter. For the construction of these spaces the well-known $Q_1 - P_0$ element (e.g., Girault and Raviart [4]) is used. We suppose that Ω is a polyhedron of R^3 and that $\{\Omega_k\}_{k=1}^N$ is a decomposition of $\bar{\Omega}$ into N hexahedrons Ω_k, which satisfy the classical assembly conditions for finite elements (see Ciarlet [2]). For every $k = 1, \cdots, N$ there exists a bijective mapping $B_k \in Q_1^3$ of the reference cube $[-1, 1]^3$ onto Ω_k, where Q_1 denotes the space of trilinear polynomials over $[-1, 1]^3$. We define X_h and Y_h by

$$\begin{aligned} X_h &:= \{v_h \in C^0(\bar{\Omega}) \colon v_h|_{\Omega_k} = z_h \circ B_k^{-1}, z_h \in Q_1, k = 1, \cdots, N\}^3 \cap X, \\ Y_h &:= \{p_h \in L^2(\Omega) \colon p_h|_{\Omega_k} = \text{const.}, k = 1, \cdots, N\}. \end{aligned} \tag{3}$$

The usual Galerkin approximation of (2) is given by (e.g., Oden [8])

(4) Find $(v_h, p_h) \in X_h \times Y_h$ with

$$[L_u(v_h + u_0, p_h), w_h]_X = 0 \quad \forall w_h \in X_h,$$
$$[L_p(v_h + u_0, p_h), q_h]_Y = 0 \quad \forall q_h \in Y_h.$$

Questions concerning the derivation of error estimates for the solutions of (4) are discussed, for instance, in Oden [8].

Problem (4) constitutes a system of highly nonlinear algebraic equations. The direct numerical solution of this system, for example, with the Newton-Raphson method, cannot be carried out efficiently, because in practical problems the number of unknowns is very large and the sparse linear systems, which have to be solved in each step, are very poorly conditioned. In order to obtain a fast algorithm for (4), which allows a high degree of parallelization, we use here an augmented Lagrangian technique, which is due to Glowinski and Le Tallec [5]. The underlying idea of this approach is to separate nonlinearities and derivatives by introducing new variables and to treat the relations between old and new variables by a combination of the penalty method and the Lagrange multiplier method.

We introduce the new variable

$$F_h := s_h(v_h + u_{0h}) \in Z_h,$$

where $Z_h := Y_h^{3\times 3}$. Here, $s_h : X_h \to Z_h$ is an approximation of $\nabla : X \to L^p(\Omega)^{3\times 3}$ (see § 6 for a possible choice of s_h) and u_{0h} is an approximation of u_0 in X_h. Mechanically, F_h is an approximation of the deformation gradient ∇u in the space Z_h. For $R > 0$ we

define the augmented Lagrangian functional $L_R^h: X_h \times Y_h \times Z_h \times Z_h$ by

$$L_R^h(v_h, p_h, F_h, \lambda_h) := \frac{C}{2}\int_\Omega |\nabla v_h + \nabla u_0 - 3|^2\, dx + \int_\Omega [g(|\text{adj } F_h|^2 - 3) - p(\det F_h - 1)]\, dx$$
$$+ \int_\Omega \lambda_h \cdot [s_h(v_h + u_{0h}) - F_h]\, dx + \frac{R}{2}\|s_h(v_h + u_{0h}) - F_h\|_{0,2}^2$$
$$+ F(v_h + u_{0h}).$$

Here, $\|\cdot\|_{0,2}$ denotes the usual L^2-norm over Ω and λ_h is the Lagrange multiplier associated with the constraint $F_h - s_h(v_h + u_{0h}) = 0$. Mechanically, λ_h can be interpreted as an approximation to the "incompressible" part of the first Piola-Kirchhoff stress tensor (Glowinski and Le Tallec [5]).

Using L_R^h instead of L we can reformulate problem (4) in the following form:

(5) Find $(v_h, p_h, F_h, \lambda_h) \in X_h \times Y_h \times Z_h \times Z_h$ with

$$[L_{Rv}^h(v_h, p_h, F_h, \lambda_h), w_h]_X = 0 \quad \forall w_h \in X_h,$$
$$[L_{Rp}^h(v_h, p_h, F_h, \lambda_h), q_h]_Y = 0 \quad \forall q_h \in Y_h,$$
$$[L_{RF}^h(v_h, p_h, F_h, \lambda_h), G_h]_Z = 0 \quad \forall G_h \in Z_h,$$
$$[L_{R\lambda}^h(v_h, p_h, F_h, \lambda_h), \mu_h]_Z = 0 \quad \forall \mu_h \in Z_h,$$

where

$$[L_{Rv}^h(v_h, p_h, F_h, \lambda_h), w_h]_X := C\int_\Omega (\nabla v_h + \nabla u_0) \cdot \nabla w_h\, dx + F(w_h)$$
$$+ \int_\Omega [\lambda_h + R(s_h(v_h + u_{0h}) - F_h)] \cdot s_h(w_h)\, dx,$$

$$[L_{Rp}^h(v_h, p_h, F_h, \lambda_h), q_h]_Y := -\int_\Omega q_h(\det F_h - 1)\, dx,$$

$$[L_{RF}^h(v_h, p_h, F_h, \lambda_h), G_h]_Z := \int_\Omega \left[g'(|\text{adj } F_h|^2 - 3)\frac{\partial \text{ adj } F_h}{\partial F_h}\text{adj } F_h - p_h \text{ adj } F_h^T\right] \cdot G_h\, dx$$
$$- \int_\Omega [R(s_h(v_h + u_{0h}) - F_h) + \lambda_h] \cdot G_h\, dx,$$

$$[L_{R\lambda}^h(v_h, p_h, F_h, \lambda_h), \mu_h]_Z := \int_\Omega [s_h(v_h + u_{0h}) - F_h] \cdot \mu_h\, dx.$$

Solving (5) means to determine stationary points of L_R^h in $X_h \times Y_h \times Z_h \times Z_h$. The validity of the transformation of (4) into (5) is shown by Lemma 3.1.

LEMMA 3.1. *The discrete problems* (4) *and* (5) *are equivalent.*

Proof. (i) Suppose that (v_h, p_h) is a solution of (4). If we define $F_h := s_h(v_h + u_{0h})$ and $\lambda_h := -p \text{ adj } F_h^T$, we have $(F_h, \lambda_h) \in Z_h \times Z_h$. Since $L(v_h + u_{0h}, p_h) = L_R^h(v_h, p_h, F_h, \lambda_h)$, it follows from (4) that $(v_h, p_h, F_h, \lambda_h)$ is a solution of (5).

(ii) Now, let $(v_h, p_h, F_h, \lambda_h)$ be a solution of (5). From (5d) we conclude $F_h = s_h(v_h + u_{0h})$ and from this together with (5c) we obtain $\lambda_h = -p_h \text{ adj } (s_h(v_h + u_{0h}))^T$. Inserting these results in (5a) and (5b) yields (4a) and (4b), which proves the lemma.

Since all nonlinear terms in L_R^h are expressed with the variable F_h, which is constant on each element Ω_k $(k = 1, \cdots, N)$, in (5) the complex nonlinearities of the

problem are reduced to the element level; a fact that will be very useful when solving (5) on a parallel computer.

4. Parallel algorithms. The main reason for the transformation of (4) into the augmented Lagrangian system (5), which at first sight seems to complicate the problem, consists in the fact that we are able to reduce the solution of (5) to the solution of a sequence of highly parallelizable subproblems. For this we use a combination of an outer simple gradient algorithm (Uzawa algorithm) and an inner block relaxation process (Glowinski and Le Tallec [5]). The simple gradient algorithm for (5) with primal variable (v_h, p_h, F_h) and dual variable λ_h reads as follows:

- Choose starting value $\lambda_h^0 \in Z_h$.
- For $n = 0, 1, 2, \cdots$ until convergence:

(6) Find $(v_h^n, p_h^n, F_h^n) \in Y_h \in Z_h$ with

$$[L_{Rv}^h(v_h^n, p_h^n, F_h^n, \lambda_h^n), w_h]_X = 0 \quad \forall w_h \in X_h,$$

$$[L_{Rp}^h(v_h^n, p_h^n, F_h^n, \lambda_h^n), q_h]_Y = 0 \quad \forall q_h \in Y_h,$$

$$[L_{RF}^h(v_h^n, p_h^n, F_h^n, \lambda_h^n), G_h]_Z = 0 \quad \forall G_h \in Z_h.$$

(7) $$\lambda_h^{n+1} := \lambda_h^n + \rho[s_h(v_h^n + u_{0h}) - F_h^n], \qquad \rho > 0.$$

The algorithm decouples the computation of (v_h, p_h, F_h) and λ_h. For the solution of the nonlinear system (6) we use the following block relaxation process, which decouples the computation of (p_h, F_h) and v_h:

- Set $v_h^{n,0} := v_h^{n-1}$.
- For $k = 0, \cdots, m$:

(8) Find $(p_h^{n,k+1}, F_h^{n,k+1}) \in Y_h \times Z_h$ with

$$[L_{Rp}^h(v_h^{n,k}, p_h^{n,k+1}, F_h^{n,k+1}, \lambda_h^n), q_h]_Y = 0 \quad \forall q_h \in Y_h,$$

$$[L_{RF}^h(v_h^{n,k}, p_h^{n,k+1}, F_h^{n,k+1}, \lambda_h^n), G_h]_Z = 0 \quad \forall G_h \in Z_h.$$

(9) Find $v_h^{n,k+1} \in X_h$ with

$$[L_{Rv}^h(v_h^{n,k+1}, p_h^{n,k+1}, F_h^{n,k+1}, \lambda_h^n), w_h]_X = 0 \quad \forall w_h \in X_h.$$

- Set $(v_h^n, F_h^n) := (v_h^{n,m}, F_h^{n,m})$.

To start the first block relaxation $(n = 0)$ we need a further starting value $v_h^{-1} \in X_h$. λ_h^0 and v_h^{-1} can be obtained, for example, from the reference configuration.

Summing up, the whole algorithm for the solution (5) reads:

- Choose $\lambda_h^0 \in Z_h$, $v_h^{-1} \in X_h$.
- For $n = 0, 1, 2, \cdots$ until convergence:
  - Set $v_h^{n,0} := v_h^{n-1}$.
  - For $k = 0, \cdots, m$:
    Compute $(p_h^{n,k+1}, F_h^{n,k+1})$ from (8).
    Compute $v_h^{n,k+1}$ from (9).
  - Set $(v_h^n, F_h^n) := (v_h^{n,m}, F_h^{n,m})$.
  - Compute λ_h^{n+1} from (7).

We remark that general results concerning the convergence of the Uzawa algorithm or the block relaxation method are not available. The major difficulty in deriving such results lies in the highly nonlinear incompressibility constraint. The convergence properties of the algorithm are illustrated by the numerical results in § 6.

Now we turn our attention to the structure and parallel solution of the subproblems arising during the above algorithm. The parallelization of the computation of λ_h^{n+1} through relation (7) poses no problems. Since $\lambda_h^{n+1}|_{\Omega_k} = \text{const.}$ for all $k = 1, \cdots, N$, all calculations can be performed simultaneously on each element Ω_k.

Solving the nonlinear system (8) means to solve N small nonlinear systems, one for each element Ω_k, of the form

$$g(y) = 0$$

with $g: R^{10} \to R^{10}$ and $y \in R^{10}$. All of these systems are completely independent of each other, and so their numerical solution, for instance by the Newton–Raphson method, can be parallelized in an optimal way.

In consequence of the splitting of the stored energy function σ, problem (9) is linear and can be decoupled with respect to the components v_{hi} $(i = 1, 2, 3)$ of v_h. Therefore, (9) is equivalent to three linear systems

$$Ax_i = b_i, \qquad i = 1, 2, 3, \tag{10}$$

with a symmetric and positive-definite matrix A, which is the same for all three systems and remains unchanged during the whole algorithm. The efficient solution of systems of the type (10) on a parallel computer will be the subject of the next section.

5. Preconditioned conjugate gradient methods. The numerical solution of the systems (10) can be performed with preconditioned conjugate gradient (PCG) methods, which works very efficiently for sparse symmetric positive-definite systems (see, e.g., Evans [3]). The rate of convergence of the basic CG method (Hestenes and Stiefel [6]) depends on the condition number $\kappa(A)$ of A: the smaller $\kappa(A)$, the more rapid the convergence. To accelerate the convergence, a preconditioning of A can be incorporated into the CG algorithm for a system $Ax = b$ in the following form (Meijerink and van der Vorst [7]):

- Choose starting value x_0.
- Set $r_0 := b - Ax_0$. Solve $C\hat{r}_0 = r_0$. Set $p_0 := \hat{r}_0$.
- For $k = 0, 1, \cdots$ until convergence:

$$\alpha_k = \frac{(\hat{r}_k, r_k)}{(p_k, Ap_k)},$$

$$x_{k+1} = x_k + \alpha_k p_k,$$

$$r_{k+1} = r_k - \alpha_k A p_k,$$

$$\text{solve } C\hat{r}_{k+1} = r_{k+1},$$

$$p_{k+1} = \hat{r}_{k+1} + \frac{(\hat{r}_{k+1}, r_{k+1})}{(\hat{r}_k, r_k)} p_k.$$

Here $(\cdot, \cdot)$ denotes the Euclidean inner product and C is the preconditioning matrix, which must be symmetric and positive definite. C can be viewed as an approximation of A and should satisfy the following criteria:

- $C\hat{r} = r$ can "easily" be solved,
- $\kappa(C^{-1}A)$ is substantially smaller than $\kappa(A)$.

Several possible ways have been proposed to obtain a suitable C (see Ortega and Voigt [9] for a survey). We concentrate here on incomplete Cholesky factorization preconditioning methods (Meijerink and van der Vorst [7]), which seem to be the most successful strategies for preconditioning on sequential computers.

We consider the following methods:

- SCG method: $C := \operatorname{diag}(A)$. This corresponds to a symmetric scaling of A, such that all main diagonal elements are equal to 1 (e.g., Ortega and Voigt [9]).
- ICCG method: $C := LL^T$, with the incomplete Cholesky decomposition LL^T of A (Meijerink and van der Vorst [7]).
- RICCG method: $C := RR^T$, with a reduced incomplete Cholesky decomposition RR^T, which results from LL^T by setting some of the diagonals in L to zero (Meijerink and van der Vorst [7]).

The major problem in the parallelization of incomplete Cholesky PCG methods consists in the solution of the system $C\hat{r} = r$, which must be carried out in each step. Clearly an exception is the SCG method, where $\hat{r}$ can be achieved by a simple parallel division. For the other methods the structure of C, which we obtain if we use the usual lexicographical numbering of the unknowns, is not amenable to parallelize the computation of $\hat{r}$. An approach to circumvent this problem is the concept of multicolor numbering of the unknowns (e.g., Adams-Ortega [1]). The nodal points are colored in such a way that there is no coupling between the unknowns, which correspond to the same color. For our problem with the discretization defined by (3) we achieve an optimal decoupling with eight colors. The resulting matrix A has the following structure:

$$A = \begin{bmatrix} D_1 & B_{21}^T & \cdots & B_{81}^T \\ B_{21} & D_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & B_{87}^T \\ B_{81} & \cdots & B_{87} & D_8 \end{bmatrix},$$

where the D_i $(i = 1, \cdots, 8)$ are diagonal and the B_{ij} $(1 \leqq j < i \leqq 8)$ are sparse.

As C, by definition, has the same structure as A, the forward-backward substitution process for solving $C\hat{r} = r$ can now be carried out in eight parallel steps. All components of the solution vector, which correspond to the same color, can be computed simultaneously.

6. Numerical results. All numerical experiments have been carried out on an ICL Distributed Array Processor (DAP) with a 64×64 array of processing elements. We have used the finite-element discretization defined by (3) and the gradient operator has been approximated for all $v_h \in X_h$ by

$$s_h(v_h)|_{\Omega_k} := \nabla v_h(M_k) \quad \forall k = 1, \cdots, N,$$

where M_k denotes the midpoint of Ω_k. For all calculations we have chosen the stored energy function

$$\sigma(\nabla u) = \frac{1}{2}(|\nabla u|^2 - 3) + \frac{7}{1{,}250}(|\operatorname{adj} \nabla u|^2 - 3).$$

Results for other constitutive relations can be found in Schäfer [12].

6.1. A model problem. First we analyse the numerical behaviour of the proposed algorithms by considering a model problem, whose analytical solution is known:

$$\Omega = (0, 1)^3, \quad \partial\Omega_1 = \{x \in \partial\Omega : x_3 = 0 \text{ or } x_3 = 1\}, \qquad f \equiv 0,$$

$$u_0 = \left(\frac{11}{10}x_1 - \frac{1}{20}, \frac{6}{5}x_2 - \frac{1}{10}, \frac{25}{33}x_3 + \frac{4}{33}\right),$$

$$t = \begin{cases} (\pm 0.21572, 0, 0) & \text{for } x_1 = 1, 0, \\ (0, \pm 0.39064, 0) & \text{for } x_2 = 1, 0. \end{cases}$$

The analytical solution is given by

$$u(x) = u_0(x), \qquad p(x) = 1 \quad \text{for } x \in \bar{\Omega}.$$

We choose a mesh of 3,375 elements and 4,096 nodes and start the Uzawa algorithm with the values $v_h^{-1} = 0$ and $\lambda_h^0 = Id$, which correspond to the reference configuration. Figure 1 shows the evolution of the errors for the unknowns v, p, F, and λ in the maximum norm. We have used $R = \rho = 3$ and 4 inner block relaxation steps. After five Uzawa iterations we already have an accuracy of $5 \cdot 10^{-4}$, which is the size of the discretization error. The corresponding computing time on the DAP amounted to only 102 seconds. We remark that two Newton-Raphson steps were sufficient to obtain the solution of (8) with a relative accuracy of 10^{-5}.

Next, we study the influence of the parameters R and ρ on the convergence properties of the Uzawa algorithm. In Figs. 2-4 the error $\|s_h(v_h + u_{0h}) - F_h\|_{0,2}$, which

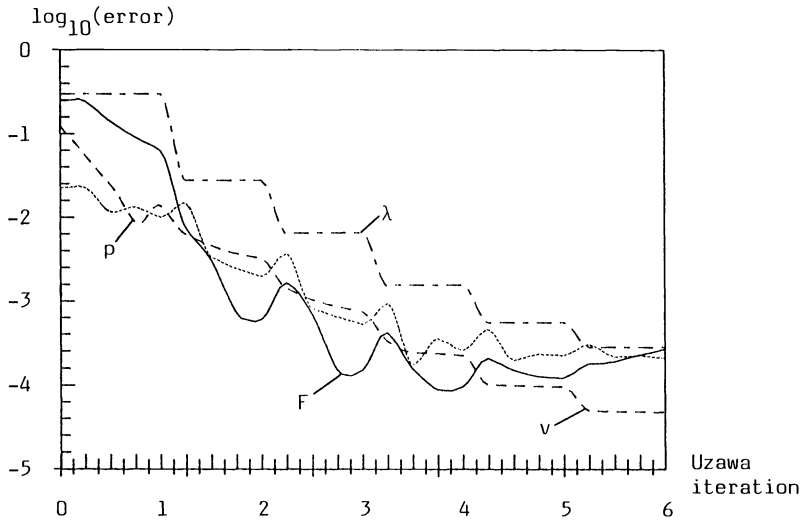


FIG. 1. *Evolution of the errors for v, p, F, and λ.*

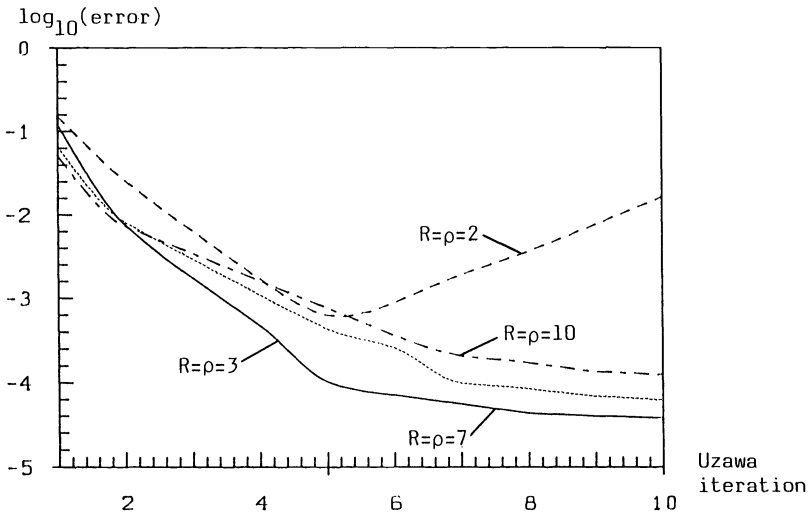


FIG. 2. *Rate of convergence* ($R = \rho$).

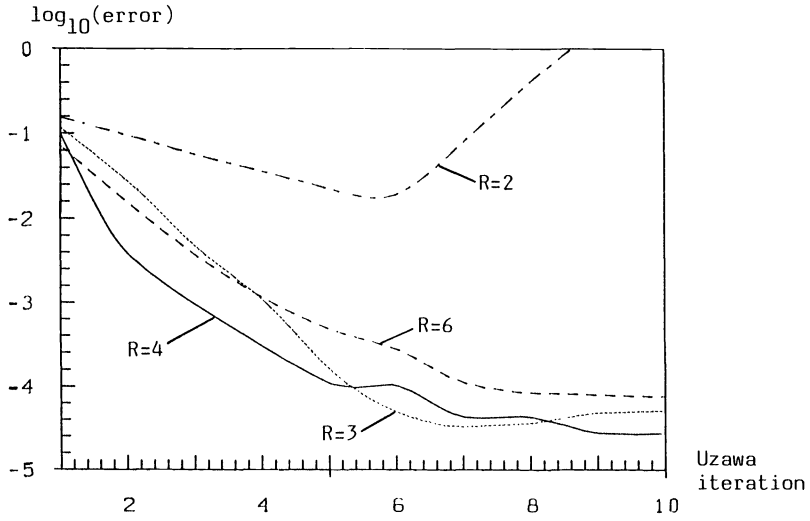


FIG. 3. *Rate of convergence* ($\rho = 4$).

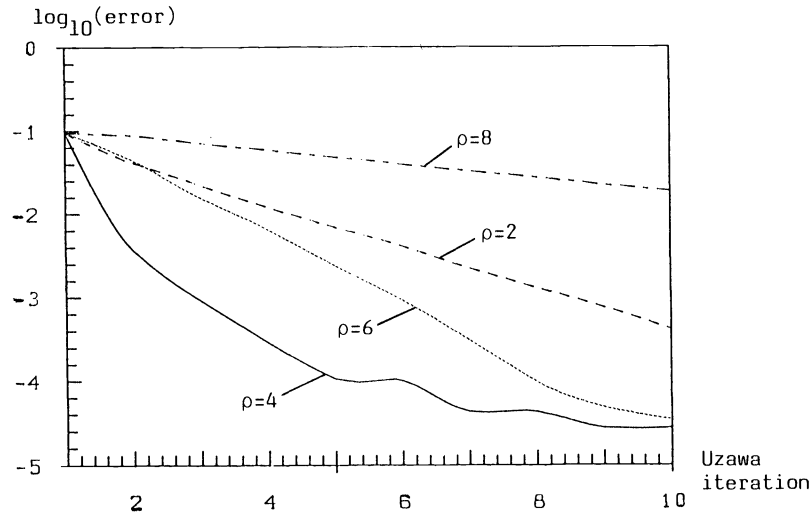


FIG. 4. *Rate of convergence* ($R = 4$).

is a measure of the convergence of the algorithm, is represented for different values of R and ρ. The associated computing times (in seconds on the DAP) for 10 Uzawa iterations are summarized in Table 1.

TABLE 1

$R\backslash\rho$	2	3	4	6	7	8
2	182	-	404	-	-	-
3	-	139	147	-	-	-
4	160	146	120	158	-	220
6	-	-	152	-	-	-
7	-	-	-	-	164	-

The following conclusions can be drawn from the results:

- If R is smaller than a certain limit value R_L, then the algorithm fails to converge.
- For $R > R_L$ the algorithm converges and the rate of convergence decreases with increasing values of R.
- The parameter ρ influences only the rate of convergence, not the convergence itself.
- The highest convergence rates are obtained with $R = \rho$.
- The closer R and ρ are to the optimal value, the smaller is the required number of iterations to solve the systems (8) and (9) within a prescribed accuracy.

We remark that these statements for the model problem also apply to more general problems. Only the limit value R_L varies as the case may be.

6.2. Extension and compression of rubber solids. We again consider the cube $(0, 1)^3$. The boundary conditions for an extension or compression of the cube in x_3-direction by fixed basal surface $x_3 = 0$ are:

$$\begin{aligned} u_0 &= x && \text{for } x_3 = 0, \\ u_0 &= (x_1, x_2, c) && \text{for } x_3 = 1, \\ t &= 0 && \text{otherwise.} \end{aligned}$$

$c \in (0, 1)$ corresponds to a compression and $c \in (1, \infty)$ to an extension.

Figures 5 and 6 show the numerically computed deformations in the cases $c = 0.5$ and $c = 2$ after 13 iterations of the Uzawa algorithm. The computing times on the DAP were 324 sec ($c = 0.5$) and 284 sec ($c = 2$).

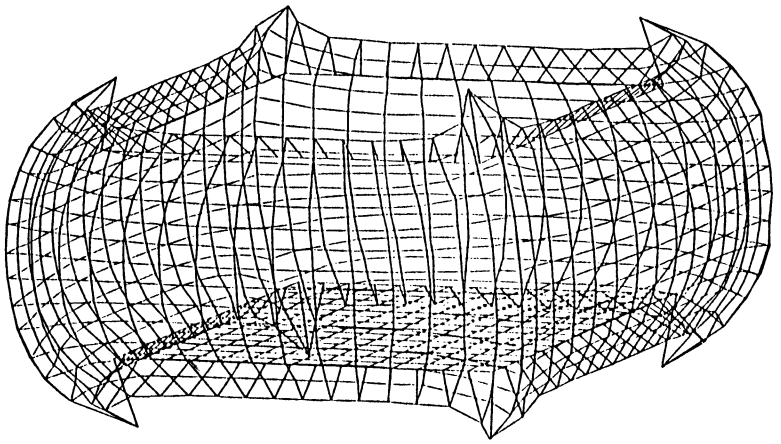


FIG. 5. *Equilibrium position* ($c = 0.5$).

Next, we show the ability of the examined algorithms for problems with bifurcation. We consider (2) with $\Omega = (0, 1)^2 \times (0, 10)$ and the boundary conditions

$$\begin{aligned} u_0 &= x && \text{for } x_3 = 0, \\ u_0 &= (x_1, x_2, 5) && \text{for } x_3 = 10, \\ t &= 0 && \text{otherwise.} \end{aligned}$$

The other data are given as in § 6.1. In the x_1- and x_2-directions we choose $h = 1/7$, and in the x_3-direction, $h = 1/63$. This results in a mesh with 3,087 elements and 4,096 nodes. Crucial for the convergence properties of the Uzawa algorithm is the choice of the starting values v_h^{-1} and λ_h^0. If we take $v_h^{-1} = 0$ and $\lambda_h^0 = Id$, the algorithm converges towards the unstable solution shown in Fig. 7(b). To obtain the stable bifurcated

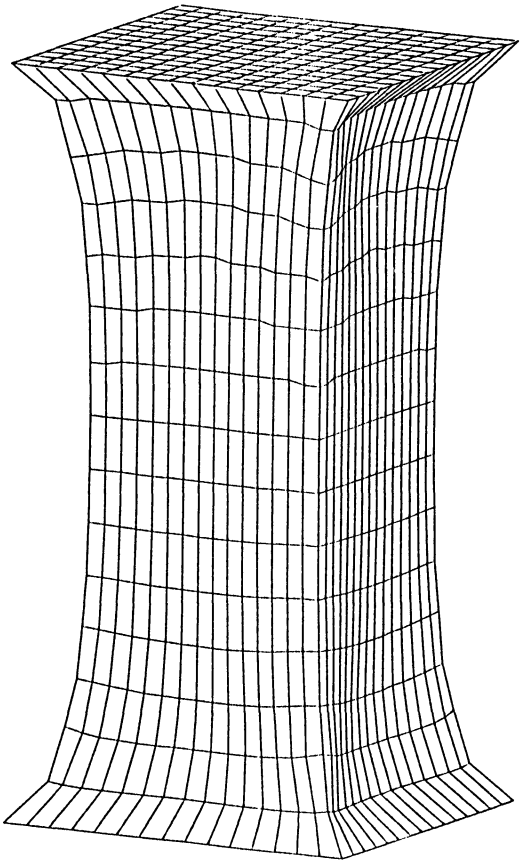


FIG. 6. *Equilibrium position* ($c = 2$).

solution illustrated in Fig. 7(c), we first replace the boundary condition $t = 0$ on the surface $x_1 = 0$ by a condition of the form

$$t = (\varepsilon, 0, 0)$$

with a suitable $\varepsilon > 0$. We carry out some Uzawa iterations for this perturbed problem, until we get a (with respect to the x_2-x_3) sufficiently asymmetric solution. With this solution we restart the algorithm, which now converges to the stable solution (Fig. 7(c)).

6.3. Comparison of PCG methods. We compare the PCG methods discussed in § 5 with regard to their efficiency on the DAP. To this, we consider the extension problem from § 6.2 with $c = 2$. In each Uzawa step $3m$ linear systems with the matrix A must be solved. Figure 8 shows for the different methods the number of iterations which were required in the mean for one of the $3m$ systems in the Uzawa steps to achieve a relative accuracy of 10^{-5}. We have chosen $R = \rho = 6$ and $m = 6$. The corresponding total computing times (in seconds on the DAP) and the total numbers of PCG iterations during the 13 Uzawa steps amounted to:

Method	SCG	CG	RICCG	ICCG
Time	284	310	701	920
Iterations	2059	2434	2036	1568

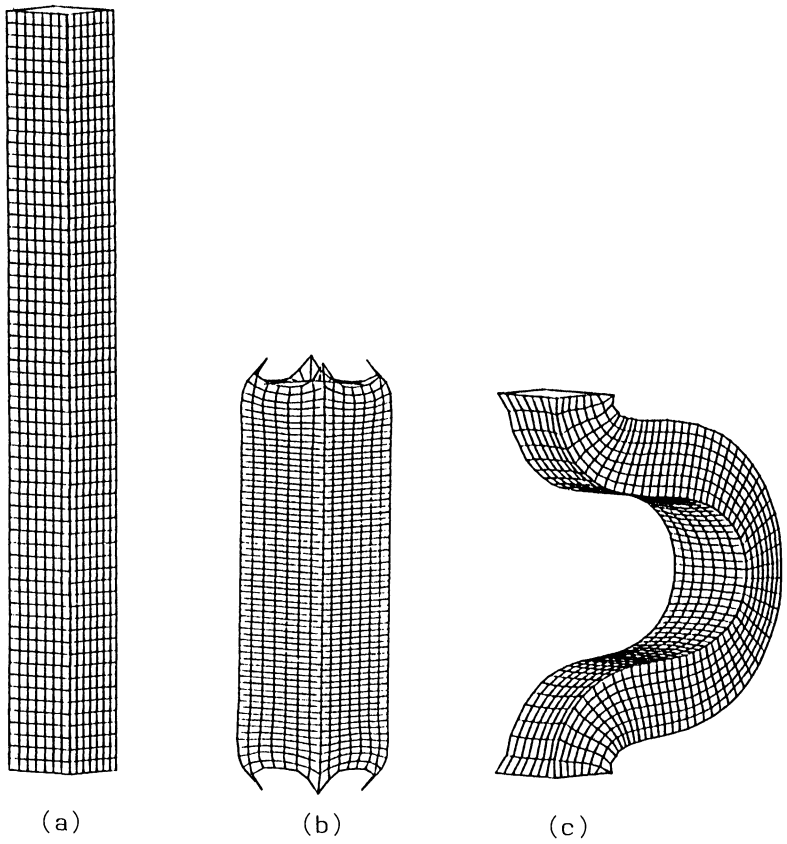


FIG. 7. *Problem with bifurcation.*

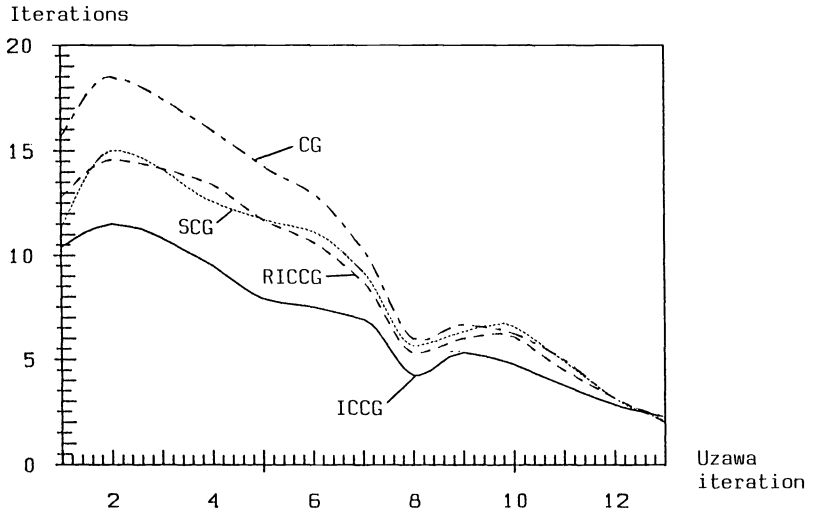


FIG. 8. *Comparison of* PCG *methods.*

In the RICCG method we have chosen the reduced incomplete Cholesky decomposition in such a way that only directly neighboured processing elements must interchange data in the solution process for the systems $C\hat{r} = r$. For the given discretization we achieve this by considering in R only those diagonals that correspond to the main diagonals of the submatrices B_{ij} $(1 \leqq j < i \leqq 8)$ of A. This results in eight nonzero diagonals in R, whereas in the full IC decomposition we have 27 nonzero diagonals.

We can conclude from the results that for our type of problems the reduction of the required number of iterations by means of an IC or RIC preconditioning is too small to compensate for the additional work, which arises from the multicolor ordering of the unknowns and the solution of the systems $C\hat{r} = r$. The SCG algorithm, which can be parallelized in a nearly optimal way without any modification, works very effectively for our problems and we conjecture that, for this type of system, other currently available incomplete factorization preconditioning methods can hardly compete with it on parallel architectures such as the DAP.

REFERENCES

[1] L. ADAMS AND J. ORTEGA, *A multi-color* SOR *method for parallel computation*, in Proc. 1982 Internat. Conference on Parallel Processing, Bellaire, MI, 1982, pp. 53–56.

[2] P. CIARLET, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.

[3] D. EVANS, *Preconditiong Methods: Analysis and Applications*, Gordon and Breach, New York, 1983.

[4] V. GIRAULT AND P. RAVIART, *Finite Element Methods for Navier–Stokes Equations*, Springer-Verlag, Berlin, 1986.

[5] R. GLOWINSKI AND P. LE TALLEC, *Numerical solution of problems in incompressible finite elasticity by augmented Lagrangian methods*, II. *Three-dimensional problems*, SIAM J. Appl. Math., 44 (1984), pp. 710–733.

[6] M. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, Nat. Bur. Standards J. Res., 49 (1952), pp. 409–436.

[7] J. MEIJERINK AND H. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[8] J. T. ODEN, *Recent developments in the theory of finite element approximations of boundary value problems in nonlinear elasticity*, Comput. Methods Appl. Mech. Engrg., 17 (1979), pp. 183–202.

[9] J. ORTEGA AND R. VOIGT, *Solution of partial differential equations on vector and parallel computers*, SIAM Rev., 27 (1985), pp. 149–240.

[10] R. RIVLIN AND D. SAUNDERS, *Large elastic deformations of isotropic materials*—VII. *Experiments on the deformation of rubber*, Philos. Trans. Roy. Soc. London Ser. A, 243 (1951), pp. 251–288.

[11] V. RUAS, *A class of asymmetric simplicial finite element methods for solving finite incompressible elasticity problems*, Comput. Methods Appl. Mech. Engrg., 27 (1981), pp. 319–343.

[12] M. SCHÄFER, *Probleme der nichtlinearen Elastizitätstheorie und ihre numerische Lösung am Parallelrechner*, Ph.D. thesis, Institut für Angewandte Mathematische, Universität Erlangen-Nürnberg, Erlangen, Germany, 1987.

[13] C. TRUESDELL AND W. NOLL, *The non-linear field theories of mechanics*, In Handbuch der Physik III/3, S. Flügge, ed., Springer-Verlag, Berlin, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 260-272, March 1991
© 1991 Society for Industrial and Applied Mathematics
003

DIAGNOSING STIFFNESS FOR RUNGE-KUTTA METHODS*

LAWRENCE F. SHAMPINE†

Abstract. Explicit Runge-Kutta methods are a popular way to solve the initial value problem for a system of ordinary differential equations. Although very effective for nonstiff problems, they are impractical for stiff problems. This paper is concerned with how to diagnose stiffness as the reason for unsatisfactory performance by a code based on explicit Runge-Kutta formulas.

Key words. stiffness, Runge-Kutta

AMS(MOS) subject classification. 65L05

1. Introduction. Explicit Runge-Kutta methods furnish a popular way to solve the initial value problem for a system of ordinary differential equations,

$$y' = f(x, y), \qquad a \leqq x \leqq b, \qquad y(a) \text{ given.}$$

Although very effective for nonstiff problems, they are impractical for stiff problems. Unfortunately, it is often not easy to decide if a problem is stiff before attempting its solution. Mathematical software of the highest quality recognizes when its performance is unsatisfactory and tries to find out why, so as to direct the user to a more appropriate code. To construct software of this quality for the solution of the initial value problem, we must be able to diagnose stiffness as the reason for unsatisfactory performance. This paper is concerned with how to do this for codes based on explicit Runge-Kutta formulas.

Practically always the user of a code based on explicit Runge-Kutta formulas will apply it to a nonstiff problem. For this reason we must devise an algorithm that has a negligible cost when the problem is clearly nonstiff. On the other hand, an unreliable scheme is worse than none at all, so when the problem might be stiff, we must be willing to pay something for a reliable diagnosis. In § 2 we propose some very inexpensive tests that serve as necessary conditions for a diagnosis of stiffness. If we decide that stiffness is possible, we compute dominant eigenvalues of a local Jacobian by a nonlinear power method. We have found that reliable computation of these eigenvalues in the present context requires careful attention to a number of practical matters that we take up in § 3. With care a check can be made with considerable reliability at a cost comparable to one or two steps with the Runge-Kutta method. Because checks are made only when there is reason to suspect stiffness, they add little to the cost of the whole integration. It is not generally appreciated how difficult it is to classify problems, even in principle. Certainly one or two dominant eigenvalues of a local Jacobian are not enough to do this. Section 3 discusses how to make a diagnosis using a couple of dominant eigenvalues and additional information that we glean whilst deciding whether to compute the eigenvalues. Substantial numerical experiments that show our algorithm to be efficient and effective are described in § 4.

2. When to check. Ideally we would check for stiffness at each step and either warn the user when stability appears to be severely restricting the stepsize, or switch

* Received by the editors August 24, 1988; accepted for publication (in revised form) October 26, 1989. This research was partially supported by North Atlantic Treaty Organization research grant 898/83 and by the Applied Mathematical Sciences program of the Office of Energy Research under Department of Energy grant DE-FG05-86ER25024.

† Mathematics Department, Southern Methodist University, Dallas, Texas 75275.

automatically to a more effective method. It is not clear that this is possible, or appropriate, but in any case, our goal is much less ambitious. We aim to diagnose stiffness as the reason for unsatisfactory performance. This less ambitious goal means that we do not test for stiffness at every step, far from it. Testing at every step is practical only if the test is very inexpensive. By testing infrequently, we can afford much more reliable tests.

The typical code intended for nonstiff problems reports to the user when a considerable amount of work has been expended on the solution of the problem. It is usual to make such a report when f has been evaluated MAXFCN times. Typical values of MAXFCN range from 2,000 to 5,000. If the cost of solution is acceptable, we do not inquire about stiffness. In the first instance, if the code makes MAXFCN evaluations of f, we ask whether this is due to stiffness. Because MAXFCN is chosen big enough to permit the solution of typical nonstiff problems, this approach means that we add nothing to the cost of integration of a problem that is clearly nonstiff. On the other hand, it represents a considerable waste of effort if the problem is stiff. The remainder of this section is devoted to simple tests that allow us to decide earlier in the integration whether to make a relatively expensive test for stiffness. Later we shall see that these new tests are also useful in diagnosing stiffness.

Monitoring the frequency of failed steps can be quite helpful. In our code we ask if there were 10 or more failed steps in the course of taking 40 successful steps. This is an alarming number of failures, and certainly justifies inquiring if stiffness is the reason for this unsatisfactory behavior, but the fact is that there is almost no other reason for a large proportion of failed steps. The matter has been studied by Shampine [13] and by Hall and Higham [5]-[7]. The behavior depends on the particular formula and the location in the complex plane of the eigenvalues of the local Jacobian. For the popular formulas, a high proportion of failures is quite likely when the problem is stiff. A (4, 5) pair of Dormand and Prince is treated as an example in [7]. The analysis applies to the modified pair called the DPS (4, 5) pair [17] that we have used in our experiments, and it says that we should expect a lot of failures. Our code also implements a (7, 8) pair of Prince and Dormand [10]. Analysis of this pair shows that it is even more likely to suffer a large number of failures in the presence of stiffness, and this agrees with our experience.

On a diagnosis of stiffness, a matter of considerable interest is how much it would cost to complete the integration. If, for example, the code reports that the stepsize is being restricted on grounds of stability, but it predicts that the integration can be completed in, say, 0.5 MAXFCN additional evaluations of f, then it would probably be best to carry on. There is some theory and a great deal of experience [3], [5]-[7], [12]-[14], [18] that says that when the stepsize is being restricted on grounds of stability, the average stepsize corresponds to being on the boundary of the stability region of the method. With an average stepsize and the cost per step of the formula pair, it is easy to predict roughly how many evaluations of f will be required to finish the integration. When combined with the cost of the integration to the current point, we can predict the total cost of the integration. There are a couple of practical issues that need some discussion. As already noted, there will generally be a lot of step failures when the stepsize is restricted by stability. It does not seem possible to account for potential failures in a simple way, and a crude prediction will suffice for our purposes, so we are content with the predictor described. An average stepsize must be computed from the sizes of the successful steps because in the presence of stiffness, steps may be tried that are much too large to succeed. It also happens that stepsize algorithms may use stepsizes that are temporarily rather smaller than necessary, so we

must average over a fair number of steps to get a representative value. In our code we average over 40 successful steps. To reduce the effects of a possible initial transient when the problem is stiff, we take 10 steps before starting the computation of an average stepsize.

The scheme for predicting the work can be used to avoid unnecessary tests for stiffness. Suppose we have reason to suspect that the problem is stiff, e.g., there have been a lot of failed steps. If we then predict that fewer than MAXFCN evaluations of f will suffice to complete the integration, there is no point in making a relatively expensive test for stiffness: If the stepsize is being restricted on grounds of stability, our prediction of the cost will be reasonably good, and the restriction is not so severe compared to the length of the interval of integration that we need concern ourselves with the matter. If the stepsize is not being restricted, the prediction is not valid, but this does not matter because the problem is then not stiff, and we certainly do not wish to test just to be told that the problem is not stiff. In point of fact, if the problem is clearly nonstiff, this way of predicting the cost is quite likely to predict a cost less than MAXFCN evaluations of f because MAXFCN is selected to be large enough to permit the solution of the typical nonstiff problem. Thus, we have here a kind of necessary condition for a diagnosis of stiffness.

The theory of stiffness supposes that the local Jacobian is constant. We shall approximate its dominant eigenvalues and compare the product of the stepsize and a dominant eigenvalue to the boundary of the stability region of the formula. Obviously we need to test the hypothesis that the local Jacobian is roughly constant. This need was brought forcefully to our attention when experimenting with the problem D5 of the set of nonstiff test problems [8]. This is a two-body problem with a rather eccentric orbit. At closest approach the integral curve is quite unstable—the local Jacobian has dominant real eigenvalues $\pm\lambda$ with $\lambda \gg 1$. The orbit is smooth enough that the average stepsize is much larger than the stepsize at close approach. If the eigenvalue $-\lambda$ is computed as the dominant one and the average stepsize is used with this eigenvalue, the problem will be diagnosed incorrectly as stiff. One way a rapidly changing Jacobian might be revealed is by the current stepsize being much different from the average. For this reason, if the current (successful) stepsize differs substantially from the average, we do not test for stiffness. A related matter is that a high-order pair, such as (7, 8), can at crude tolerances experience a number of step failures when the behavior of the solution changes sharply. This is the only reason other than stiffness that is at all likely to lead to a lot of failed steps. Comparing the current to the average stepsize is an effective device for avoiding an unnecessary test for stiffness in this situation, too.

In summary, we test for stiffness only when we have done MAXFCN evaluations of f, or we have observed a lot of step failures—in our tests, 10 or more failed steps in the course of taking 40 successful steps. Even then, we do not test when the current successful stepsize differs considerably from the average—in our tests, by a factor of five larger or smaller—nor when we predict that fewer than MAXFCN evaluations will suffice to complete the integration. This prediction is done using the average successful stepsize—in our tests, after an initial 10 steps, the average is computed every 40 (successful) steps. Specifically, it is assumed that the rest of the integration will consist of successful steps of this length.

Our experiments confirm the arguments here that these simple tests avoid testing for stiffness when the problem is clearly not stiff, minimize the cost of testing when the problem is marginally stiff, and diagnose stiffness early, i.e., well before expending MAXFCN evaluations, when the problem is quite stiff. These new devices are, therefore, of considerable value in their own right. However, we shall see that the information

gathered and the circumstances in which we go to the expense of computing eigenvalues of a local Jacobian are essential to diagnosing stiffness.

3. How to check. In this section we discuss the computation of dominant eigenvalues of a local Jacobian for use in a diagnosis of stiffness. It is important that we avoid diagnosing a problem as stiff when it is not. This is not possible unless we can rely upon the eigenvalues that we compute. It is not necessary to compute the eigenvalues very accurately, but we must be confident that we have some accuracy. For this reason, our approach is quite different from that of many authors who seek a "free," or very inexpensive, indicator of possible stiffness. We are prepared to pay a substantial price for reliability if necessary. Fortunately, it is possible to get reliable results at a modest cost—the cost, say, of one or two steps. We begin by briefly recalling the origin of some "free" estimates of dominant eigenvalues. It is worth pointing out why we cannot be confident of the accuracy of these estimates, but our main reason is to prepare for an investigation of an idea of Robertson [12] for starting a nonlinear power iteration for the computation of dominant eigenvalues. We shall show that this idea provides a very good starting vector. We go on to take up practical issues that we have found essential for the reliable computation of dominant eigenvalues in the present context.

We consider a single step of an explicit Runge-Kutta formula applied to

$$\frac{dy}{dx} = f(y), \qquad y(x_n) = y_n. \tag{1}$$

(We do not assume that the equation is autonomous; we merely suppress the independent variable in f here and in expressions that follow for notational convenience.) An explicit Runge-Kutta scheme has the form

$$u_0 = y_n, \qquad f_0 = f(u_0), \tag{2a}$$

and for $i = 1, \cdots, s-1$,

$$u_i = u_0 + h \sum_{j=0}^{i-1} \beta_{i,j} f_j, \qquad f_i = f(u_i). \tag{2b}$$

We consider Runge-Kutta pairs, so there is a result

$$y_{n+1} = y_n + h \sum_{i=0}^{s-1} \gamma_i f_i \tag{3}$$

of order $p \geqq 1$,

$$y(x_n + h) - y_{n+1} = \mathcal{O}(h^{p+1}),$$

and a companion result

$$y^*_{n+1} = y_n + h \sum_{i=0}^{s-1} \gamma^*_i f_i \tag{4}$$

of order $p+1$ that is used to estimate the local error by

$$y^*_{n+1} - y_{n+1} = (y(x_n + h) - y_{n+1}) + \mathcal{O}(h^{p+2}).$$

The considerations that follow are exact when f has the form

$$f(y) = Jy + g \tag{5}$$

for a constant matrix J and a constant vector g. It is usual to take $g=0$, but this is unnecessary and limits the applicability of the results. Results obtained for f of the form (5) are approximately true for more general f because locally

$$f(y)\approx f(y_n)+\left.\frac{\partial f}{\partial y}\right|_{y_n}(y-y_n). \tag{6}$$

Of course, this is only reasonable for y near y_n. If f is not autonomous, there are additional terms in a linearization like (6), with the consequence that we are restricted to x near x_n, too. In any case, we must allow nonzero g in (5) if we wish to approximate general f.

It is easily seen from the definition of the Runge-Kutta formulas that for f of the form (5), the computable quantity

$$\Delta_i = h(f_i - f_0) = Q_i(hJ)hf_0, \tag{7}$$

where $Q_i(hJ)$ is a polynomial of degree i in hJ. By linearization as in (6), the representation of Δ_i is approximately true for more general f. If we write $v=hf_0$, then it is clear from (7) that a suitable linear combination of $\Delta_1, \cdots, \Delta_i$ will yield $(hJ)^i v$, except in degenerate cases when only a power smaller than i is possible. Using the computable quantities v and the Δ_j obtained in this way underlies a number of attempts to approximate the dominant eigenvalues, the spectral radius, or a norm of hJ (see, for example, Gladwell, Shampine, and Brankin [4], Robertson [12], Shampine [15], Sottas [19], and Wolfbrandt [20]). It is appealing because the stages f_j are in any case evaluated, and the extra computation is small enough that the estimate may be described reasonably as "free." There are some practical difficulties that have received little comment. Some of the early work [11], [15], [19], [20] is based on only a few stages, too few in our experience to be confident of convergence of the power method. A matter that has been neglected in this work is a test for convergence. A serious difficulty with the free estimates is that quantities may not be evaluated close enough to (x_n, y_n) for linearization to provide a good description of the function f. This is easy to appreciate in terms of the independent variable, but we should not overlook the fact that with high-order formulas, y_{n+1} can differ significantly from y_n. It is important to keep in mind that we are not computing the dominant eigenvalues of a known matrix. For a general f, the matrix J is merely a "representative" approximation to the underlying Jacobian. Indeed, we cannot form products hJv directly; they are computed as linear combinations (basically differences) of values of the nonlinear function f. An equally serious related difficulty is that quantities may be so close to (x_n, y_n) that their differences are not significant in finite precision arithmetic.

For the reasons presented, the author believes that the free estimates are not sufficiently reliable to serve as the basis for diagnosing stiffness in quality mathematical software. On the other hand, we need a good starting vector for a nonlinear power method, and it is possible to construct a very satisfactory one in this way. In [15] Shampine used hf_0 to start an iteration to compute $h\rho(J)$. It works well for f of the form (5), and linearization suggests that it is a reasonable choice for general f. In his Master of Science thesis [12], Robertson proposed using instead the estimated error vector $e=y^*_{n+1}-y_{n+1}$, and his experiments showed this to be a much better choice. A theoretical justification was provided for problems of the form $f(y)=Jy$ that is based on the behavior of the error over a sequence of steps of constant size. We provide a different and illuminating justification that eliminates several restrictions: The homogenous form of (5) entails no loss of generality for some questions, but, as we observed earlier, it cannot be used to model the solution for general f in the way

needed here. The linearization that underlies the application of the model to general problems is much less convincing when it is to hold over several steps. Runge-Kutta codes may alter the stepsize at any step.

When examining Robertson's experimental results, it struck the author that with a code based on a six stage (4, 5) pair, m iterations from the start e gave virtually the same result as $m+4$ iterations from the start hf_0. On investigation it became apparent that this phenomenon is general and can be explained easily. Consistency ($p \geqq 1$) implies that

$$\sum_{i=0}^{s-1} \gamma_i = 1.$$

This fact and a little manipulation of the formulas for y_{n+1} and y^*_{n+1} results in

$$e = y^*_{n+1} - y_{n+1} = \sum_{i=0}^{s-1} (\gamma^*_i - \gamma_i) h(f_i - f_0).$$

Then for linear problems (5),

$$e = \sum_{i=0}^{s-1} (\gamma^*_i - \gamma_i) Q_i(hJ) hf_0 = S(hJ) hf_0,$$

with $S(hJ)$ a polynomial of degree $s-1$ in hJ. The stability polynomial $P(hJ)$ of the formula (3) of order p has the form

$$P(hJ) = I + (hJ)/1! + \cdots + (hJ)^p/p! + \delta_{p+1}(hJ)^{p+1} + \cdots + \delta_s(hJ)^s,$$

where the coefficients $\delta_{p+1}, \cdots, \delta_s$ are characteristic of the formula. Using this expression and the corresponding one for the stability polynomial of the formula (4), it is easy to show that

$$S(hJ) = [(\delta^*_{p+1} - \delta_{p+1})(hJ)^p + \cdots + (\delta^*_s - \delta_s)(hJ)^{s-1}].$$

The fact that we require from this analysis is that e is a linear combination of terms $(hJ)^i hf_0$ with powers $i \geqq p$. This means that Robertson's starting vector e corresponds to having started with hf_0, done $s-1$ iterations of a power method, and formed a linear combination of the iterations from p on. Roughly speaking, this is like doing p iterations with a power method. It can be much better than this because when an eigenvalue dominates strongly, the combination is more like having done $s-1$ iterations. Considering that hf_0 is itself likely to be rich in directions corresponding to large eigenvalues, the reader might share the author's hope that for a high-order pair, only one iteration would be needed to get convergence from this starting vector. Alas, experiment has shown that this need not be the case. When an eigenvalue dominates strongly it *is* the case, and the starting vector is very often an excellent start, but when the problem is nonstiff, and especially when there are subdominant eigenvalues of magnitude close to the spectral radius, more iterations may be needed.

After we have decided that stiffness is sufficiently likely to warrant checking, we compute the dominant eigenvalues of the local Jacobian and use them to diagnose stiffness. The tests made to assist in this decision give us some confidence that the Jacobian is not changing rapidly, hence that the usual theory of absolute stability is applicable. Let h be the *average* stepsize. If the dominant eigenvalues of hf_y are well inside the stability region of the formula, the stepsize is not being restricted on grounds of stability and the computation is stable. For this reason we introduce the factor h, and in what follows, when we speak of computing dominant eigenvalues, we are speaking of the eigenvalues of the scaled local Jacobian hf_y. Later we observe that this

scaling serves other purposes as well. We use a nonlinear power method for computing the dominant eigenvalues that is much the same as that described in [15] and is related to that of [9]. Important practical matters are taken up there that we do not repeat. Here we take up matters that are different because the task is different, and we take up other practical matters that are essential to reliable computation.

For notational convenience we continue to omit the independent variable when we write the function f. When we have reached a point (x_n, y_n) in the integration, we wish to approximate the dominant eigenvalues of $hJ = hf_y(x_n, y_n)$. In the following we write y instead of y_n. The fundamental operation is to compute from a vector v a quantity

$$T(v) \approx hJv.$$

We normally have $f(y)$ at our disposal from the integration. We scale v to $w = \delta v$, and then we compute

$$T(v) = \frac{h}{\delta}[f(y+w) - f(y)] \approx \frac{h}{\delta} Jw = hJ.$$

Noting that $T(v)$ amounts to a difference approximation to a derivative in the direction of v, it is clear that we must take w to be "small" to get an accurate approximation. As usual with difference approximations, we cannot let w be "too" small, else the difference will not be significant. Because the attempts to get free estimates do not control the sizes of such quantities, they can get into trouble. The scaling we use is a development of that discussed in [15].

With a way of approximating the product $(hJ)v$, we can consider a power method for approximating the dominant eigenvalues of hJ. Starting with a vector v, we compute $(hJ)v$, $(hJ)^2v$, $(hJ)^3v$, etc. In general there is a danger of overflow with the power method. We start with a vector v scaled to unit length. Let λ be an eigenvalue of J. If stability is restricting the stepsize, $h\lambda$ will be in the stability region, and hence will be bounded in magnitude by the radius of a disk that encloses the stability region. Because the stability regions of explicit Runge–Kutta formulas are not large, the iterates cannot grow terribly fast. Thus, working with hJ instead of J allows us to compute several iterates without fear of overflow.

To calculate the two dominant eigenvalues of hJ, three successive iterates v, $w \approx (hJ)v$, and $z \approx (hJ)^2v$ are used, and coefficients α, β that minimize the residual $\|z + \alpha w + \beta v\|$ are found. The roots of

$$\lambda^2 + \alpha\lambda + \beta = 0$$

are taken as approximations to the two dominant eigenvalues of hJ. This is a standard approach. We followed the algorithm in Franklin [2] with some important differences due to the special circumstances. Some authors do not even check for convergence in the present context; in our experience a robust and reliable algorithm requires great care.

If the initial vector v is very close to an eigenvector, the process degenerates and the formulas in [2] for the least squares fits α and β break down. This is not merely a formal degeneracy. Because it is frequent that a real eigenvalue μ is strongly dominant when the problem is stiff and because Robertson's starting vector represents a number of iterations when the Runge–Kutta formula is of moderate to high order, it is often the case that v is *very* close to an eigenvector. When this is the case, we have $w \approx \mu v$ and $z \approx \mu w \approx \mu^2 v$. The residual is made very small by $\alpha = -\mu$, $\beta = 0$, which results in approximate eigenvalues $\lambda = 0$, μ. An equally small residual is provided by $\alpha = 0$, $\beta = -\mu^2$, which results in $\lambda = -\mu, \mu$. Clearly the computed eigenvalues are very sensitive

to the data. It is true that the dominant eigenvalue μ is not sensitive, but for our purposes the difference in the approximation to the subdominant eigenvalue is disastrous. If $h\mu \ll -1$, we would diagnose a problem with eigenvalues 0, μ as stiff, but we would describe a problem with eigenvalues $\pm\mu$ as unstable, hence *non*stiff. Because of the sensitivity of the fit, the negative eigenvalue is as likely to be the bigger in magnitude as the positive one; this causes trouble in classifying the problem.

Franklin advises that the case of a dominant real root be recognized before proceeding to compute a pair. For diagnosing stiffness obviously it is helpful always to compute two eigenvalues. We did this until we found that not recognizing the case of a strongly dominant real eigenvalue could lead to an incorrect diagnosis of stiffness. Now we check as often as possible whether there is a dominant real eigenvalue. A dominant real eigenvalue is computed by a Rayleigh quotient, which corresponds to minimizing another residual, namely $\|w+\alpha v\|$, and taking $\lambda = -\alpha$. Franklin suggests that convergence be tested by comparing the size of the residual to a tolerance times the size of w. If the matrix is symmetric, this assures us that we have computed an accurate eigenvalue. If there is no convergence, he goes on to consider a pair of eigenvalues and again tests for convergence by examining the size of a residual. Unfortunately, the Jacobian matrices that arise in the present context are rarely symmetric and the eigenproblem can be quite ill-conditioned. We found that even a very stringent test on the size of the residual could lead to unacceptable approximate eigenvalues. To counter this, we decided to test convergence by comparing successive iterates. Again we found in experiment that very close agreement of iterates did not imply accurate eigenvalues. It is important to appreciate that we have no use for very accurate eigenvalues. Just one correct digit would suffice, but we must be able to trust this digit, and if it is necessary to a lot of work to get a credible value, we must pay the price. In our code we make both kinds of tests. At each iteration we first test for a dominant real eigenvalue: We ask if the computed residual is exactly zero. (This is not unusual with the first iterate, and it can happen with a later iterate.) If it is not, then with all iterates except the first we ask if the current Rayleigh quotient agrees with the last one to a relative accuracy of 0.1 percent *and* provides a residual that is no more than 0.1 percent of the size of the current iterate. When these tests do not show a dominant real eigenvalue to be present, we approximate a pair of eigenvalues. As in the simpler case, we check both a residual and the convergence of successive approximate eigenvalues. The convergence test on the eigenvalues is made only on the eigenvalue of larger magnitude, taking the one with positive imaginary part when the pair consists of complex conjugates. On general grounds, and specifically in this context for reasons pointed out earlier, the subdominant eigenvalue might be computed (very) inaccurately. Because of this we do not ask for convergence to the subdominant eigenvalue, and we use its approximation only when its size is comparable to that of the dominant eigenvalue so that it should be acceptably accurate.

These convergence tests strike us as remarkably stringent, but we found them necessary in order to reliably compute approximations to the dominant eigenvalues that have a correct digit. Besides, in our experience, when the problem is quite stiff, the tests are passed easily. Indeed, we have usually gotten convergence in just two or three iterations. The situation with nonstiff problems is different. It is quite possible then that the nonlinear power method fail to converge in the number of iterations allowed. This does not cause trouble with the diagnosis because the problem is presumed to be nonstiff until evidence to the contrary is found. Because a convergence failure is relatively expensive, we try to recognize that a problem is nonstiff before even attempting to compute dominant eigenvalues.

Convergence to a dominant real eigenvalue can occur at any iteration, but for a pair we must form at least v, $(hJ)v$, $(hJ)^2v$, $(hJ)^3v$ for our convergence tests. Because convergence is usually so fast, we have organized our code to compute exactly these quantities. If we do not get convergence, we normalize the last iterate $(hJ)^3v$ and take it to be a new v for another cycle. Along with the scaling by h discussed earlier, this controls the size of quantities quite adequately. Because the iterations are relatively inexpensive and because we do not check often, we have been willing to go through as many as eight cycles in the computation of the dominant eigenvalues. This corresponds to a maximum of 24 evaluations of f, the cost of a few steps, so it is a very modest price to pay for explaining unsatisfactory performance by the integrator.

4. Diagnosing stiffness. Now we consider how to use the information gathered by the devices of §§ 2 and 3 to diagnose stiffness. Surprisingly little attention has been given to this matter. We begin by considering problems having eigenvalues with positive real part. If the eigenvalue of largest magnitude has positive real part, the problem is locally unstable and is classified as nonstiff because the stepsize is not being restricted on grounds of stability. The matter is less clear when the dominant eigenvalue has negative real part, but a subdominant eigenvalue has a positive real part. This case has been ignored, perhaps in part because many authors do not compute a subdominant eigenvalue. We do try to compute such an eigenvalue, but we do not trust its computation unless it is comparable in size to the dominant eigenvalue. In our code we diagnose the problem as not stiff if a subdominant eigenvalue has a positive real part and the magnitude of this eigenvalue is at least 90 percent of the magnitude of the dominant eigenvalue. It is possible that both the eigenvalues computed have negative real parts, but that there is a subdominant eigenvalue with a positive real part of sufficient magnitude that it prevents the problem from being stiff. In the next test on the dominant eigenvalue, this local instability is recognized by its effect on the stepsize.

When the dominant eigenvalue has negative real part, we compare it to the boundary of the stability region. If the problem is nonstiff or if it is locally unstable, the dominant eigenvalue will lie well within the stability region. (Recall that we have already tried two inexpensive ways of recognizing these possibilities, namely by comparing the current stepsize to the average one and predicting the cost of completing the integration, before going to the expense of computing eigenvalues.) A difficulty with this obvious tactic is that the shape of a stability region is not easy to quantify. We have chosen to provide our code with the radius of a circle that approximates the boundary of the stability region in a sector of the left half complex plane. It is, unfortunately, necessary to restrict our attention to a sector because the stability regions for important formulas change sharply near the imaginary axis. Obviously some judgment must be exercised in the selection of these parameters. We want to approximate the boundary by a segment of a circle in an appropriate sector, but this conflicts with our wish to make the sector as big as possible. For the (7, 8) pair [10] in our code we take the radius to be 5.2 and we restrict our attention to the sector of the left half complex plane with points z such that $|\mathrm{Im}\,(z)/\mathrm{Re}\,(z)| \leqq 11$. The corresponding quantities are 3.3 and 5.3 for the (4, 5) pair [17]. We are not scaling the stability regions for equal work, so the (4, 5) pair is actually the more efficient when the stepsize is restricted on grounds of stability. Unfortunately, its behavior near the imaginary axis is rather less satisfactory, as indicated by the sector specified. If the dominant eigenvalue lies in the sector and its magnitude is less than 90 percent of this radius, we say that it is well within the stability region and that the stepsize is not being restricted on grounds of stability.

An obviously unsatisfactory aspect of this way of proceeding is that we are "blind" to dominant eigenvalues that are outside the sector, i.e., close to the imaginary axis. In part this is intrinsic; it is far from clear how to classify problems with eigenvalues that are close to the imaginary axis. In part it reflects the fact that the formula pair is not all that we might hope for. It does not seem to be generally appreciated that the rather sharp reduction of stability seen near the imaginary axis with many popular formulas means that a relatively small change in the problem can lead to a quite different behavior of the integration with a corresponding difference in how stiff the problem is. For one reason or another, there are problems that our approach does not classify, and our code returns with a statement that it is "unsure."

In [15] Shampine has observed that a standard test problem, D5 in the nonstiff test set [8], has a Jacobian that at the initial point of the integration has eigenvalues $\pm\sqrt{2{,}000}$ and $\pm i\sqrt{1{,}000}$. Because the interval of integration is [0, 20], these eigenvalues are "large." During the present investigation a number of anomalous experimental results forced the author to ask how general this situation is. The conclusion is a sobering one. Systems of equations of the form $\ddot{x} = f(t, x)$ are of great practical importance, and, correspondingly, are well represented in standard test sets. Although there are special codes for these special problems, general purpose codes require them to be converted to first-order systems. It is usual to do this by introducing the velocity vector $v = \dot{x}$ as an additional set of dependent variables. The Jacobian of this new system is

$$M = \begin{bmatrix} 0 & I \\ f_y & 0 \end{bmatrix}.$$

It is easy to see that if λ is an eigenvalue of f_y, both the square roots of λ are eigenvalues of M. Thus if f_y has a dominant eigenvalue λ that is real and positive, M has the dominant eigenvalues $\pm\sqrt{\lambda}$. If f_y has a dominant eigenvalue λ that is real and negative, M has the dominant eigenvalues $\pm i\sqrt{|\lambda|}$. More troublesome computationally is the case that f_y has a dominant eigenvalue λ that is complex because its conjugate is also a dominant eigenvalue of this real matrix, hence M has (at least) four distinct eigenvalues of the same magnitude that dominate. This class of common problems looks quite innocent, but it presents a number of difficulties for the diagnosis of stiffness. Computing just one dominant eigenvalue is not enough: In § 2 we related how we once made a wrong diagnosis with D5 because we computed the negative one of the pair as the dominant one and the average stepsize was very much larger than the current stepsize. Because a pair of pure imaginary eigenvalues is quite possible, it is necessary to consider what action is appropriate in this case. There is no general agreement on the classification of such problems; we have chosen to say that we are "unsure" how to classify it. Many authors have used a power method to calculate a single dominant eigenvalue; this will generally fail to converge for this class of problems because there are at least two eigenvalues of maximum magnitude. It is even possible that computing a pair of dominating eigenvalues with a power method will fail because there are more than two that dominate.

5. Numerical results. The ideas presented here have been implemented and tested in several codes in both single and double precision on machines with both short and long word lengths. However, all the numerical results reported in this section were computed with a suite of Runge–Kutta codes being developed by Shampine, Gladwell, and Brankin that implements both DPS (4, 5), Shampine's modification [17] of a (4, 5) pair of Dormand and Prince, and PD (7, 8), a (7, 8) pair of Prince and Dormand [10].

The computations were done in double precision on an IBM AT, which corresponds to about 16 decimal digits.

We have integrated all the problems in the set of stiff test problems of Enright, Hull, and Lindberg [1]. Although described as a set of stiff problems, the problems range from easy nonstiff problems through mildly stiff to rather stiff problems. A discussion of the matter is to be found in [16], where it is also pointed out that it is important to use an appropriate error control. We have adopted the suggestion made there of taking the error in a solution component relative to the largest magnitude of this component over the range of integration (as determined in a preliminary computation). For these tests we set MAXFCN to 2,000 evaluations. In each case we continued the integration until it was completed successfully or more than MAXFCN evaluations were made.

We solved all the problems with both DPS (4, 5) and PD (7, 8) at the two tolerances 10^{-3} and 10^{-5}. Problem E2 is van der Pol's equation, but the value of the parameter specified results in a very easy problem. At the more stringent tolerance the (4, 5) pair integrates the problem in 109 evaluations of f and the (7, 8) pair integrates it in 129 evaluations. Clearly this is not a stiff problem, and the code has done what we want by simply carrying out the integration. The behavior of the eigenvalues is interesting because they vary considerably along the solution curve. We have done computations in which we forced a check for stiffness at every step. Most of the situations described in §§ 3 and 4 were exercised, and our schemes for computing eigenvalues and diagnosing stiffness performed as designed.

The B family of problems in the test set reveals the behavior of codes when applied to problems with Jacobians having complex eigenvalues. As pointed out in [16], some members of this family are no more stiff than some of the nonstiff test problems of [8]. Problem B2 was integrated at the more stringent tolerance by the (4, 5) pair in 550 evaluations and by the (7, 8) pair in 780 evaluations. The corresponding numbers for the B3 problem are 706 and 798, and for the B4 problem, 1,504 and 1,740. The stepsize *is* being restricted by stability (the lower-order pair is more efficient here because it is more stable), but if the problems can be integrated this inexpensively, this fact is of little interest. If the test on the work required to complete the integration is suppressed, the code does recognize a restriction, at least sometimes. In the case of the (4, 5) pair for B2 and B3 and at both tolerances, a test is made after 50 successful steps which says that the stepsize is being restricted on grounds of stability. The behavior for B4 was much the same except that there were three or four tests made, all of which diagnosed a stability restriction. Frequent failures were much less likely to trigger a check on stiffness with the (7, 8) pair. Indeed, the only such checks were made at tolerances 10^{-3} and 10^{-5} for B3. When checks were made, a stability restriction was recognized (along with the fact that it would be inexpensive to complete the integration).

Another problem that is inexpensive to integrate is E1. At the more stringent tolerance, this problem is integrated in 487 evaluations with the (4, 5) pair and 541 evaluations with the (7, 8) pair. One of the points made in [16] is that in the original tests [1], the error control for this problem is so lax that practically no accuracy is required of the solution. Despite the fact that we do have a proper control and are getting accurate results, the integrations are very inexpensive. This problem is in the test set because the local Jacobian has dominant eigenvalues that are complex with substantial imaginary part. We have solved this problem making checks at every step and verified that we compute these eigenvalues accurately and efficiently. The problem would be stiff if it were posed on a longer interval. On the interval specified, it is at

most mildly stiff, and because the performance of the explicit Runge-Kutta formulas is perfectly satisfactory, there is no reason to inquire about a stability restriction, and the code does not.

Problem B5 was constructed to have eigenvalues close to the imaginary axis. In the present context it is especially interesting because the dominant eigenvalues are "invisible" to the (4, 5) pair and "visible"' to the (7, 8) pair. After MAXFCN function evaluations with the (4, 5) pair, the code checked for a stability restriction. Interestingly, there were no failed steps at either tolerance. The dominant pair of eigenvalues was computed accurately and inexpensively, but because they do not lie in the sector appropriate to DPS (4, 5), the subroutine reported that it was "unsure" about a stability restriction. There were not enough failed steps with the (7, 8) pair to trigger a check: At the cruder tolerance there were two failures, at the other there were none. When MAXFCN function evaluations were done, the code diagnosed a stability restriction, but predicted that the integration could be completed in only 1.6 MAXFCN evaluations at the tolerance 10^{-3} and 1.7 MAXFCN at the tolerance 10^{-5}.

Only one other problem in the set exhibited points of interest. When solving D3 at the tolerance 10^{-3} with the (4, 5) pair, the code had enough failures to trigger a check for stiffness every 40 steps. On three of these occasions it found the stepsize to be well within the stability region, and on the other three it diagnosed a stability restriction and predicted that about 4.9 MAXFCN evaluations would be needed to complete the integration. At the other tolerance and at both tolerances using the other pair, a stability restriction was diagnosed at every check, and it was predicted that about 4.9 MAXFCN evaluations would suffice to complete the integration.

All the other problems in the test set were diagnosed as stiff at both tolerances with both pairs of formulas. In every case stiffness was diagnosed at successful step number 50. For the present tests we continued the integration, and in practically every case stiffness was diagnosed every 40 successful steps thereafter up to termination of the run because of too much work. Evidently monitoring of step failures was very successful at recognizing stiffness early in the integration. It is worth remarking that some of the problems are easy even for a code intended for nonstiff problems. One of the observations in [16] about the tests of [1] is that a better choice of error control is needed for the family of problems C2-C5. With our reasonable choice, stiffness is diagnosed for each member of the C family, but none is predicted to need more than 2 MAXFCN evaluations of f to complete the integration. In such a case it would probably be a waste of time to change to a code intended for stiff problems. In contrast, when stiffness is diagnosed for D6, the cost is predicted to be on the order of 37,000 MAXFCN.

6. Conclusion. The author has developed a number of schemes for diagnosing stiffness. The one presented here seems to be the most reliable and effective. Considerable effort has been devoted to avoiding false returns. When in doubt, the code says that it is "unsure." The scheme is usually very inexpensive, but we have been willing to do as much work as a couple of Runge-Kutta steps if necessary to make the diagnosis credible.

Acknowledgments. The author is indebted to M. Paprzycki of Southern Methodist University, who spent a summer experimenting with schemes for diagnosing stiffness. The results provided insight that led to the developments reported here. R. W. Brankin of Numerical Algorithms Group Ltd. and I. Gladwell of Southern Methodist University helped the author in many ways with this investigation. Comments by anonymous referees and the editor have improved the exposition substantially.

REFERENCES

[1] W. H. ENRIGHT, T. E. HULL, AND B. LINDBERG, *Comparing numerical methods for stiff systems of* ODEs, BIT, 15 (1975), pp. 10-48.

[2] J. N. FRANKLIN, *Matrix Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1968.

[3] I. GLADWELL, *Initial value routines in the* NAG *library*, ACM Trans. Math. Software, 5 (1979), pp. 386-400.

[4] I. GLADWELL, L. F. SHAMPINE, AND R. W. BRANKIN, *Automatic selection of the initial step size for an* ODE *solver*, J. Comput. Appl. Math., 18 (1987), pp. 175-192.

[5] G. HALL, *Equilibrium states of Runge-Kutta schemes*, ACM Trans. Math. Software, 11 (1985), pp. 289-301.

[6] ———, *Equilibrium states of Runge-Kutta schemes*—II, ACM Trans. Math. Software, 12 (1986), pp. 183-192.

[7] D. J. HIGHAM AND G. HALL, *Embedded Runge-Kutta formulas with stable equilibrium states*, Report 140, Department of Mathematics, University of Manchester, Manchester, U.K., 1987.

[8] T. E. HULL, W. H. ENRIGHT, B. M. FELLEN, AND A. E. SEDGWICK, *Comparing numerical methods for ordinary differential equations*, SIAM J. Numer. Anal., 9 (1972), pp. 603-637.

[9] B. LINDBERG, IMPEX 2—*A procedure for solution of stiff differential equations*, Report TRITA-NA-7303, Department of Information Processing, Royal Institute of Technology, Stockholm, Sweden, 1973.

[10] P. J. PRINCE AND J. R. DORMAND, *High order embedded Runge-Kutta formulae*, J. Comput. Appl. Math., 7 (1981), pp. 67-75.

[11] P. RENTROP, *Partionierte Runge-Kutta Verfahren zur numerischen Lösung von nichtsteifen und steifen Anfangswertenaufgaben*, Report TUM-M8304, Department of Mathematics, Technical University of Munich, Munich, FRG, 1983.

[12] B. C. ROBERTSON, *Detecting stiffness with explicit Runge-Kutta formulas*, Report 193/87, Department of Computer Sciences, University of Toronto, Toronto, Ontario, Canada, 1986.

[13] L. F. SHAMPINE, *Stiffness and nonstiff differential equation solvers*, in Numerische Behandlung von Differentialgleichungen, R. Ansorge, L. Collatz, G. Hämmerlin, and W. Törnig, eds., Birkhauser, Basel, 1975, pp. 287-301.

[14] ———, *Stiffness and nonstiff differential equation solvers*, II: *detecting stiffness with Runge-Kutta methods*, ACM Trans. Math. Software, 3 (1977), pp. 44-53.

[15] ———, *Lipschitz constants and robust* ODE *codes*, in Computational Methods in Nonlinear Mechanics, J. T. Oden, ed., North-Holland, Amsterdam, 1980, pp. 427-449.

[16] ———, *Evaluation of a test set for stiff* ODE *solvers*, ACM Trans. Math. Software, 7 (1981), pp. 409-420.

[17] ———, *Some practical Runge-Kutta formulas*, Math. Comp., 46 (1986), pp. 135-150.

[18] L. F. SHAMPINE AND K. L. HIEBERT, *Detecting stiffness with the Fehlberg* (4, 5) *formulas*, Comput. Math. Appl., 3 (1977), pp. 41-46.

[19] G. P. SOTTAS, *Etudes de méthodes numériques pour la résolution de systemes d'équations différentielles raides*, Ph.D. thesis, Department of Mathematics, University of Geneva, Geneva, Switzerland, 1984.

[20] A. WOLFBRANDT, *Dynamic adaptive selection of integration algorithms when solving* ODE's, BIT, 22 (1982), pp. 361-367.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 273-297, March 1991
© 1991 Society for Industrial and Applied Mathematics
004

COMPUTATIONAL USE OF GROUP THEORY IN BIFURCATION ANALYSIS OF SYMMETRIC STRUCTURES*

KAZUO MUROTA† AND KIYOHIRO IKEDA‡

Abstract. An efficient computational procedure is proposed for identifying singular points in global bifurcation analysis of the static behavior of symmetric discrete structures such as symmetric truss domes. Assuming group equivariance of the system of equations describing the steady state, and making use of group representation theory, the proposed method decomposes the Jacobian matrix (or tangent stiffness matrix) into block-diagonal form, with possibly repeated occurrences of identical blocks, by means of a suitable "local" coordinate transformation. The "local" transformation is computationally favorable in that it requires a small amount of computation and preserves the sparsity of the original Jacobian matrix fairly well. A concrete procedure is described for symmetric truss structures which are equivariant to dihedral groups; an explicit formula is given to the number of diagonal blocks into which the Jacobian matrix splits, and an estimate of the required number of computations shows the efficiency of the proposed method.

Key words. symmetric structure, bifurcation analysis, block diagonalization, computational complexity, group representation theory, parallelism, sparse matrix

AMS(MOS) subject classifications. 73K30, 15A21, 65F50, 65N30

1. Introduction. In the last decade the so-called group-theoretic approach (Golubitsky, Stewart, and Schaeffer [7], Sattinger [17]) has turned out to be fruitful in the local and global bifurcation analysis. Group-theoretic viewpoints are also useful and essential in designing successful numerical discretization schemes (Fujii-Yamaguti [6]), and in enhancing the computational efficiency in global numerical analysis (Healey [9], [10]).

In this paper we consider a static (or stationary, steady) bifurcation problem expressed by a system of equations

$$f(v, \lambda) = 0$$

in a finite-dimensional vector v with a bifurcation parameter λ. We are concerned with the bifurcation diagram, i.e., the loci of (v, λ) satisfying the equations in a region of our interest.

As a typical problem that we have in mind, we mention the problem of determining the deformation of the spherical diamond shell of Fig. 1 under a uniform vertical loading, where v is a 93 $(=3\times 31)$ dimensional vector representing displacements of 31 free nodes, and λ is a loading parameter.

In applications we often encounter bifurcation problems that have group symmetry (in the sense to be made precise in § 3). For instance, the system of equations describing the spherical diamond shell has a symmetry expressed by a dihedral group if all the physical properties are symmetric. Also finite-dimensional approximations to group-symmetric bifurcation problems described by differential equations often fall into the category we shall consider.

* Received by the editors April 26, 1989; accepted for publication (in revised form) December 5, 1989.

† Department of Mathematical Engineering and Information Physics, University of Tokyo, Bunkyo-ku, Tokyo 113, Japan. This work was done during this author's stay at the Institute for Econometrics and Operations Research, University of Bonn, Bonn, Federal Republic of Germany, and was supported by the Alexander von Humboldt Foundation.

‡ Department of Civil Engineering, Nagaoka University of Technology, Nagaoka, Niigata 940-21, Japan.

A typical numerical procedure for obtaining the bifurcation diagram would involve the following two phases:

(i) Extending and tracing a path (or locus), and

(ii) Identifying bifurcation points as well as the directions of new branches.

The first phase (i) amounts to solving a system of nonlinear equations by, e.g., a predictor-corrector (Newton–Raphson) type method, whereas the second amounts to determining the rank of the Jacobian matrix J of f with respect to v, as well as its null space.

When the bifurcation problem has group symmetry we can make use of this fact in various ways to enhance the computational efficiency of the numerical analysis. In particular, a recent paper [10] by Healey illustrates to the full extent how we can reduce the computational effort in the path-tracing phase (i) by exploiting symmetry (see § 3 for detail).

The main concern of this paper is to exploit symmetry in the second phase (ii) of singularity detection, in which we are to see if the Jacobian matrix is singular at particular points. The basic idea of the proposed method is to find a suitable (unitary) matrix H, which is independent of particular points, such that the transformed Jacobian matrix H^*JH takes a block-diagonal form.

The advantage of this block diagonalization is twofold. Obviously, the test for singularity of J is decomposed into a series of tests for singularity of diagonal blocks, which are much smaller in size and hence much easier to handle. Moreover, the diagonal blocks of H^*JH are not all distinct but contain several pairs (or tuples, depending on the symmetry) of identical matrices. Thus we have only to deal with distinct representatives among the diagonal blocks, and we can check for their singularities independently of one another, or in parallel if such a machinery is available.

The proposed method is based on the well-established mathematical facts about the (nonunique) decomposition of a group representation into a direct sum of irreducible representations (see § 2). Such a transformation matrix H, as well as the block-diagonal form H^*JH, is not uniquely determined from the group representation theoretic point of view, which fact allows us to select a "good" one from the computational point of view.

For a symmetric structure like the one in Fig. 1, the proposed choice of H is induced naturally from the underlying geometric symmetry, and is a block-diagonal matrix composed of many small diagonal blocks. In other words, the block-diagonal form H^*JH is obtained through a series of "local" coordinate transformations. This "locality" is computationally advantageous in that it requires a small amount of computation and preserves, to some extent, the sparsity of the original Jacobian matrix J.

For the example problem of spherical diamond shell (Fig. 1, § 3.3), the proposed method puts the Jacobian matrix J of size 93 into a block-diagonal matrix H^*JH with eight diagonal blocks, among which are six distinct blocks of sizes 6, 7, 8, 10, 15, and 16 (see § 4 for detailed description).

This paper is organized as follows. Preliminaries from group representation theory are given in § 2 with particular stress on dihedral groups. Sections 3 and 4 consider the block-diagonalization of the Jacobian matrix under a group symmetry from a computational point of view. The former affords general framework for bifurcation problems with symmetry expressed by a finite group G, and the latter specializes it to symmetric truss structures, the symmetry of which is represented by dihedral groups. For symmetric truss structures, we give in § 4 an explicit formula for the transformation matrix H and for the number and the sizes of the obtained diagonal blocks. We also estimate the computational complexity of the proposed procedure to show its

efficiency. Finally, in § 5 we discuss some questions on how the transformation matrix H should be changed at a symmetry-breaking bifurcation point.

2. Preliminaries on group representation. In this section we introduce the notation as well as some fundamental facts about linear representations of finite groups to be used in this paper. For complete accounts, the reader is referred to works such as Curtis and Reiner [4], Hammermesh [8], Jacobson [14], and Miller [16].

Let G be a finite group, whose order is denoted by $|G|$. Let V be a finite-dimensional vector space over $F = \mathbf{R}$ or $\mathbf{C}$, where $\mathbf{R}$ is the field of real numbers and $\mathbf{C}$ the field of complex numbers. Denote by $GL(V)$ the group of all nonsingular linear transformations of V onto itself, and by $GL(N, F)$ the group of all nonsingular matrices over F of order N.

By a representation of G on V is meant a homomorphism $\hat{T}: G \to GL(V)$, i.e.,

$$\hat{T}(gh) = \hat{T}(g)\hat{T}(h), \qquad g, h \in G.$$

We call V the representation space and $N = \dim V$ the dimension or the degree of the representation. Fixing a basis $B = (u_1, \cdots, u_N)$ for V, we can associate with $\hat{T}(g)$ an $N \times N$ nonsingular matrix $T(g) = T_B(g) = (T_{ij}(g))_{1 \leqq i,j \leqq N}$ in the usual manner:

$$\hat{T}(g)u_j = \sum_{i=1}^{N} T_{ij}(g)u_i, \qquad g \in G.$$

The homomorphism $T: G \to GL(N, F)$ is called a matrix representation of G. A change of basis B to $C = (v_1, \cdots, v_N)$ where $v_j = \sum_{i=1}^{N} H_{ij}u_i$ with $H = (H_{ij}) \in GL(N, F)$ replaces the matrix T_B by another matrix $T_C = H^{-1}T_BH$.

We say that $\hat{T}$ (respectively, T) is unitary if $\hat{T}(g)$ (respectively, $T(g)$) is unitary for each $g \in G$.

The character $\chi: G \to F$ of $\hat{T}$ is defined by

$$\chi(g) = \operatorname{Tr} \hat{T}(g) = \operatorname{Tr} T(g) = \sum_{i=1}^{N} T_{ii}(g), \qquad g \in G.$$

Note that χ does not depend on the choice of a basis.

Since we are interested in computational efficiency in this paper, we are concerned not only with representations in the abstract sense but also with their concrete matrix representations with respect to particular choices of bases. In fact, it is one of the central issues of this paper to choose "good" bases that yield matrix representations which are convenient to numerical computations.

Let P be a finite set, and assume that G acts on P through permutations $\pi(g)$, $g \in G$, of P. The permutation representation $\hat{T}$ is a representation of dimension $|P|$ defined as follows. Consider the vector space $V = F^P$, i.e., the linear space consisting of "formal" sums $\sum_{p \in P} f_p u_p (f_p \in F)$ with basis $(u_p \mid p \in P)$, and define $\hat{T}: G \to GL(V)$ by

$$\hat{T}(g)u_p = u_{\pi(g)p}.$$

A permutation representation is called the regular representation when $P = G$, and the unit representation when $|P| = 1$.

Let $\hat{T}_1$ and $\hat{T}_2$ be representations of G with respective representation spaces V_1 and V_2 over F. We say that $\hat{T}_1$ and $\hat{T}_2$ are equivalent if there exists a nonsingular linear map $\hat{H}: V_1 \to V_2$ such that

$$\hat{T}_1(g) = \hat{H}^{-1}\hat{T}_2(g)\hat{H}, \qquad g \in G.$$

We also say that two matrix representations T_1 and T_2 of G are equivalent if there exists a nonsingular matrix H such that

$$T_1(g) = H^{-1}T_2(g)H, \qquad g \in G.$$

The direct sum $\hat{T}_1 \oplus \hat{T}_2$ of $\hat{T}_1$ and $\hat{T}_2$ is a representation on the direct sum $V_1 \oplus_F V_2$ of the representation spaces V_1 and V_2, and is defined by

$$(\hat{T}_1 \oplus \hat{T}_2)(g) = \hat{T}_1(g) \oplus \hat{T}_2(g), \qquad g \in G.$$

The tensor product $\hat{T}_1 \otimes \hat{T}_2$ of $\hat{T}_1$ and $\hat{T}_2$ is a representation on the tensor product $V_1 \otimes_F V_2$ of V_1 and V_2, and is defined by

$$(\hat{T}_1 \otimes \hat{T}_2)(g) = \hat{T}_1(g) \otimes \hat{T}_2(g), \qquad g \in G.$$

A subspace W of V is said to be ($\hat{T}(G)$- or G-) invariant if $\hat{T}(g)w \in W$ for $w \in W$, $g \in G$. If W is invariant, the subrepresentation of $\hat{T}$ on W, denoted as $T|W$, is defined by the restriction of $\hat{T}(g)$ to W for each $g \in G$. $\hat{T}$ is said to be irreducible if there exists no nontrivial invariant subspace W, where W is nontrivial if W is neither $\{0\}$ nor V. $\hat{T}$ is absolutely irreducible if it is irreducible as a representation over $\mathbf{C}$.

There exist a finite number of nonequivalent irreducible representations of G (over a fixed field F). We denote by

$$\{\hat{T}^{(\mu)} | \mu \in R(G)\}$$

a family of all nonequivalent irreducible representations of G, where $R(G)$ denotes an index set for the irreducible representations of G, and put

$$N^{(\mu)} = \dim \hat{T}^{(\mu)}, \qquad \chi^{(\mu)} = \operatorname{Tr} \hat{T}^{(\mu)}.$$

The unit representation is indicated by $\mu = \mu_0$, i.e., $\chi^{(\mu_0)}(g) = 1$, $g \in G$. We also fix a family

$$\{T^{(\mu)} | \mu \in R(G)\}$$

of nonequivalent irreducible matrix representations of G.

It is known as Maschke's theorem that any representation $\hat{T}$ of G on V can be expressed as a direct sum of irreducible representations. To be more precise, V is decomposed as

$$V = \bigoplus_{\mu \in R(G)} V^{\mu} \tag{1}$$

with

$$V^{\mu} = \bigoplus_{i=1}^{a^{\mu}} V_i^{\mu}, \tag{2}$$

where V_i^{μ} is invariant, and $\hat{T}|V_i^{\mu}$ is irreducible and equivalent to $\hat{T}^{(\mu)}$. Substituting (2) into (1), we obtain

$$V = \bigoplus_{\mu \in R(G)} \bigoplus_{i=1}^{a^{\mu}} V_i^{\mu}, \tag{3}$$

a direct sum decomposition into irreducible components.

The decomposition (1) is unique and called the isotypic (or standard) decomposition, each V^{μ} being called an isotypic (or homogeneous) component. On the other hand, the decomposition (2) is not unique though the multiplicity a^{μ} ($\geqq 0$) is uniquely determined. Hence, the decomposition (3) into irreducible components is not unique either.

The direct sum decomposition (3) means that the matrix representation is put into a block-diagonal form

$$T(g) = \bigoplus_{\mu \in R(G)} \bigoplus_{i=1}^{a^\mu} T_i^\mu(g), \qquad g \in G \tag{4}$$

with T_i^μ being irreducible, if we choose a basis of V consistent with the decomposition (3), i.e., if we first choose a basis $(h_{ij}^\mu | j = 1, \cdots, N^{(\mu)})$ for each V_i^μ and adopt their union as a basis of V. In this paper, we take advantage of the fact that we can impose a further condition

$$T_i^\mu(g) = T^{(\mu)}(g), \qquad g \in G, \qquad 1 \leqq i \leqq a^\mu. \tag{5}$$

In the case of $F = \mathbf{C}$, the multiplicity a^μ is given by

$$a^\mu = \frac{1}{|G|} \sum_{g \in G} \chi(g) \chi^{(\mu)}(g^{-1}). \tag{6}$$

LEMMA (Schur). *Let T_1 and T_2 be irreducible matrix representations of G over F, and J be a matrix over F. Assume that*

$$T_1(g)J = JT_2(g), \qquad g \in G.$$

(i) *If T_1 and T_2 are not equivalent, then $J = O$.*

(ii) *If T_1 and T_2 are equivalent, then $J = O$ or J is nonsingular.*

(iii) *If $T_1(g) = T_2(g)$, $g \in G$, and T_1 is absolutely irreducible, then $J = \alpha I$ for some $\alpha \in F$.* □

The dihedral groups and their representations play substantial roles in this paper. The dihedral group D_n of order $2n$ is defined as

$$D_n = \{1, r, \cdots, r^{n-1}; s, sr, \cdots, sr^{n-1}\},$$

where

$$r^n = s^2 = (sr)^2 = 1.$$

A matrix representation T of D_n is determined by $T(r)$ and $T(s)$.

We index the family of nonequivalent irreducible representations of D_n by

$$R(D_n) = \{(1, j) | j = 1, 2, 3, 4\} \cup \{(2, j) | j = 1, \cdots, (n-2)/2\} \quad \text{for } n \text{ even},$$

$$R(D_n) = \{(1, j) | j = 1, 2\} \cup \{(2, j) | j = 1, \cdots, (n-1)/2\} \quad \text{for } n \text{ odd},$$

where the first component d of the index $\mu = (d, j)$ indicates the dimension of the representation.

The one-dimensional irreducible representations $T^{(1,j)}$ of D_n are uniquely determined and given by

$$T^{(1,1)}(r) = 1, \qquad T^{(1,1)}(s) = 1,$$

$$T^{(1,2)}(r) = 1, \qquad T^{(1,2)}(s) = -1,$$

$$T^{(1,3)}(r) = -1, \qquad T^{(1,3)}(s) = 1,$$

$$T^{(1,4)}(r) = -1, \qquad T^{(1,4)}(s) = -1.$$

The two-dimensional irreducible matrix representations $T^{(2,j)}$ are not unique but the following choice seems to be convenient, as will be explained in §§ 4 and 5:

$$T^{(2,j)}(r) = R^j, \qquad T^{(2,j)}(s) = S,$$

where

$$R = \begin{pmatrix} \cos(2\pi/n) & -\sin(2\pi/n) \\ \sin(2\pi/n) & \cos(2\pi/n) \end{pmatrix}, \qquad S = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{7}$$

The corresponding characters are given by

$$\chi^{(2,j)}(r) = 2\cos(2\pi j/n), \qquad \chi^{(2,j)}(s) = 0.$$

These irreducible representations over $\mathbf{R}$ are absolutely irreducible.

3. Exploiting general group symmetry. In this section we propose a computational method for the numerical bifurcation analysis of structures with group symmetry in general. The method will be applied to structures with D_n-symmetry in the next section.

3.1. Group equivariance. We consider a bifurcation problem described by a system of equations

$$f_i(v, \lambda) = 0, \qquad i = 1, \cdots, N, \tag{8}$$

in a finite-dimensional real vector $v = (v_j \mid j = 1, \cdots, N) \in \mathbf{R}^N = V$ with a bifurcation parameter $\lambda \in \mathbf{R}$. Suppose further that this problem is equivariant to a finite group G through a unitary matrix representation T of G on V:

$$T(g)f(v, \lambda) = f(T(g)v, \lambda), \qquad g \in G,$$

where $f = (f_i \mid i = 1, \cdots, N)$. Then the Jacobian matrix $J = (J_{ij})$, $J_{ij} = \partial f_i / \partial v_j$, commutes with the group action:

$$T(g)J(v, \lambda) = J(v, \lambda)T(g), \qquad g \in G, \tag{9}$$

if v is G-symmetric, i.e., if $T(g)v = v$, $g \in G$.

For $v \in V$, the isotropy subgroup of v is defined [7] as

$$\Sigma(v) = \{g \in G \mid T(g)v = v\},$$

whereas for a subgroup $G' \subseteq G$, the G'-symmetric subspace is defined [10] as

$$\mathcal{S}(G') = \{v \in V \mid T(g)v = v, g \in G'\}.$$

In particular, G-symmetric subspace coincides with the isotypic component in (1) corresponding to the unit representation μ_0:

$$\mathcal{S}(G) = V^{\mu_0}. \tag{10}$$

(See § 5 for an extension of this relation.)

A typical numerical procedure for finding the bifurcation diagram would involve the following two phases:

(i) Extending and tracing a path (or locus), and

(ii) Identifying bifurcation points as well as the directions of new branches.

The first phase amounts to solving a system of nonlinear equations by, e.g., a predictor-corrector type method, whereas the second amounts to determining the rank of J as well as its null space.

During phase (i), the solutions v to (8) do not change their symmetry $G' = \Sigma(v)$ ($\subseteq G$) and $f(v, \lambda) \in \mathcal{S}(G')$. As observed by several authors (e.g., [6], [10]), this fact enables us to reduce the problem (8) to that on the subspace $\mathcal{S}(G')$ having a smaller number of unknown variables. In particular, the main path (with the full symmetry of G) can be determined from the reduction to the G-symmetric subspace. Furthermore, Healey [10] suggests a method to enhance the computational efficiency in phase (ii) based on the fact that the reduced Jacobian matrix remains nonsingular at a symmetry-breaking bifurcation point. Since $\mathcal{S}(G) = V^{\mu_0}$, as noted above, we may say that this technique is based on the direct sum decomposition of V into two parts:

$$V = V^{\mu_0} \oplus \Big(\bigoplus_{\mu \neq \mu_0} V^{\mu} \Big),$$

which is an aggregation of the isotypic decomposition (1).

We shall explain in the following section how we can make use of the nonunique decomposition (3), which is a refinement of the isotypic decomposition, for efficient computation in the phase (ii) of singularity detection.

3.2. Block diagonalization. Since T is assumed to be unitary, we can find (cf. § 2) a unitary (orthogonal) matrix H such that

$$H^* T(g) H = \bigoplus_{\mu \in R(G)} \bigoplus_{i=1}^{a^\mu} T_i^\mu(g) = \bigoplus_{\mu \in R(G)} \bigoplus_{i=1}^{a^\mu} T^{(\mu)}(g), \qquad g \in G, \tag{11}$$

where (5) is used. Note that $T^{(\mu)}(g)$ is a square matrix of order $N^{(\mu)}$. If we accordingly partition the column set of H as

$$H = (H_i^\mu \mid \mu \in R(G), i = 1, \cdots, a^\mu) \tag{12}$$

with H_i^μ being an $N \times N^{(\mu)}$ matrix, we have

$$T(g) H_i^\mu = H_i^\mu T^{(\mu)}(g), \tag{13}$$

which shows how to determine H (see § 4).

A block-diagonal matrix $\tilde{J}$ is obtained by transforming J with H:

$$\tilde{J} = H^* J H.$$

If we define

$$\tilde{J}_{ij}^{\mu\nu} = (H_i^\mu)^* J H_j^\nu,$$

$\tilde{J}$ is partitioned consistently with (11):

$$\tilde{J} = (\tilde{J}_{ij}^{\mu\nu} \mid \mu, \nu \in R(G); i = 1, \cdots, a^\mu; j = 1, \cdots, a^\nu).$$

Then Schur's lemma (i) implies that

$$\tilde{J}_{ij}^{\mu\nu} = O \quad \text{if } \mu \neq \nu, \tag{14}$$

since $(H^* T(g) H)\tilde{J} = \tilde{J}(H^* T(g) H)$ by (9). It follows that

$$\tilde{J} = \bigoplus_{\mu \in R(G)} \tilde{J}^\mu, \tag{15}$$

where

$$\tilde{J}^\mu = (\tilde{J}_{ij}^{\mu\mu} \mid i, j = 1, \cdots, a^\mu), \qquad \mu \in R(G),$$

is a square matrix of order $a^\mu N^{(\mu)}$.

Each diagonal block $\tilde{J}^{\mu}$ can have a further structure by virtue of our choice (5) and Schur's lemma (iii). That is, if $T^{(\mu)}$ is absolutely irreducible, then

$$\tilde{J}_{ij}^{\mu\mu} = \alpha_{ij}^{\mu} I_{N^{(\mu)}}, \qquad i,j = 1, \cdots, a^{\mu},$$

for some $\alpha_{ij}^{\mu} \in F$. If we define

$$A^{\mu} = (\alpha_{ij}^{\mu} \mid i,j = 1, \cdots, a^{\mu}), \tag{16}$$

this means that $\tilde{J}^{\mu}$ is in a block-diagonal form (after suitable arrangement of rows and columns) which has A^{μ} as its $N^{(\mu)}$ identical diagonal blocks of size a^{μ}:

$$\tilde{J}^{\mu} = \bigoplus_{l=1}^{N^{(\mu)}} A^{\mu}.$$

To sum up, we obtain

$$\tilde{J} = H^{*}JH = \left(\bigoplus_{\mu \in R_a(G)} \bigoplus_{l=1}^{N^{(\mu)}} A^{\mu} \right) \oplus \left(\bigoplus_{\mu \in R(G) - R_a(G)} \tilde{J}^{\mu} \right), \tag{17}$$

where $R_a(G)$ denotes the subsets of $R(G)$ consisting of absolutely irreducible representations of G. This type of block diagonalization, as well as the one of (15) corresponding to the isotypic decomposition (1), is well known in many application areas. In bifurcation theory, as shown by Fujii and Yamaguti [6], (15) plays an essential role in the analysis of bifurcation at simple points. This principle is also used in quantum mechanics [8], [16] in deriving "selection rules."

Obviously, J is singular if and only if A^{μ} or $\tilde{J}^{\mu}$ is singular for some $\mu \in R(G)$. Hence, instead of testing for the singularity of J, of size N, we may test for the singularity of $|R(G)|$ smaller matrices: A^{μ}, of size a^{μ}, for $\mu \in R_a(G)$ and $\tilde{J}^{\mu}$, of size $a^{\mu}N^{(\mu)}$, for $\mu \in R(G) - R_a(G)$. This procedure involves an extra work of computing the matrices A^{μ} and $\tilde{J}^{\mu}$; the amount of computation for it depends on the choice of H. The following section shows a natural choice of H, which results in an overall improvement in efficiency.

3.3. "Good" basis. This section suggests a choice of the matrix H in (11) and (17) for a class of bifurcation problems including those arising from truss structures.

Let us recall the spherical diamond shell of Fig. 1, and suppose the steady state of this structure is described in terms of displacements of the nodes. We denote by P the set of the free nodes, $|P| = 31$. As a consequence of the geometric symmetry of this structure, the bifurcation problem is equivariant to the dihedral group D_6, if the physical characteristics and the loading pattern also have D_6 symmetry; $r \in D_6$ corresponds to the anticlockwise rotation around Z-axis at angle $\pi/3$ and $s \in D_6$ to the reflection with respect to the XZ-plane. The space V of the displacements can be regarded as the tensor product of $\mathbf{R}^P$ and the three-dimensional Euclidean space $\mathbf{R}^3$, and the action of D_6 on V conforms with this tensor product. That is, the representation T of D_6 on V is the tensor product of a permutation representation T_P on P (or $\mathbf{R}^P$) and a representation T_E on $\mathbf{R}^3$.

On the basis of this observation, we shall discuss in a rather general framework in this section. The following section will give a more concrete and detailed description for truss structures with D_n-symmetry; in particular, Example 4.1 will serve to illustrate the following arguments.

Now we shall assume for (8) in general that

$$V = \mathbf{R}^P \otimes \mathbf{R}^E \tag{18}$$

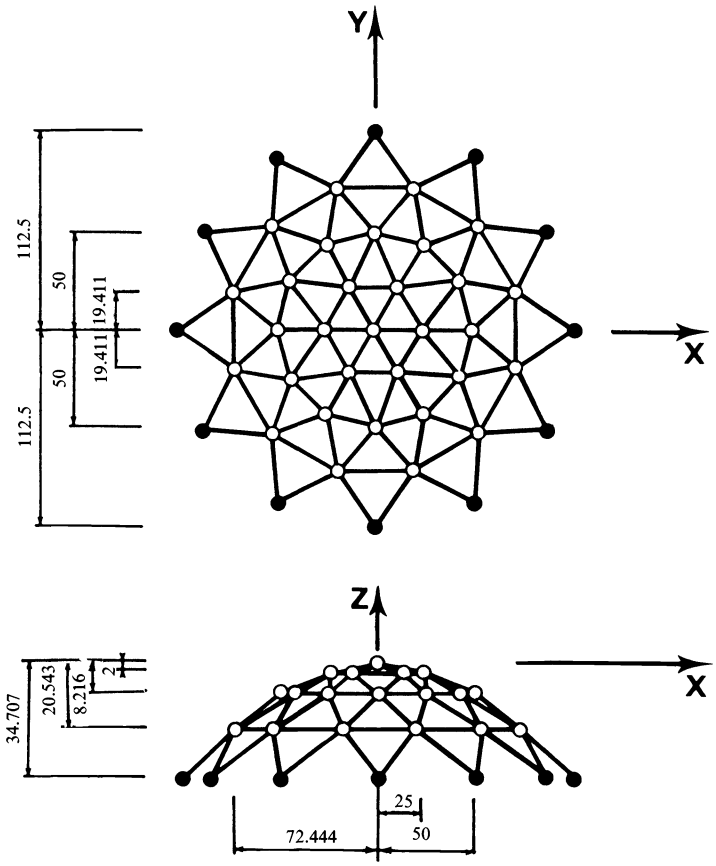


FIG. 1. *Spherical diamond shell.*

for some finite sets P and E, and that

$$T = T_P \otimes T_E,$$

where T_P is a permutation representation of G on P and T_E a representation of G on $\mathbf{R}^E$. Note that the matrix T is of size $N = |P| \times |E|$ and each row (or column) set is indexed by $P \times E$.

Let

$$P = \bigcup_{k=1}^{K} P_k \tag{19}$$

be the decomposition of P into disjoint orbits with respect to the action of G. Obviously, $\mathbf{R}^{P_k}$ is a G-invariant subspace for each k. Also let

$$E = \bigcup_{l=1}^{L} E_l$$

be a decomposition of E into disjoint subsets such that $\mathbf{R}^{E_l}$ is a G-invariant subspace for each l. Then $P \times E$ is partitioned into disjoint sets as

$$P \times E = \bigcup_{k=1}^{K} \bigcup_{l=1}^{L} P_k \times E_l = \bigcup_{\kappa} (P \times E)_\kappa. \tag{20}$$

Here and henceforth, we put

$$\kappa = (k, l), \qquad (P \times E)_\kappa = P_k \times E_l,$$

and the summation over κ will mean that for $k = 1, \cdots, K$ and $l = 1, \cdots, L$. Put

$$N_\kappa = |(P \times E)_\kappa| = |P_k| \times |E_l|.$$

Since $V_\kappa = \mathbf{R}^{P_k} \otimes \mathbf{R}^{E_l}$ is G-invariant, we have a block-diagonal decomposition of T:

$$T(g) = \bigoplus_\kappa T_\kappa(g), \qquad g \in G.$$

Note that this decomposition is of combinatorial nature, in that it is obtained only through partition of the set of coordinates without involving rotation of coordinate axes.

Each diagonal block T_κ of T is to be decomposed further into irreducible components through a suitable "local" change of the basis of V_κ. If we denote by H_κ the unitary matrices of size N_κ representing such "local" transformations, we have (cf. (11), (12), (13))

$$(H_\kappa)^* T_\kappa(g) H_\kappa = \bigoplus_{\mu \in R(G)} \bigoplus_{i=1}^{a_\kappa^\mu} T^{(\mu)}(g), \qquad g \in G.$$

The whole transformation matrix H is a block-diagonal matrix:

$$H = \bigoplus_\kappa H_\kappa \tag{21}$$

and

$$H^* T(g) H = \bigoplus_\kappa \bigoplus_{\mu \in R(G)} \bigoplus_{i=1}^{a_\kappa^\mu} T^{(\mu)}(g) = \bigoplus_{\mu \in R(G)} \bigoplus_{i=1}^{a^\mu} T^{(\mu)}(g), \qquad g \in G,$$

where

$$a^\mu = \sum_\kappa a_\kappa^\mu.$$

Then, as explained in the previous section, $\tilde{J} = H^* J H$ is in a block-diagonal form (17) with $|R(G)|$ distinct blocks; A^μ is of size a^μ, and $\tilde{J}^\mu$ is of size $a^\mu N^{(\mu)}$.

It should be clear that the partition (20) above does not put $\tilde{J}$ into a block-diagonal form. That is, the submatrix $\tilde{J}_{\kappa\lambda}$ of $\tilde{J}$ with row-set $(P \times E)_\kappa$ and column-set $(P \times E)_\lambda$ does not vanish in general even if $\kappa \neq \lambda$, since T_κ and T_λ may have some irreducible components in common.

Nevertheless, $\tilde{J}_{\kappa\lambda}$ is likely to vanish for many pairs of (κ, λ) in practical applications. This is not because of the group symmetry but because of the sparsity of J and the locality of our transformation H. In fact, if $J_{\kappa\lambda} = O$ (where $J_{\kappa\lambda}$ is the submatrix of J with the same row/column subsets as $\tilde{J}_{\kappa\lambda}$), then $\tilde{J}_{\kappa\lambda} = (H_\kappa)^* J_{\kappa\lambda} H_\lambda = O$. This means that each A^μ or $\tilde{J}^\mu$ in (17) inherits the sparsity of J in the sense that its (κ, λ) block vanishes if $J_{\kappa\lambda} = O$.

3.4. Computational cost. We conclude this section with a rough estimate of the computational efficiency of the above procedure when each irreducible representation of G is absolutely irreducible.

The computational cost for obtaining the nonzero elements of $\tilde{J}$ from J is estimated as follows. For each (κ, λ), the distinct nonzero elements of $\tilde{J}_{\kappa\lambda} = (H_\kappa)^* J_{\kappa\lambda} H_\lambda$ can be computed with

$$\sum_{\mu \in R(G)} (a_\kappa^\mu N_\kappa N_\lambda + a_\kappa^\mu N_\lambda a_\lambda^\mu)$$

multiplications and additions. Hence all the elements of $\tilde{J}$ can be computed with

$$\sum_\kappa \sum_\lambda \sum_{\mu \in R(G)} (a_\kappa^\mu N_\kappa N_\lambda + a_\kappa^\mu N_\lambda a_\lambda^\mu) = \sum_{\mu \in R(G)} \left((N + a^\mu) \left(\sum_\kappa a_\kappa^\mu N_\kappa \right) \right) \tag{22}$$

operations, where

$$\sum_\kappa N_\kappa = N, \qquad \sum_\kappa a_\kappa^\mu = a^\mu$$

are used.

Assuming the Gaussian elimination for testing for singularity of a matrix, the overall computational cost of the proposed procedure is estimated by

$$\sum_{\mu \in R(G)} \left((N + a^\mu) \left(\sum_\kappa a_\kappa^\mu N_\kappa \right) + \frac{1}{3} (a^\mu)^3 \right). \tag{23}$$

In the case of D_n-symmetric structures, we can derive an explicit form for this expression (see § 4). Note also that if J is known to be a symmetric matrix, then the above estimate reduces to its half.

Note that the cost (22) for computing $\tilde{J}$ is of the order of N^2 (if $|G|$ and $|E|$ are of constant order). To see this, first observe

$$\sum_\mu a_\kappa^\mu \leqq \sum_\mu a_\kappa^\mu N^{(\mu)} = N_\kappa = |P_k||E_l| \leqq |G||E|, \qquad \kappa = (k, l).$$

It then follows that

$$\begin{aligned} \sum_\mu (N + a^\mu) \left(\sum_\kappa a_\kappa^\mu N_\kappa \right) &\leqq 2N \sum_\mu \sum_\kappa a_\kappa^\mu N_\kappa = 2N \sum_\kappa \left(\sum_\mu a_\kappa^\mu \right) N_\kappa \\ &\leqq 2N|G||E| \sum_\kappa N_\kappa = 2|G||E|N^2. \end{aligned}$$

It is to be emphasized that this bound is due to our choice of "local" transformation H; if H were a dense matrix, the computation of $\tilde{J}$ alone would involve $O(N^3)$ operations.

In the above argument we have assumed that all the entries of J are computed first. In many applications (e.g., truss structures, finite-element method, or other discretization schemes), however, the matrix J is often sparse, being an assembly of $O(N)$ elementary matrices. Then the computation of $\tilde{J}$ can be done more efficiently, compatibly with such assembly process, as follows.

Let J_e be the elementary matrix, say, of size N_e, corresponding to element e; we denote by $\hat{J}_e$ the $N \times N$ matrix obtained by augmenting J_e with zeros. We assume that J is expressed as

$$J = \sum_{e=1}^{M} \hat{J}_e$$

with M elementary matrices. Referring to H_κ in (21), we denote by $\hat{H}_\kappa$ the $N \times N$ matrix obtained by augmenting H_κ with zeros; then (21) is written as $H = \sum_\kappa \hat{H}_\kappa$. We partition the column set of H_κ as

$$H_\kappa = (H_\kappa^\mu \,|\, \mu \in R(G)),$$

where H_κ^μ denotes the $N \times a_\kappa^\mu N^{(\mu)}$ submatrix corresponding to the representation μ. Accordingly we express $\hat{H}_\kappa$ as

$$\hat{H}_\kappa = (\hat{H}_\kappa^\mu \mid \mu \in R(G))$$

with $N \times a^\mu N^{(\mu)}$ submatrices $\hat{H}_\kappa^\mu$, which are obtained by augmenting H_κ^μ with zeros. Using these expressions we have

$$\begin{aligned}
\tilde{J} &= H^* J H \\
&= \sum_{e=1}^{M} \sum_{\kappa} \sum_{\kappa'} (\hat{H}_\kappa)^* \hat{J}_e \hat{H}_{\kappa'} \\
&= \sum_{e=1}^{M} \sum_{\kappa} \sum_{\kappa'} ((\hat{H}_\kappa^\mu)^* \hat{J}_e \hat{H}_{\kappa'}^{\mu'} \mid \mu, \mu' \in R(G)) \\
&= \bigoplus_{\mu \in R(G)} \left(\sum_{e=1}^{M} \sum_{\kappa} \sum_{\kappa'} (\hat{H}_\kappa^\mu)^* \hat{J}_e \hat{H}_{\kappa'}^{\mu} \right).
\end{aligned}$$

The last equality follows from (14); note also that $(\hat{H}_\kappa^\mu)^* \hat{J}_e \hat{H}_{\kappa'}^{\mu'} \neq O$ in general. Since J_e is small in size, there are only small number (at most N_e^2) of pairs (κ, κ') that contribute to this sum. For each $(e, \mu, \kappa, \kappa')$, the contribution of the matrix product $(\hat{H}_\kappa^\mu)^* \hat{J}_e \hat{H}_{\kappa'}^{\mu}$ to A^μ can be computed with $a_\kappa^\mu N_e (N_e + a_{\kappa'}^\mu)$ operations. Thus, if a_κ^μ and N_e are small, being independent of N and $|G|$ (which is often the case), we can compute $\tilde{J}$ with $O(M|G|)$ operations. This implies that the first term in (23) could be replaced by a term of $O(N|G|)$ if $M = O(N)$. See [1] and [12].

The sparsity of J is partially inherited to $\tilde{J}$ (or to A^μ), as has been mentioned at the end of § 3.3. There are many computational techniques that exploit sparsity in computing determinants, and hence the second term $(a^\mu)^3/3$ in (23) could be reduced considerably.

4. Symmetric truss structures. The method discussed in § 3 is applied to D_n-symmetric truss structures. It is noteworthy in connection to the block diagonalization (17) of J that each irreducible representation of D_n is absolutely irreducible, i.e., $R_a(D_n) = R(D_n)$.

4.1. Block-diagonal form. Suppose a D_n-symmetric truss structure is described by a D_n-equivariant system of equations (8) in terms of the node displacement vector $v \in V = \mathbf{R}^P \otimes \mathbf{R}^E$, where P is the set of the free nodes and $E = \{X, Y, Z\}$ the set of coordinate axes. We assume that $r \in D_n$ corresponds to the rotation around Z-axis at angle $2\pi/n$ and $s \in D_n$ to the reflection with respect to XZ-plane.

The orbital decomposition (19) is naturally induced from the geometric symmetry. It is easy to see that $|P_k| \in \{1, n, 2n\}$; i.e., if we put

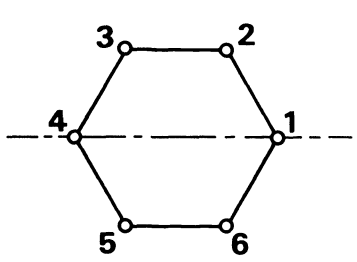
$$K_0 = |\{k \mid |P_k| = 1\}|, \quad K_1 = |\{k \mid |P_k| = n\}|, \quad K_2 = |\{k \mid |P_k| = 2n\}|,$$

we have

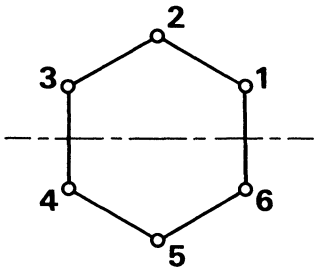
$$K = K_0 + K_1 + K_2.$$

By convention we put $K_1 = 0$ when $n = 1$.

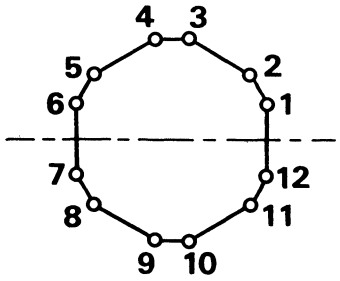
For those orbits P_k with $|P_k| = n$, we distinguish two further types, namely, type V and type M illustrated in Figs. 2(a) and 2(b). For n even, an orbit of type V has



Cases 1ZV, 1XYV



Cases 1ZM, 1XYM



Cases 2Z, 2XY

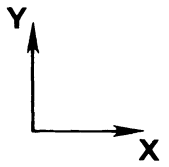
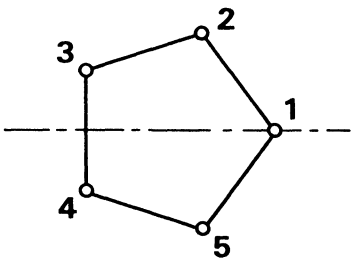
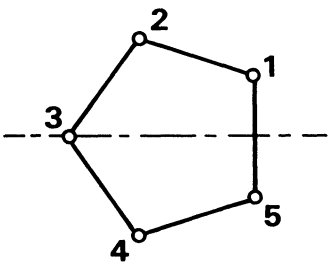


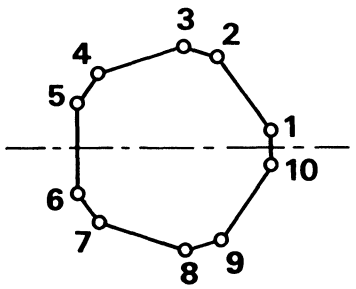
FIG. 2(a). *Orbits and node numbers.* $n = 6$.



Cases 1ZV, 1XYV



Cases 1ZM, 1XYM



Cases 2Z, 2XY

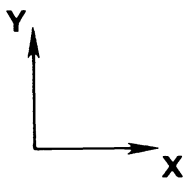


FIG. 2(b). *Orbits and node numbers.* $n = 5$.

two nodes on the XZ-plane, while one of type M has none. For n odd, an orbit of either type has one node on the XZ-plane. We put

$$K_{1V} = |\{k \mid |P_k| = n,\ P_k \text{ is of type } V\}|,$$

$$K_{1M} = |\{k \mid |P_k| = n,\ P_k \text{ is of type } M\}|.$$

Obviously, $K_{1V} + K_{1M} = K_1$.

Let χ_{P_k} denote the character of the permutation representation of D_n on $\mathbf{R}^{P_k}$; $\chi_{P_k}(g)$ is equal to the number of nodes in P_k which do not change their position under the action of $g \in D_n$. It is decomposed as

$$\chi_{P_k} = \sum_{\mu \in R(D_n)} a_k^{\mu} \chi^{(\mu)},$$

where the multiplicities a_k^{μ} are determined by (6) and are given below. Recall the notation $\mu = (d, j)$ for the irreducible representations of D_n introduced in § 2.

(24)

	$k: \lvert P_k\rvert = 1$	$k: \lvert P_k\rvert = n$ Type V	$k: \lvert P_k\rvert = n$ Type M	$k: \lvert P_k\rvert = 2n$
$\mu = (1,1)$:	1	1	1	1
$\mu = (1,2)$:	0	0	0	1
$\mu = (1,3)$:	0/-	1/-	0/-	1/-
$\mu = (1,4)$:	0/-	0/-	1/-	1/-
$\mu = (2,j)$:	0	1	1	2

Here an entry with / indicates two cases (n: even/n: odd), and $j = 1, \cdots, \lfloor (n-1)/2 \rfloor$, where $\lfloor \cdot \rfloor$ is the Gauss symbol meaning the integer part.

We may divide E into $L = 2$ parts, $E_1 = \{Z\}$ and $E_2 = \{X, Y\}$. The representation of D_n on $\mathbf{R}^{E_1}$ is the unit representation, whereas that on $\mathbf{R}^{E_2}$ for $n \geqq 3$ is the two-dimensional irreducible representation denoted as (2, 1) in § 2; i.e.,

$$\chi_{E_1} = \chi^{(1,1)}, \qquad \chi_{E_2} = \begin{cases} \chi^{(1,1)} + \chi^{(1,2)} & \text{if } n = 1, \\ \chi^{(1,3)} + \chi^{(1,4)} & \text{if } n = 2, \\ \chi^{(2,1)} & \text{if } n \geqq 3. \end{cases}$$

The representation T_κ is the tensor product of the representations on $\mathbf{R}^{P_k}$ and $\mathbf{R}^{E_l}$, where $\kappa = (k, l)$, i.e.,

$$\chi_\kappa = \chi_{P_k} \chi_{E_l}, \qquad \kappa = (k, l).$$

Decomposition of χ_κ

$$\chi_\kappa = \sum_{\mu \in R(D_n)} a_\kappa^{\mu} \chi^{(\mu)}$$

into irreducible representations is now straightforward. For $\kappa = (k, l)$ with $l = 1$, we have $\chi_\kappa = \chi_{P_k}$ since $\chi_{E_1} = 1$, and hence the multiplicities a_κ^{μ} are given by (24) above. The multiplicities a_κ^{μ} for $\kappa = (k, l)$ with $l = 2$ are given below (types V and M need not be distinguished).

(25)

For $n=1$:	$k:\|P_k\|=1$		$k:\|P_k\|=2$
$\mu=(1,1)$:	1		2
$\mu=(1,2)$:	1		2
For $n=2$:	$k:\|P_k\|=1$	$k:\|P_k\|=2$	$k:\|P_k\|=4$
$\mu=(1,1)$:	0	1	2
$\mu=(1,2)$:	0	1	2
$\mu=(1,3)$:	1	1	2
$\mu=(1,4)$:	1	1	2
For $n\geqq 3$:	$k:\|P_k\|=1$	$k:\|P_k\|=n$	$k:\|P_k\|=2n$
$\mu=(1,1)$:	0	1	2
$\mu=(1,2)$:	0	1	2
$\mu=(1,3)$:	0/-	1/-	2/-
$\mu=(1,4)$:	0/-	1/-	2/-
$\mu=(2,1)$:	1	2	4
$\mu=(2,j)$:	0	2	4

Here an entry with / indicates two cases (n: even/n: odd), and $j=2,\cdots,\lfloor(n-1)/2\rfloor$.

Finally, we obtain the following expressions of $a^\mu=\sum_\kappa a_\kappa^\mu$ and $\sum_\kappa a_\kappa^\mu N_\kappa$ for $\mu\in R(D_n)$.

(26)

For $n=1$:	a^μ	$\sum_\kappa a_\kappa^\mu N_\kappa$
$\mu=(1,1)$:	$2K_0+3K_2$	$3K_0+10K_2$
$\mu=(1,2)$:	K_0+3K_2	$2K_0+10K_2$
For $n=2$:	a^μ	$\sum_\kappa a_\kappa^\mu N_\kappa$
$\mu=(1,1)$:	$K_0+2K_1+3K_2$	$K_0+2(3K_1+10K_2)$
$\mu=(1,2)$:	K_1+3K_2	$2(2K_1+10K_2)$
$\mu=(1,3)$:	$K_0+K_1+K_{1V}+3K_2$	$2(K_0+2K_1+K_{1V}+10K_2)$
$\mu=(1,4)$:	$K_0+K_1+K_{1M}+3K_2$	$2(K_0+2K_1+K_{1M}+10K_2)$
For $n\geqq 3$:	a^μ	$\sum_\kappa a_\kappa^\mu N_\kappa$
$\mu=(1,1)$:	$K_0+2K_1+3K_2$	$K_0+n(3K_1+10K_2)$
$\mu=(1,2)$:	K_1+3K_2	$n(2K_1+10K_2)$
$\mu=(1,3)$:	$K_1+K_{1V}+3K_2$/-	$n(2K_1+K_{1V}+10K_2)$/-
$\mu=(1,4)$:	$K_1+K_{1M}+3K_2$/-	$n(2K_1+K_{1M}+10K_2)$/-
$\mu=(2,1)$:	$K_0+3K_1+6K_2$	$2K_0+n(5K_1+20K_2)$
$\mu=(2,j)$:	$3K_1+6K_2$	$n(5K_1+20K_2)$

Again "/" indicates two cases (n: even/n: odd), and $j=2,\cdots,\lfloor(n-1)/2\rfloor$.

Recall that the multiplicities a^μ given above are equal to the sizes of the diagonal blocks A^μ. For example, when n ($\geqq 4$) is even, we have one block of size $K_0+2K_1+3K_2$, one of size K_1+3K_2, one of size $K_1+K_{1V}+3K_2$, one of size $K_1+K_{1M}+3K_2$, a pair of identical blocks of size $K_0+3K_1+6K_2$, and $(n/2-2)$ pairs of identical blocks of size $3K_1+6K_2$.

Example 4.1. Recall the spherical diamond shell of Fig. 1, which is D_6-equivariant. The node set P ($|P|=31$) is divided into $K=5$ orbits, as shown in Fig. 3:

$$|P_1|=1,\qquad |P_2|=|P_3|=|P_4|=6\ (=n),\qquad |P_5|=12\ (=2n),$$

and P_2 and P_4 are of type V, and P_3 of type M. Hence we have

$$K_0=1,\qquad K_{1V}=2,\qquad K_{1M}=1,\qquad K_2=1.$$

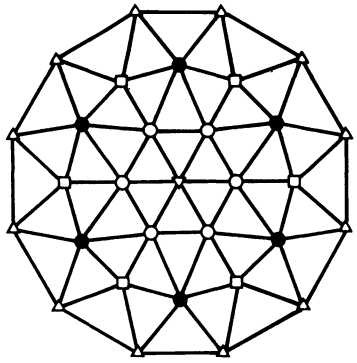


FIG. 3. *Orbital decomposition* ($P_1(\nabla)$; $P_2(\bigcirc)$; $P_3(\bullet)$; $P_4(\square)$; $P_5(\triangle)$).

Also note that

$$N = 3|P| = 3(K_0 + 6K_1 + 12K_2) = 93, \qquad N_\kappa \in \{1, 6, 12, 24\}.$$

Hence H is a block-diagonal matrix of order $93(=N)$ consisting of $10(=2K)$ blocks of the following sizes:

block size:	1	2	6	12	24
number of blocks:	1	1	3	4	1

We see from (26) that the Jacobian matrix J of size 93 is brought into $\tilde{J}$, which is a block-diagonal matrix consisting of the blocks of the following sizes:

representation μ:	$(1,1)$	$(1,2)$	$(1,3)$	$(1,4)$	$(2,1)$	$(2,2)$
block size a^μ:	10	6	8	7	16	15
number of blocks $N^{(\mu)}$:	1	1	1	1	2*	2*

Here 2* means that the two blocks are identical. Each block retains sparsity (and band structure) to some extent; for example, the block of size 16 looks as follows, where × denotes nonzero entries:

×	×	×	×	0	0	0	0	0	0	0	0	0	0	0	0
×	×	×	×	×	×	×	×	×	×	0	0	0	0	0	0
×	×	×	×	×	×	×	×	×	×	0	0	0	0	0	0
×	×	×	×	×	×	×	×	×	×	0	0	0	0	0	0
0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	0
0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	0
0	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
0	0	0	0	×	×	×	×	×	×	×	×	×	0	×	×
0	0	0	0	×	×	×	×	×	×	×	×	×	×	×	0
0	0	0	0	×	×	×	×	×	×	×	×	×	×	0	×
0	0	0	0	×	×	×	×	×	×	0	×	×	×	×	×
0	0	0	0	×	×	×	×	×	×	×	×	0	×	×	×
0	0	0	0	×	0	×	×	0	×	×	0	×	×	×	×

4.2. Transformation matrix. Explicit forms for the "local" transformation matrices H_κ are given for convenience in applications. They can be determined by the method of projection (see Miller [16]) or directly by (13). We treat the case with $n \geqq 2$.

We have eight cases to consider, which we shall name as follows:

	κ	V_κ	dim V_κ
Case $0Z$:	$\kappa=(k,1), \lvert P_k\rvert=1$	V_{0Z}	1
Case $0XY$:	$\kappa=(k,2), \lvert P_k\rvert=1$	V_{0XY}	2
Case $1ZV$:	$\kappa=(k,1), \lvert P_k\rvert=n$	V_{1ZV}	n
Case $1ZM$:	$\kappa=(k,1), \lvert P_k\rvert=n$	V_{1ZM}	n
Case $1XYV$:	$\kappa=(k,2), \lvert P_k\rvert=n$	V_{1XYV}	$2n$
Case $1XYM$:	$\kappa=(k,2), \lvert P_k\rvert=n$	V_{1XYM}	$2n$
Case $2Z$:	$\kappa=(k,1), \lvert P_k\rvert=2n$	V_{2Z}	$2n$
Case $2XY$:	$\kappa=(k,2), \lvert P_k\rvert=2n$	V_{2XY}	$4n$

The corresponding geometric configurations of the nodes are depicted in Fig. 2. Note the difference in the configurations for two types V and M. Cases $0Z$, $1ZV$, $1ZM$, and $2Z$ correspond to the displacements in the Z-direction, and cases $0XY$, $1XYM$, $1XYV$, and $2XY$ to those in the XY-directions.

Let V_κ^μ denote the isotypic components of V_κ; remember that the multiplicities a_κ^μ are given in (24) and (25). The basis of V_κ^μ is given by the column vectors of H_κ^μ to be listed below; when $a_\kappa^\mu \geqq 2$, the matrix H_κ^μ contains a_κ^μ groups of columns separated by semicolon (;) and each group corresponds to an irreducible component, denoted as H_i^μ in (12). The matrix H_κ is obtained as

$$H_\kappa=(H_\kappa^\mu \mid \mu \in R(D_n)).$$

We shall use the mnemonics such as $0Z$, $0XY$ for κ and (d,j) for μ; for example, $H_{0Z}^{(1,1)}$ will denote the matrix H_κ^μ with κ in the case $0Z$ and $\mu=(1,1)$ (the unit representation). The components of H_κ^μ are ordered consistently with the node numbers as shown in Fig. 2; for example, the first two components of $H_{1XYV}^{(d,j)}$ correspond to the displacement of the first node in the X- and the Y-direction, respectively. Remember that $\mu=(1,3)$ and $(1,4)$ apply only for n even.

Put

$$\omega=\exp\frac{2\pi i}{n}, \qquad \omega^{1/4}=\exp\frac{\pi i}{2n},$$

$$\gamma_n(a,b)=(ab^{j-1}/\sqrt{n/2} \mid j=1,\cdots,n)=\sqrt{\frac{2}{n}}\begin{pmatrix} a \\ ab \\ \vdots \\ ab^{n-1}\end{pmatrix},$$

$$\Gamma_n(a,b)=(\Re(ab^{j-1}/\sqrt{n}), \Im(ab^{j-1}/\sqrt{n}) \mid j=1,\cdots,n)=\frac{1}{\sqrt{n}}\begin{pmatrix} \Re(a) \\ \Im(a) \\ \Re(ab) \\ \Im(ab) \\ \vdots \\ \Re(ab^{n-1}) \\ \Im(ab^{n-1})\end{pmatrix},$$

where $\Re$ and $\Im$ denote, respectively, the real and the imaginary part of a complex number.

Case $0Z$. dim $V_{0Z}=1$.

$$H_{0Z}^{(1,1)}=1.$$

Case $0XY$. dim $V_{0XY}=2$.

$$H_{0XY}^{(2,1)}=I_2.$$

Case $1ZV$. dim $V_{1ZV}=n$.

$$H_{1ZV}^{(1,1)}=\gamma_n(1/\sqrt{2},1),\qquad H_{1ZV}^{(1,3)}=\gamma_n(1/\sqrt{2},-1),$$
$$H_{1ZV}^{(2,j)}=(\Re\gamma_n(1,\omega^j),\Im\gamma_n(1,\omega^j)).$$

Case $1ZM$. dim $V_{1ZM}=n$.

$$H_{1ZM}^{(1,1)}=\gamma_n(1/\sqrt{2},1),\qquad H_{1ZM}^{(1,4)}=\gamma_n(1/\sqrt{2},-1),$$
$$H_{1ZM}^{(2,j)}=(\Re\gamma_n(\omega^{j/2},\omega^j),\Im\gamma_n(\omega^{j/2},\omega^j)).$$

Case $1XYV$. dim $V_{1XYV}=2n$.

$$H_{1XYV}^{(1,1)}=\Gamma_n(1,\omega),\qquad H_{1XYV}^{(1,2)}=\Gamma_n(i,\omega),$$
$$H_{1XYV}^{(1,3)}=\Gamma_n(1,-\omega),\qquad H_{1XYV}^{(1,4)}=\Gamma_n(i,-\omega),$$
$$H_{1XYV}^{(2,j)}=(\Gamma_n(1,\omega^{-j+1}),\Gamma_n(i,\omega^{-j+1});\Gamma_n(1,\omega^{j+1}),\Gamma_n(-i,\omega^{j+1})).$$

Case $1XYM$. dim $V_{1XYM}=2n$.

$$H_{1XYM}^{(1,1)}=\Gamma_n(\omega^{1/2},\omega),\qquad H_{1XYM}^{(1,2)}=\Gamma_n(i\omega^{1/2},\omega),$$
$$H_{1XYM}^{(1,3)}=\Gamma_n(i\omega^{1/2},-\omega),\qquad H_{1XYM}^{(1,4)}=\Gamma_n(\omega^{1/2},-\omega),$$
$$H_{1XYM}^{(2,j)}=(\Gamma_n(\omega^{(-j+1)/2},\omega^{-j+1}),\Gamma_n(i\omega^{(-j+1)/2},\omega^{-j+1});$$
$$\Gamma_n(\omega^{(j+1)/2},\omega^{j+1}),\Gamma_n(-i\omega^{(j+1)/2},\omega^{j+1})).$$

Case $2Z$. dim $V_{2Z}=2n$.

$$H_{2Z}^{(1,1)}=\gamma_{2n}(1/\sqrt{2},1),\qquad H_{2Z}^{(1,2)}=\gamma_{2n}(1/\sqrt{2},-1),$$
$$H_{2Z}^{(1,3)}=\Re\gamma_{2n}((1+i)/\sqrt{2},i),\qquad H_{2Z}^{(1,4)}=\Im\gamma_{2n}((1+i)/\sqrt{2},i),$$
$$H_{2Z}^{(2,j)}=(\Re\gamma_{2n}(\omega^{j/4},\omega^{j/2}),\Im\gamma_{2n}(\omega^{j/4},\omega^{j/2});$$
$$\Re\gamma_{2n}(\omega^{j^*/4},\omega^{j^*/2}),\Im\gamma_{2n}(-\omega^{j^*/4},\omega^{j^*/2})),$$

where $j^*=n-j$.

Case $2XY$. dim $V_{2XY}=4n$.

$$H_{2XY}^{(1,1)}=(\Gamma_{2n}(\omega^{1/4},\omega^{1/2});\Gamma_{2n}(i\omega^{1/4},-\omega^{1/2})),$$
$$H_{2XY}^{(1,2)}=(\Gamma_{2n}(i\omega^{1/4},\omega^{1/2});\Gamma_{2n}(\omega^{1/4},-\omega^{1/2})),$$
$$H_{2XY}^{(1,3)}=\left(\Gamma_{2n}\left(\frac{1-i}{\sqrt{2}}\omega^{1/4},-i\omega^{1/2}\right);\Gamma_{2n}\left(\frac{1+i}{\sqrt{2}}\omega^{1/4},i\omega^{1/2}\right)\right),$$
$$H_{2XY}^{(1,4)}=\left(\Gamma_{2n}\left(\frac{1+i}{\sqrt{2}}\omega^{1/4},-i\omega^{1/2}\right);\Gamma_{2n}\left(\frac{1-i}{\sqrt{2}}\omega^{1/4},i\omega^{1/2}\right)\right),$$
$$H_{2XY}^{(2,j)}=(\Gamma_{2n}(\omega^{(-j+1)/4},\omega^{(-j+1)/2}),\Gamma_{2n}(i\omega^{(-j+1)/4},\omega^{(-j+1)/2});$$
$$\Gamma_{2n}(\omega^{(j+1)/4},\omega^{(j+1)/2}),\Gamma_{2n}(-i\omega^{(j+1)/4},\omega^{(j+1)/2});$$
$$\Gamma_{2n}(\omega^{(-j^*+1)/4},\omega^{(-j^*+1)/2},\Gamma_{2n}(-i\omega^{(-j^*+1)/4},\omega^{(-j^*+1)/2});$$
$$\Gamma_{2n}(\omega^{(j^*+1)/4},\omega^{(j^*+1)/2}),\Gamma_{2n}(i\omega^{(j^*+1)/4},\omega^{(j^*+1)/2})),$$

where $j^*=n-j$.

4.3. Computational cost. In line with § 3.4 we consider here the total amount of computation in the case of D_n-symmetric structures with $n \geqq 3$. We shall derive an estimate under the reasonable assumption that K_0 is negligibly small as compared with N/n.

Substitution of (26) into (23) yields an expression in terms of K_0, K_1, K_2, and n for the total amount of computation. Using the relation

$$K_0+nK_1+2nK_2=|P|=\frac{N}{3},$$

estimates for $\sum_\kappa a_\kappa^\mu N_\kappa$ such as

$$K_0+n(3K_1+10K_2)=|P|+2n(K_1+4K_2)\leqq\frac{5N}{3}$$

and those for a^μ such as

$$K_0+2K_1+3K_2\leqq K_0+2(K_1+2K_2)=\frac{2N}{3n}+\left(1-\frac{2}{n}\right)K_0\sim\frac{2N}{3n},$$

we obtain

$$\sum_{\mu\in R(G)}\left((N+a^\mu)\left(\sum_\kappa a_\kappa^\mu N_\kappa\right)\right)\leqq\left(\frac{5}{3}n+6\right)N^2+O(N)$$

as an estimate for the cost of computing the matrix $\tilde{J}$ from J. The second term of (23) for testing for the singularity of the blocks A^μ is estimated similarly as

$$\frac{1}{3}\sum_{\mu\in R(G)}(a^\mu)^3\leqq\frac{1}{6n^2}N^3.$$

The total cost is therefore approximately bounded by

$$\left(\frac{5}{3}n+6\right)N^2+\frac{1}{6n^2}N^3. \tag{27}$$

(When J is known to be a symmetric matrix, this bound should be divided by two.) Note that the cost for computing $\tilde{J}$ from J is bounded by the order of N^2 and that this is due to our choice of H; if H were a dense matrix, then the computation of $\tilde{J}=H^*JH$ itself would require $O(N^3)$ computations. The above estimate (27) shows that the proposed method is more efficient than the direct calculation of the determinant of J, which will require $N^3/3$ multiplications.

As has been discussed in § 3.4, it would be fair to mention in this connection the issue of sparsity of J, which is not taken into consideration in the above argument. According to the method of § 3.4, it is possible to compute $\tilde{J}$ with $O(nN)$ operations instead of $O(nN^2)$ operations as in the estimate (27).

The estimate (27) (or its modification incorporating sparsity) also suggests the possibility of partial use of group symmetry to enhance the overall computational efficiency, as follows. Let n' be a divisor of n. Since a D_n-symmetric structure is also $D_{n'}$-symmetric, we can apply the method of block-diagonalization for $D_{n'}$ symmetry, making partial use of group symmetry. We may choose n' so that the computational cost, given by (27) with n replaced by n', is to be minimized. For example, this criterion yields $n'=3$ for the spherical diamond shell of Fig. 1.

5. Basis change at symmetry-breaking points. In the previous sections we have considered a computational procedure for a G-symmetric branch in the bifurcation diagram. When we encounter a symmetry-breaking bifurcation point, we are to trace another branch having less symmetry; put $G'=\Sigma(v)$ for a solution point (v, λ) on this branch. Then the Jacobian matrix J commutes no longer with the action of G but with the action of the subgroup G'. In other words, we are to consider the restriction, denoted as $T\downarrow G'$, of the representation T of G on G'. In this section we are interested in how the transformation matrix H, chosen with reference to G, should be changed with such symmetry breaking.

5.1. Basis of G'-symmetric space. We consider here the G'-symmetric subspace $\mathcal{S}(G')$ which plays a primary role in the path-tracing phase (i). Recall that the column vectors of H constitute a basis of V which is compatible with the isotypic decomposition (1) with respect to G. In particular, the column vectors of $(H_i^{\mu_0} \mid i=1, \cdots, a^{\mu_0})$ in (12) affords a basis of $\mathcal{S}(G)=V^{\mu_0}$. Since $\mathcal{S}(G)\subseteq\mathcal{S}(G')$, we may hope for a situation where a basis of $\mathcal{S}(G')$ is obtained as a subfamily of the column vectors of H.

Define

$$\ker T^{(\mu)}=\{g\in G \mid T^{(\mu)}(g)=I\}, \qquad \mu\in R(G).$$

It is easy to see that $\ker T^{(\mu)}$ is a normal (or invariant) subgroup of G and

$$\mathcal{S}(\ker T^{(\mu)})\supseteq V^{\mu}, \qquad \mu\in R(G).$$

Since $G'\subseteq\ker T^{(\mu)}$ implies

$$\mathcal{S}(G')\supseteq\mathcal{S}(\ker T^{(\mu)}),$$

we see

$$\mathcal{S}(G')\supseteq\bigoplus\{V^{\mu} \mid G'\subseteq\ker T^{(\mu)}\}. \tag{28}$$

If this holds with equality, the column vectors of

$$(H_i^{\mu} \mid G'\subseteq\ker T^{(\mu)};\ i=1, \cdots, a^{\mu})$$

give a basis of $\mathcal{S}(G')$.

Let us consider a simple bifurcation point, at which $\ker J$ is one-dimensional by definition. Let $\mu_1\in R(G)$ be such that $\ker J\subseteq V^{\mu_1}$. As has been pointed out by Fujii and Yamaguti [6], the subgroup G' representing the symmetry of the bifurcating branch is given by

$$G'=\ker T^{(\mu_1)}.$$

This implies, in particular, that G' is a normal subgroup of G. Combining these observations with the lemma below, we obtain the following proposition. Note also that (10) is a special case of the lemma with $G'=G$.

PROPOSITION. *For a branch bifurcating from a simple bifurcation point on the G-symmetric path,* (28) *holds with equality and therefore a basis of $\mathcal{S}(G')$ is obtained as a subfamily of the column vectors of H.* □

LEMMA. *If G' is a normal subgroup of G, then*

$$\mathcal{S}(G')=\bigoplus\{V^{\mu} \mid G'\subseteq\ker T^{(\mu)}\},$$

where V^{μ} are the isotypic components with respect to G.

Proof. Since (28) has been shown, it suffices to prove that

$$G' \not\subseteq \ker T^{(\mu)} \Rightarrow \mathcal{P}(G') \cap V^{\mu} = \{0\}.$$

The left-hand side is equivalent to assuming that $T_i^{\mu}\downarrow G'$ (for each i) is not a multiple of the unit representation. By Clifford's theorem (see, e.g., [4], [14]), on the other hand, all of the irreducible components of $T_i^{\mu}\downarrow G'$ are conjugate with one another (and have the same multiplicity). It follows then that $T_i^{\mu}\downarrow G'$ does not contain the unit representation as its irreducible component, since the unit representation of G' constitutes a conjugacy class by itself. This implies $\mathcal{P}(G') \cap V^{\mu} = \{0\}$ since the subrepresentation of $T_i^{\mu}\downarrow G'$ on $\mathcal{P}(G') \cap V^{\mu}$ must be a multiple of the unit representation. □

In the lemma above, the assumption of G' being a normal subgroup is necessary. For example, let $G = D_6$ and $G' = \{1, s\}$. For the two two-dimensional irreducible representations $\mu = (2, 1)$ and $\mu = (2, 2)$, we have $G' \not\subseteq \ker T^{(\mu)}$ $(= \{1\})$ but $\dim(\mathcal{P}(G') \cap V^{\mu}) = (\dim V^{\mu})/2$, the multiplicity of μ.

5.2. Chain-adapted basis. Here we consider the direct sum decomposition (11) of $T(g)$, $g \in G$, by means of a matrix H. Since $T^{(\mu)}\downarrow G'$ may be reducible, each diagonal block of (11) may possibly be decomposed further with an additional change of the basis. If this additional change is not necessary, i.e., if $H^*T(g)H$, $g \in G'$, is also in a block-diagonal form such that the diagonal blocks are irreducible representations of G' with the same matrix for equivalent representations, then the basis matrix H is said to be *adapted* to the chain: $G \supseteq G'$. More generally, a basis is said to be adapted to a chain of subgroups; $G \supseteq G_1 \supseteq \cdots \supseteq G_r$ if it is adapted to $G \supseteq G_i$ for $i = 1, \cdots, r$. The idea of "adapted basis" has been used in quantum chemistry (e.g., [15]) and its notion is explicitly formulated by Clausen [3] with the observation that for any chain of (a finite group) G, there exists a basis adapted to it. In some applications, it turns out to be convenient to slightly (but not essentially) extend the notion of adaptedness. We shall say that H is *adapted up to sign*, if H is adapted when some of its columns are multiplied by -1.

Suppose a chain $G \supseteq G_1 \supseteq \cdots \supseteq G_r$ represents how the symmetry breaks down with a progression of bifurcations. In a favorable case where this chain is known in advance before the numerical computation, we can find a matrix H which is adapted (up to sign) to this chain. This means that we can use the same H throughout to make J into a block-diagonal form, though the positions of the nonzero elements in H^*JH vary with G_i. In most practical situations, however, the matrix H is to be chosen at the beginning of computation without reference to such a chain.

In this connection, the following two facts are to be noted.

(i) We have assumed in § 2 that a family $\{T^{(\mu)} | \mu \in R(G)\}$ of nonequivalent matrix representations of G is given; then the basis matrix H is determined by (13) with reference to the family. Though the choice of $T^{(\mu)}$ does not matter for the decomposition of J with symmetry G, it affects the "adaptedness" of H to $G \supset G'$.

(ii) If (v, λ) is a solution to (8) with $\Sigma(v) = G'$ and $g \in G$, then $(w, \lambda) = (T(g)v, \lambda)$ is also a solution to it. We shall call such branches conjugate to each other. The symmetry group $\Sigma(w)$ of w is conjugate to G', i.e., $\Sigma(w) = gG'g^{-1}$. Since conjugate branches can be regarded as representing "physically equivalent" phenomena, we may choose the one which is convenient from the computational viewpoint. In particular, if H is adapted (up to sign) to $G \supset gG'g^{-1}$ for some $g \in G - G'$ and not to $G \supset G'$, it is advantageous to choose the branch of solutions $(T(g)v, \lambda)$ instead of that of (v, λ) which has the symmetry $\Sigma(v) = G'$.

The above observations would lead to the following choice of $T^{(\mu)}$ and of the bifurcation branches to be traced.

(i) Fix a maximal chain of subgroups of G and choose $T^{(\mu)}$ so that the basis matrix H determined by (13) is adapted (up to sign) to the chain.

(ii) Among the conjugate bifurcation branches, choose one, if any, such that H is adapted (up to sign) to $G \supseteq G'$, where G' denotes the symmetry group of the branch. If such a branch does not exist, pick up any one of them and modify the part of H which is not adapted.

The arguments above are now illustrated by symmetric structures with $G = D_n$. The subgroups are given by

$$\{C_m \mid m \mid n\} \cup \{D_m^l \mid m \mid n, l = 1, \cdots, n/m\},$$

where $m \mid n$ means that m is a divisor of n and

$$D_m^l = \{r^{kn/m}, sr^{l-1+kn/m} \mid k = 0, 1, \cdots, m-1\},$$
$$C_m = \{r^{kn/m} \mid k = 0, 1, \cdots, m-1\}.$$

Note first that D_m^l is conjugate to D_m^1 or D_m^2 according as l is odd or even, and moreover, that D_m^1 and D_m^2 are conjugate if n/m is odd. Hence $G' \subseteq D_n$ is conjugate to one of the following:

$$\{D_m^1, C_m \mid m \mid n\} \cup \{D_m^2 \mid m \mid n, n/m \text{ is even}\}.$$

Due to our choice (7) of $T^{(\mu)}$ for D_n, the basis matrix H is adapted (up to sign) to the chain $D_n \supset C_m$ for all m, and it is adapted (up to sign) to $D_n \supseteq D_m^l$ if and only if

$$\left(\frac{2j}{m} \notin \mathbf{Z} \text{ or } \frac{2(l-1)j}{n} \in \mathbf{Z}\right) \quad \text{for } j = 1, \cdots, \left\lfloor \frac{n-1}{2} \right\rfloor, \tag{29}$$

where $\mathbf{Z}$ is the set of integers. In particular, H is adapted (up to sign) to $D_n \supset D_m^1$ for all m since the second condition in (29) is satisfied. H is not adapted (up to sign) to $D_n \supset D_m^2$ for some m, but even for such m we can sometimes find l such that

(30) D_m^l is conjugate to D_m^2 and H is adapted (up to sign) to $D_n \supset D_m^l$.

It is not difficult to see that if n/m is odd, (30) is satisfied by $l = 1$; if n/m is even and n is not a multiple of 4, (30) is satisfied by $l = n/(2m) + 1$; and if n/m is even and n is a multiple of 4, (30) is never satisfied. For $n \leqq 20$, e.g., the following pairs (n, m) admit no l which satisfies (30):

$$n = 4\text{: } m = 1, 2,$$
$$n = 8\text{: } m = 1, 2, 4,$$
$$n = 12\text{: } m = 1, 2, 3, 6,$$
$$n = 16\text{: } m = 1, 2, 4, 8,$$
$$n = 20\text{: } m = 1, 2, 10.$$

Note here D_m^1 and $D_m^{n/(2m)+1}$ have, respectively, the XZ-plane and the YZ-plane as the planes of symmetry.

These facts imply the following:

(1) If n is not a multiple of four, the matrix H remains valid throughout the bifurcation analysis if we choose appropriate bifurcation branches. To be more concrete, we may choose any one of the C_m-symmetric conjugate solutions, which are not symmetric with respect to any plane containing the Z-axis. When we encounter a solution which is symmetric with respect to a plane containing the Z-axis, we should

and can choose the solution which is conjugate to it and is D_m^1-or $D_m^{n/(2m)+1}$-symmetric, i.e., symmetric with respect either to the XZ-plane or to the YZ-plane. With this strategy, we obtain the descending chain of symmetry groups $G \supset G_1 \supset \cdots \supset G_r$ of the form

$$D_n \supset D_{m_1}^1 \supset \cdots \supset D_{m_p}^1 \supset D_{m_{p+1}}^{l_{p+1}} \supset \cdots \supset D_{m_q}^{l_q} \supset C_{m_{q+1}} \supset \cdots \supset C_{m_r}, \tag{31}$$

where $m_k \mid m_{k-1}$ for $k=1, \cdots, r$ (with $m_0=n$), m_k is odd for $k=p+1, \cdots, q$, and $l_k = n/(2m_k)+1$ for $k=p+1, \cdots, q$. The basis matrix H is adapted (up to sign) to this chain.

(2) If n is a multiple of four, we may take the chain of the form (31), where $m_k \mid m_{k-1}$ for $k=1, \cdots, r$ (with $m_0=n$) and both n/m_k and l_k are even for $k=p+1, \cdots, q$. By conjugacy, it suffices to consider the following three cases.

(2-1) $p=q$. Then H is adapted (up to sign) to the chain.

(2-2) $p=0$ and $l_k=2$ for $k=1, \cdots . q$. When we encounter the $D_{m_1}^2$-symmetric solution, we should replace H_i^μ by $H_i^\mu Q(j, n)$ for all $\mu=(2, j)$ and $i=1, \cdots, a^\mu$, where $Q(j, n)$ is a 2×2 orthogonal matrix defined by

$$Q(j, n) = \begin{pmatrix} \cos(\pi j/n) & \sin(\pi j/n) \\ -\sin(\pi j/n) & \cos(\pi j/n) \end{pmatrix}.$$

Note that $Q(j, n)^* S R^j Q(j, n) = S$, where R and S are defined by (7).

(2-3) $1 \leqq p < q$, n/m_p is odd and $l_k = n/m_p + 1$ for $k=p+1, \cdots, q$. When we encounter the $D_{m_{p+1}}^{n/m_p+1}$-symmetric solution, we should replace H_i^μ by $H_i^\mu Q(j, m_p)$ for all $\mu=(2, j)$ and $i=1, \cdots, a^\mu$.

Note that the three cases can be recognized at the point when H is to be modified. The computational cost for the modification of H along a chain is bounded by $O(nN)$.

Example 5.1. Recall again the spherical diamond shell of Fig. 1. The equilibrium paths of this structure under a parametric loading with a D_6-symmetric vertical loading pattern have been computed in [13]. The bifurcation behavior may be summarized by the schematic diagram in Fig. 4. Note that this is the case (1) above and that H need not be modified.

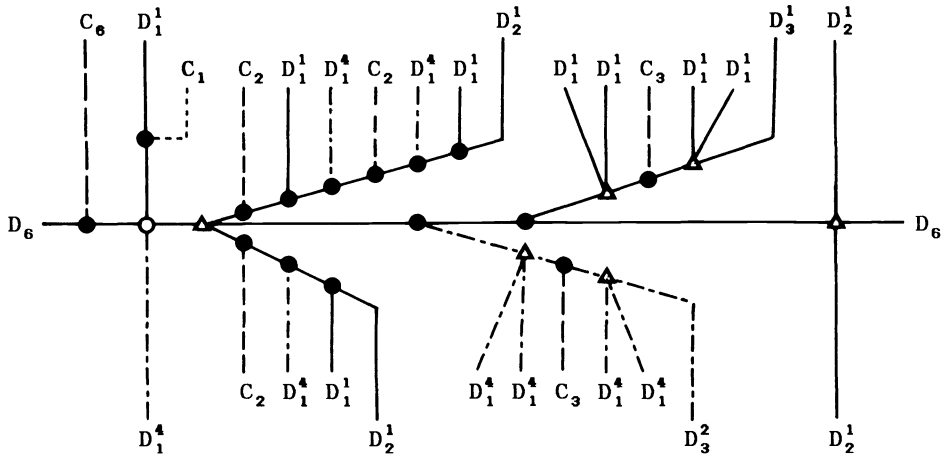


FIG. 4. *Schematic bifurcation diagram of the spherical diamond shell.* ●, *Simple bifurcation point.* ○, *Symmetric double bifurcation point.* △, *Asymmetric double bifurcation point.*

6. Conclusion. In this paper we have proposed a computational method to be used in bifurcation analysis. The idea is to bring the Jacobian matrix into a block-diagonal form by means of a "local" change of coordinates. The "local" transformation is conceptually simple in that it is induced from the underlying geometric symmetry, and is computationally efficient in that the transformation can be done with $O(N^2)$ or $O(N)$ operations and that the transformed Jacobian matrix remains fairly sparse.

In a very recent paper [2], Chen and Sameh demonstrated the computational efficiency of their "SAS decomposition method," which decomposes a matrix based on its symmetry with respect to reflections of coordinate axes. The underlying mathematical principle of the SAS decomposition method can be recognized as the one described in § 3 of this paper applied to D_1 (or its direct product). Note that D_1 has no irreducible representations of degree greater than one, so that no identical blocks appear among the diagonal blocks in (17). In view of the success of the SAS decomposition method, the method of the present paper seems promising from the practical computational point of view.

This paper has been devoted mostly to the description and the analysis of the proposed method. Its application to real-world problems will be reported in a forthcoming paper [12].

One of the referees kindly informed the authors that the recent papers by Dellnitz and Werner [5] and Healey [11] consider other group-theoretic techniques for the phase (ii). The authors thank the referees for their constructive comments.

REFERENCES

[1] A. BOSSAVIT, *Symmetry, groups, and boundary value problems—A progressive introduction to noncommutative harmonic analysis of partial differential equations in domains with geometrical symmetry*, Comput. Methods Appl. Mech. Engrg., 56 (1986), pp. 167–215.

[2] H.-C. CHEN AND A. H. SAMEH, *A matrix decomposition method for orthotropic elasticity problems*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 39–64.

[3] M. CLAUSEN, *Beiträge zum Entwurf schneller Spektraltransformationen*, Ph.D. thesis, Fakultät für Informatik, Universität Karlsruhe, Germany, 1988.

[4] C. W. CURTIS AND I. REINER, *Representation Theory of Finite Groups and Associative Algebras*, Wiley-Interscience, New York, 1962.

[5] M. DELLNITZ AND B. WERNER, *Computational methods for bifurcation problems with symmetries—with special attention to steady state and Hopf bifurcation points*, J. Comput. Appl. Math., 26 (1989), pp. 97–123.

[6] H. FUJII AND M. YAMAGUTI, *Structure of singularities and its numerical realization in nonlinear elasticity*, J. Math. Kyoto Univ., 20 (1980), pp. 489–590.

[7] M. GOLUBITSKY, I. STEWART, AND D. G. SCHAEFFER, *Singularities and Groups in Bifurcation Theory*, Vol. 2, Springer-Verlag, Berlin, New York, 1988.

[8] M. HAMMERMESH, *Group Theory and Its Application to Physical Problems*, Addison-Wesley, Reading, MA, 1962.

[9] T. J. HEALEY, *Symmetry, bifurcation and computational methods in nonlinear structural mechanics*, Ph.D. thesis, University of Illinois, Urbana, IL, 1985.

[10] ———, *A group-theoretic approach to computational bifurcation problems with symmetry*, Comput. Methods Appl. Mech. Engrg., 67 (1988), pp. 257–295.

[11] ———, *Numerical bifurcation with symmetry: diagnosis & computation of singular points*, in Proc. International Conference on Bifurcation Theory and Its Numerical Analysis, L. Kaitai, J. Marsden, M. Golubitsky, and G. Iooss, eds., Xi'an Jiaotong University Press, People's Republic of China, 1989, pp. 218–227.

[12] K. IKEDA AND K. MUROTA, *Bifurcation analysis of symmetric structures using block-diagonalization*, Comput. Methods Appl. Mech. Engrg., to appear.

[13] K. IKEDA, K. MUROTA, AND H. FUJII, *Bifurcation hierarchy of symmetric structures*, Internat. J. Solids and Structures, to appear.

[14] N. JACOBSON, *Basic Algebra* II, W. H. Freeman, San Francisco, 1980.

[15] D. J. KLEIN, C. H. CARLISLE, AND F. A. MATSEN, *Symmetry adaptation to sequences of finite groups*, in Advances in Quantum Chemistry, Vol. 5, P. Löwdin, ed., Academic Press, New York, 1970, pp. 219–260.

[16] W. MILLER, JR., *Symmetry Groups and Their Applications*, Academic Press, New York, 1972.

[17] D. H. SATTINGER, *Group Theoretic Methods in Bifurcation Theory*, Lecture Notes in Mathematics,Vol. 762, Springer-Verlag, Berlin, New York, 1979.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 298-310, March 1991
© 1991 Society for Industrial and Applied Mathematics
005

FAST CELL-STRUCTURED ALGORITHM FOR DIGIT REVERSAL OF ARBITRARY LENGTH*

VÍTĚZSLAV VESELÝ†

Abstract. The method for fast cell-structured bit reversal proposed in [R. N. Bracewell, *The Hartley Transform*, Oxford University Press, New York, 1986] is generalized to arbitrary length N. In addition, the cell size is optional. Four simple FORTRAN77 subprograms implementing the new approach are presented for each type of symmetry requirement on factorization of N (none, partial, or full symmetry, and bit reversal). Full (partial) symmetry allows full (partial) in-place permutation. A comparison shows them to be much faster for large composite lengths (e.g., by about 57 percent for $N=2^{16}$) than other appropriate commonly used algorithms. The time efficiency rapidly increases with the magnitude of N and with the number of factors of N.

Key words. digit-reversal permutation, bit-reversal permutation, fast algorithm, Fast Fourier Transform, Fast Hartley Transform

AMS(MOS) subject classifications. 65F30, 65T05, 15A36

1. Introduction.

A generalization of the method proposed by Bracewell in [1, p. 66] for fast bit reversal is presented. There the $2^m \times 2^m$ bit-reversal permutation matrix is decomposed into 4×4 or 8×8 bit-reversal submatrices (the so-called "cells"). The row-column cell coordinates permute according to the bit reversal of length $\sqrt{2^m/4}$ or $\sqrt{2^m/8}$, respectively. Thus we achieve more than a square root reduction of the original permutation length.

In the present paper a general decomposition theorem for digit reversal of arbitrary composite length $N = N_1 \cdots N_m$ is proved where the cell size may be any integer dividing N. For highly composite lengths N a suitable choice of the cell size (about $\sqrt[3]{N}$) may lead up to a cube root length reduction. Four Fortran77 subprograms based on this result are given in Appendix A for each of the four types of symmetry requirements for factorization of N. Their computational efficiency is compared with that of other appropriate commonly used algorithms listed in Appendix B. The test results are summarized in Table 1 and demonstrate a significant savings in time (about 57 percent for $N=2^{16}$) growing with increasing N and m. This and other similar topics are dealt with in [2], as well.

Throughout the paper we shall use the following notation:

$\delta_{i,j}$: Kronecker's symbol ($\delta_{i,j}=1$ if $i=j$ and zero otherwise).

$N = N_1 N_2 \cdots N_m$, $m \geqq 3$: decomposition of permutation length N, $N_i \geqq 1$.

$N_{i,j} = N_i N_{i+1} \cdots N_j$ if $i \leqq j$ and $N_{i,j}=1$ otherwise.

$\mathcal{N} = (N_1, N_2, \cdots, N_m)$: mixed radix number system.

$\mathcal{N}' = (N_m, \cdots, N_2, N_1)$: reversed mixed radix number system.

$n = [n_1, n_2, \cdots, n_m]_{\mathcal{N}}$: mixed radix integer representation of $n = n_1 N_{2,m} + n_2 N_{3,m} + \cdots + n_{m-1} N_m + n_m$ with respect to the number system $\mathcal{N}$, $0 \leqq n_i < N_i$, $i = 1, 2, \cdots, m$. We shall often omit the subscript $\mathcal{N}$ whenever the associated number system $\mathcal{N}$ is implicitly determined from the context.

$\mathbf{I}_N$: identity matrix of size $N \times N$.

* Received by the editors October 17, 1988; accepted for publication (in revised form) December 21, 1989.

† Institute of Physical Metallurgy, Czechoslovak Academy of Sciences, Žižkova 22, 61662 Brno, Czechoslovakia.

$\mathbf{S}_{\mathcal{N}}$: digit reversal matrix of size $N \times N$ with respect to $\mathcal{N}$ having entries $S_{\mathcal{N}}(n, [k_1, \cdots, k_m]_{\mathcal{N}}) = \delta_{n,[k_m,\cdots,k_1]_{\mathcal{N}}}$. The same symbol $S_{\mathcal{N}}$ will be used for the associated mapping $S_{\mathcal{N}}([n_1, \cdots, n_m]) = [n_m, \cdots, n_1]$ as well. We shall use also a simplified notation $\mathbf{S}_{i,j}$ instead of $\mathbf{S}_{(N_i,\cdots,N_j)}$, $i \leqq j$.

$\otimes$: Kronecker product of matrices.

2. The main decomposition theorem.

DEFINITION 1. Let i, j be arbitrary but fixed integers, $1 < i \leqq j < m$. We denote $\mathcal{L} = (N_1, \cdots, N_{i-1})$, $\mathcal{C} = (N_i, \cdots, N_j)$, and $\mathcal{R} = (N_{j+1}, \cdots, N_m)$ as the number systems associated with $L = N_{1,i-1}$ (left part), $C = N_{i,j}$ (central part), and $R = N_{j+1,m}$ (right part), respectively. Then the number system $\tilde{\mathcal{N}} = (L, C, R)$ is called an LCR-*decomposition of the number system* $\mathcal{N}$.

LEMMA 2. *It holds that* $\mathbf{S}_{\mathcal{N}} = \mathbf{S}_{\tilde{\mathcal{N}}}(\mathbf{S}_{\mathcal{L}} \otimes \mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{R}}) = (\mathbf{S}_{\mathcal{R}} \otimes \mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{L}})\mathbf{S}_{\tilde{\mathcal{N}}}$.

Proof. It is easy to see that

$$\mathbf{S}_{\mathcal{N}} = \mathbf{S}_{1,m} = \mathbf{S}_{(N_{1,i-1},N_{i,m})}(\mathbf{S}_{1,i-1} \otimes \mathbf{S}_{i,m}) = \mathbf{S}_{(L,CR)}(\mathbf{S}_{\mathcal{L}} \otimes \mathbf{S}_{i,m})$$

and

$$\mathbf{S}_{i,m} = \mathbf{S}_{(N_{i,j},N_{j+1,m})}(\mathbf{S}_{i,j} \otimes \mathbf{S}_{j+1,m}) = \mathbf{S}_{(C,R)}(\mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{R}}).$$

Together

$$\begin{aligned}\mathbf{S}_{\mathcal{N}} &= \mathbf{S}_{(L,CR)}[\mathbf{S}_{\mathcal{L}} \otimes \mathbf{S}_{(C,R)}(\mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{R}})] \\ &= \mathbf{S}_{(L,CR)}(\mathbf{I}_L \otimes \mathbf{S}_{(C,R)})(\mathbf{S}_{\mathcal{L}} \otimes \mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{R}}) = \mathbf{S}_{(L,C,R)}(\mathbf{S}_{\mathcal{L}} \otimes \mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{R}})\end{aligned}$$

because $\mathbf{S}_{(L)} = \mathbf{I}_L$.

It also holds that $\mathbf{S}_{\tilde{\mathcal{N}}}(\mathbf{S}_{\mathcal{L}} \otimes \mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{R}})\mathbf{S}_{\tilde{\mathcal{N}}}^{T} = (\mathbf{S}_{\mathcal{R}} \otimes \mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{L}})$ by a well-known property of the Kronecker product, which gives the latter identity $\mathbf{S}_{\mathcal{N}} = \mathbf{S}_{\tilde{\mathcal{N}}}(\mathbf{S}_{\mathcal{L}} \otimes \mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{R}})\mathbf{S}_{\tilde{\mathcal{N}}}^{T}\mathbf{S}_{\tilde{\mathcal{N}}} = (\mathbf{S}_{\mathcal{R}} \otimes \mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{L}})\mathbf{S}_{\tilde{\mathcal{N}}}$. □

MAIN THEOREM 3. *The digit reversal matrix* $\mathbf{S}_{\mathcal{N}}$ *is structured into* $R \times L$ *blocks* $\mathbf{S}_{\mathcal{N}}^{(r,l)} = \mathbf{S}_{\mathcal{C}} \otimes \mathbf{U}^{(l',r')}$ *of size* $CL \times CR$ *where* $l' = S_{\mathcal{L}}(l)$, $r' = S_{\mathcal{R}}^{-1}(r)$, $0 \leqq l < L$, $0 \leqq r < R$ *and* $\mathbf{U}^{(l',r')}$ *denotes the matrix of size* $L \times R$ *having unit entry in the* l'*th row and* r'*th column and zeros elsewhere, i.e.,* $\mathbf{U}^{(l',r')}(n, k) = \delta_{n,l'}\delta_{r',k}$.

Proof. By Lemma 2, $\mathbf{S}_{\mathcal{N}} = \mathbf{S}_{(L,C,R)}\tilde{\mathbf{S}}$, where $\tilde{\mathbf{S}} = \mathbf{S}_{\mathcal{L}} \otimes \mathbf{S}_{\mathcal{C}} \otimes \mathbf{S}_{\mathcal{R}}$. Hence we have for any $0 \leqq l, n_1 < L$, $0 \leqq n_2, k_2 < C$, and $0 \leqq r, k_3 < R$:

$$\begin{aligned}S_{\mathcal{N}}^{(r,l)}([n_2, n_1], [k_2, k_3]) &= S_{\mathcal{N}}([r, n_2, n_1], [l, k_2, k_3]) \\ &= \tilde{S}([n_1, n_2, r], [l, k_2, k_3]) \\ &= S_{\mathcal{L}}(n_1, l)S_{\mathcal{C}}(n_2, k_2)S_{\mathcal{R}}(r, k_3) \\ &= S_{\mathcal{C}}(n_2, k_2)\delta_{n_1,S_{\mathcal{L}}(1)}\delta_{r,S_{\mathcal{R}}(k_3)} \\ &= S_{\mathcal{C}}(n_2, k_2)\delta_{n_1,S_{\mathcal{L}}(1)}\delta_{S_{\mathcal{R}}^{-1}(r),k_3} \\ &= S_{\mathcal{C}}(n_2, k_2)U^{(l',r')}(n_1, k_3) \\ &= (S_{\mathcal{C}} \otimes U^{(l',r')})([n_2, n_1], [k_2, k_3]).\end{aligned}$$ □

Theorem 3 reveals a cell structure in the matrix $\mathbf{S}_{\mathcal{N}}$ which may be characterized as follows:

Cell pattern = permutation matrix $\mathbf{S}_{\mathcal{C}}$, which is a digit reversal with respect to the number system $\mathcal{C} = (N_i, \cdots, N_j)$.

lth cell in the rth row $= \mathbf{S}_{\mathscr{C}} \otimes \mathbf{U}^{(l',r')}$ is the cell pattern shifted and expanded in both directions. The Kronecker product makes the cell pattern expand L times vertically and R times horizontally in view of the $L \times R$ size of the matrix $\mathbf{U}^{(l',r')}$ and shift by $l' = S_{\mathscr{L}}(l)$ downwards and by $r' = S_{\mathscr{R}}^{-1}(r)$ to the right in accordance with the (l', r')th unit entry of $\mathbf{U}^{(l',r')}$. Thus the shift fluctuations are determined by the cell coordinates via the digit reversal permutations $S_{\mathscr{L}}$ and $S_{\mathscr{R}}^{-1}$.

Atoms per cell $= C = N_{i,j}$.

Number of cells $= RL = N/C$.

Vertical grid spacing in cell = number of cells per row
$= L = N_{1,i-1}$.

Horizontal grid spacing in cell = number of cells per column
$= R = N_{j+1,m}$.

Cell size $= CL \times CR = N/R \times N/L$.

Horizontal coordinate of the upper left element of the lth cell in the rth row:

$$k(l, r) = Nl/L + S_{\mathscr{R}}^{-1}(r). \tag{1}$$

Vertical coordinate of the upper left element of the lth cell in the rth row:

$$n(r, l) = Nr/R + S_{\mathscr{L}}(l). \tag{2}$$

3. Fast algorithm. The steps of the fast cell-structured digit reversal algorithm for $\mathbf{y} = \mathbf{S}_{\mathscr{N}}\mathbf{x}$, $\mathbf{x} = (x_0, x_1, \cdots, x_{N-1})^T$ and $\mathbf{y} = (y_0, y_1, \cdots, y_{N-1})^T$ are:

1) Evaluation of short permutations $S_{\mathscr{C}}$, $S_{\mathscr{L}}$, and $S_{\mathscr{R}}^{-1}$.

2) Accomplishing the elementary cell transform for each r, l by
   — computing coordinates $k(l, r)$ and $n(r, l)$ of the lth cell in the rth row by (1) and (2), and
   — evaluating the elementary transform

$$\mathbf{y}^{(r,l)} = \mathbf{S}_{\mathscr{C}}\mathbf{x}^{(l,r)} \tag{3}$$

where

$$\mathbf{y}^{(r,l)} = (y_0^{(r,l)}, \cdots, y_{C-1}^{(r,l)})^T, \qquad \mathbf{x}^{(l,r)} = (x_0^{(l,r)}, \cdots, x_{C-1}^{(l,r)})^T$$

and

$$y_q^{(r,l)} = y_{n(r,l)+qL}, \quad x_q^{(l,r)} = x_{k(l,r)+qR} \quad \text{for each } q, \qquad 0 \leqq q < C.$$

The computational time spent by step 1) is small in comparison with $S_{\mathscr{N}}$. In most cases $S_{\mathscr{C}}$, $S_{\mathscr{L}}$, and $S_{\mathscr{R}}^{-1}$ may be precomputed and stored in memory for several short lengths only once in advance to accommodate a sufficiently wide range of large N. Alternatively we can design a recursive procedure that reduces the length of the permutations $S_{\mathscr{C}}$, $S_{\mathscr{L}}$, and $S_{\mathscr{R}}^{-1}$ applying steps 1) and 2) repeatedly. Very soon either a stored permutation will be obtained or the length will equal some N_i, whereupon the recursion stops.

The algorithm does not allow in-place computation in general unless there is a certain additional symmetry in $\mathscr{N}$. Let us note that $S_{\mathscr{R}}^{-1}(r) = S_{\mathscr{R}'}(r)$. Consequently, the partial symmetry of $\mathscr{N}$ given by $\mathscr{R}' = \mathscr{L}$ results in $S_{\mathscr{R}}^{-1} = S_{\mathscr{L}}$ and in LCL-decomposition of $\mathscr{N}$. Then (1) and (2) attain the form

$$k(l, r) = Nl/L + S_{\mathscr{L}}(r) \tag{1'}$$

and

$$n(r, l) = Nr/L + S_{\mathscr{L}}(l), \tag{2'}$$

which says that $k(r, l) = n(r, l)$ and therefore, in (3), $\mathbf{x}^{(l,r)}$ may be transformed directly to $\mathbf{x}^{(r,l)}$ and $\mathbf{x}^{(r,l)}$ directly back to $\mathbf{x}^{(l,r)}$ for each l, r, $l \leqq r$ provided that there is a buffer of length C where the original values of $\mathbf{x}^{(r,l)}$ must be saved prior to being overwritten by the transformed values $\mathbf{x}^{(r,l)} = \mathbf{S}_{\mathscr{C}}\mathbf{x}^{(l,r)}$. Clearly we can dispense with the buffer provided that there is an additional symmetry $\mathscr{C} = \mathscr{C}'$ ($\mathscr{R}' = \mathscr{L}'$ gives, together with $\mathscr{C} = \mathscr{C}'$, a full symmetry $\mathscr{N} = \mathscr{N}'$) because in this case $S_{\mathscr{C}}^{-1}(q) = S_{\mathscr{C}'}(q) = S_{\mathscr{C}}(q)$ allows a simple interchange of values $x_q^{(l,r)}$ and $x_{S_{\mathscr{C}}(q)}^{(r,l)}$ for each q.

The power of the fast cell-structured digit reversal becomes apparent for $N = p^m$, $p \geqq 2$. Let $m = 3m' + m''$, $1 \leqq m'$, $0 \leqq m'' \leqq 2$. We shall make a *balanced* LCL-decomposition of $\mathscr{N} = (p, p, \cdots, p)$ in dependence on m'' as follows:

— $L = C = p^{m'}$ for $m'' = 0$,

— $L = p^{m'}$, $C = p^{m'+1}$ for $m'' = 1$,

— $L = p^{m'+1}$, $C = p^{m'}$ for $m'' = 2$.

A cell-structured algorithm working for all $N = p^m \leqq N_{\max} = p^{3\mu}$ reduces $S_{\mathscr{N}}$ to $S_{\mathscr{C}}$ and $S_{\mathscr{L}} = S_{\mathscr{L}}^{-1}$ with lengths not exceeding $\sqrt[3]{N_{\max}}$. Moreover, this is an in-place algorithm due to the full symmetry of $\mathscr{N} = (p, p, \cdots, p)$. This dramatic length reduction allows us to avoid step 1) by keeping all permutations $S_{\mathscr{K}}$ of the lengths $K = p^2, p^3, \cdots, p^\mu$ in the computer memory. This can be best illustrated by the bit reversal (see CBRR in Appendix A) where only six saved (array BRD) permutations of lengths 2, 4, 8, 16, 32, and 64 (=126 memory locations) allow extremely fast bit reversal up to the length $2^{18} = 262{,}144$.

4. Comparison of speed. Four Fortran77 subprograms CDRR, CDRRPS, CDRRFS, and CBRR are listed in Appendix A. These implement the cell-structured digit reversal (CDR) of a real sequence stored in input array X with increment INC. The LCR-decomposition is specified by the input arrays LP, CP, and RP, which define the permutations $S_{\mathscr{L}}$, $S_{\mathscr{C}}$, and $S_{\mathscr{R}}$, respectively, in a way indicated by the comment lines. CDRR accomplishes CDR for the general case where no symmetry requirements are posed on $\mathscr{N}$. CDRRPS and CDRRFS assume partial symmetry (PS) and full symmetry (FS) of $\mathscr{N}$, respectively. CBRR performs bit reversal (FS with $N_i = 2$ for $i = 1, 2, \cdots, m$) up to the length $N = 2^{18}$ and is fully self-contained because none of the reduced permutations LP, CP, or RP is needed. The permuted sequence is stored in an output array XP with the increment INC as well. Except for CDRR an in-place permutation (X = XP) is allowed. Note that X and XP must not be overlaid with a nonzero shift which is an integer multiple of INC.

To properly compare these subprograms, three other commonly used algorithms were chosen for examination. The subprograms ODRR, ODRRPS, and ODRRFS from Appendix B perform a function analogous to that of CDRR, CDRRPS, and CDRRFS, respectively, but they permute the data in a standard (ordinary) way (ODR = Ordinary Digit Reversal). ODRRPS is the Fortran code extracted from the well-known Singleton general purpose Fast Fourier Transform (FFT) program [3].

Table 1 shows the execution times in milliseconds for all these seven subprograms and for a wide range of lengths up to 100,000. The times come from runs on an ICL 2950/10 computer (the VME operating system) with 2 MB of memory and 300,000 instr./sec. and floating-point hardware, at the Institute of Physical Metallurgy in Brno, Czechoslovakia, using the ICLFORTRAN77 compiler (without optimization). The overhead times spent by calculations of LP, CP, and RP permutations are indicated in parentheses. For this purpose a little more efficient version of ODRR was used with the number of nested loops reduced to seven instead of the original 18 (seven is the maximal number of factors in $\mathscr{L}$, $\mathscr{C}$, or $\mathscr{R}$).

TABLE 1

Time in milliseconds to calculate a length N digit reversal by the seven algorithms tested.

Length				Time [msec]						
	Decomposition			No symmetry		Partial symmetry		Full symmetry		
N	L	C	R	ODRR	CDRR	ODRRPS	CDRRPS	ODRRFS	CDRRFS	CBRR
16	2	2×2	2	5	3 (+8)	7	4 (+6)	5	3 (+6)	3
32	2×2	2	2×2	6	5 (+7)	6	6 (+6)	7	4 (+5)	5
	2	2×2×2	2	6	4 (+8)	11	6 (+6)	7	4 (+6)	-
63	3	7	3	8	6 (+7)	8	10 (+5)	9	7 (+5)	-
64	2×2	2×2	2×2	10	7 (+8)	18	10 (+5)	11	7 (+5)	8
72	2×3	2	3×2	10	9 (+8)	12	11 (+5)	12	7 (+5)	-
120	2×2	2×3	5	11	11 (+8)	-	-	-	-	-
	2	2×3×5	2	14	10 (+10)	38	17 (+8)	-	-	-
128	2×2	2×2×2	2×2	17	11 (+8)	32	16 (+6)	19	11 (+6)	12
130	2	13	5	11	11 (+10)	-	-	-	-	-
252	2×3	7	3×2	27	21 (+8)	37	29 (+6)	31	19 (+6)	-
255	3	17	5	20	20 (+9)	-	-	-	-	-
256	2×2×2	2×2	2×2×2	31	24 (+9)	62	30 (+7)	36	19 (+6)	21
504	2×3	2×7	3×2	51	39 (+9)	116	55 (+7)	-	-	-
507	13	3	13	34	49 (+9)	45	59 (+6)	41	36 (+6)	-
512	2×2×2	2×2×2	2×2×2	59	42 (+10)	116	55 (+7)	68	34 (+7)	35
1,024	2×2×2	2×2×2×2	2×2×2	116	77 (+10)	262	105 (+8)	132	64 (+8)	66
	2×2×2×2	2×2	2×2×2×2	116	92 (+11)	227	109 (+7)	134	68 (+7)	-
1,025	5	41	5	71	73 (+11)	100	109 (+9)	89	70 (+9)	-
1,026	2×3	19	3×3	84	76 (+10)	-	-	-	-	-
	3	2×19×3	3	85	72 (+19)	255	119 (+17)	-	-	-
2,048	2×2×2×2	2×2×2	2×2×2×2	230	162 (+12)	452	205 (+8)	264	126 (+8)	129
2,052	2×2×3	19	3×3	166	151 (+10)	-	-	-	-	-
	2×3	3×19	3×2	195	145 (+13)	508	208 (+10)	-	-	-
2,057	11	17	11	125	152 (+10)	179	202 (+7)	162	126 (+7)	-
4,089	3	47	29	237	294 (+13)	-	-	-	-	-
4,096	2×2×2×2	2×2×2×2	2×2×2×2	473	306 (+13)	1,029	397 (+9)	524	239 (+9)	244
	2×2×2×2×2	2×2	2×2×2×2×2	462	371 (+16)	924	427 (+9)	525	258 (+10)	-

TABLE 1—continued

4,104	2×2×3	19	3×3×2	400	306 (+12)	–	–	–	–	–
	2×3	2×19×3	3×2	403	285 (+20)	986	418 (+17)	–	–	–
8,192	2×2×2×2×2	2×2×2×2×2×2	2×2×2×2×2	1,048	587 (+15)	2,114	777 (+11)	1,094	471 (+11)	473
	2×2×2×2×2×2	2×2×2×2	2×2×2×2×2×2	1,053	657 (+16)	1,884	805 (+10)	1,090	493 (+10)	–
8,195	5	149	11	489	568 (+21)	–	–	–	–	–
8,208	2×2×2×3	19	3×3×2	865	616 (+14)	–	–	–	–	–
	2×2×3	3×19	3×2×2	876	583 (+14)	2,018	772 (+11)	–	–	–
16,380	3×7	2×13	5×3×2	1,846	1,207 (+15)	–	–	–	–	–
	2×3	5×13×7	3×2	1,853	1,162 (+54)	3,875	1,665 (+50)	–	–	–
16,384	2×2×2×2×2×2	2×2×2×2×2	2×2×2×2×2×2	2,130	1,225 (+18)	4,243	1,535 (+11)	2,197	938 (+12)	944
	2×2×2×2×2×2×2	2×2	2×2×2×2×2×2×2	2,108	1,499 (+25)	3,778	1,699 (+14)	2,193	1,053 (+14)	–
16,399	23	31	23	1,211	1,197 (+13)	1,449	1,511 (+9)	1,258	955 (+9)	–
32,759	17	47	41	2,389	2,383 (+16)	–	–	–	–	–
32,760	2×2×7	3×13	5×3×2	3,654	2,409 (+17)	–	–	–	–	–
	2×3	2×5×13×7	3×2	3,682	2,481 (+97)	7,906	3,548 (+94)	–	–	–
32,768	2×2×2×2×2	2×2×2×2×2	2×2×2×2×2	4,233	2,369 (+20)	8,456	2,978 (+14)	4,426	1,869 (+13)	1,907
	2×2×2×2×2×2	2×2×2	2×2×2×2×2×2	4,231	2,674 (+25)	7,514	3,165 (+15)	4,399	1,976 (+14)	–
65,520	2×3×7	3×13	5×2×2×2	8,095	4,942 (+20)	–	–	–	–	–
	2×2×3	5×13×7	3×2×2	7,979	5,233 (+55)	15,610	6,877 (+50)	–	–	–
65,533	71	13	71	4,711	5,201 (+20)	5,870	6,229 (+12)	5,195	3,876 (+12)	–
65,536	2×2×2×2×2	2×2×2×2×2×2	2×2×2×2×2	8,427	4,789 (+25)	17,856	6,025 (+20)	8,810	3,771 (+18)	3,755
	2×2×2×2×2×2×2	2×2	2×2×2×2×2×2×2	8,440	6,300 (+43)	15,090	6,800 (+23)	8,828	4,345 (+23)	–
93,312	2×2×3×3	2×3×2×3×2	3×3×2×2	11,545	7,050 (+26)	23,948	8,748 (+19)	11,891	5,463 (+19)	–
99,979	11	149	61	7,108	7,326 (+27)	–	–	–	–	–
100,000	2×5×5	2×2×5×2	5×5×2	10,982	7,693 (+23)	23,253	9,315 (+15)	–	–	–

As reference lengths powers of two were chosen. Where appropriate, two LCR-decompositions were selected for powers of two, one being balanced, the other using 4×4 or 8×8 cells in accordance with [1]. To each reference length two other closest values are added which have different factors N_i ($i=1,2,\cdots,m$), one with small m, the other with greater m. For the latter two, LCR-decompositions were applied, one preferring balance (the interval comprising L, C, and R should be as small as possible), the other, symmetry (partial or full), and both having the same set of factors but not necessarily in the same order.

Inspecting Table 1 in more detail we can draw the following conclusions about the time efficiency of CDR (as far as not stated otherwise the percentage values relate to the balanced LCR-decomposition of $N=2^{16}$ which exhibits best effectiveness—underlined in Table 1):

—CDRRFS is always faster than ODRRFS; the best rate is achieved with $N_i=2$ for $i=1,2,\cdots,m$ (time savings of 57 percent). The times for CBRR do not significantly differ from those for CDRRFS. In general, CDRRFS and CBRR are the fastest of all subprograms (by 38 percent for CDRRPS and by 21 percent for CDRR) for any N and m.

—CDRRPS is usually (except for very small m) faster than ODRRPS (by 66 percent) but slower than CDRR (by 25.5 percent). ODRRPS exhibits similar behaviour in comparison with ODRR where the time drawback is much more significant (by 110 percent). Thus the cost of buffered in-place calculation is generally greater for Singleton's ODRRPS than for our CDRRPS.

—CDRR is usually (except for very small m) faster than ODRR (by 43 percent).

—Within each type of symmetry the effectiveness of CDR versus ODR grows with increasing m. This is mostly apparent in the close lengths $N_1=65{,}536$ ($m=16$) and $N_2=65{,}533$ ($m=3$) for which CDRR is 43 percent faster for N_1 but 11 percent slower for N_2 compared to ODRR; and CDRRPS is 66 percent faster for N_1 but 6 percent slower for N_2 compared to ODRRPS. Only CDRRFS is faster than ODRRFS in both cases, for N_1 by 57 percent and for N_2 only by 25 percent. Hence we conclude that CDR is recommended especially for well-factorizable lengths. The effectiveness grows also with increasing N as indicated by the execution times for power-of-two lengths.

—The best LCR-decomposition seems to be the balanced one. This is best evidenced with the power-of-two lengths where balanced decomposition is somewhat faster than that of Bracewell ($C=4$ or $C=8$)—by 24 percent for CDRRPS, and 13 percent for CDRRFS. If time, and not memory, is important, then balance seems to be a little bit more favourable than symmetry for most other power-of-two lengths as well.

As the measured execution times might be strongly distorted due to disk operations of the virtual computer memory accomplishing the actual data transfers, no other parallel job was allowed to run with timing tests of Table 1. To make sure, another test run was made where the data transfers were artificially suppressed by modifying the Fortran code. Although different (shorter) times were measured, the conclusions were the same.

5. Final conclusions. A simple theory has been developed allowing calculation of digit reversal of length N factorizable into at least three factors via digit reversals of much shorter lengths L, C, and R such that $N=\mathrm{LCR}$. For each of the four symmetry requirements on factorization of N (none, partial, or full symmetry, and bit reversal) a new algorithm based on that theory, called cell-structured digit reversal (CDR), has

been presented in the form of Fortran77 subroutines which, in general, were shown to be much faster than other algorithms of appropriate type using the standard approach. The short Fortran code and a clear indexing scheme make the new subroutines very simple to understand and to relate to the theory.

For a given length N the shortest execution times are achieved with the longest (prime) factorization and with balanced LCR-decomposition where L, C, and R are as close as possible, which results in reduced permutation length equal to $\sqrt[3]{N}$. Highly composite lengths N with small fluctuations of factor magnitudes should be preferred. From this point of view powers of two are the best possible lengths, e.g., for $N = 2^{16}$, CDR is 57 percent faster than the commonly used standard approach.

The in-place calculation is possible only if there is a symmetry such that the factorization of L is equal to the reversed factorization of R. In the case of partial symmetry (no symmetrical factorization was used for C) an additional working storage of length C is needed.

Appendix A.

```
      SUBROUTINE CDRR(X,XP,NINC,INC,LP,L,CP,C,RP,R)
C
C  !!USE F77 COMPILER!!
C  CELL-STRUCTURED DIGIT REVERSAL PERMUTATION FOR
C  ARBITRARY LENGTH AND REAL DATA BY THE METHOD OF
C  LCR-DECOMPOSITION
C
C  X (NINC) ...INPUT DATA
C  XP(NINC) ...PERMUTED DATA (MUST BE X<>XP)
C  NINC     ...  = N*INC
C  N        ...NUMBER OF DATA
C  INC      ...INCREMENT
C  LP(L)    ...LEFT PERMUTATION l:
C              LP(l(I)+1)=I*C*R*INC, I=0,1,...,L-1
C  CP(C)    ...CENTRAL PERMUTATION c:
C              CP(c(I)+1)=I*R*INC+1, I=0,1,...,C-1
C  RP(R)    ...RIGHT PERMUTATION r:
C              RP(r(I)+1)=I*INC, I=0,1,...,R-1
C
C
      INTEGER LP(L),C,CP(C),R,RP(R)
      REAL X(NINC),XP(NINC)
C
      LINC =L*INC
      LCINC=LINC*C
      IRINC=1
      DO 30 IR=1,R
        KPINC=IRINC
        DO 20 IL=1,L
          KINC=RP(IR)+LP(IL)
          KP=KPINC
          DO 10 IC=1,C
            XP(KP)=X(KINC+CP(IC))
   10       KP=KP+LINC
   20     KPINC=KPINC+INC
   30   IRINC=IRINC+LCINC
      RETURN
      END

      SUBROUTINE CDRRPS(X,XP,NINC,INC,CP,C,RP,R,WS)
C
C  !!USE F77 COMPILER!!
C  PARTIALLY SYMMETRICAL CELL-STRUCTURED DIGIT
C  REVERSAL PERMUTATION FOR ARBITRARY LENGTH AND
C  REAL DATA BY THE METHOD OF LCR-DECOMPOSITION
C
C  X (NINC) ...INPUT DATA
C  XP(NINC) ...PERMUTED DATA (MAY BE X=XP)
C  NINC     ...  = N*INC
C  N        ...NUMBER OF DATA
C  INC      ...INCREMENT
C  CP(C)    ...CENTRAL PERMUTATION c:
C              CP(c(I)+1)=I*R*INC  ,I=0,1,...,C-1
C  RP(R)    ...LEFT/RIGHT PERMUTATION r:
C              RP(r(I)+1)=I*INC+1,I=0,1,...,R-1
C  WS(C)    ...WORKING STORAGE ARRAY
C
```

```
C
      INTEGER C,CP(C),R,RP(R),RINC,CRINC
      REAL X(NINC),XP(NINC),WS(C)
C
      RINC =R*INC
      CRINC=C*RINC
      IRINC=0
      DO 30 IR=1,R
        KINC=RP(IR)
        DO 20 IL=1,IR
          KPINC=IRINC+RP(IL)
          DO 5 IC=1,C
 5          WS(IC)=X(KPINC+CP(IC))
          KP=KPINC
          DO 10 IC=1,C
            XP(KP)=X(KINC+CP(IC))
10          KP=KP+RINC
          K=KINC
          DO 15 IC=1,C
            XP(K)=WS(IC)
15          K=K+RINC
20        KINC=KINC+CRINC
30    IRINC=IRINC+CRINC
      RETURN
      END

      SUBROUTINE CDRRFS(X,XP,NINC,INC,CP,C,RP,R)
C
C   !!USE F77 COMPILER!!
C   FULLY SYMMETRICAL CELL-STRUCTURED DIGIT REVERSAL
C   PERMUTATION FOR ARBITRARY LENGTH AND REAL DATA
C   BY THE METHOD OF LCR-DECOMPOSITION
C
C   X (NINC) ...INPUT DATA
C   XP(NINC) ...PERMUTED DATA (MAY BE X=XP)
C   NINC     ... = N*INC
C   N        ...NUMBER OF DATA
C   INC      ...INCREMENT
C   CP(C)    ...CENTRAL PERMUTATION c:
C               CP(c(I)+1)=I*R*INC  ,I=0,1,...,C-1
C   RP(R)    ...LEFT/RIGHT PERMUTATION r:
C               RP(r(I)+1)=I*INC+1,I=0,1,...,R-1
C
C
      INTEGER C,CP(C),R,RP(R),RINC,CRINC
      REAL X(NINC),XP(NINC)
C
      RINC =R*INC
      CRINC=C*RINC
      IRINC=0
      DO 40 IR=1,R
        KINC=RP(IR)
        DO 20 IL=1,IR-1
          KPINC=IRINC+RP(IL)
          KP=KPINC
          DO 10 IC=1,C
            K=KINC+CP(IC)
            WS=X(KP)
            XP(KP)=X(K)
            XP(K)=WS
10          KP=KP+RINC
20        KINC=KINC+CRINC
        KP=KINC
        DO 30 IC=1,C
          K=KINC+CP(IC)
          IF(KP-K)30,25,25
25        WS=X(KP)
          XP(KP)=X(K)
          XP(K)=WS
30        KP=KP+RINC
40    IRINC=IRINC+CRINC
      RETURN
      END

      SUBROUTINE CBRR(X,XP,NINC,INC,M)
C
C   !!USE F77 COMPILER!!
C   CELL-STRUCTURED BIT REVERSAL PERMUTATION FOR
C   REAL DATA BY THE METHOD OF LCR-DECOMPOSITION
C
C   X (NINC)    ...INPUT DATA
C   XP(NINC)    ...PERMUTED DATA (MAY BE X=XP)
C   NINC        ... = N*INC
C   N           ... = 2**M = NUMBER OF DATA, M < 19
C   INC         ...INCREMENT
C
```

```
C
      INTEGER CP(64),RP(64),CR(2),C,R,BRD(0:126),
     *        RINC
      REAL X(NINC),XP(NINC)
      EQUIVALENCE (C,CR(1)),(R,CR(2))
      DATA BRD/ 1,  1,  2,  1,  3,  2,  4,  1,  5,  3,  7,  2,
     *          6,  4,  8,  1,  9,  5, 13,  3, 11,  7, 15,  2,
     *         10,  6, 14,  4, 12,  8, 16,  1, 17,  9, 25,  5,
     *         21, 13, 29,  3, 19, 11, 27,  7, 23, 15, 31,  2,
     *         18, 10, 26,  6, 22, 14, 30,  4, 20, 12, 28,  8,
     *         24, 16, 32,  1, 33, 17, 49,  9, 41, 25, 57,  5,
     *         37, 21, 53, 13, 45, 29, 61,  3, 35, 19, 51, 11,
     *         43, 27, 59,  7, 39, 23, 55, 15, 47, 31, 63,  2,
     *         34, 18, 50, 10, 42, 26, 58,  6, 38, 22, 54, 14,
     *         46, 30, 62,  4, 36, 20, 52, 12, 44, 28, 60,  8,
     *         40, 24, 56, 16, 48, 32, 64/
C
      IF(M.GT.6)GOTO 20
      INC1=INC-1
      J1=NINC/INC-1
      J2=J1+J1
      IINC=1
      DO 10 J=J1,J2
        IINCP=INC*BRD(J)-INC1
        IF(IINCP-IINC)10,5,5
    5   WS=X(IINCP)
        XP(IINCP)=X(IINC)
        XP(IINC)=WS
   10   IINC=IINC+INC
        RETURN
C
   20 M3=M/3
      MR=MOD(M,3)
      C=2**M3
      R=C
      IF(MR.GT.0)CR(MR)=CR(MR)+CR(MR)
      RINC=R*INC
      JCINC=0
      J1=C-1
      J2=J1+J1
      IF(INC.GT.1)GOTO 40
C
      DO 30 J=J1,J2
        CP(BRD(J))=JCINC
   30   JCINC=JCINC+RINC
      CALL CDRRFS(X,XP,NINC,INC,CP,C,BRD(R-1),R)
      RETURN
C
   40 JRINC=1
      IF(C.NE.R)GOTO 60
      DO 50 J=J1,J2
        CP(BRD(J))=JCINC
        RP(BRD(J))=JRINC
        JCINC=JCINC+RINC
   50   JRINC=JRINC+INC
      GOTO 90
C
   60 DO 70 J=J1,J2
        CP(BRD(J))=JCINC
   70   JCINC=JCINC+RINC
      J1=R-1
      J2=J1+J1
      DO 80 J=J1,J2
        RP(BRD(J))=JRINC
   80   JRINC=JRINC+INC
C
   90 CALL CDRRFS(X,XP,NINC,INC,CP,C,RP,R)
      RETURN
      END
```

Appendix B.

```
      SUBROUTINE ODRR(X,XP,NINC,INC,NFAC,M)
C
C   !!USE F77 COMPILER!!
C   ORDINARY DIGIT REVERSAL PERMUTATION FOR
C   ARBITRARY LENGTH AND REAL DATA
C
C   X (NINC)     ...INPUT DATA
C   XP(NINC)     ...PERMUTED DATA (MUST BE X<>XP)
C   NINC         ...  = N*INC
C   N            ...NUMBER OF DATA
C   INC          ...INCREMENT
C   NFAC(M)      ...FACTORS OF N DETERMINING THE
C                   MIXED RADIX NUMBER SYSTEM OF
```

```
C                THE INPUT VECTOR X, M < 19
C                NFAC(1) ...MOST SIGNIFICANT DIGIT
C                   .
C                   .
C                NFAC(M) ...LEAST SIGNIFICANT DIGIT
C
      PARAMETER (MMAX=18)
      DIMENSION L(0:MMAX),X(NINC),XP(NINC),NFAC(M)
      KL=MMAX-M
      IF(KL)60,30,10
   10 DO 20 K=1,KL
   20   L(K)=INC
   30 L(0)=INC
      DO 40 K=KL+1,MMAX
   40   L(K)=NFAC(K-KL)*L(K-1)
      K=1
      DO  50  K1  =  1  ,L(1) , L(0)
      DO  50  K2  = K1  ,L(2) , L(1)
      DO  50  K3  = K2  ,L(3) , L(2)
      DO  50  K4  = K3  ,L(4) , L(3)
      DO  50  K5  = K4  ,L(5) , L(4)
      DO  50  K6  = K5  ,L(6) , L(5)
      DO  50  K7  = K6  ,L(7) , L(6)
      DO  50  K8  = K7  ,L(8) , L(7)
      DO  50  K9  = K8  ,L(9) , L(8)
      DO  50  K10 = K9  ,L(10), L(9)
      DO  50  K11 = K10 ,L(11), L(10)
      DO  50  K12 = K11 ,L(12), L(11)
      DO  50  K13 = K12 ,L(13), L(12)
      DO  50  K14 = K13 ,L(14), L(13)
      DO  50  K15 = K14 ,L(15), L(14)
      DO  50  K16 = K15 ,L(16), L(15)
      DO  50  K17 = K16 ,L(17), L(16)
      DO  50  KP  = K17 ,L(18), L(17)
        XP(KP)=X(K)
   50   K=K+INC
   60 RETURN
      END

      SUBROUTINE ODRRPS(X,XP,NINC,INC,NFAC,M,KT,NP,
     *                  MAXP)
C
C  !!USE F77 COMPILER!!
C  PARTIALLY SYMMETRICAL DIGIT REVERSAL PERMUTATION
C  FOR ARBITRARY LENGTH AND REAL DATA BY A METHOD
C  USED IN SINGLETON'S FFT SUBROUTINE
C
C  X (NINC) ...INPUT DATA
C  XP(NINC) ...PERMUTED DATA (MAY BE X=XP)
C  NINC     ...  = N*INC
C  N        ...NUMBER OF DATA
C  INC      ...INCREMENT
C  NFAC(M)  ...FACTORS OF N DETERMINING
C              THE MIXED RADIX NUMBER
C              SYSTEM OF THE INPUT VECOR X
C              NFAC(1) ...MOST SIGNIFICANT DIGIT
C                 .
C                 .
C              NFAC(M) ...LEAST SIGNIFICANT DIGIT
C  KT       ...NUMBER OF SQUARE FACTORS
C  NP(MAXP) ...WORKING ARRAY
C  MAXP     ...>= PRODUCT OF SQUARE-FREE FACTORS
C                 MINUS ONE
C               > M
C
      PARAMETER (MMAX=18,MAXF=23)
      INTEGER NFAC(M),NNFAC(MMAX),NP(MAXP)
      REAL X(NINC),XP(NINC),XT(MAXF)
C
C     PERMUTE THE RESULTS TO NORMAL ORDER---DONE
C     IN TWO STAGES
C     PERMUTATION FOR SQUARE FACTORS OF N
      KS=NINC
      JC=INC
      NP(1)=KS
      IF (KT.EQ.0) GO TO 450
      K=KT+KT+1
      IF (M.LT.K) K=K-1
      J=1
      NP(K+1)=JC
  370 NP(J+1)=NP(J)/NFAC(J)
      NP(K)=NP(K+1)*NFAC(J)
      J=J+1
      K=K-1
      IF (J.LT.K) GO TO 370
      K3=NP(K+1)
      KSPAN=NP(2)
```

```
      KK=JC+1
      K2=KSPAN+1
      J=1
  380 AK=X(KK)
      XP(KK)=X(K2)
      XP(K2)=AK
      KK=KK+INC
      K2=KSPAN+K2
      IF (K2.LT.KS) GO TO 380
  390 K2=K2-NP(J)
      J=J+1
      K2=NP(J+1)+K2
      IF (K2.GT.NP(J)) GO TO 390
      J=1
  400 IF (KK.LT.K2) GO TO 380
      KK=KK+INC
      K2=KSPAN+K2
      IF (K2.LT.KS) GO TO 400
      IF (KK.LT.KS) GO TO 390
      JC=K3
  450 IF (2*KT+1.GE.M) RETURN
      KSPNN=NP(KT+1)
C     PERMUTATION FOR SQUARE-FREE FACTORS OF N
      J=M-KT
      NNFAC(J+1)=1
  460 NNFAC(J)=NFAC(J)*NNFAC(J+1)
      J=J-1
      IF (J.NE.KT) GO TO 460
      KT1=KT+1
      NN=NNFAC(KT1)-1
      JJ=0
      J=0
      GO TO 490
  470 JJ=JJ-K2
      K2=KK
      K=K+1
      KK=NNFAC(K)
  480 JJ=KK+JJ
      IF (JJ.GE.K2) GO TO 470
      NP(J)=JJ
  490 K2=NNFAC(KT1)
      K=KT1+1
      KK=NNFAC(K)
      J=J+1
      IF (J.LE.NN) GO TO 480
C     DETERMINE THE PERMUTATION CYCLES OF LENGTH
C     GREATER THAN 1
      J=0
      GO TO 510
  500 K=KK
      KK=NP(K)
      NP(K)=-KK
      IF (KK.NE.J) GO TO 500
      K3=KK
  510 J=J+1
      KK=NP(J)
      IF (KK.LT.0) GO TO 510
      IF (KK.NE.J) GO TO 500
      NP(J)=-J
      IF (J.NE.NN) GO TO 510
      MMAXF=INC*MAXF
      NT=KS
C     REORDER X TO XP, FOLLOWING THE PERMUTATION
C     CYCLES
      GO TO 580
  520 J=J-1
      IF (NP(J).LT.0) GO TO 520
      JJ=JC
  530 KSPAN=JJ
      IF (JJ.GT.MMAXF) KSPAN=MMAXF
      JJ=JJ-KSPAN
      K=NP(J)
      KK=JC*K+II+JJ
      K1=KK+KSPAN
      K2=0
  540 K2=K2+1
      XT(K2)=X(K1)
      K1=K1-INC
      IF (K1.NE.KK) GO TO 540
  550 K1=KK+KSPAN
      K2=K1-JC*(K+NP(K))
      K=-NP(K)
  560 XP(K1)=X(K2)
      K1=K1-INC
      K2=K2-INC
      IF (K1.NE.KK) GO TO 560
      KK=K2
```

```
      IF (K.NE.J) GO TO 550
      K1=KK+KSPAN
      K2=0
  570 K2=K2+1
      XP(K1)=XT(K2)
      K1=K1-INC
      IF (K1.NE.KK) GO TO 570
      IF (JJ.NE.0) GO TO 530
      IF (J.NE.1) GO TO 520
  580 J=K3+1
      NT=NT-KSPNN
      II=NT-INC+1
      IF (NT.GE.0) GO TO 520
      RETURN
      END


      SUBROUTINE ODRRFS(X,XP,NINC,INC,NFAC,M)
C
C  !!USE F77 COMPILER!!
C  ORDINARY FULLY SYMMETRICAL DIGIT REVERSAL
C  PERMUTATION FOR ARBITRARY LENGTH AND REAL DATA
C
C  X (NINC)     ...INPUT DATA
C  XP(NINC)     ...PERMUTED DATA (MUST BE X<>XP)
C  NINC         ...  = N*INC
C  N            ...NUMBER OF DATA
C  INC          ...INCREMENT
C  NFAC(M)      ...FACTORS OF N DETERMINING THE
C                  MIXED RADIX NUMBER SYSTEM OF THE
C                  INPUT VECTOR X, M < 19
C                  NFAC(1) ...MOST SIGNIFICANT DIGIT
C                    .
C                    .
C                  NFAC(M) ...LEAST SIGNIFICANT DIGIT
C
      PARAMETER (MMAX=18)
      DIMENSION L(0:MMAX),X(NINC),XP(NINC),NFAC(M)
      KL=MMAX-M
      IF(KL)60,30,10
   10 DO 20 K=1,KL
   20   L(K)=INC
   30 L(0)=INC
      DO 40 K=KL+1,MMAX
   40   L(K)=NFAC(K-KL)*L(K-1)
      K=1
      DO  50  K1  =  1  ,L(1) , L(0)
      DO  50  K2  = K1  ,L(2) , L(1)
      DO  50  K3  = K2  ,L(3) , L(2)
      DO  50  K4  = K3  ,L(4) , L(3)
      DO  50  K5  = K4  ,L(5) , L(4)
      DO  50  K6  = K5  ,L(6) , L(5)
      DO  50  K7  = K6  ,L(7) , L(6)
      DO  50  K8  = K7  ,L(8) , L(7)
      DO  50  K9  = K8  ,L(9) , L(8)
      DO  50  K10 = K9  ,L(10), L(9)
      DO  50  K11 = K10 ,L(11), L(10)
      DO  50  K12 = K11 ,L(12), L(11)
      DO  50  K13 = K12 ,L(13), L(12)
      DO  50  K14 = K13 ,L(14), L(13)
      DO  50  K15 = K14 ,L(15), L(14)
      DO  50  K16 = K15 ,L(16), L(15)
      DO  50  K17 = K16 ,L(17), L(16)
      DO  50  KP  = K17 ,L(18), L(17)
         IF(KP-K)50,45,45
   45    WS=X(KP)
         XP(KP)=X(K)
         XP(K)=WS
   50    K=K+INC
   60 RETURN
      END
```

REFERENCES

[1] R. N. BRACEWELL, *The Hartley Transform*, Oxford University Press, New York, 1986.

[2] D. FRASER, *Array permutation by index-digit permutation*, J. Assoc. Comput. Mach., 23 (1976), pp. 298–309.

[3] R. C. SINGLETON, *An algorithm for computing the mixed radix Fast Fourier Transform*, IEEE Trans. Audio Electroacoustics, 17 (1969), pp. 93–103.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 311-341, March 1991
© 1991 Society for Industrial and Applied Mathematics
006

DETECTION, CLASSIFICATION, AND MEASUREMENT OF DISCONTINUITIES*

DAVID LEE†

Abstract. The detection, classification, and measurement of discontinuities from sampled noisy data of a function are studied. A function has a discontinuity of degree k at a point, if the kth-order left and right derivatives at that point are different. The difference is the size of the discontinuity. Discontinuities are classified by their degrees and measured by their sizes. The problem of discontinuity detection and measurement is complex because of the noise corrupting the data. It is difficult to distinguish the discontinuities of the underlying function from the false discontinuities of the noise. A detector, which consists of a pair of a pattern and a linear filter, is proposed. It is shown that for a discontinuity in the input function there is a scaled pattern in the filter response. The location of the pattern is the location of the discontinuity, and the scaling factor of the pattern is the size of the discontinuity. Necessary and sufficient conditions for the one-to-one correspondence between the discontinuities of the input function and the scaled patterns in the filter response are given. Therefore, the problem of discontinuity detection and measurement is reduced to searching for the (scaled) pattern in the filter response. In the presence of noise, the pattern matching is approximate. A statistical method for the pattern search is proposed, and optimal detectors are studied. It is shown that for white noise the optimal detectors are natural splines. As an application of discontinuity detection, curve fitting that preserves discontinuities is discussed.

Key words. discontinuity, detection, classification, measurement, optimal detector, stochastic process, algorithm, curve fitting, smoothing spline

AMS(MOS) subject classifications. 26A15, 41A15, 60G35, 65D10, 68U10

1. Introduction.

The problem of discontinuity detection arises in many applications in science and technology, such as computer vision, computer graphics, signal processing, image processing, pattern recognition, geology, tomography, and remote sensing. A number of approaches have been proposed in statistics literature, such as generalized cross validation (see [33]). The problem is studied mostly in computer vision in terms of edge detection. There are Marr-Hildreth-type detectors [4], [22], detectors using regularization [24], [37], Wiener-type detectors [3], [42], methods based on Markov random field models [13], [23], [34], detectors from residual analysis [14], [21], and a number of other detectors [2], [7], [17], [25], [31]. Most of the approaches are heuristic methods. Function value discontinuities have been studied, yet very little is known about the detection of derivative discontinuities. Almost no effort has been made for the measurement of the size of discontinuities, except in [33], where the size is estimated during the smoothing process *after* the location of discontinuities is detected.

In this work, we study the detection, classification, and measurement of discontinuities from sampled data of a function. Informally, a function $f(t)$ has a discontinuity of degree k at t_0, $k=0, 1, \cdots$, if the kth-order left and right derivatives at t_0 are different, i.e., $f^{(k)}(t_0+) \neq f^{(k)}(t_0-)$. The difference $f^{(k)}(t_0+) - f^{(k)}(t_0-)$ is the size of the discontinuity. Discontinuities are classified by their degrees and measured by their sizes. The problem of discontinuity detection is complex due to the noise corrupting the data. It is difficult to identify the discontinuities of the underlying function, which is buried in noise. We propose a detector, which consists of a pair of a pattern and a linear filter, and the problem is reduced to searching for the (scaled) pattern in a

* Received by the editors August 31, 1988; accepted for publication (in revised form) November 15, 1989.
† AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

filter response. More specifically, for degree k discontinuity detection and measurement, we use a detector $(\phi, \phi^{(k+1)})$, where ϕ is the pattern and its $(k+1)$st derivative $\phi^{(k+1)}$ is the corresponding filter. If function $f(t)$ has a degree k discontinuity at t_0, then in the filter response $f * \phi^{(k+1)}$, where $*$ is the convolution operator, there is a pattern $\alpha\phi$ at location t_0, where α is the size of the discontinuity. Therefore, we reduce the problem to searching for the (scaled) pattern in the filter response. However, in the presence of noise, we can never find the exact pattern, and the pattern matching is approximate. We study optimal detectors to cope with the random noise. We propose a statistical method for the pattern matching. As an application of discontinuity detection, we discuss curve fitting preserving discontinuities. After identifying the locations of function value and first-order derivative discontinuities, we use a cubic smoothing spline to fit data, preserving the discontinuities.

In § 2, we study ideal discontinuity detection and measurement when there is no noise. The problem is reduced to searching for scaled patterns in the filter response. In the presence of noise, the problem is more complicated, and this is the topic of §§ 3-6. We study optimal detectors and propose a statistical method for the pattern search. After discussing some computational issues, we summarize our approach in an algorithm. In §§ 7 and 8, we investigate the problem in more detail when the noise is white. We derive optimal detectors, which are natural splines. We give two examples to explain the idea. As an application, in §§ 9 and 10, we discuss curve fitting preserving discontinuities, and provide experimental results. Miscellaneous issues are addressed in the remaining sections.

2. Ideal discontinuity detection. We study discontinuities of real-valued functions defined on the set of real numbers R. We assume throughout this paper that all the functions are square integrable and have finitely many function value and derivative discontinuities. We further assume that the functions do not have *removable* discontinuities, which are of no practical interest to detect and are almost impossible to detect from finitely many samples. We denote the ith derivative of a function f by $f^{(i)}$, $i = 0, 1, \cdots$, and denote the right (left) limit of f at t_0 by $f(t_0+)(f(t_0-))$. For $r \geqq 0$ and $A > 0$, let $W_r[-A, A]$ be a set of real-valued functions satisfying the following conditions: (i) It has absolutely continuous rth derivative and square integrable $(r+1)$st derivative almost everywhere; (ii) It has support $[-A, A]$, i.e., it is identically zero outside the interval $[-A, A]$.

Since we only consider functions with finitely many function value and derivative discontinuities, in an interval between two consecutive discontinuities, the function has arbitrary order absolutely continuous derivatives.

DEFINITION 2.1. A function $f(t)$ has a *normal discontinuity* of *degree k* at t_0 with a *base* $[t_0 - L, t_0 + L]$, where $k = 0, 1, \cdots$ and $L > 0$, if (i) $f(t)$ has arbitrary order absolutely continuous derivatives in $[t_0 - L, t_0 + L]$ except at t_0; (ii) $f^{(i)}(t_0+) = f^{(i)}(t_0-)$, for all $i < k$; (iii) $f^{(k)}(t_0+) \neq f^{(k)}(t_0-)$, where $f^{(k)}(t_0+) - f^{(k)}(t_0-)$ is the *size* of the discontinuity. □

In general, the detection and measurement of normal discontinuities are difficult. In this work, we study the detection of a special case of normal discontinuities.

DEFINITION 2.2. A function $f(t)$ has an *ideal discontinuity* of *degree k* at t_0 with a *base* $[t_0 - L, t_0 + L]$, where $k = 0, 1, \cdots$ and $L > 0$, if (i) $f(t)$ is identical to a polynomial of degree k in $[t_0, -L, t_0)$, and is identical to a (different) polynomial of degree k in $(t_0, t_0 + L]$; (ii) $f^{(i)}(t_0+) = f^{(i)}(t_0-)$, for all $i < k$; (iii) $f^{(k)}(t_0+) \neq f^{(k)}(t_0-)$, where $f^{(k)}(t_0+) - f^{(k)}(t_0-)$ is the *size* of the discontinuity. □

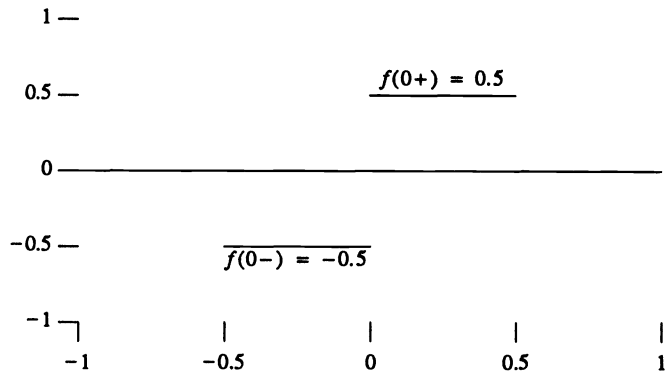


FIG. 1. *Function $f(t)$ has an ideal discontinuity of degree 0 at 0 with a base $[-0.5, 0.5]$ and size $f(0+)-f(0-)=1$.*

Example 2.1. In Fig. 1, $f(t)$ has an ideal discontinuity of degree 0 at 0 with a base $[-0.5, 0.5]$:

$$f(t)=\begin{cases}-0.5, & -0.5\leqq t<0,\\ 0.5, & 0\leqq t\leqq 0.5.\end{cases}$$

The function values at $t\notin[-0.5, 0.5]$ are arbitrary. The size of the discontinuity is $f(0+)-f(0-)=1$, which measures the magnitude of the *jump* at 0.

Example 2.2. In Fig. 2, $f(t)$ has an ideal discontinuity of degree 1 at 0 with a base $[-0.5, 0.5]$:

$$f(t)=\begin{cases}t+0.5, & -0.5\leqq t<0,\\ -t+0.5, & 0\leqq t\leqq 0.5.\end{cases}$$

The function values at $t\notin[-0.5, 0.5]$ are arbitrary. The size of the discontinuity is $f'(0+)-f'(0-)=-2$, which measures the sharpness of the *corner* at 0.

Example 2.3. In Fig. 3, $f(t)$ has an ideal discontinuity of degree 2 at 0 with a base $[-0.5, 0.5]$:

$$f(t)=\begin{cases}-t^2+0.5, & -0.5\leqq t<0,\\ -2t^2+0.5, & 0\leqq t\leqq 0.5.\end{cases}$$

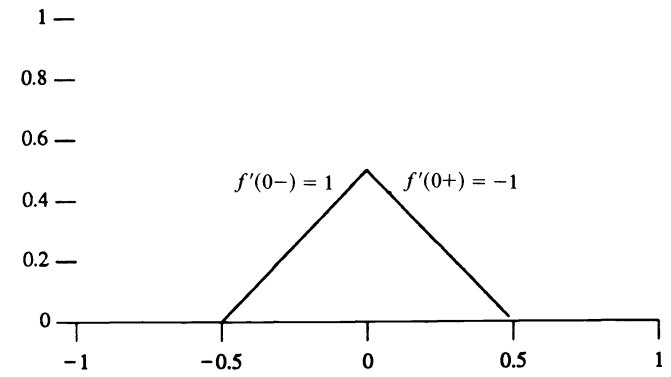


FIG. 2. *Function $f(t)$ has an ideal discontinuity of degree 1 at 0 with a base $[-0.5, 0.5]$ and size $f'(0+)-f'(0-)=-2$.*

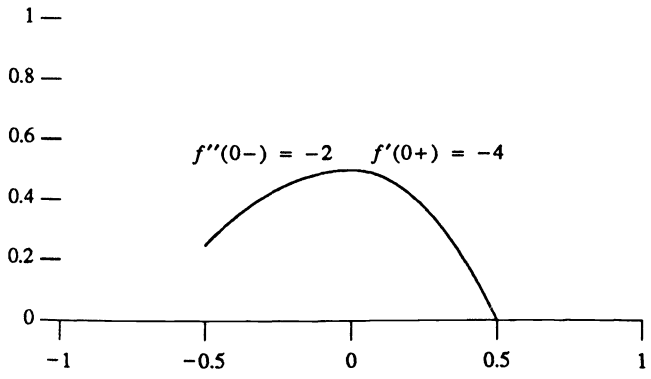


FIG. 3. *Function $f(t)$ has an ideal discontinuity of degree 2 at 0 with a base $[-0.5, 0.5]$ and size $f''(0+)-f''(0-)=-2$.*

The function values at $t \notin [-0.5, 0.5]$ are arbitrary. The size of the discontinuity is $f''(0+)-f''(0-)=-2$.

Discontinuities are classified by their degrees and measured by their sizes. Our approach for detecting discontinuities is based on Theorem 2.1.

THEOREM 2.1. *Let $f(t)$ have an ideal discontinuity of degree k at t_0 with a base $[t_0-L, t_0+L]$, where $k=0, 1, \cdots$ and $L>0$. Let $\phi \in W_{k+1}[-A, A]$, where $0<A \leqq L/2$. The convolution of f and $\phi^{(k+1)}$ is*

$$T(t)=f * \phi^{(k+1)}=\int_R f(t-\tau)\phi^{(k+1)}(\tau)\,d\tau. \tag{2.1}$$

Then for $t \in [-A, A]$,

$$T(t_0+t)=[f^{(k)}(t_0+)-f^{(k)}(t_0-)]\cdot\phi(t). \tag{2.2}$$

Proof. A shorter proof can be obtained by using generalized functions [12]. We provide a longer but conventional proof here.

For simplicity, assume that $t_0=0$ for $t \in [-A, A]$ and $|\tau|>L$, since $0<A \leqq L/2$, $|t-\tau|>A$. Since $\phi^{(k+1)}$ has support $[-A, A]$, $\phi^{(k+1)}(t-\tau)=0$ for $t \in [-A, A]$ and $|\tau|>L$. Therefore,

$$\begin{aligned} T(t) &= \int_R f(t-\tau)\phi^{(k+1)}(\tau)\,d\tau \\ &= \int_R f(\tau)\phi^{(k+1)}(t-\tau)\,d\tau \\ &= \int_{-L}^{L} f(\tau)\phi^{(k+1)}(t-\tau)\,d\tau \\ &= \int_{-L}^{0} f(\tau)\phi^{(k+1)}(t-\tau)\,d\tau+\int_0^L f(\tau)\phi^{(k+1)}(t-\tau)\,d\tau. \end{aligned} \tag{2.3}$$

Since f has a degree k ideal discontinuity at 0 with a base $[-L, L]$, f is identical to a degree k polynomial in $[-L, 0]$. On the other hand, for $t \in [-A, A]$, since $A \leqq L/2$, we have $t+L \geqq A$, and

$$\phi^{(i)}(t+L)=0, \qquad i=0, 1, \cdots, k. \tag{2.4}$$

Integrating by parts k times and using (2.4), we have

$$\int_{-L}^{0} f(\tau)\phi^{(k+1)}(t-\tau)\,d\tau = \int_{-L}^{0} f^{(k)}(\tau)\phi'(t-\tau)\,d\tau - \sum_{i=0}^{k-1} f^{(i)}(0-)\cdot\phi^{(k-i)}(t). \tag{2.5}$$

Since f is identical to a degree k polynomial for $t\in[-L, 0]$, $f^{(k)}(\tau)\equiv f^{(k)}(0-)$. We have

$$\begin{aligned}\int_{-L}^{0} f^{(k)}(\tau)\phi'(t-\tau)\,d\tau &= f^{(k)}(0-)\int_{-L}^{0}\phi'(t-\tau)\,d\tau = f^{(k)}(0-)\int_{t}^{t+L}\phi'(\tau)\,d\tau \\ &= f^{(k)}(0-)\cdot[\phi(t+L)-\phi(t)] = f^{(k)}(0-)\cdot[0-\phi(t)] \\ &= f^{(k)}(0-)\cdot[-\phi(t)].\end{aligned} \tag{2.6}$$

From (2.5) and (2.6),

$$\int_{-L}^{0} f(\tau)\phi^{(k+1)}(t-\tau)\,d\tau = f^{(k)}(0-)\cdot[-\phi(t)] - \sum_{i=0}^{k-1} f^{(i)}(0-)\cdot\phi^{(k-i)}(t). \tag{2.7}$$

Similarly,

$$\int_{0}^{L} f(\tau)\phi^{(k+1)}(t-\tau)\,d\tau = f^{(k)}(0+)\cdot[\phi(t)] + \sum_{i=0}^{k-1} f^{(i)}(0+)\cdot\phi^{(k-i)}(t). \tag{2.8}$$

Since f has a degree k ideal discontinuity at 0,

$$f^{(i)}(0-) = f^{(i)}(0+), \qquad i = 0, 1, \cdots, k-1. \tag{2.9}$$

From (2.3), (2.7), (2.8), and (2.9), for $t\in[-A, A]$,

$$T(t) = [f^{(k)}(0+) - f^{(k)}(0-)]\cdot\phi(t). \qquad \square$$

In signal processing,[1] $\phi^{(k+1)}$ plays the role of a *linear filter*, and T in (2.1) is the *filter response* to the *input signal* f (see [5]). Theorem 2.1 suggests the following approach for discontinuity detection. Assume that the input signal f has a degree k ideal discontinuity at location t_0. From (2.2), if ϕ has a feature point at 0, which is invariant with respect to scaling, then there is a feature point at t_0 in the filter response T. Strict extremum is an obvious choice as a feature point, and we use ϕ with a strict maximum at 0. In the filter response (2.1), we identify extrema as locations of possible discontinuities of the input signal.[2]

The locations of the extrema in the filter response do not contain information on the size of the discontinuities. Furthermore, the filter response may contain extrema, which do not correspond to discontinuities of the input signal, and this will introduce false discontinuities. To reduce such *spurious responses* and to measure the size of the discontinuities, Theorem 2.1 provides an efficient approach. From (2.2), the filter response $T(t_0+t)$ is identical to $[f^{(k)}(t_0+) - f^{(k)}(t_0-)]\cdot\phi(t)$ for $t\in[-A, A]$. Therefore, we match the pattern ϕ, scaled by $f^{(k)}(t_0+) - f^{(k)}(t_0-)$, with the filter response T, at the location of the extremum t_0. To facilitate the processing, we choose a *pattern* function ϕ for degree k ideal discontinuity detection according to the following.

[1] We use the terminologies from signal processing for mnemonics. As a matter of fact, the motivation of this work is from signal processing and image analysis.

[2] Up to this point, for $k = 0$, it is similar to Marr-Hildreth-type filters [22], where derivatives of Gaussian are used as filters. Since Gaussian does not have compact support, equation (2.2) does not hold, and there is no theoeretical justification of the correspondence between the location of the discontinuity and the zero-crossing or extremum of the filter response, unless the discontinuity has infinite base, which is not realistic.

Criteria 2.1. (i) $\phi \in W_{k+1}[-A, A]$; (ii) ϕ is symmetric with respect to zero and has a strict maximum at 0; (iii) ϕ is normalized, i.e., $\phi(0) = 1$. □

By criterion (i), for discontinuities of a particular degree, we have to choose sufficiently smooth pattern functions to obtain a corresponding filter. Furthermore, the pattern (and hence the filter) has a compact support. From criterion (ii), the pattern is symmetric and its first-order derivative is antisymmetric. Therefore, if the input signal f is a degree k polynomial, then we can easily check that the filter response in (2.1) is zero. By criterion (ii) and (2.2), a discontinuity in the input signal f corresponds to an extremum in the filter response T, and we can use all the extrema in T as candidates of discontinuities. Criterion (iii) makes the scaled pattern matching easier, and provides the size of the discontinuities directly, since (2.2) becomes

$$T(t_0+t) = T(t_0) \cdot \phi(t) = [f^{(k)}(t_0+) - f^{(k)}(t_0-)] \cdot \phi(t).$$

We have Theorem 2.2.

THEOREM 2.2 (Detection Theorem). *Let $f(t)$ have an ideal discontinuity of degree k at t_0 with a base $[t_0 - L, t_0 + L]$, where $k = 0, 1, \cdots$ and $L > 0$. Let ϕ satisfy Criteria* 2.1, *where $0 < A \leqq L/2$. Then the filter response T has an extremum at t_0, and for $t \in [-A, A]$,*

$$T(t_0+t) = T(t_0) \cdot \phi(t) \tag{2.10}$$

where $T(t_0) = f^{(k)}(t_0+) - f^{(k)}(t_0-)$ is the size of the discontinuity. □

Theorem 2.2 suggests the following discontinuity detection scheme. For degree k discontinuity detection, we first choose a pattern ϕ, which satisfies Criteria 2.1, and take its $(k+1)$st-order derivative $\phi^{(k+1)}$ as a filter. If there is an extremum in the filter response $T(t)$ at t_0, we match $T(t_0) \cdot \phi(t)$ with $T(t_0+t)$ for $t \in [-A, A]$. A successful matching indicates that a degree k discontinuity of size $T(t_0)$ is detected at t_0. A detector naturally contains a pattern and a corresponding filter, and we have Definition 2.3.

DEFINITION 2.3. For the detection and measurement of discontinuities of degree k, $k = 0, 1, \cdots$, a *detector* consists of a pair of a pattern and filter $(\phi, \phi^{(k+1)})$, where the pattern ϕ satisfies Criteria 2.1. □

From Theorem 2.2, if there is a degree k ideal discontinuity in the input signal, then there is a scaled pattern in the filter response at the same location. However, scaled patterns in the filter response might not correspond to any discontinuities in the input signal, resulting in false discontinuities declared. We discuss this issue next.

For simplicity, we assume that $A = \pi$ and that $t_0 = 0$. Assume that there is a scaled pattern in the filter response, i.e., (2.10) holds. Integrating (2.1) by parts k times, since $\phi^{(i)}(\pm\pi) = 0$, $i = 1, 2, \cdots, k$, the following equation is equivalent to (2.10):

$$T(t) = \int_{-\pi}^{\pi} f^{(k)}(t-\tau)\phi'(\tau)\, d\tau = T(0) \cdot \phi(t) \quad \text{for } t \in [-\pi, \pi]. \tag{2.11}$$

We want to determine whether the input signal f has a degree k ideal discontinuity at 0 and of size $T(0)$. This is the case if and only if $f^{(k)}$ is a step function with a base $[-2\pi, 2\pi]$ and a step at 0 of size $T(0)$. Obviously, (2.11) is equivalent to

$$T(t) = \int_{-2\pi}^{2\pi} f^{(k)}(\tau)\phi'(t-\tau)\, d\tau = T(0) \cdot \phi(t) \quad \text{for } t \in [-\pi, \pi].$$

If we take $f^{(k)}$ as an unknown function, then this is a Fredholm integral equation of the first kind [39, p. 51]. Note that the solutions are not necessarily continuous and

that $\phi'(t)$ is antisymmetric and $\varphi'(0) = 0$ by Criteria 2.1. We want to determine whether the solutions are always step functions as described before.

Let S be such a step function with a base $[-2\pi, 2\pi]$ and a step at 0 of size $T(0)$. From Theorem 2.2, S is a solution of (2.11). Therefore, if (2.11) has no other solutions, then a scaled pattern in the filter response must correspond to an ideal discontinuity in the input signal. Let

$$f^{(k)} = S + D \tag{2.12}$$

be another solution, where D is square integrable. Substituting (2.12) into (2.11), since S is a solution of (2.11),

$$\int_{-\pi}^{\pi} D(t-\tau)\phi'(\tau)\, d\tau = 0 \quad \text{for } t \in [-\pi, \pi]. \tag{2.13}$$

If the solutions of (2.13) are constants, then from (2.12) $f^{(k)}$ is a step function, and there is a degree k ideal discontinuity at 0 in the input signal f. Otherwise, from (2.12) $f^{(k)}$ is not a step function as described before, and there is no degree k ideal discontinuity at 0 in the input signal.

Obviously, any constant is a solution of (2.13), since ϕ' is antisymmetric. In general, the solution set of (2.13) may contain functions other than constants. We have to make some assumptions on D to discuss the uniqueness. We assume that D is a periodic function with period 2π. With this assumption, the uniqueness of the solution of (2.13) depends on the pattern ϕ. Since ϕ is symmetric, its Fourier expansion in $[-\pi, \pi)$ is [36]:

$$\phi(t) = \frac{c_0}{2} + \sum_{n=1}^{\infty} c_n \cos(nt) \quad \text{for } t \in [-\pi, \pi)$$

where

$$c_n = \frac{1}{\pi} \int_{-\pi}^{\pi} \phi(\tau) \cos(n\tau)\, d\tau. \tag{2.14}$$

A pattern ϕ is *strongly complete* if $c_n \neq 0$ for all $n \geqq 1$.

The following theorem provides a necessary and sufficient condition for a one-to-one correspondence between the ideal discontinuities in the input signal and the scaled patterns in the filter response.

THEOREM 2.3 (Strong Completeness Theorem). *Assume that* (i) *The input signal f has continuous $(k-1)$st derivative and square integrable kth derivative in the interval $[t_0-2A, t_0+2A]$*; (ii) *Equation* (2.10) *holds for $t \in [-A, A]$*; (iii) *For a step function S with a base $[t_0-2A, t_0+2A]$ and a step at t_0 of size $T(t_0)$, the difference between $f^{(k)}$ and S is a periodic function with period $2A$. Then the input signal f has an ideal discontinuity of degree k at t_0 with a base $[t_0-2A, t_0+2A]$ and of size $T(t_0)$ if and only if the pattern ϕ is strongly complete.*

Proof. For simplicity, we assume that $t_0 = 0$, $T(0) = 1$, and $A = \pi$. From the previous discussion, we only have to show that any solution of (2.13) is a constant if and only if the pattern ϕ is strongly complete. Since D is a periodic function with period 2π, its Fourier expansion in $[-2\pi, 2\pi)$ is

$$D(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(nx) + b_n \sin(nx)]. \tag{2.15}$$

Substituting (2.15) into (2.13),

$$\int_{-\pi}^{\pi} D(t-\tau)\phi'(\tau)\,d\tau = \sum_{n=1}^{\infty}\left\{a_n\int_{-\pi}^{\pi}\cos[n(t-\tau)]\phi'(\tau)\,d\tau + b_n\int_{-\pi}^{\pi}\sin[n(t-\tau)]\phi'(t)\,d\tau\right\} \tag{2.16}$$

$$= \sum_{n=1}^{\infty}\alpha_n[b_n\cos(nt) - a_n\sin(nt)] = 0 \quad \text{for } t\in[-\pi,\pi]$$

where

$$\alpha_n = \pi n c_n, \qquad n = 1, 2, \cdots$$

where c_n is in (2.14).

Assume that ϕ is not strongly complete. Then there exists $n \geqq 1$, such that $c_n = 0$ and hence $\alpha_n = 0$. From (2.16), the corresponding a_n and b_n can be arbitrary. Since a_n and b_n are Fourier coefficients of $D(t)$, by the uniqueness of the Fourier series [36], $D(t)$ is not a constant.

Conversely, assume that ϕ is strongly complete. Then the $\alpha_n \neq 0$ for all $n \geqq 1$. By the uniqueness of the Fourier series, $\alpha_n a_n = \alpha_n b_n = 0$, and $a_n = b_n = 0$, $n = 1, 2, \cdots$. Therefore, from (2.15), $D(t) = a_0/2$, since $D(t)$ has no removable discontinuities [36, Chap. 13.8].

We have shown that ϕ is strongly complete if and only if $D(t)$ is a constant, and this is the case if and only if f has a degree k ideal discontinuity at 0. The proof is completed. □

It is a strong assumption that ϕ is strongly complete. A weaker assumption is that only finitely many Fourier coefficients in (2.14) can be zero, and we call ϕ *weakly complete.* If ϕ is weakly complete, then only finitely many α_n can be zero, and from (2.16) only finitely many a_n and b_n can be arbitrary. From (2.15), $D(x)$ contains finitely many terms and is an analytic function. From (2.12), $f^{(k)}$ is an analytic function in $[-2\pi, 2\pi]$ except at 0, where $f^{(k)}(0+) - f^{(k)}(0-) = T(0)$. From Definition 2.1, f has a degree k normal discontinuity at 0. Therefore we have Theorem 2.4.

THEOREM 2.4 (Weak Completeness Theorem). *Assume that* (i) *The input signal f has continuous $(k-1)$st derivative and square integrable kth derivative in the interval $[t_0-2A, t_0+2A]$*; (ii) *Equation* (2.10) *holds for $t\in[-A, A]$*; (iii) *For a step function S with a base $[t_0-2A, t_0+2A]$ and a step at t_0 of size $T(t_0)$, the difference between $f^{(k)}$ and S is a periodic function with period $2A$. If the pattern ϕ is weakly complete, then the input signal f has a normal discontinuity of degree k at t_0 with a base $[t_0-2A, t_0+2A]$ and of size $T(t_0)$.* □

For degree k discontinuities, we first choose a detector $(\phi, \phi^{(k+1)})$ as in Definition 2.3, and then search for the scaled pattern ϕ in the filter response. From the detection theorem, we shall find all the ideal discontinuities. On the other hand, if the pattern is weakly complete and the periodicity condition holds, then by the weak completeness theorem no false discontinuities will be declared. Furthermore, if the pattern is strongly complete, then by the strong completeness theorem the discontinuities declared are ideal.

However, in practice, the matching in (2.10) can never succeed due to the noise corrupting the signal. We now discuss the design of detectors to cope with the random noise.

3. Random noise and optimal detectors. A random noise n can be modeled as a stochastic process [26]. There are two ways we can compute averages of a stochastic

process. We can average together the values of all member functions evaluated at some particular time, and this is called *ensemble average.* We can compute a *time average* of a particular member function over all time. A stochastic process is *ergodic* if (i) The ensemble average is constant with time; (ii) The time averages of all member functions are equal; (iii) The ensemble and time averages are numerically equal. Ergodic processes model commonly encountered random noises quite well, including white and Gaussian noise [5], [26].

For notational convenience, we denote the ensemble average by $E\{\cdot\}$ and the time average over interval $[-T, T]$ by $E_{t[-T,T]}\{\cdot\}$. More specifically, for a member function of a process n, $E_{t[-T,T]}\{n(\tau)\} = 1/(2T)\int_{-T}^{T} n(\tau)\,d\tau$. As $T\to\infty$, $E_{t[-\infty,\infty]}\{n(\tau)\} = \lim_{T\to\infty} 1/(2T)\int_{-T}^{T} n(\tau)\,d\tau$. For an ergodic process n, $E\{n\} = E_{t[-\infty,\infty]}\{n(\tau)\}$.

A process n is *correlation-ergodic* [26], if for any member function of the process $n(t)$ the *autocorrelation* of the process

$$R_n(t) = \lim_{T\to\infty} \frac{1}{2T}\int_{-T}^{T} n(t+\tau)n(\tau)\,d\tau.$$

Both white and Gaussian noise are correlation-ergodic. From now on, we assume that the random noise we are dealing with is ergodic and correlation-ergodic. We need the following lemma, and omit the routine proof.

LEMMA 3.1. *Let Φ be a square integrable function. For the process $(n * \Phi)^2$, the ensemble average $E\{[n * \Phi(t_0)]^2\}$ is invariant with respect to time t_0, and*

$$E\{(n * \Phi)^2\} = \int_R |F[\Phi](s)|^2 P_n(s)\,ds \tag{3.1}$$

where F is the Fourier transform and P_n is the power spectrum of n. □

Assume that the input signal f is corrupted by an additive noise n. To detect and measure degree k discontinuities, we choose a detector $(\phi, \phi^{(k+1)})$ as in Definition 2.3. The filter response becomes

$$\tilde{T} = (f+n) * \phi^{(k+1)} = f * \phi^{(k+1)} - n * \phi^{(k+1)}. \tag{3.2}$$

If f has a degree k ideal discontinuity at t_0, then, from (2.10), for $t\in[-A, A]$,[3]

$$\tilde{T}(t_0+t) = T(t_0)\cdot\phi(t) + n * \phi^{(k+1)}(t_0+t) \tag{3.3}$$

where $T(t_0) = f^{(k)}(t_0+) - f^{(k)}(t_0-)$ is the size of the discontinuity.

The filter response $\tilde{T}$ consists of two parts. The first part $T(t_0)\cdot\phi(t)$ is the response to the signal f, and the second part $n * \phi^{(k+1)}(t_0+t)$ is the response to the noise. We want to find the scaled pattern $T(t_0)\cdot\phi(t)$, which is buried in the noise response. To facilitate the processing, we choose a detector $(\phi, \phi^{(k+1)})$ with the following considerations.

First, to detect a discontinuity, we verify that the (scaled) pattern ϕ appears where the filter response has an extremum. From (3.3) and Criteria 2.1, the (ensemble) average magnitude of the extremum is proportional to $E\{[T(t_0)\cdot\phi(0)]^2\}$. On the other hand, from (3.2), the average magnitude of the filter response to the noise at the same location is proportional to $E\{[n * \phi^{(k+1)}(t_0)]^2\}$. From Lemma 3.1, it is invariant with respect to t_0, and we denote this quantity by $E\{[n * \phi^{(k+1)}]^2\}$. The *signal-to-noise ratio* is

$$\rho = \frac{E\{[T(t_0)\cdot\phi(0)]^2\}}{E\{[n * \phi^{(k+1)}]^2\}}. \tag{3.4}$$

We choose a detector such that the amplitude of the filter response will be highly

[3] $n * \phi^{(k+1)}$ in (3.3) is interpreted as the convolution of a sample of n with $\phi^{(k+1)}$.

dependent on the presence or absence of the discontinuity and relatively insensitive to the fluctuations in the noise n. This is equivalent to maximizing the signal-to-noise ratio ρ. Since the choice of ϕ should not depend on any particular value of $T(t_0)$, we set $T(t_0) = 1$ in the following analysis.

Second, we identify the extrema in the filter response, and use their locations as candidates of discontinuities. For each candidate, we conduct a pattern matching to eliminate false responses. Since the pattern matching is approximate due to noise, there is a possiblity of accepting false responses. Furthermore, pattern matching is usually expensive. Therefore, we want to minimize the number of extrema in the filter response to noise when there are no discontinuities in the input signal. The filter response to noise is $n * \phi^{(k+1)}$. We minimize its expected number of extrema in the unit interval, which is approximately [1, Chap. 3.5-2]

$$\frac{1}{\pi}\left(\frac{E\{\{[n * \phi^{(k+1)}]'\}^2\}}{E\{[n * \phi^{(k+1)}]^2\}}\right)^{1/2}.$$

Equivalently, we maximize

$$\mu = \frac{E\{[n * \phi^{(k+1)}]^2\}}{E\{[n * \phi^{(k+2)}]^2\}}. \tag{3.5}$$

From the above considerations, we construct a detector, which maximizes a combination of ρ and μ. The combination could be a monotonically increasing function of ρ and μ. For simplicity, we choose a detector, which maximizes the product $\rho\mu$. From (3.4) and (3.5), we find ϕ, which maximizes

$$\rho\mu = \frac{[\phi(0)]^2}{E\{[n * \phi^{(k+2)}]^2\}}. \tag{3.6}$$

From Lemma 3.1,

$$\begin{aligned} \rho\mu &= \frac{[\phi(0)]^2}{\int_R |F[\phi^{(k+2)}](s)|^2 P_n(s)\, ds} \\ &= \frac{1}{\int_R |F[\phi^{(k+2)}](s)|^2 P_n(s)\, ds}. \end{aligned} \tag{3.7}$$

Therefore, we find ϕ, which minimizes[4]

$$\int_R |F[\phi^{(k+2)}](s)|^2 P_n(s)\, ds. \tag{3.8}$$

The minimization problem can be solved as follows. We first find $F[\phi^{(k+2)}]$, which minimizes (3.8), using calculus of variation [11]. We then use the inverse Fourier transform to obtain $\phi^{(k+2)}$, and finally compute ϕ. The minimization problem is subject to the constraints from Criteria 2.1. A solution provides an *optimal* detector. In summary we have Definition 3.1.

DEFINITION 3.1. For the detection and measurement of discontinuities of degree k, an optimal detector $(\phi, \phi^{(k+1)})$ is given by a pattern ϕ, which minimizes (3.8) subject to the conditions from Criteria 2.1. □

In general, for arbitrary noise process n, it is not easy to solve the minimization problem. The solution may not exist or may not be unique. We shall not pursue the

[4] Assume that ϕ is not required to be normalized as in Criteria 2.1. It is obvious that if ϕ maximizes (3.6), then $\phi/\phi(0)$ also maximizes (3.6). Therefore, with the additional constraint $\phi(0) = 1$, if ϕ maximizes (3.6), then it also maximizes (3.6) among all functions without such constraint. Thus the condition $\phi(0) = 1$ in Criteria 2.1 is not restrictive.

issue here. In § 7, we provide a solution when the noise n is white. We show that the solution exists and is unique, which is a natural spline.

Because of the noise, the location of an extremum in the filter response may deviate from the location of the corresponding discontinuity in the signal. We want to have a detector, which minimizes such deviation. Assume that f has a degree k ideal discontinuity at 0 and that filter response has an extremum at δ, instead, due to the noise. From (3.3),

$$\begin{aligned}\tilde{T}'(\delta) &= [f^{(k)}(0+) - f^{(k)}(0-)] \cdot \phi'(\delta) + [n * \phi^{(k+1)}(\delta)]' \\ &= [f^{(k)}(0+) - f^{(k)}(0-)] \cdot \phi'(\delta) + [n * \phi^{(k+2)}(\delta)] = 0.\end{aligned}$$

Since ϕ has absolutely continuous first-order derivative and has a strict extremum at 0, $\phi'(0) = 0$. Therefore, $\phi'(\delta) \approx \phi'(0) + \phi''(0)\delta = \phi''(0)\delta$, and

$$\delta \approx -\frac{n * \phi^{(k+2)}(\delta)}{[f^{(k)}(0+) - f^{(k)}(0-)] \cdot \phi''(0)}. \tag{3.9}$$

Assume that $E\{n\} = 0$. Since n is ergodic, $E\{n * \phi^{(k+2)}(\delta)\} = 0$, and from (3.9) $E\{\delta\} = 0$. We minimize the variance off the deviation δ. From (3.9) and Lemma 3.1,

$$E\{\delta^2\} \approx \frac{E\{[n * \phi^{(k+2)}(\delta)]^2\}}{[f^{(k)}(0+) - f^{(k)}(0-)]^2 [\phi''(0)]^2} = \frac{\int_R |F[\phi^{(k+2)}](s)|^2 P_n(s)\, ds}{[f^{(k)}(0+) - f^{(k)}(0-)]^2 [\phi''(0)]^2}.$$

The numerator is (3.8). Therefore, the optimal detector also tends to minimize the deviation of the extremum in the filter response from the location of the corresponding discontinuity.

4. Searching for the scaled pattern in the filter response. The basic idea of our approach is simple. For the detection and measurement of discontinuities of a particular degree, we first determine an optimal detector according to Definition 3.1. After convolving the input signal with the filter, we find extrema in the filter response, and then match the (scaled) pattern at the location of each extremum. More specifically, assume that the input signal f has a degree k discontinuity at t_0 and that the filter response $\tilde{T}$ has an extremum at t_0 and $\tilde{T}(t_0) \approx T(t_0)$. From (3.3), we verify

$$\tilde{T}(t_0 + t) - \tilde{T}(t_0) \cdot \phi(t) = n * \phi^{(k+1)}(t_0 + t), \qquad -A \leqq t \leqq A. \tag{4.1}$$

If there is no noise, i.e., $n \equiv 0$, then we match $\tilde{T}(t_0 + t)$ and $\tilde{T}(t_0) \cdot \phi(t)$ exactly for $t \in [-A, A]$. In the presence of noise, i.e., $n \neq 0$, this is an approximate pattern matching problem [27]. An obvious approach is: compute the L_2 or L_∞ norm of $\tilde{T}(t_0 + t) - \tilde{T}(t_0) \cdot \phi(t)$; if it is smaller than certain threshold value, then the matching is declared successful. While L_2 norm tolerates sharp deviations, L_∞ norm seems to be too conservative. Furthermore, neither approaches incorporate the knowledge of the noise involved, and both are ad hoc.

We suggest a statistical method based on the ergodicity of the noise process. There is a variety of ergodic theorems, starting with the works by Birkhoff and von Neumann (see [26]). For our purpose, we prove a simple theorem, which provides the statistics of a sample of a process $n * \Phi$.

THEOREM 4.1. *Let Φ be a square integrable function. Then*

$$\begin{aligned} E_{t[-\infty,\infty]}\{n * \Phi\} &= E\{n\} \int_R \Phi(\tau)\, d\tau, \\ E_{t[-\infty,\infty]}\{[n * \Phi]^2\} &= \int_R |F[\Phi](s)|^2 P_n(s)\, ds \end{aligned} \tag{4.2}$$

where $E_{t[-\infty,\infty]}\{\cdot\}$ is the time average, $E\{\cdot\}$ is the ensemble average, F is the Fourier transform, and P_n is the power spectrum of n.

Proof.

$$E_{t[-\infty,\infty]}\{n * \Phi\} = E_{t[-\infty,\infty]}\left\{\int_R n(t-\tau)\cdot\Phi(\tau)\,d\tau\right\}$$

$$= \int_R \{E_{t[-\infty,\infty]}\{h(t-\tau)\}\}\cdot\Phi(\tau)\,d\tau$$

$$= \int_R \{E_{t[-\infty,\infty]}\{n(t)\}\}\cdot\Phi(\tau)\,d\tau.$$

Since n is ergodic, $E_{t[-\infty,\infty]}\{n(t)\} = E\{n\}$, and

$$E_{t[-\infty,\infty]}\{n * \Phi] = \int_R E\{n\}\cdot\Phi(\tau)\,d\tau$$

$$= E\{n\}\int_R \Phi(\tau)\,d\tau.$$

We have proved the first equality. For the second, let R_n be the autocorrelation of the noise n. Let $i = \sqrt{-1}$, then

$$E_{t[-\infty,\infty]}\{[n * \Phi]^2\} = E_{t[-\infty,\infty]}\left\{\int_R n(t-x)\cdot\Phi(x)\,dx\cdot\int_R n(t-y)\cdot\Phi(y)\,dy\right\}$$

$$= \int_{R^2}\Phi(x)\Phi(y)\cdot\{E_{t[-\infty,\infty]}\{n(t-x)\cdot n(t-y)\}\}\,dx\,dy.$$

Since n is correlation-ergodic, $E_{t[-\infty,\infty]}\{n(t-x)\cdot n(t-y)\} = R_n(x-y)$. Therefore,

$$E_{t[-\infty,\infty]}\{[n * \Phi]^2\} = \int_{R^2}\Phi(x)\cdot\Phi(y)\cdot R_n(x-y)\,dx\,dy$$

$$= \int_{R^2}\Phi(x)\cdot\Phi(y)\left[\int_R P_n(s)\,e^{i2\pi s(x-y)}\,ds\right]dx\,dy$$

$$= \int_R\left[\int_R\Phi(x)\cdot e^{i2\pi sx}\,dx\cdot\int_R\Phi(y)\cdot e^{-i2\pi sy}\,dy\right]\cdot P_n(s)\,ds$$

$$= \int_R F[\Phi](-s)\cdot F[\Phi](s)\cdot P_n(s)\,ds$$

$$= \int_R |F[\Phi](s)|^2 P_n(s)\,ds. \qquad \square$$

We assume that the time mean and variance of the process $n * \phi^{(k+1)}$ over interval $[-A, A]$ approximately equal the time mean and variance over interval $[-\infty, \infty]$, respectively. From Theorem 4.1, where $\Phi = \phi^{(k+1)}$, we have Theorem 4.2.

THEOREM 4.2. *If* $\tilde{T}(t_0+t)-\tilde{T}(t_0)\cdot\phi(t)$ *is a sample of the process* $n*\phi^{(k+1)}$ *in the time interval* $[-A, A]$, *then*

$$E_{t[-A,A]}\{[\tilde{T}(t_0+t)-\tilde{T}(t_0)\cdot\phi(t)]\}\approx E\{n\}\int_R \phi^{(k+1)}(\tau)\,d\tau,$$

$$E_{t[-A,A]}\{[\tilde{T}(t_0+t)-\tilde{T}(t_0)\cdot\phi(t)]^2\}\approx\int_R |F[\phi^{(k+1)}](s)|^2 P_n(s)\,ds. \qquad (4.3)$$

□

Note that the right sides of (4.3) can be precomputed, if we know the mean and power spectrum of the noise n. The left sides of (4.3) can be computed from the filter response $\tilde{T}$.

We are now ready to discuss the approximate pattern matching of $\tilde{T}(t_0+t)$ and $\tilde{T}(t_0)\cdot\phi(t)$ for $t\in[-A, A]$. Equation (4.1) can be interpreted as follows. For a stochastic process $n*\phi^{(k+1)}$, we have a *sample* (or *observation*) of this process in a time interval $t\in[-A, A]$: $\tilde{T}(t_0+t)-\tilde{T}(t_0)\cdot\phi(t)$. We want to verify that it is indeed a sample of the stochastic process. This can be done by verifying the statistics of the sample. More specifically, from Theorem 4.2, if $\tilde{T}(t_0+t)-\tilde{T}(t_0)\cdot\phi(t)$ is a sample of the process, its time mean and variance, which are the left sides of (4.3), should approximately equal the right sides of (4.3). If (4.3) does not hold, then $\tilde{T}(t_0+t)-\tilde{T}(t_0)\cdot\phi(t)$ is not likely to be a sample of the process $n*\phi^{(k+1)}$. Therefore, we declare that there is no matching between $\tilde{T}(t_0+t)$ and $\tilde{T}(t_0)\cdot\phi(t)$, and there is no degree k discontinuity at t_0. Otherwise, we accept t_0; there is a degree k discontinuity at t_0 of size $\tilde{T}(t_0)$.

In summary, for each extremum t_0 in the filter response $\tilde{T}(t)$, we perform approximate pattern matching of $\tilde{T}(t_0+t)$ and $\tilde{T}(t_0)\cdot\phi(t)$ for $t\in[-A, A]$. We compute the left sides of (4.3) and compare their values with the right sides of (4.3), which can be precomputed. If (4.3) holds, then there is a degree k discontinuity at t_0 with size $\tilde{T}(t_0)$. Otherwise, t_0 is rejected.

5. Some computational issues. In practice, we have only sampled noisy data of a signal: $\tilde{f}=f+n$ at t_i, $i=1, 2, \cdots$, and sometimes the data are sparse. On the other hand, to obtain the filter response, we compute an integral

$$\tilde{T}(t)=(\tilde{f}*\phi^{(k+1)})(t)\int \tilde{f}(t-\tau)\phi^{(k+1)}(\tau)\,d\tau. \qquad (5.1)$$

This is a computation of a weighted integral using a set of sampled data, where the weight $\phi^{(k+1)}$ is absolutely continuous. There are results in approximation theory for integration problems (see [8], [28], [38]). The difficulty here is: (i) We integrate a function with discontinuities and we do not know the location and size of the discontinuities; (ii) The sampled data are corrupted by noise and sometimes the noise level is high. We might want to approximate or interpolate the integrand first and then integrate the resulting function. This approach does not seem to be satisfactory. To approximate or interpolate a function with discontinuities is a difficult task itself if we do not know the location of the discontinuities. Furthermore, the input data are corrupted by noise, and this makes the task more difficult. As a matter of fact, discontinuity detection could be a reasonable first step for such approximation or interpolation problems (see [20], [33]).

For our case, we use the following simple procedure. For degree 0 discontinuities (jumps), we use a step function interpolation of the data for the integrand. After locating the jumps, we detect degree 1 discontinuities (corners). We can assume that between jumps, the function is continuous, and we use a piecewise linear interpolation

of the data between jumps for the integrand. This procedure provides reasonable results for jump and corner detection (see § 8).

For detecting discontinuities of degree more than 1, we construct a *cubic smoothing-spline* from the data between discontinuities detected earlier, and use it for the integrand. For a discussion of cubic smoothing-splines, see [8] and [30].

Another method is using the Fourier transform. Since

$$F[\tilde{T}] = F[\tilde{f} * \phi^{(k+1)}] = F[\tilde{f}] \cdot (2\pi si)^{k+1} F[\phi],$$
$$\tilde{T} = F^{-1}[(2\pi si)^{k+1} F[\tilde{f}] \cdot F[\phi]],$$

where F and F^{-1} are the Fourier transform and inverse Fourier transform, respectively.

In the filter response, we search for extrema as candidates of discontinuities. More specifically, from (3.3), we search for extrema of $T(t_0) \cdot \phi(t)$, which is buried in the noise response $n * \phi^{(k+1)}$. This problem by itself is interesting and yet complicated. We do not intend to investigate the issue here. For our case, since the filter response is obtained after a smoothing process by a convolution with the filter, which is smooth, we use the following simple technique. For a set of $2k+1$ consecutive data points in a processing window, if the $(k+1)$st sample is an *infimum* or *supremum* in the window, then take it as an extremum in the filter response. We shift the window from left to right and obtain extrema in the filter response. This procedure may adopt extrema, which do not correspond to discontinuities in the input function. However, our pattern matching procedure will effectively eliminate those false responses.

We discuss discontinuity detection of signals over time interval $[-\infty, \infty]$. In practice, we only have sampled data in a finite range, and we detect discontinuities in this range. From (5.1), for a discontinuity at t_0, the values of $\tilde{f}$ outside the interval $[t_0-2A, t_0+2A]$ have no effects on the values of $\tilde{T}(t_0+t)$ for $t \in [-A, A]$, where $[-A, A]$ is the support of the detector. From Theorems 2.2 and 4.2, the values of the input signal $\tilde{f}$ outside the interval $[t_0-2A, t_0+2A]$ have no effects on the detection and measurement processes of a discontinuity at t_0. Therefore, discontinuities and their detection are local processes, and our approach and analysis remain valid if the input signal is in a finite range. Note that near the two extreme sampling locations, anomaly may arise.

6. An algorithm for discontinuity detection. We classify discontinuities according to their degrees. We detect and measure discontinuities in order: $k = 0, 1, \cdots$. We first choose an optimal detector by Definition 3.1. We then compute the filter reponse of the input signal and search for the extrema in the filter response, as described in § 5. For each extremum, we do pattern matching using Theorem 4.2. A successful matching indicates that a discontinuity is detected at that location with measured size. We summarize as follows.

ALGORITHM 6.1 (discontinuity detection).
Input: sampled data $(t_i, \tilde{f}(t_i))$, $i = 1, 2, \cdots$.
Output: locations and sizes of discontinuities of degree k, $k = 0, 1, \cdots$.
for $k = 0, 1, \cdots$ **do**

(i) Choose a detector $(\phi, \phi^{(k+1)})$, where ϕ is determined by Definition 3.1, and compute the right sides of (4.3);

(ii) Compute the filter response:

$$\tilde{T}(t) = \int \tilde{f}(t-\tau)\phi^{(k+1)}(\tau)\, d\tau$$

(iii) Find extrema of $\tilde{T}(t)$ at t_j, $j = 1, 2, \cdots$;

(iv) For each t_j, compute

$$E_{t[-A,A]}\{[\tilde{T}(t_j+t)-\tilde{T}(t_j)\cdot\phi(t)]\} \quad \text{and} \quad E_{t[-A,A]}\{[\tilde{T}(t_j+t)-\tilde{T}(t_j)\cdot\phi(t)]^2\}$$

If

$$E_{[-A,A]}\{[\tilde{T}(t_j+t)-\tilde{T}(t_j)\cdot\phi(t)]\}\approx E\{n\}\int_R \phi^{(k+1)}(\tau)\,d\tau$$

$$E_{[-A,A]}\{[\tilde{T}(t_j+t)-\tilde{T}(t_j)\cdot\phi(t)]^2\}\approx \int_R |F[\phi^{(k+1)}](s)|^2 P_n(s)\,ds$$

then a discontinuity of degree k and size $\tilde{T}(t_j)$ is detected at t_j. □

end

7. White noise and optimal detectors. In practice, we usually assume that n is a white noise, i.e., it has mean zero, and is spectrally white with $P_n = N_0^2$, a constant (see [1], [5]). In this case, we show that the pattern ϕ in an optimal detector is a *natural spline.* A natural spline defined in a finite interval $[a, b]$ is a piecewise polynomial such that at each knot it satisfies certain constraints for its function and derivative values and that at the two endpoints some of its function and derivative values are zero (see [8]).

From Rayleigh's theorem, (3.8) becomes $N_0^2\int_R [\phi^{(k+2)}]^2\,ds$, and we minimize

$$\int_{-\infty}^{\infty} [\phi^{(k+2)}(s)]^2\,ds \tag{7.1}$$

subject to the conditions from Criteria 2.1. We have Theorem 7.1.

THEOREM 7.1. *Assume that the noise n is white. For the detection and measurement of discontinuities of degree k, the pattern ϕ in an optimal detector $(\phi, \phi^{(k+1)})$ is the natural spline of degree $2k+3$ with one knot at 0, satisfying the $2(2k+4)$ conditions:* $\phi^{(i)}(\pm A)=0$, $i=0,1,\cdots,k+1$, $\phi(0)=1$, *and* $\phi^{(2j+1)}(0)=0$, $j=0,1,\cdots,k$. □

For the proof, see Appendix 1.

Note that when n is a white noise, the right sides of (4.3) become $E\{n\}\int_R \phi^{(k+1)}(\tau)\,d\tau = 0$, and $\int_R |F[\phi^{(k+1)}](s)|^2 P_n(s)\,ds = N_0^2\int_R [\phi^{(k+1)}(s)]^2\,ds$.

In practice, the two most important cases of discontinuities are degree 0 (function value discontinuity or jump) and degree 1 (first-order derivative discontinuity or corner) discontinuity detection. We have Corollary 7.1.

COROLLARY 7.1. *Assume that the noise n is white.* (i) *For the detection and measurement of jumps, $k=0$, we have an optimal detector (ϕ, ϕ'), where the pattern ϕ is the unique cubic natural spline with one knot at 0, and the corresponding filter ϕ' is a quadratic spline*:

$$\phi(t)=\begin{cases}\dfrac{-t^2}{A^3}(2t+3A)+1, & -A\leqq t\leqq 0,\\[2ex] \dfrac{t^2}{A^3}(2t-3A)+1, & 0<t\leqq A,\\[2ex] 0, \quad \textit{otherwise,}\end{cases} \tag{7.2}$$

$$\phi'(t)=\begin{cases}-\dfrac{6t(t+A)}{A^3}, & -A\leqq t\leqq 0,\\[2ex] \dfrac{6t(t-A)}{A^3}, & 0<t\leqq A,\\[2ex] 0, \quad \textit{otherwise.}\end{cases} \tag{7.3}$$

(ii) *For the detection and measurement of corners, $k=1$, we have an optimal detector (ϕ, ϕ''), where the pattern ϕ is the unique quintic natural spline with one knot at 0, and the corresponding filter ϕ'' is a cubic spline*:

$$\phi(t)=\begin{cases}\dfrac{(t+A)^3}{3A^5}(8t^2-9At+3A^2), & -A\leqq t\leqq 0,\\ -\dfrac{(t-A)^3}{3A^5}(8t^2+9At+3A^2), & 0\leqq t\leqq A,\\ 0, \quad \textit{otherwise}, \end{cases} \tag{7.4}$$

$$\phi''(t)=\begin{cases}\dfrac{20}{3A^5}(t+A)(8t^2+At-A^2), & -A\leqq t\leqq 0,\\ -\dfrac{20}{3A^5}(t-A)(8t^2-At-A^2), & 0\leqq t\leqq A,\\ 0, \quad \textit{otherwise}. \end{cases} \tag{7.5}$$

□

With some assumptions, the Completeness Theorems 2.3 and 2.4 establish a one-to-one correspondence between the discontinuities of the input signal and the scaled patterns in the filter response. Therefore, an important question is whether the patterns in Theorem 7.1 and Corollary 7.1 are complete such that a scaled pattern in the filter response does correspond to a discontinuity in the input signal. We have Theorem 7.2.

THEOREM 7.2. *The cubic and quintic natural splines in Corollary* 7.1 *are strongly complete, and the natural splines in Theorem* 7.1 *are weakly complete.* □

For the proof, see Appendix 2.

8. Examples. We use two examples to explain Algorithm 6.1. Here, we assume that the noise n is white.

Example 8.1. Input signal f has function value discontinuities (jumps) at $t=-0.5$, 0, and 0.5, with size 0.25, 0.5, and -0.25, respectively (see Fig. 4(a)). It is corrupted by a random noise n, i.e., $\tilde{f}=f+n$, and the sampled noisy data are in Fig. 4(b). We know that

$$E\{[n*\phi']\}=0.001 \quad \text{and} \quad E\{[n*\phi']^2\}=0.002.$$

We apply Algorithm 6.1 and use the detector in Corollary 7.1 to detect jumps. The filter response $\tilde{T}(t)=\tilde{f}*\phi'(t)$ is in Fig. 4(c), and has 16 extrema (see Table 1).

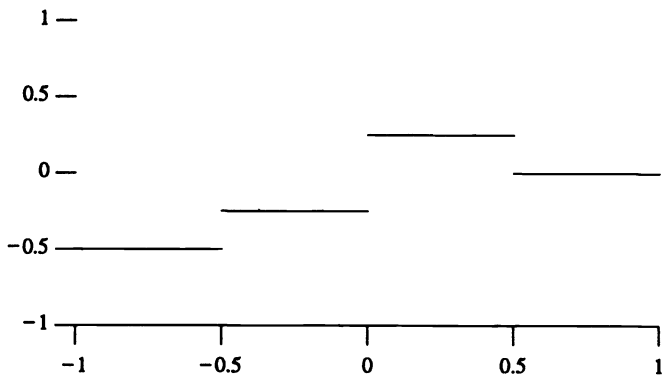


FIG. 4(a). *Input signal $f(t)$ has jumps at $t=-0.5$, 0, and* 0.5, *with sizes* 0.25, 0.5, *and* -0.25, *respectively.*

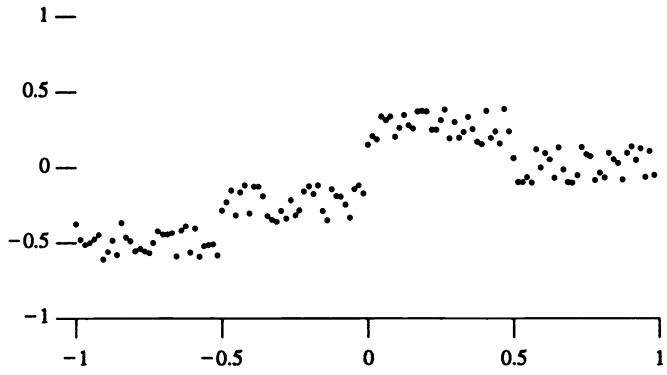


FIG. 4(b). *Sampled data from input signal $f(t)$ in Fig.* 4(a), *corrupted by noise.*

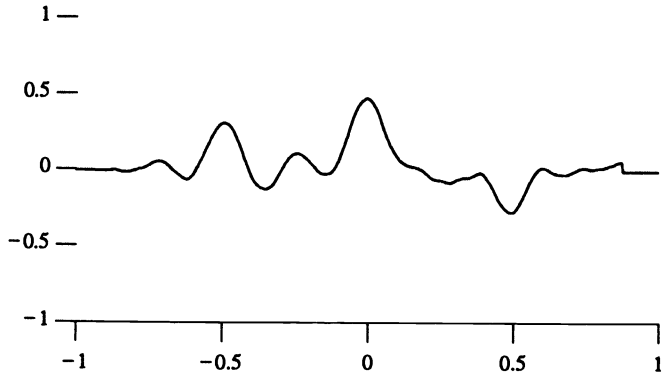


FIG. 4(c). *Filter response $\tilde{T} = \tilde{f} * \phi'$.*

TABLE 1

Extrema	t_0	$\tilde{T}(t_0)$	$E_t\{[\cdot]\}$	$E_t\{[\cdot]^2\}$
max	−0.874023	0.006097	−0.081482	3.042131
min	−0.832031	−0.009734	−1.728958	7.329125
max	−0.714844	0.063585	−0.382837	0.260425
min	−0.623047	−0.057566	−1.747315	5.731243
max*	−0.495117	0.317577	−0.104486	0.021425
min	−0.353516	−0.120287	−0.804558	1.149351
max	−0.244141	0.115055	−0.423485	0.323542
min	−0.151367	−0.021025	−5.862922	68.182121
max*	−0.003906	0.477164	0.035627	0.002611
min	0.277344	−0.076244	−0.049658	0.026657
min	0.338867	−0.042458	1.134892	3.502952
min*	0.490234	−0.274425	−0.077595	0.011872
max	0.597656	0.023210	−3.734250	32.249680
min	0.670898	−0.023613	−0.347841	1.454527
max	0.743164	0.022591	0.021995	1.261203
max	0.868164	0.061889	−0.232219	0.173846

The extrema without an asterisk are rejected either by step (iv) of Algorithm 6.1 or because the size is almost zero. Therefore, we have three jumps detected at $t_0 = -0.495117$, -0.003906, and 0.490234, with size 0.317577, 0.477164, and -0.274425, respectively.

Example 8.2. Input signal f has first-order derivative discontinuities (corners) at $t = -0.5$, 0, and 0.5, with size 2, -4, and 2, respectively (see Fig. 5(a)). It is corrupted

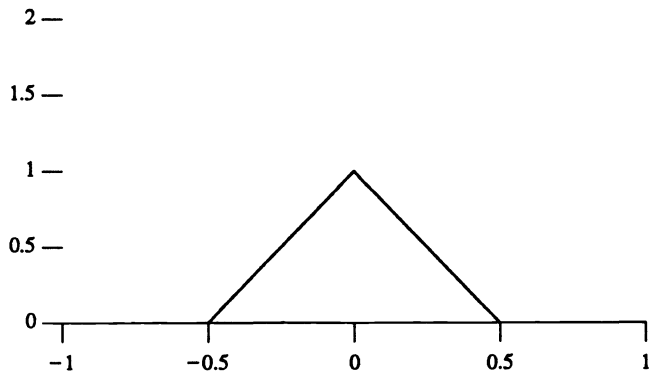


FIG. 5(a). *Input signal $f(t)$ has corners at $t = -0.5$, 0, and 0.5, with sizes 2, -4, and 2, respectively.*

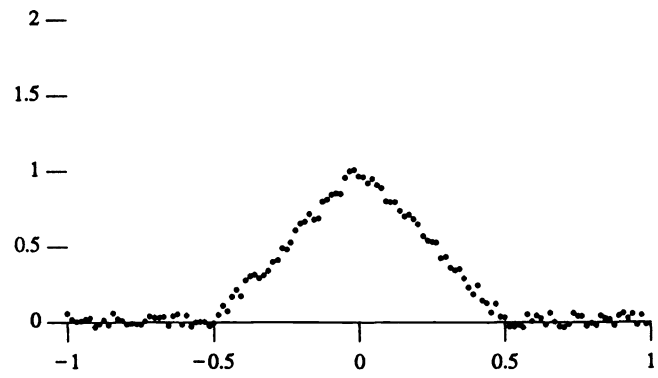


FIG. 5(b). *Sampled data from input signal $f(t)$ in Fig.* 5(a), *corrupted by noise.*

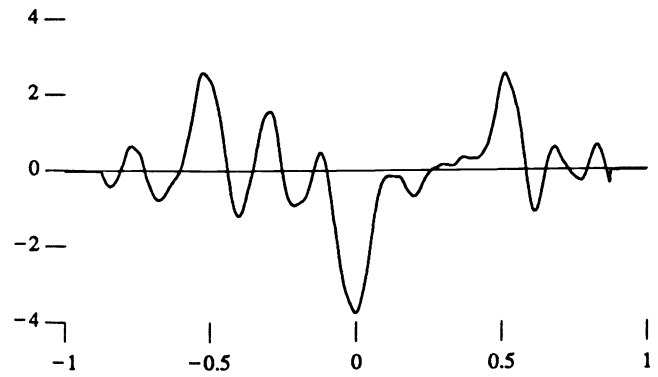


FIG. 5(c). *Filter response $\tilde{T} = \tilde{f} * \phi''$.*

by a random noise n, i.e., $\tilde{f} = f + n$, and the sampled noisy data are in Fig. 5(b). We know that

$$E\{[n * \phi'']\} = -0.002693 \quad \text{and} \quad E\{[n * \phi'']^2\} = 0.073068.$$

We apply Algorithm 6.1 and use the detector in Corollary 7.1 to detect corners. The filter response $\tilde{T}(t) = \tilde{f} * \phi''(t)$ is in Fig. 5(c), and has 20 extrema (see Table 2).

TABLE 2

Extrema	t_0	$\tilde{T}(t_0)$	$E_t\{[\cdot]\}$	$E_t\{[\cdot]^2\}$
max	−0.875000	0.016151	0.902324	83.490273
min	−0.843750	−0.199873	−0.649532	0.816400
max	−0.771484	0.332539	−0.574586	0.512505
min	−0.675781	−0.379699	−0.704066	1.089924
max*	−0.509766	2.255528	−0.065369	0.011573
min	−0.396484	−0.555298	−1.640432	4.508598
max	−0.291016	0.795009	−0.454970	0.284503
min	−0.209961	−0.452710	−0.602008	0.919224
max	−0.120117	0.245875	−4.847034	51.409389
min*	−0.000977	−3.871765	0.002698	0.000906
min	0.139648	−0.081026	8.462642	237.320831
min	0.200195	−0.341225	−0.047593	0.294620
max	0.300781	0.085180	−0.245052	4.827452
max	0.369141	0.179818	1.816077	13.401153
max*	0.510742	2.257509	−0.063539	0.015152
min	0.615234	−0.545270	−1.502938	4.431358
max	0.683594	0.314577	−0.344363	1.378340
min	0.772461	−0.132239	−1.096505	2.358089
max	0.828125	0.338393	−0.326054	0.249912
min	0.874023	−0.209279	−0.596498	0.981090

The extrema without an asterisk are rejected either by step (iv) of Algorithm 6.1 or because the size is almost zero. Therefore, we have three corners detected at $t_0 = -0.509766$, -0.000977, and 0.510742, with size 2.255528, -3.871765, and 2.257509, respectively.

Remark 8.1. In step (iv) of Algorithm 6.1, depending on whether the two quantities involved are *approximately* the same, we accept or reject the corresponding extremum. Two quantities are approximately the same if (an appropriate norm of) their difference is approximately zero. We also reject the extrema whose values are *approximately* zero. To claim that a value is approximately zero, a commonly used technique in engineering is *thresholding*, i.e., when the absolute value does not exceed a certain *threshold* value, we claim that it is approximately zero. Choosing an appropriate threshold value is a complicated process. It may depend on the implementation environment, the practical needs, or other factors. Intelligent and adaptive thresholding has been considered in edge detection literature (see [4]). However, most of them are not totally objective.

Signal-to-noise ratio ρ could be another alternative used to reject false responses, where ρ is given in (3.4). For practical reasons, we can use $\rho = [\tilde{T}(t_0)]^2 / E_t\{[\tilde{T} - \tilde{T}(t_0) \cdot \phi]^2\}$, instead. This method also depends on a threshold value of ρ.

For the quantities involved in the two examples, there is a noticeable difference between that of a discontinuity and that of a false response, and the threshold values are not difficult to find.

9. Curve fitting preserving discontinuities. We apply results of discontinuity detection to curve fitting. For a survey of curve fitting algorithms, see [15]. For noisy data, a commonly used technique is spline smoothing [8], [30]. However, such splines are smooth, i.e., a degree $2k+1$ smoothing spline has continuous $2k$th derivatives. The discontinuities such as jumps and corners will be smoothed out, and this is not desirable. To cope with such an anomaly, different approaches have been proposed for curve fitting, preserving discontinuities, such as multiple knots [8], splines in tension [32], splines with discontinuities [18], [20], [33], and many others. Most of these approaches assume that the location and type of discontinuities are known.

We first detect discontinuities. Assume that the function f is corrupted by an additive white noise n: $\tilde{f}=f+n$. From m sampled data

$$\tilde{f}(t_i)=z_i, \qquad i=1, 2, \cdots, m \tag{9.1}$$

we apply Algorithm 6.1 to detect discontinuities of different degrees. To distinguish different types of discontinuities, we use an index sequence $\{\xi_i : i=1, 2, \cdots, m\}$, where ξ_i is the degree of discontinuity at t_i. We then use cubic smoothing spline to fit data, preserving discontinuities. To construct a smoothing spline, we consider the class of functions defined on $[t_1, t_m]$ with absolutely continuous second-order derivatives except at t_i, where $\xi_i \leqq 2$. We consider the smoothing splines which, among all functions $z(t)$ in this class, minimize[5]

$$\theta(z)=\int_{t_1}^{t_m} [z''(t)]^2\, dt+\lambda \sum_{i=1}^{m} \left[\frac{z(t_i)-z_i}{\delta}\right]^2 \tag{9.2}$$

where δ is the standard deviation of the noise n.

General results of such smoothing-splines are obtained in [33], using reproducing kernels. Here we consider cubic smoothing splines only, and we derive the splines directly using the calculus of variation. We show the existence and uniqueness of the splines, interpretations of different continuity constraints, and implementations.

The splines are obtained by minimizing θ in (9.2) without imposing any boundary conditions. Therefore, it can be easily shown that it is decomposable with respect to function value discontinuities, where $\xi_i=0$. We first decompose the problem, and then construct the smoothing splines in each subinterval. After decomposition, $\xi_i \geqq 1$ within each subinterval, and we only have to construct smoothing splines with $\xi_i \geqq 1$. From now on, we confine ourselves to such smoothing splines. Without loss of generality, we assume that m data points are given in the unit interval: $0=t_1<t_2<\cdots<t_m=1$, where $\xi_i \geqq 1$. Following Reinsch [30], and using the calculus of variations [11], we can easily prove [20] Theorem 9.1.

THEOREM 9.1. *The smoothing spline σ is the unique cubic spline, a piecewise cubic polynomial, which satisfies the following conditions at knot t_i:*

(i) *For the two extreme knots $t_1=0$ and $t_m=1$:*

$$\sigma''(0+)=0, \qquad \sigma'''(0+)=-\frac{\lambda}{2}\frac{\sigma(0+)-z_1}{\delta^2},$$

$$\sigma''(1-)=0, \qquad \sigma'''(1-)=\frac{\lambda}{2}\frac{\sigma(1-)-z_m}{\delta^2}.$$

[5] For $\xi_i=0$, the corresponding term in (9.2) should be defined differently. However, the problem is decomposable with respect to jumps, and, after decomposition, $\xi_i \geqq 1$ within each subinterval. Therefore, we only have to consider the seminorm in (9.2).

(ii) *For the interior knots* t_i, $i = 2, 3, \cdots, m-1$,

(A) $\xi_i = 1$ (*a degree* 1 *discontinuity at* t_i):

$$\sigma(t_i-) = \sigma(t_i+), \qquad \sigma''(t_i-) = \sigma''(t_i+) = 0,$$

$$\sigma'''(t_i+) - \sigma'''(t_i-) = -\lambda \frac{\sigma(t_i) - z_i}{\delta^2}.$$

(B) $\xi_i = 2, 3$ (*a degree* 2 *discontinuity or with continuous second order derivative at* t_i):

$$\sigma(t_i-) = \sigma(t_i+), \quad \sigma'(t_i-) = \sigma'(t_i+), \quad \sigma''(t_i-) = \sigma''(t_i+),$$

$$\sigma'''(t_i+) - \sigma'''(t_i-) = -\lambda \frac{\sigma(t_i) - z_i}{\delta^2}.$$

In Case (ii)(A), the constraints at t_i, where $z(t)$ does not have continuous first-order derivatives, are called the *Weierstrass–Erdmann corner conditions* [11]. These conditions preserve the corners and avoid over-smoothing.

Since σ is a cubic spline, $\xi_i \leqq 3$. A degree 2 discontinuity detection process can distinguish a knot with $\xi_i = 2$ from that with $\xi_i = 3$. However, from (ii)(B), there is no need to detect degree 2 discontinuities, since the resulting smoothing splines are the same. Therefore, additional smoothness constraints are obtained from using the smoothing splines at the knots, where only the first-order derivatives are required to be continuous, i.e., $\xi_i = 2$. This observation can be used to simplify the discontinuity detection process. That is, we only have to locate the knots t_i with $\xi_i \leqq 1$. Experience shows that detecting discontinuities of degree 2 or higher is difficult, since they becomes very sensitive to noise.

Since $\sigma(t)$ is a cubic polynomial in each subinterval (t_{i-1}, t_i), $i = 2, 3, \cdots, m$, we have

$$\sigma(t) = A_i + B_i(t - t_i) + C_i(t - t_i)^2 + D_i(t - t_i)^3 \quad \text{for } t \in (t_{i-1}, t_i). \tag{9.3}$$

The cubic polynomials join at their common endpoints, and satisfy the conditions in Theorem 9.1. From these conditions, we obtain relations among the coefficients of the cubic polynomials in (9.3), and the problem is reduced to solving a system of linear equations, where the variables are the coefficients A_i, B_i, C_i, and D_i. After some routine manipulations (see Appendix 3) the coefficient matrix of the system becomes symmetric and strictly diagonally dominant with at most five nonzero entries in each row. Therefore, it has a unique solution, which can be obtained by iterative methods [16] or direct methods such as Gaussian elimination without pivoting [9].

The parameter λ establishes a compromise between our desire to obtain a smooth curve and to stay close to the given data. Different λ values determine different smoothing splines. As λ varies from zero to infinity, σ varies from the L_2-approximation to the interpolating-spline of the data. Choosing an appropriate λ was studied in [30], [33], and [35], and cross-validation techniques were developed [6], [10], [33], [40], [41].

10. Experiments. The method for discontinuity detection and curve fitting was implemented in the *C* language and was tested on a VAX 8550 machine running the UNIX operating system (UNIX is a trademark of AT&T).

In Fig. 6(a) there is a sampled data set, which is corrupted by a random noise. We use the detectors of Corollary 7.1. Applying Algorithm 6.1, there is a jump detected at 0.49 of size 0.247 and a corner detected at 0.264 of size −1.83.

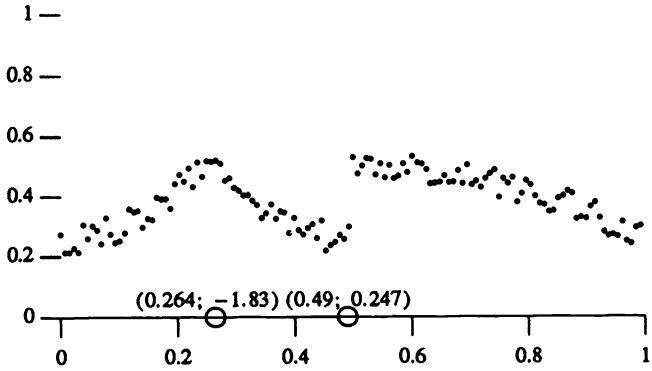


FIG. 6(a). *Sampled data from a function corrupted by noise. A jump at* 0.49 *of size* 0.247 *and a corner at* 0.264 *of size* −1.83 *are detected using Algorithm* 6.1.

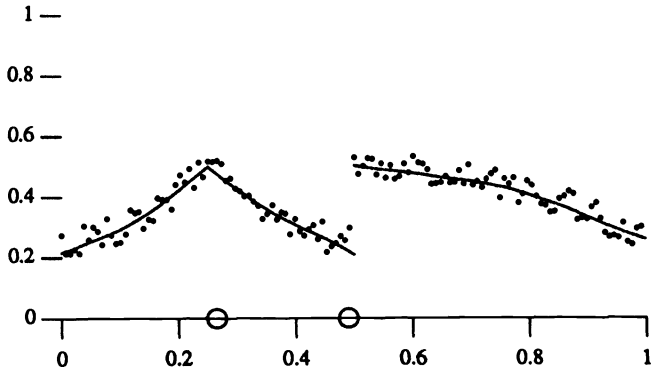


FIG. 6(b). *Fitting data using a cubic smoothing spline, preserving discontinuities, where* $\lambda = 100$.

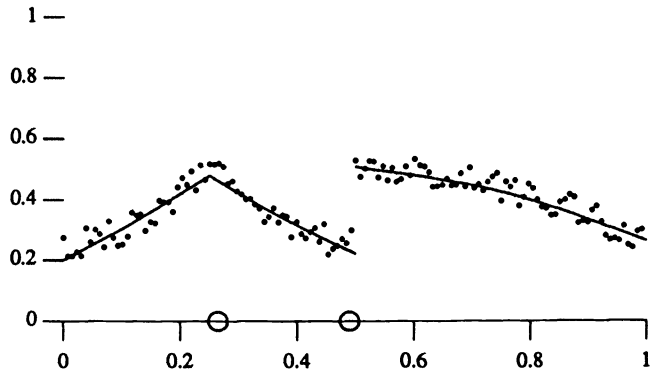


FIG. 6(c). *Fitting data using a cubic smoothing spline, preserving discontinuities, where* $\lambda = 10$.

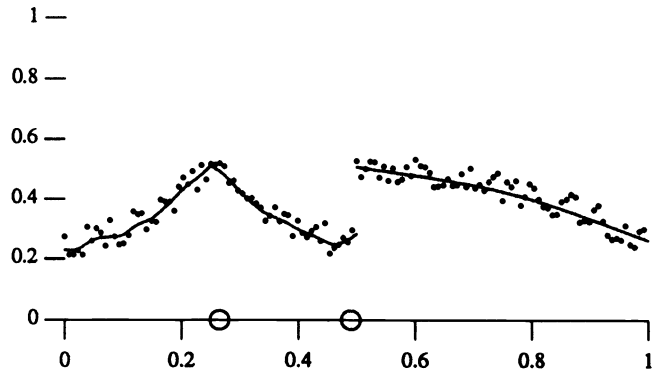


FIG. 6(d)

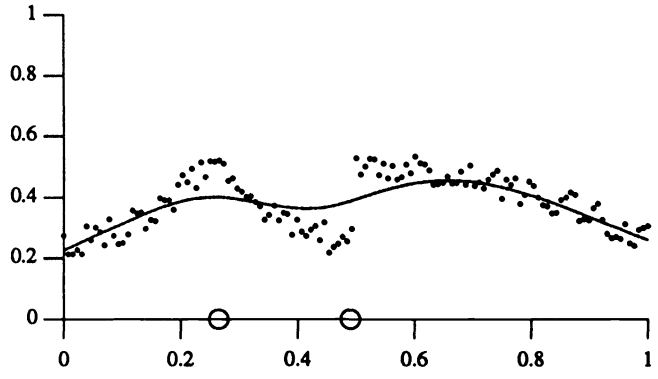


FIG. 6(e)

We then use the cubic smoothing spline (with $\lambda = 100$) described in § 9 to fit data, preserving the jump and corner (see Fig. 6(b)). For smaller λ values, the curves are more smooth but less close to data (see Fig. 6(c)), where $\lambda = 10$. For larger λ values, the curves stay closer to the data and wiggle (see Fig. 6(d)), where $\lambda = 10{,}000$. On the other hand, if we apply Reinsch's cubic smoothing splines, the jump and corner are smoothed out (see Fig. 6(e)).

We use detectors with support $[-A, A]$ for both jumps and corners, where $A = 0.125$. For $0.05 \leqq A \leqq 0.25$, results for jump detection are similar. For $A < 0.05$, the detector becomes very sensitive to noise. For $0.06 \leqq A \leqq 0.15$, results for corner detection are similar. For $A < 0.06$, the detector becomes very sensitive to noise. Note that the jump has a base $[0, 1]$ and the corner has a base $[0, 0.5]$. By Theorem 2.2, for jump detection, $A \leqq 0.25$, and for a corner detection, $A \leqq 0.125$. The moral is: choose a detector with support $[-A, A]$ such that A is close to $L/2$, where a discontinuity has a base of length $2L$.

11. Irregularities. We have discussed detection and measurement of ideal discontinuities. Besides noise, *irregularity* may further complicate the situation. We now discuss briefly the effects of irregularities to the detection and measurement processes of discontinuities.

For simplicity, assume that function f has a degree k ideal discontinuity at 0. It is superimposed by a function p, which represents the *irregularity*. The input function

is $f+p$. Let $p=p_1+p_2$, where

$$p_1(t)=\frac{p(t)+p(-t)}{2} \quad \text{and} \quad p_2(t)=\frac{p(t)-p(-t)}{2}.$$

Obviously, p_1 is an even function and p_2 is an odd function. p_1 is the *even irregularity*, and p_2 is the *odd irregularity*. If p is a degree r polynomial, then we have *polynomial irregularity* of degree r. The filter response becomes

$$T(t)=\alpha\phi(t)+(p*\phi^{(k+1)})(t)=\alpha\phi(t)+(p^{(k+1)}*\phi)(t) \tag{11.1}$$

where α is the size of the discontinuity. A routine computation yields Theorem 11.1.

THEOREM 11.1. *For degree k discontinuity detection, polynomial irregularities of degree no more than k have no effects on the location and size of the discontinuities detected. If k is even (odd), then even irregularities have no effects on the discontinuity size (location) and odd irregularities have no effects on the discontinuity location (size).* □

We use degree 0 discontinuity detection and measurement as an example. If the input function is shifted up, which corresponds to adding a constant, then it has no effects on the location and size of the discontinuities detected.

If the input function is tilted, which corresponds to adding a linear function $ct+h$, then it has no effects on the location of the discontinuities detected. However, from (11.1), the discontinuity size is modified by adding $(c*\phi)(0)=c\xi$, where

$$\xi=\int_{-A}^{A}\phi(\tau)\,d\tau.$$

If the input is superimposed by a quadratic polynomial p with $p'=dt+c$, then the irregularities will perturb not only the size but also the location of the discontinuities detected. Using (11.1), we check that the discontinuity size is perturbed by $c\xi$.

We now estimate the deviation of the discontinuity location. Using the detection theorem, the extremum of $T(t)$ is the discontinuity location. Setting the first order derivative of $T(t)$ to zero, a straightforward computation yields

$$T'(\delta)=\alpha\phi'(\delta)+d\xi=0$$

where δ is the deviation of the discontinuity location. Since $\phi'(0)=0$,

$$\delta=-\frac{d\xi}{\alpha\phi''(0)}.$$

It is obvious that the deviation is proportional to the magnitude of the leading coefficient of p, and that if the discontinuity size α is large compared with d, the perturbation is negligible.

If we add a cubic term to p, then it only has effects on the discontinuity size.

Therefore, if the degree of polynomial irregularities is not high, then they may not cause errors at all, or they may only cause errors in the measurement of discontinuity size. However, when the degree of polynomial irregularities is getting higher, they may cause errors both in the location and size of the discontinuities detected. On the other hand, experiments show that the major source of errors is still the noise [19].

12. Discontinuity detection in high dimensions. We have only studied one-dimensional discontinuity detection. For higher dimensions, the problem is also important and yet much more complicated. For the two-dimensional case, we could apply our algorithm along two orthogonal directions (or a few different directions), and

combine the discontinuities detected. However, this is essentially a one-dimensional approach, and for dimensions more than two, this is not an efficient method.

Is it possible to generalize our approach to high dimensions? The following arguments show that there is no direct generalization.

For simplicity, we consider discontinuity detection in two dimensions, and we only discuss degree 0 discontinuities. Assume that $f(x, y)$ has a degree 0 discontinuity of size 1 at $x=0$. More specifically, $f(x, y)=0$ if $x<0$ and $f(x, y)=1$ if $x \geqq 0$. We choose ϕ such that its partial derivatives with respect to x and y are not identically zero for $r=(x^2+y^2)^{1/2} \leqq A$, where $A>0$. Let $P(\partial/\partial x, \partial/\partial y)$ be an arbitrary polynomial of differential operators $\partial/\partial x$ and $\partial/\partial y$. Assume that $P(\partial/\partial x, \partial/\partial y)\phi$ forms a filter such that for some constant α,

$$f * \left(P\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)\phi \right) = \alpha \cdot \phi \quad \text{for } r \leqq A.$$

Denote $P\phi = P(\partial/\partial x, \partial/\partial y)\phi$, and a straightforward computation yields

$$\int_{-\infty}^{x} \int_{-\infty}^{\infty} (P\phi)(u, v)\, du\, dv = \alpha\phi(x, y) \quad \text{for } r \leqq A.$$

The partial derivative with respect to y of the left side is identically zero, and that of the right side is not. This is a contradiction.

13. Conclusion. We study discontinuity detection, classification, and measurement. We propose a detector, which consists of a pair of a pattern and a filter. After convolving the input signal with the filter, the problem is reduced to pattern search in the filter response. This approach enables us to detect the location of discontinuities of different degrees, to eliminate false discontinuities, and to measure the size of discontinuities.

We conclude with remarks on a few issues we have not discussed. Some are worth further investigation.

We have discussed discontinuity detection of a function. For applications such as image analysis and signal processing, we are dealing with *band-limited* signals [29]. More specifically, due to the effects of the *point-spread-function* G of the recording system, we have input signal $S=f * G$, and we want to detect discontinuities of the original signal f from samples of the bandlimited signal S. We can no longer apply our method directly. It can be shown that for a degree k discontinuity of the original signal $f(t)$ at t_0, there is a scaled pattern $\phi * G$ in the filter response $S * \phi^{(k+1)}$ at location t_0 and the scaling factor is the size of the discontinuity of $f(t)$ at t_0. For more details and applications to edge detection, see [19].

We have only considered additive noise. In practice, the signal may also be corrupted by a multiplicative noise m:

$$\tilde{S}(t) = m \int_R f(t-\tau)G(\tau)\, d\tau + n.$$

To cope with both additive and multiplicative noise, an optimal discontinuity detector remains to be explored.

We assume that we have some knowledge about the noise, such as ergodicity, mean, variance, power spectrum, etc. To obtain information of system noise as a research topic itself, see [1] and [5]. Modeling error may result in missing discontinuities or declaring false discontinuities.

For the approximate pattern matching in § 4, we verify that the left side of (4.1) is indeed a sample of a stochastic process $n * \phi^{(k+1)}$. In general, to verify that a function

is a sample of a stochastic process is a fundamental problem in statistics [26]. For our purpose, we verify its time mean and variance. Obviously, these are only necessary conditions, and two quantities can not characterize a process. If n is correlation-ergodic, similar to the proof of Theorem 4.1, we can compute the autocorrelation of $n * \phi^{(k+1)}$. We then compute the autocorrelation of the left side of (4.1) and compare it with that of $n * \phi^{(k+1)}$. Now we compare two functions instead of two pairs of quantities. Apparently, we do more verification. However, from our preliminary implementations, this approach does not provide more promising results.

Our technique is designed for ideal discontinuity detection. For the more general normal discontinuity detection, a practical method with theoretical justification is an important and yet challenging research topic.

In image analysis, there are a number of *optimal* detectors (filters), based on different optimality criteria. They cannot replace each other; there is no convincing evidence that any particular one is superior to the rest. Usually, the design of an *optimal* detector involves two stages: (i) Propose some optimality criteria; (ii) Construct a detector satisfying the criteria. In our approach, we discuss the general conditions for choosing the detectors. When constructing the detectors, we follow suit: we propose some optimality criteria and then find the detectors satisfying the criteria. Even though we also use the term *optimal detector*, it should be interpreted as *conditionally optimal*. The meaning of the word *optimal* is *most desirable* or *satisfactory*.[6] By this definition, *optimal* algorithms for discontinuity detection, classification, and measurement remain to be explored.

Appendix 1.

Proof of Theorem 7.1. We find ϕ, which minimizes (7.1) and satisfies conditions from Criteria 2.1. Since ϕ is symmetric, we only have to consider ϕ defined on $[0, A]$. Since $\phi \in W_{k+1}[-A, A]$ has support $[-A, A]$,

$$\phi^{(i)}(A) = 0, \qquad i = 0, 1, \cdots, k+1. \tag{A1}$$

Since ϕ is symmetric, $\phi^{(2j+1)}(-t) = -\phi^{(2j+1)}(t)$, $1 \leqq 2j+1 \leqq k+1$. On the other hand, $\phi^{(i)}(0-) = \phi^{(i)}(0+)$, $i = 0, 1, \cdots, k+1$, we have

$$\begin{aligned} &\phi(0) = 1, \\ &\phi^{(2j+1)}(0) = 0, \qquad 1 \leqq 2j+1 \leqq k+1. \end{aligned} \tag{A2}$$

We now derive the natural spline using the calculus of variation.

Let $\delta \in W_{k+1}[-A, A]$ satisfy conditions (A1) and (A2) with $\delta(0) = 0$. For all such δ,

$$\int_0^A [\phi^{(k+2)} + \delta^{(k+2)}]^2 \, dt - \int_0^A [\phi^{(k+2)}]^2 \, dt \geqq 0.$$

Thus,

$$\int_0^A \phi^{(k+2)} \delta^{(k+2)} \, dt \equiv 0.$$

Integrating by parts,

$$\int_0^A \phi^{(k+2)} \delta^{(k+2)} \, dt = [\phi^{(k+2)} \delta^{(k+1)}]_0^A - [\phi^{(k+3)} \delta^{(k)}]_0^A + \cdots + (-1)^{k+1} [\phi^{(2k+3)} \delta]_0^A + (-1)^{k+2} \int_0^A \phi^{(2k+4)} \delta \, dt.$$

[6] Webster's Third New International Dictionary.

We have the Euler equation

$$\phi^{(2k+4)} \equiv 0. \tag{A3}$$

Since $\delta^{(i)}(A) = 0$, $i = 0, 1, \cdots, k+1$,

$$\phi^{(k+2)}\delta^{(k+1)}(0) = \phi^{(k+3)}\delta^{(k)}(0) = \cdots = \phi^{(2k+2)}\delta'(0) = \phi^{(2k+3)}\delta(0) = 0.$$

Since $\delta(0) = 0$ and δ also satisfies condition (A2), i.e., $\delta^{(2j+1)}(0) = 0$, $1 \leqq 2j+1 \leqq k+1$,

$$\phi^{(2k+1)}\delta''(0) = \phi^{(2k-1)}\delta^{(4)}(0) = \cdots = \phi^{(i)}\delta^{(2k-i+3)}(0) = 0.$$

Therefore,

$$\phi^{(2k+1)}(0) = \phi^{(2k-1)}(0) = \cdots = \phi^{(i)}(0) = 0 \tag{A4}$$

where $i = 2\lceil (k+1)/2 \rceil + 1$.

The theorem follows from (A1)-(A4).

Appendix 2.

Proof of Theorem 7.2. For simplicity, assume that $A = \pi$. A routine computation shows that the natural splines in Corollary 7.1 are strongly complete.

We now show that for $k \geqq 2$ the natural splines in Theorem 7.1 are weakly complete. Assume on the contrary that they were not weakly complete. Then there are infinitely many n such that

$$c_n = \frac{1}{2\pi}\int_{-2\pi}^{2\pi} \phi(\tau)\cos(n\tau/2)\, d\tau = 0.$$

Since ϕ has support $[-\pi, \pi]$ and is symmetric,

$$\begin{aligned} c_n &= \frac{1}{\pi}\int_0^{\pi} \phi(\tau)\cos(n\tau/2)\, d\tau \\ &= \frac{1}{\pi}\left(\frac{2}{n}\right)\left\{\sin(n\pi/2)\sum_{i=0}^{k+1}(-1)^i\left(\frac{2}{n}\right)^{2i}\phi^{(2i)}(\pi) + \cos(n\pi/2)\right. \\ &\qquad \left.\cdot \sum_{i=0}^{k+1}(-1)^i\left(\frac{2}{n}\right)^{2i+1}\phi^{(2i+1)}(\pi) + (-1)^{k+2}\left(\frac{2}{n}\right)^{2k+3}\phi^{(2k+3)}(0)\right\}. \end{aligned} \tag{A5}$$

There are infinitely many $n \geqq 1$ such that $c_n = 0$:

$$\sum_{i=0}^{2k+3}\left(\frac{2}{n}\right)^i a_{n,i} = 0 \tag{A6}$$

where

$$a_{n,i} = \begin{cases} \sin(n\pi/2)(-1)^{i/2}\phi^{(i)}(\pi), & i = 0, 2, \cdots, 2k+2, \\ \cos(n\pi/2)(-1)^{(i-1)/2}\phi^{(i)}(\pi), & i = 1, 3, \cdots, 2k+1, \\ (-1)^{k+1}(\cos(n\pi/2) - 1)\phi^{(2k+3)}(0), & i = 2k+3. \end{cases}$$

Case (i). There are at least $2(2k+4)$ odd $n = 2m+1$ such that $c_n = 0$. If there are at least $(2k+4)$ such odd n where m is even, then $\sin(n\pi/2) = 1$, and $a_{n,i} = a_i$ for such n. From (A6), we have a system of homogeneous linear equations with variables a_i, where the number of equations is no less than the number of variables $2k+4$. The coefficient matrix is a Vandermonde matrix. Therefore, $a_i = 0$, $i = 0, 2, \cdots, 2k+2$, and $a_{2k+3} = 0$. We have $\phi^{(2i)}(\pi) = 0$, $i = 0, 1, \cdots, k+1$, and $\phi^{(2k+3)}(0) = 0$. On the other hand, since $\phi^{(2i+1)}(0) = 0$, $i = 0, 1, \cdots, k$, $\phi(t) \equiv 0$. If there are at least $(2k+4)$ such odd n where m is odd, similarly we have $\phi(t) \equiv 0$.

Case (ii). There are at least $2(2k+4)$ even $n=2m$ such that $c_n=0$. Either there are at least $(2k+4)$ such n where m is even or there are at least $(2k+4)$ such n where m is odd. For both cases, similar to Case (i), we have $\phi^{(2i+1)}(\pi)=0$, $i=0,1,\cdots,k$. On the other hand, since $\phi^{(i)}(\pi)=0$, $i=0,1,\cdots,k+1$, and $\phi^{(2i+1)}(0)=0$, $i=0,1,\cdots,k$, we have $\phi(t)\equiv 0$.

We have shown that if the pattern were not weakly complete, then $\phi(t)\equiv 0$. This is a contradiction, since $\phi(0)=1$.

Appendix 3. Cubic smoothing-splines preserving discontinuities. Let $0=t_1<t_2<\cdots<t_m=1$, and let $h_i=t_{i+1}-t_i$, $i=1,2,\cdots,m-1$. Assume that

$$\sigma(t_i)=a_i, \qquad i=1,2,\cdots,m$$

and that

$$\sigma''(t_i+)=b_i, \quad \sigma''(t_i-)=c_i, \quad i=1,2,\cdots,m$$

where $b_m=c_1=0$. We have for $t_i\leqq t\leqq t_{i+1}$, $i=1,2,\cdots,m-1$,

$$\sigma(t)=a_i+\left[\frac{a_{i+1}-a_i}{h_i}-\frac{h_i}{3}b_i-\frac{h_i}{6}c_{i+1}\right](t-t_i)+\frac{b_i}{2}(t-t_i)^2+\frac{c_{i+1}-b_i}{6h_i}(t-t_i)^3.$$

We now compute the parameters of $\sigma(t)$ using the constraints of Theorem 9.1. From (i) of Theorem 9.1,

$$b_1=c_m=0,$$

$$\text{(A7)} \qquad \frac{c_2}{h_1}=-\frac{\lambda}{2}\frac{a_1-z_1}{\delta^2} \quad \text{and} \quad \frac{b_{m-1}}{h_{m-1}}=-\frac{\lambda}{2}\frac{a_m-z_m}{\delta^2}.$$

We now apply (ii) of Theorem 9.1. We first use conditions on third-order derivatives, which are invariant with respect to the ξ_i values,

$$\text{(A8)} \qquad \frac{c_{i+1}-b_i}{h_i}-\frac{c_i-b_{i-1}}{h_{i-1}}=-\lambda\frac{a_i-z_i}{\delta^2}, \qquad i=2,3,\cdots,m-1.$$

The conditions on first- or second-order derivatives depend on the ξ_i values.

(α) $\xi_i=1$. From (ii)(A),

$$b_i=c_i=0$$

(β) $\xi_i=2, 3$. From (ii)(B),

$$b_i=c_i.$$

Note that for both cases, $b_i=c_i$, $i=1,2,\cdots,m$. Therefore, the conditions from the first-order derivatives are

$$\frac{1}{h_{i-1}}a_{i-1}-\left(\frac{1}{h_{i-1}}+\frac{1}{h_i}\right)a_i+\frac{1}{h_i}a_{i+1}=\frac{1}{6}h_{i-1}c_{i-1}+\frac{1}{3}(h_{i-1}+h_i)c_i+\frac{1}{6}h_ic_{i+1}.$$

Now (A7) becomes

$$\frac{c_2}{h_1}=-\frac{\lambda}{2}\frac{a_1-z_1}{\delta^2} \quad \text{and} \quad \frac{c_{m-1}}{h_{m-1}}=-\frac{\lambda}{2}\frac{a_m-z_m}{\delta^2}$$

and (A8) becomes

$$\frac{1}{h_{i-1}}c_{i-1}-\left(\frac{1}{h_{i-1}}+\frac{1}{h_i}\right)c_i+\frac{1}{h_i}c_{i+1}=-\lambda\frac{a_i-z_i}{\delta^2}.$$

We summarize as follows.

Constraint I.

$$b_1 = c_1 = b_m = c_m = 0, \tag{A9}$$

$$\frac{c_2}{h_1} = -\frac{\lambda}{2}\frac{a_1 - z_1}{\delta^2} \quad \text{and} \quad \frac{c_{m-1}}{h_{m-1}} = -\frac{\lambda}{2}\frac{a_m - z_m}{\delta^2}. \tag{A10}$$

Constraint II. For $i = 2, 3, \cdots, m-1$,

Case (α). $\xi_i = 1$.

$$b_i = c_i = 0. \tag{A11}$$

Case (β). $\xi_i = 2, 3$.

$$\frac{1}{h_{i-1}} c_{i-1} - \left(\frac{1}{h_{i-1}} + \frac{1}{h_i}\right) c_i + \frac{1}{h_i} c_{i+1} = -\lambda \frac{a_i - z_i}{\delta^2} \tag{A12}$$

and

$$\begin{aligned} &\frac{1}{h_{i-1}} a_{i-1} - \left(\frac{1}{h_{i-1}} + \frac{1}{h_i}\right) a_i + \frac{1}{h_i} a_{i+1} \\ &\quad = \frac{1}{6} h_{i-1} c_{i-1} + \frac{1}{3}(h_{i-1} + h_i) c_i + \frac{1}{6} h_i c_{i+1}. \end{aligned} \tag{A13}$$

We solve for a_i, b_i, and c_i, $i = 1, 2, \cdots, m$ as follows. Assume that there are $m - r - 2$ interior knots, where $\xi_i = 1$. We have $b_i = c_i = 0$, by Constraint II, Case (α). For the rest of r interior knots, where $\xi_i = 2, 3$, there are $2r$ equations from Constraint II, Case (β). We renumber the r subscripts preserving the ordering such that $m = r + 2$. Then (A12) becomes

$$\frac{1}{\tilde{h}_{i-1}} \tilde{c}_{i-1} - \left(\frac{1}{\tilde{h}_{i-1}} + \frac{1}{\tilde{h}_i}\right) \tilde{c}_i + \frac{1}{\tilde{h}_i} \tilde{c}_{i+1} = -\lambda \frac{\tilde{a}_i - \tilde{z}_i}{\delta^2} \tag{A14}$$

where $i = 2, 3, \cdots, r+1$. From (A9), (A10), and (A14),

$$P^T \tilde{c} = \frac{\lambda}{\delta^2}(\tilde{z} - \tilde{a}) \tag{A15}$$

where P is an $r \times (r+2)$ tridiagonal matrix:

$$p_{i-1,i} = \frac{1}{\tilde{h}_{i-1}}, \quad p_{i,i} = -\left(\frac{1}{\tilde{h}_{i-1}} + \frac{1}{\tilde{h}_i}\right), \quad p_{i+1,i} = \frac{1}{\tilde{h}_i} \quad i = 2, 3, \cdots, r+1$$

where

$$\begin{aligned} \tilde{a} &= (\tilde{a}_1, \cdots, \tilde{a}_{r+2})^T, \\ \tilde{b} = \tilde{c} &= (\tilde{c}_2, \cdots, \tilde{c}_{r+1})^T, \\ \tilde{z} &= (\tilde{z}_1, \cdots, \tilde{z}_{r+2})^T. \end{aligned}$$

On the other hand, (A13) becomes

$$\begin{aligned} &\frac{1}{\tilde{h}_{i-1}} \tilde{a}_{i-1} - \left(\frac{1}{\tilde{h}_{i-1}} + \frac{1}{\tilde{h}_i}\right) \tilde{a}_i + \frac{1}{\tilde{h}_i} \tilde{a}_{i+1} \\ &\quad = \frac{1}{6} \tilde{h}_{i-1} \tilde{c}_{i-1} + \frac{1}{3}(\tilde{h}_{i-1} + \tilde{h}_i) \tilde{c}_i + \frac{1}{6} \tilde{h}_i \tilde{c}_{i+1}, \qquad i = 2, 3, \cdots, r+1 \end{aligned} \tag{A16}$$

where $\tilde{a}_1 = a_1/2$, $\tilde{c}_1 = c_1$, $\tilde{a}_{r+2} = a_m/2$, $\tilde{c}_{r+2} = c_m$, $\tilde{z}_1 = z_1/2$, and $\tilde{z}_{r+2} = z_m/2$.

From (A9) and (A16),

$$P\tilde{a} = Q\tilde{c} \tag{A17}$$

where Q is an $r \times r$ positive definite and tridiagonal matrix:

$$q_{i,i} = \frac{1}{6}\tilde{h}_{i-1}, \quad q_{i,i+1} = q_{i+1,i} = \frac{1}{3}(\tilde{h}_{i-1} + \tilde{h}_i), \quad i = 2, 3, \cdots, r+1.$$

From (A15) and (A17),

$$\left(\frac{\delta^2}{\lambda} PP^T + Q\right)\tilde{c} = P\tilde{z}. \tag{A18}$$

The coefficient matrix of (A18) is obviously symmetric. A routine and yet lengthy computation shows that there are no more than five nonzero entries in a row and that the matrix is diagonally dominant. Therefore, (A18) can be solved using iterative methods [16] or direct methods such as Gaussian elimination without pivoting [9]. Thus, we have obtained all the values of b_i and c_i, $i = 1, 2, \cdots, m$. The values of a_i can be obtained from (A7) and (A8).

Acknowledgments. I am grateful for the careful reviews and insightful comments from the referees. Reference [18] was brought to my attention by both of them. I am indebted to Henry Landau and Jyh-jen Shiau for numerous stimulating discussions. Valuable comments and helpful discussions with Leopold Flatto, Eric Grosse, and Theo Pavlidis are deeply appreciated.

REFERENCES

[1] J. S. BENDAT, *Principles and Applications of Random Noise Theory*, John Wiley, New York, 1958.
[2] F. BERGHOLM, *Edge focusing*, IEEE Trans. Pattern Anal. Machine Intelligence, 9 (1987), pp. 726–741.
[3] R. A. BOIE AND I. J. COX, *Two dimensional edge recognition using matched and Wiener filters for machine vision*, in Proc. First Internat. Conference on Computer Vision, London, 1987, pp. 450–456.
[4] J. CANNY, *A computational approach to edge detection*, IEEE Trans. Pattern Anal. and Machine Intelligence, 8 (1987), pp. 679–698.
[5] K. R. CASTLEMAN, *Digital Image Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1979.
[6] P. CRAVEN AND G. WAHBA, *Smoothing noisy data with spline functions; estimating the correct degree of smoothing by the method of generalized cross-validation*, Numer. Math., 31 (1979), pp. 337–403.
[7] L. S. DAVIS, *A Survey of edge detection techniques*, Comput. Graphics Image Process., 4 (1975), pp. 248–270.
[8] C. DE BOOR, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.
[9] G. FORSYTHE AND C. MOLER, *Computer Solutions of Linear Algebraic Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1963.
[10] H. A. GAMBER, *Choice of an optimal shape parameter when smoothing noisy data*, Comm. Statist. A—Theory Methods, 8 (1979), pp. 1425–1435.
[11] I. M. GELFAND AND S. V. FOMIN, *Calculus of Variation*, Prentice-Hall, Englewood Cliffs, NJ, 1963.
[12] I. M. GELFAND AND G. E. SHILOV, *Generalized Functions*, Academic Press, New York, London, 1964.
[13] S. GEMAN AND D. GEMAN, *Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images*, IEEE Trans. Pattern Anal. Machine Intelligence, 6 (1984), pp. 721–741.
[14] W. E. L. GRIMSON AND T. PAVLIDIS, *Discontinuity detection for visual surface reconstruction*, Comput. Vision Graphics Image Process., 30 (1985), pp. 316–330.
[15] E. GROSSE, *A catalog of algorithms for approximation*, in Algorithms for Approximation II, J. Mason, ed., Chapman & Hall, London, 1989.
[16] L. A. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.
[17] B. K. P. HORN, *Robot Vision*, The MIT Press, Cambridge, MA, 1986.
[18] P. J. LAURENT AND F. I. UTRERAS, *Optimal smoothing of noisy data using spline function*, J. Approx. Theory Appl., 3 (1986), pp. 71–94.

[19] D. LEE, *Coping with discontinuities in computer vision: their detection, classification, and measurement*, IEEE Trans. Pattern Anal. Machine Intelligence, 12 (1990), pp. 321–344.

[20] D. LEE AND T. PAVLIDIS, *One-dimensional regularization with discontinuities*, IEEE Trans. Pattern Anal. Machine Intelligence, 10 (1988), pp. 822–829.

[21] D. LEE, T. PAVLIDIS, AND K. HUANG, *Edge detection through residual analysis*, in Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Ann Arbor, MI, IEEE Computer Society, Washington, DC, 1988, pp. 215–222.

[22] D. MARR AND E. C. HILDRETH, *Theory of edge detection*, Proc. Roy. Soc. London Ser. B, 207 (1980), pp. 187–217.

[23] J. L. MARROQUIN, S. MITTER, AND T. POGGIO, *Probabilistic solution of ill-posed problems in computational vision*, J. Amer. Statist. Assoc., 82 (1987), pp. 76–89.

[24] D. MUMFORD AND J. SHAH, *Boundary detection by minimizing functionals*, in Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, IEEE Computer Society, Washington, DC, 1985, pp. 22–26.

[25] V. S. NALWA AND T. O. BINFORD, *On detecting edges*, IEEE Trans. Pattern Anal. Machine Intelligence, 8 (1986), pp. 699–714.

[26] A. PAPOULIS, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 1984.

[27] T. PAVLIDIS, *Structural Pattern Recognition*, Springer-Verlag, Berlin, New York, 1977.

[28] M. J. D. POWELL, *Curve fitting by splines in one variable*, in Numerical Approximation to Functions and Data, J. G. Hayes, ed., Athlone Press, London, 1970, pp. 65–83.

[29] L. R. RABINER AND B. GOLD, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[30] C. H. REINSCH, *Smoothing by spline functions*, Numer. Math., 10 (1967), pp. 177–183.

[31] A. ROSENFELD AND M. THURSTON, *Edge and curve detection for visual scene analysis*, IEEE Trans. Comput., 20 (1971), pp. 562–567.

[32] D. G. SCHWARTZ, *An interpolation curve using a spline in tension*, J. Math. Phys., 45 (1966), pp. 312–317.

[33] J. H. SHIAU, *Smoothing spline estimation of functions with discontinuities*, Ph.D. thesis, Department of Statistics, University of Wisconsin, Madison, WI, 1985.

[34] J. F. SILVERMAN AND D. B. COOPER, *Bayesian clustering for unsupervised estimation of surface and texture models*, IEEE Trans. Pattern Anal. Machine Intelligence, 10 (1988), pp. 482–495.

[35] A. N. TIKHONOV AND V. Y. ARSENIN, *Solutions of Ill-posed Problems*, V. H. Winston, Washington, DC; John Wiley, New York, 1977.

[36] E. C. TITCHMARSH, *The Theory of Functions*, Oxford University Press, Oxford, U.K., 1952.

[37] V. TORRE AND T. A. POGGIO, *On edge detection*, IEEE Trans. Pattern Anal. Machine Intelligence, 8 (1986), pp. 147–163.

[38] J. F. TRAUB AND H. WOZNIAKOWSKI, *A General Theory of Optimal Algorithms*, Academic Press, New York, 1980.

[39] V. VOLTERRA, *Theory of Functionals and of Integral and Integro-Differential Equations*, Dover, New York, 1959.

[40] G. WAHBA, *Cross-validated spline methods for the estimation of multivariate functions from data on functionals*, in Statistics: An Appraisal, Proc. of the 50th Anniversary Conference, H. A. David and H. T. David, eds., Iowa State Statistical Laboratory, Iowa State University, Ames, IA, 1984.

[41] G. WAHBA AND J. WENDELBERGER, *Some new mathematical methods for variational objective analysis using splines and cross-validation*, Monthly Weather Rev., 108 (1980), pp. 36–57.

[42] N. WIENER, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*, John Wiley, New York, 1949.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 342-361, March 1991
© 1991 Society for Industrial and Applied Mathematics
007

REGULARIZED SOLUTIONS TO THE AEROSOL DATA INVERSION PROBLEM*

J. KENNETH WOLFENBARGER† AND JOHN H. SEINFELD†

Abstract. Regularized solutions to the aerosol data inversion problem are presented. An approximate form of generalized cross validation is developed that is applicable to this linearly constrained inverse problem. The results obtained with this algorithm for choosing the smoothing parameter are compared with those obtained by the method of discrepancy and by minimizing an unbiased estimate of the inverted errors. Examples are presented that demonstrate the importance of using generalized cross validation to choose the smoothing parameter when the magnitude of the errors in the data is difficult to estimate.

Key words. inverse problems, generalized cross validation, aerosols

AMS(MOS) subject classification. 45

1. Introduction. We consider the problem of determining a suitable aerosol size distribution $f(x)$ from a set of p measurements $\mathbf{y}^M$. Here, the product $f(x)\,dx$ can represent the mass, surface area, or number of particles in the size interval $(x, x+dx)$. Since the diameter of aerosol particles can range over several orders of magnitude, it is convenient to define particle size by

$$x = \frac{\log (d/d_1)}{\log (d_2/d_1)} \tag{1}$$

where d is the particle diameter, and (d_1, d_2) is the interval of particle diameters over which the solution will be computed. For conventional aerosol instruments the relationship between the aerosol size distribution and the data is

$$y_i^M = \int_0^1 k_i(x) f(x)\,dx + \varepsilon_i^t, \qquad i = 1, \cdots, p \tag{2}$$

where $k_i(x)$ is commonly referred to as the kernel function and is proportional to the known response of the instrument to a monodisperse aerosol sample of size x. Note that the interval (d_1, d_2) must be defined so that the support of the product $k_i(x)f(x)$ lies in the interval $(0, 1)$. We will assume that the errors in the data $\boldsymbol{\varepsilon}^t$ are independent normal random variables. This assumption is rarely valid; however, as discussed in [19], if we can represent $\boldsymbol{\varepsilon}^t$ as a linear combination of independent normal random variables, error sources for example, then (2) can be transformed so that the resulting $\boldsymbol{\varepsilon}^t$ are independent normal random variables with unit variance.

Further information about the size distribution can be provided by linear inequality constraints on $f(x)$ that arise because the size distribution is nonnegative, bounds may exist for the true values of the measured data, the value of the distribution may be known at certain x's (e.g., the endpoints), and as discussed in [19], certain linear combinations of data may be known.

The problem of finding solutions to (2) is ill-posed: for a given $\mathbf{y}^M$ and $\boldsymbol{\varepsilon}^t$, a solution may not exist, and if a solution exists then it is usually not unique. When the number of data is large, we may attempt to eliminate the ill-posedness by approximating (2) by an invertible system of linear equations, but this approach has long been known to yield unstable results [16].

* Received by the editors August 28, 1989; accepted for publication November 11, 1989. This research was supported by National Science Foundation grant ATM-8503103.

† Department of Chemical Engineering, California Institute of Technology, Pasadena, California 91125.

Regularization is a powerful technique for finding solutions to the ill-posed (2). Here, the set of solutions defined by (2) is replaced by the set of smooth solutions $f_\lambda(x)$, $\lambda \geqq 0$, that minimize

$$\sum_{i=0}^{p} \left(\int_0^1 k_i(x) f(x)\, dx - y_i^M \right)^2 + \lambda J(f) \tag{3}$$

subject to constraints. $J(f)$ is the regularizing functional that penalizes undesirable solutions; for example, in this paper we use

$$J(f) = \int_0^1 (f^{(m)}(x))^2\, dx \tag{4}$$

since high-frequency oscillations are undesirable. The unspecified λ measures the relative importance of the solution's smoothness and its fidelity to the data. If λ is too small then smoothness becomes unimportant, and the solution will be too sensitive to perturbations in the data and will likely exhibit unrealistic oscillations. On the other hand, if λ is too large then structure that exists in the true distribution will be suppressed.

One technique for choosing λ that we have used with some success and that is straightforward to apply to this constrained inversion is the method of discrepancy [5]: set $\lambda = \lambda_D$ where

$$\|\mathbf{y}_{\lambda_D}^R - \mathbf{y}^M\|^2 = \mathbf{E}\|\mathbf{y}^t - \mathbf{y}^M\|^2 \tag{5}$$

and where the superscript t denotes true, the superscript R denotes predicted, boldface variables represent column vectors, and $\|\cdot\|$ denotes the Euclidean norm. Wahba and others [18] have presented theoretical and numerical evidence on related inversion problems that shows that λ_D tends to oversmooth $f_\lambda(x)$. The weakness of choosing the solution to match the statistics of the error in the data is made clear in following example presented in [19]. If

$$y_i^t - y_i^M = (\varepsilon_i + \varepsilon p + 1) y_i^t, \qquad i = 1, \cdots, p$$

where ε_i is an independent normal random variable with a variance of σ^2, then we can show that

$$\lambda_D \to \infty \quad \text{as } \sigma p \to 1$$

independent of instruments and the data. Here λ_D is a poor regularization parameter: it overestimates the amount of smoothing, and the oversmoothing becomes worse as more data are included in the inversion.

An additional difficulty with using λ_D is that the standard deviations of the errors in the data are usually not well known, and a poor estimate of the magnitude of these errors can easily lead to unrealistic solutions [19].

In this paper we use methods similar to those of Kimeldorf and Wahba [12], and Villalobos and Wahba [17] to find solutions to the aerosol data inversion problem subject to inequality constraints. Smoothing parameters are chosen to minimize an unbiased estimate of the inverted errors, and by generalized cross validation (GCV) when the number of data are large. We present examples that demonstrate the importance of taking the constraints into consideration when using GCV to choose λ. Additional examples compare the solutions obtained while using λ_D with those obtained for the regularization parameters described in this paper and demonstrate the importance of using the generalized cross validation approach as one's ability to estimate the magnitude of the errors diminishes.

2. Solution to the regularized problem. The solution to (2) is not easy to find in general because the subset $\{f(x): f(x) \geqq 0\}$, of $\mathscr{L}^2(0, 1)$ for example, defines an infinite cone, and minimization techniques are not well developed on this type of set.

One approach we can use is described in [17] in relation to multidimensional interpolation. Here, we assume $f(x)$ lies in the Sobolev space $W_2^m(0, 1)$, where $f^{m}(x) \in \mathscr{L}^2(0, 1)$ and $f^{(i)}(x)$, $i = 1, \cdots, m-1$, is absolutely continuous [1]. The inner product is defined as

$$\langle f, g\rangle = \sum_{i=0}^{m-1} f^{(i)}(0)g^{(i)}(0) + \int_0^1 f^{(m)}(x)g^{(m)}(x)\, dx. \tag{6}$$

The constraint that the solution be nonnegative is replaced with the finite set of constraints

$$f(x_i) \geqq 0, \qquad i = 1, \cdots, \bar{n} \leqq n \tag{7}$$

so that there are a finite number of linear inequality constraints, and we write these as

$$c_i(f) \geqq b_i, \qquad i = 1, \cdots, n. \tag{8}$$

If $T(f)$ is a linear functional on $W_2^m(0, 1)$, then the Riesz representation theorem [1] guarantees the existence of the representer of T such that

$$T(f) = \langle \theta, f\rangle \tag{9}$$

for all $f \in W_2^m(0, 1)$. For example, if

$$T(f) = \int_0^1 t^0(x)f(x)\, dx \tag{10}$$

then integration by parts and the independence on f shows that θ satisfies the differential equation

$$\theta^{(2m)}(x) = (-1)^m t^0(x) \tag{11}$$

with the boundary conditions

$$(-1)^i\theta^{(m+i)}(0) - \theta^{(m-1-i)}(0) = \theta^{(m+i)}(1) = 0, \qquad i = 0, \cdots, m-1. \tag{12}$$

We readily obtain

$$(-1)^m\theta(x) = t^{2m}(x) + \sum_{i=0}^{m-1} \frac{x^i}{i!}(t^{i-1}(0) + (-1)^{m-k}t^{2m-i}(0)) \tag{13}$$

where

$$t^i(x) = \int_x^1 t^{i-1}(s)\, ds, \qquad i = 1, \cdots, 2m. \tag{14}$$

Thus, if we define the representers $\boldsymbol{\kappa}(x)$ and $\boldsymbol{\gamma}(x)$ by

$$\langle f, \kappa_i\rangle = \int_0^1 k_i(x)f(x)\, dx, \tag{15}$$

$$\langle f, \gamma_i\rangle = c_i(f), \tag{16}$$

then $f_\lambda(x)$, the minimizer of (3), is also the minimizer of

$$\sum_{i=0}^{p} (\langle \kappa_i, f\rangle - y_i^M)^2 + \lambda\langle P_1 f, P_1 f\rangle \tag{17}$$

subject to

$$\langle \gamma_i, f \rangle = b_i, \qquad i = 1, \cdots, n \tag{18}$$

where P_1 is the projection on $W_2^m(0, 1)$ defined by

$$P_1(f(x)) = f(x) - \sum_{i=0}^{m-1} \frac{f^{(i)}(0)x^i}{i!}. \tag{19}$$

We denote the range and kernel of P_1 on $W_2^m(0, 1)$ by $H_0(0, 1)$ and $H_1(0, 1)$, respectively.

There exist coefficients $\mathbf{k}$, $\mathbf{c}$, and $\mathbf{w}$ such that the minimizer of (3) as well as any element of $W_2^m(0, 1)$ can be written as

$$f_\lambda(x) = \sum_{i=0}^{p} \mathbf{k}_i P_1 \kappa_i(x) + \sum_{i=0}^{n} \mathbf{c}_i P_1 \gamma_i(x) + \sum_{i=0}^{m-1} \mathbf{w}_i \omega_i(x) + f_\perp(x) \tag{20}$$

where

$$\omega_i(x) = \frac{x^{i-1}}{i!}, \qquad i = 1, \cdots, m \tag{21}$$

is a basis for $H_0(0, 1)$, and where $f_\perp(x) \in H_1(0, 1)$ is orthogonal to all $\kappa_i(x)$ and $\gamma_i(x)$. This representation may not be unique, however, because the set of representers $\{\boldsymbol{\gamma}(x), \mathbf{k}(x)\}$ can be linearly independent. For example, we may require the solution to satisfy

$$b_{Li} \leqq \int_0^1 k_i(x) f(x)\, dx \leqq b_{Ui}. \tag{22}$$

Therefore, it is desirable to identify a subset of the representers $\{\hat{\boldsymbol{\gamma}}(x), \hat{\boldsymbol{\kappa}}(x)\}$ with $\hat{p} + \hat{n} \leqq p + n$ elements, such that $\{P_1\hat{\boldsymbol{\gamma}}, P_1\hat{\boldsymbol{\kappa}}\}$ are linearly independent, and such that the span of $\{P_1\hat{\boldsymbol{\gamma}}, P_1\hat{\boldsymbol{\kappa}}\}$ equals the span of $\{P_1\boldsymbol{\gamma}, P_1\boldsymbol{\kappa}\}$. With $\{\hat{\boldsymbol{\gamma}}(x), \hat{\boldsymbol{\kappa}}(x)\}$ identified, elements of $W_2^m(0, 1)$ can be written uniquely as

$$f_\lambda(x) = \sum_{i=0}^{p} \hat{\mathbf{k}}_i P_1 \hat{\kappa}_i(x) + \sum_{i=0}^{n} \hat{\mathbf{c}}_i P_1 \hat{\gamma}_i(x) + \sum_{i=0}^{m-1} \mathbf{w}_i \omega_i(x) + f_\perp(x). \tag{23}$$

Substituting (23) into (17) shows that $f_\perp(x) = 0$ and that $\hat{\mathbf{k}}$, $\hat{\mathbf{c}}$, and $\mathbf{w}$ minimize

$$\| \mathcal{K}\mathbf{x} - \mathbf{y}^M \|^2 + \lambda \| \mathcal{D}\mathbf{x} \|^2 \tag{24}$$

subject to

$$\mathcal{C}\mathbf{x} \geqq \mathbf{b} \tag{25}$$

where

$$\mathcal{K} = [\langle P_1\boldsymbol{\kappa}, P_1\hat{\boldsymbol{\kappa}}^T \rangle, \langle P_1\boldsymbol{\kappa}, P_1\hat{\boldsymbol{\gamma}}^T \rangle, \langle P_0\boldsymbol{\kappa}, \boldsymbol{\omega}^T \rangle], \tag{26}$$

$$\mathcal{D} = [\langle P_1\hat{\boldsymbol{\kappa}}, P_1\hat{\boldsymbol{\kappa}}^T \rangle, \langle P_1\hat{\boldsymbol{\gamma}}, P_1\hat{\boldsymbol{\gamma}}^T \rangle, 0], \tag{27}$$

$$\mathcal{C} = [\langle P_1\boldsymbol{\gamma}, P_1\hat{\boldsymbol{\kappa}}^T \rangle, \langle P_1\boldsymbol{\gamma}, P_1\hat{\boldsymbol{\gamma}}^T \rangle, \langle P_0\boldsymbol{\gamma}, \boldsymbol{\omega}^T \rangle], \tag{28}$$

$$\mathbf{x} = \begin{pmatrix} \hat{\mathbf{k}} \\ \hat{\mathbf{c}} \\ \mathbf{w} \end{pmatrix}. \tag{29}$$

Here $\langle P_1\boldsymbol{\gamma}, P_1\boldsymbol{\kappa}^T \rangle$, for example, represents a matrix with ijth component equal to $\langle P_1\gamma_i, P_1\kappa_j \rangle$, and if A is an $n \times m$ matrix and B is an $n \times p$ matrix, then $[A, B]$ denotes the $n \times m + p$ augmented matrix with the first m columns corresponding to A and the last p columns corresponding to B.

If the feasible set defined by (25) is not empty, then a minimizer of (24) must exist. If the rank of the matrix $\langle \boldsymbol{\omega}, \boldsymbol{\kappa}^T \rangle$ is m, then it is straightforward to show that the minimizer is unique; this condition is sufficient though not necessary. A more detailed discussion of the existence of a unique solution can be found in [19].

2.1. Some numerical considerations. To calculate the elements of these matrices, $\langle P_1\kappa_i, P_1\kappa_j \rangle$, for example, (6) and (13) are not suitable for numerical calculations, and simpler expressions are readily obtained for a general linear functional by introducing the reproducing kernel for $W_2^m(0, 1)$ [12]. The reproducing kernel $K(x, y)$ for the Hilbert space H satisfies the following:

1) $K_x(y) \in H$;
2) $K_x(y) = K_y(x)$;
3) $\langle K_x(y), u(y) \rangle = u(x)$;

where we write $K_x(y)$ to mean that $K(x, y)$ should be viewed as a function of y. From this definition we can see that the representer $\theta_i(x)$ for the linear functional T_i satisfies

$$\theta_i(x) = T_i K_x(y) \tag{30}$$

and

$$\langle \theta_i, \theta_j \rangle = T_i T_j K(x, y). \tag{31}$$

Note also that if K_i is the reproducing kernel for H_i where $W_2^m(0, 1) = \sum_{\oplus} H_i$, then we can show [10] that $K = \sum K_i$.

From the definitions we can write

$$\int_0^1 u(y)\delta_x(y)\, dy = \langle K_{1;x}(y), u(y) \rangle \tag{32}$$

$$= \int_0^1 K_{1;x}^{(m)}(y) u^{(m)}(y)\, dy \tag{33}$$

where $\delta_x(y)$ is the dirac delta function. Integration by parts plus the independence of $K_{1;x}(y)$ on $u(y)$ shows that $K_{1;x}(y)$ satisfies the distributional equation

$$K_{1;x}^{(2m)}(y) = (-1)^m \delta_x(y) \tag{34}$$

subject to the boundary conditions

$$K_{1;x}^{(m+i)}(1) = K_{1;x}^{(i)}(0) = 0, \qquad i = 0, \cdots, m-1 \tag{35}$$

or

$$K_1(x, y) = \sum_{i=0}^{m-1} \frac{(-1)^k x^{m-1-i} y^{m+i}}{(m-1-i)!(m+i)!}, \qquad x \geqq y. \tag{36}$$

We can easily verify that

$$K_0(x, y) = \sum_{i=0}^{m-1} \frac{x^i y^i}{2(i!)}. \tag{37}$$

Thus

$$\langle P_1\kappa_i, P_1\kappa_j \rangle = \int_0^1 \int_0^x (k_i(x)k_j(y) + k_i(y)k_j(x)) \times \sum_{\nu=0}^{m-1} \frac{(-1)^\nu x^{m-1-\nu} y^{m+\nu}}{(m-1-\nu)!(m+\nu)!}\, dy\, dx \tag{38}$$

or, if it is not efficient to keep the sum inside the integral, we can show that

$$\langle P_1\kappa_i, P_1\kappa_j\rangle = \frac{(-1)^m}{(2m-1)!}\int_0^1\int_0^x k_i(x)k_j(y)(x-y)^{2m-1}\,dy\,dx + \sum_{\nu=0}^{m-1}\frac{(-1)^\nu M_{ki}^{m-\nu-1}M_{kj}^{m+\nu}}{(m-\nu-1)!(m+\nu)!} \tag{39}$$

where M_{ki}^j is the jth moment of $k_i(x)$ over the interval $(0, 1)$. Efficient algorithms for these two-dimensional integrals have been developed [6], [11], [13]. Also, if $\gamma_i(x)$ is the evaluation functional, then

$$\langle P_1\gamma_i, P_1\kappa_j\rangle = \frac{(-1)^m}{(2m-1)!}\int_0^{x_i} k_j(y)(x_i-y)^{2m-1}\,dy + \sum_{\nu=0}^{m-1}\frac{(-1)^k x_i^{m+\nu}M_{kj}^{m-1-\nu}}{(m-1-\nu)!(m+\nu)!}. \tag{40}$$

We find the minimizer of (24) with the active-set algorithm described in [7] and [8]. Briefly, a minimum is found by iteratively choosing a set of active constraints, constraints that will be treated as equalities, and performing the minimization while ignoring the remaining inequalities. A single constraint is added and/or deleted per iteration.

The upper triangular matrix R that satisfies

$$R^TR = \mathcal{K}^T\mathcal{K} + \lambda\mathcal{D}^T\mathcal{D} = H^0 \tag{41}$$

is used in place of the Hessian H^0 to reduce the computation and increase the numerical stability. The straightforward method of computing R is to simply form the Hessian and then perform the Cholesky decomposition; however, for small λ this approach can lead to a troublesome loss of precision. Instead, if λ is small then we compute upper triangular Δ_R that satisfies

$$R^TR = (\hat{R}+\Delta_R)^T(\hat{R}+\Delta_R) \tag{42}$$

where $\hat{R}$ is the upper triangular matrix that satisfies

$$\hat{R}^T\hat{R} = \mathcal{K}^T\mathcal{K}. \tag{43}$$

Note that $\hat{R}$ will be available if the minimizer of

$$\sum_{i=1}^{p}\left(\int_0^1 k_i(x)f(x)\,dx - y_i^M\right)^2 \tag{44}$$

is computed. In the Cholesky decomposition, we compute the ith row of R given the first $i-1$ rows by solving

$$R_{i,i}R_{i,j} = H_{i,j}^{i-1}, \qquad j = i, \cdots, n \tag{45}$$

where

$$H^i = H^{i-1} - \mathbf{r}_i\mathbf{r}_i^T, \qquad i = 1, \cdots, n. \tag{46}$$

The column vector $\mathbf{r}_i$ corresponds to the ith column of R^T. Similarly, we can compute the ith row of Δ_R given the first $i-1$ rows by solving

$$\Delta_{Ri,i}\Delta_{Ri,j} = D_{i,j}^{i-1}, \qquad j = i, \cdots, n \tag{47}$$

where

$$D^i = D^{i-1} - \hat{\mathbf{r}}_i\Delta_{\mathbf{r}_i}^T - \Delta_{\mathbf{r}_i}\hat{\mathbf{r}}_i^T - \Delta_{\mathbf{r}_i}\Delta_{\mathbf{r}_i}^T \tag{48}$$

and

$$D^0 = \lambda \mathscr{D}^T \mathscr{D} \tag{49}$$

where the column vector $\hat{\mathbf{r}}_i$ corresponds to the ith column of $\hat{R}^T$, and the column vector $\Delta_{\mathbf{r}_i}$ corresponds to the ith column of Δ_R^T. Note that the product $\hat{R}^T\hat{R}$ is not formed. Roughly speaking, we avoid precision loss by working with terms on the order of $\sqrt{\lambda}\mathscr{D}$ relative to $\hat{R}$ in contrast to working with terms on the order of $\lambda\mathscr{D}^T\mathscr{D}$ relative to $\hat{R}^T\hat{R}$.

One feature of the algorithm described in [7] is that the inequality constraints are divided into general constraints and bound constraints:

$$\mathbf{b}_U \leqq \begin{bmatrix} I \\ A \end{bmatrix} \mathbf{x} \leqq \mathbf{b}_L \tag{50}$$

where the matrix I corresponds to the bound constraints and the A corresponds to the general constraints. If, for example, $\mathbf{x}$ is a $\hat{p}$-vector, $\hat{m}$ general constraints are active, and $\hat{n}$ bound constraints are active, we can show [7] that $\hat{n}(6\hat{p} - 2\hat{m} - (3/2)\hat{n})$ multiplications are saved per iteration when a single general constraint is added if the constraints are treated separately instead of treating them all as general constraints.

In the preceding formulation for the solution to (3), note that all of the constraints must be treated as general constraints since the condition $f(x_i) \geqq 0$ leads to a matrix with entries given in (28) instead of the identity matrix. The calculations will be expensive if a large number of constraints of this type are chosen and the initial active set chosen is not close to the final active set. Additionally, computing the entries for the matrices $\mathscr{K}$, $\mathscr{D}$, and $\mathscr{C}$ can become expensive. For example, if we have 40 measurements and 40 constraints, then we will need to numerically evaluate 800 double and single integrals.

An undesirable feature of the previous formulation is that for any finite number of constraints of the form $f(x_i) \geqq 0$, we always have

$$\sum_{i=1}^{p} \left(\int_0^1 k_i(x) f_\lambda(x)\, dx - y_i^M \right)^2 \to 0 \tag{51}$$

as $\lambda \to 0$, when in fact we encounter cases where the magnitude of

$$\inf_{f \geqq 0} \sum_{i=0}^{p} \left(\int_0^1 k_i(x) f(x)\, dx - y_i^M \right)^2 \tag{52}$$

is significant. This can occur, for example, if

$$\sum_{i \neq j} k_i(x) > k_j(x) \quad \text{and} \quad \sum_{i \neq j} y_i^M \leqq \alpha y_j^M \tag{53}$$

where $\alpha < 1$. The inability of the previous formulation to predict that (52) can be nonzero could become important if the method used to choose λ depends on the difference between the predicted data and the measured data.

2.2. A finite-dimensional solution. An alternative, though similar, approach to solving (3) is to assume that the solution can be approximated as an n-dimensional linear spline. The difficulty associated with this assumption is obvious: how can we be certain that n is large enough to capture all of the information provided by the data? For now we simply assume that n is of the same order of magnitude as p, for example, $n = 2p$. Note that in this setting, $f_\lambda(x)$ is able to reflect the lack of a positive solution to (2) for $\boldsymbol{\varepsilon}^t = \mathbf{0}$.

With this assumption we seek $\mathbf{f}_\lambda$, the solution to the following quadratic program:

$$\text{minimize} \quad \mathbf{f}^T(\mathcal{K}_n^T\mathcal{K}_n + p\lambda D^{(m)^T}D^{(m)})\mathbf{f} - 2\mathbf{y}^{M^T}\mathcal{K}_n\mathbf{f} \tag{54}$$

$$\text{subject to} \quad C\mathbf{f} \geqq b \tag{55}$$

where

$$f_\lambda(x) = \mathbf{g}^T(x)\mathbf{f}_\lambda, \tag{56}$$

and

$$\mathcal{K}_{ni,j} = \int_0^1 g_j(x)k_i(x)\,dx \tag{57}$$

and where $\mathcal{K}_n$ is the matrix that represents the integral operator for the linear splines and $\{g_i(x), i = 1, \cdots, n\}$ is a basis for the n-dimensional linear splines. $D^{(m)}$ is a difference matrix such that if $\mathbf{f}^T\mathbf{g}(x)$ is an approximation to $f(x)$, then

$$\int_0^1 (f^{(m)}(x))^2\,dx \approx \mathbf{f}^T D^{(m)^T}D^{(m)}\mathbf{f}. \tag{58}$$

For example, if the splines are equally spaced we use

$$D^{(1)}_{i,j} = \frac{\delta_{i,i} - \delta_{i,i-1}}{n-1}, \tag{59}$$

and for m even

$$D^{(m)} = (D^{(1),T}D^{(1)})^{m/2}. \tag{60}$$

In the remainder of this paper we assume that the solution can be approximated as an n-dimensional linear spline. If the data are consistent and few constraints are required to keep the solution positive, then it would be better to keep the solution in $W_2^m(0, 1)$; however, the following analysis is similar for both approaches.

We can show that the solution in the absence of constraints $C = 0$ is

$$D^{(m)}\mathbf{f}_\lambda = K_D^T(K_DK_D^T + \lambda pI)^{-1}\mathbf{y}^M \tag{61}$$

where $K_D = D^{-(m)}\mathcal{K}_n$ and $D^{-(m)}$ is the inverse of $D^{(m)}$. Note that K_D can be computed from $\mathcal{K}_n$ with only $\frac{1}{2}mn(n-1)$ additions and that the inverse defined in (61) is order p instead of the larger n. Following [4] we observe that the predicted data are linearly related to the measured data

$$\mathbf{y}_\lambda^R = K_DK_D^T(K_DK_D^T + p\lambda I)^{-1}\mathbf{y}^M = A(\lambda)\mathbf{y}^M. \tag{62}$$

If the set of constraints that are satisfied at $\mathbf{f}_\lambda$ are known, then a simple expression for the solution can be found by projecting the quadratic functional onto the null-space of the active constraints. Let the set of linearly independent constraints that are active at the minimizing $\mathbf{f}_\lambda$ be denoted by

$$C_a\mathbf{f} = \mathbf{b}_a; \tag{63}$$

then we can show that

$$D^{(m)}\mathbf{f}_\lambda = V_{Ca}\Lambda_{Ca}^{-1}U_C^T\mathbf{b}_a + K_{V0}^T(K_{V0}K_{V0}^T + p\lambda I_0)^{-1}\mathbf{y}_0^M \tag{64}$$

where

$$[K_{V0}, K_{Va}] = K_D[V_{C0}, V_{Ca}], \tag{65}$$

$$\mathbf{y}_0^M = \mathbf{y}^M - K_{Va}\Lambda_{Ca}^{-1}U_C^T\mathbf{b}_a \tag{66}$$

and where the singular value decomposition of C_a is represented by

$$C_a = U_C[\Lambda_{Ca}, 0]\begin{bmatrix} V_{Ca}^T \\ V_{C0}^T \end{bmatrix}. \tag{67}$$

Here we find an expression similar to (62) for the predicted data,

$$\mathbf{y}_0^R = \mathbf{y}_\lambda^R - K_{Va}\Lambda_{Ca}^{-1}U_C^T\mathbf{b}_a \tag{68}$$

$$= K_{V0}K_{V0}^T(K_{V0}K_{V0}^T + p\lambda I_0)^{-1}\mathbf{y}_0^M \tag{69}$$

or we can write

$$\mathbf{y}_0^R = A_0(\lambda)\mathbf{y}_0^M. \tag{70}$$

3. Minimizing the expected recovered error. If the standard deviation of the errors in the data is known (and in this case is equal to one), then we can calculate an unbiased estimate of the expected errors in the predicted data $\mathbf{E}R(\lambda)$ and the regularization parameter that minimizes $\mathbf{E}R(\lambda)$, λ_R, can be found.

If $C = 0$, then the analysis in [4] can be applied directly. The expression we want to minimize is

$$\mathbf{E}R(\lambda) = \mathbf{E}\|\mathbf{y}^t - \mathbf{y}_\lambda^R\|^2, \tag{71}$$

which, when combined with (2) and (62), yields

$$\mathbf{E}R(\lambda) = \mathbf{E}\|(I - A(\lambda))\mathbf{y}^t\|^2 + \mathbf{tr}\, A^2(\lambda). \tag{72}$$

Observe that an unbiased estimate of $\mathbf{E}R(\lambda)$ is given by $\hat{R}(\lambda)$:

$$\hat{R}(\lambda) = \|(I - A(\lambda))\mathbf{y}^M\|^2 + \mathbf{tr}\, A^2(\lambda) - \mathbf{tr}\,(I - A(\lambda))^2. \tag{73}$$

The singular value decomposition of K_D

$$K_D = U_K \Lambda V^T \tag{74}$$

can be used to show that (73) is equivalent to

$$\hat{R}(\lambda) = p + \sum_{j=1}^{p} r_i(\lambda)^2 y_{U;i}^{M^2} - 2r_i(\lambda) \tag{75}$$

where

$$\mathbf{y}_U^M = U_K\mathbf{y}^M, \tag{76}$$

$$r_i(\lambda) = \frac{p\lambda}{\lambda_i + p\lambda}, \tag{77}$$

$$\lambda_i = \sqrt{\Lambda_{ii}}. \tag{78}$$

Therefore a good choice of λ is λ_R, the minimizer (75), and λ_R is easily found since $\mathbf{y}_U^M$ and λ_i are independent of λ.

If the regularized solution has active constraints, defined by (63), then (70) must be substituted for (62) in the previous analysis. Thus we find

$$\mathbf{E}R(\lambda) = \mathbf{E}\|\mathbf{y}_\lambda^R - \mathbf{y}^t\|^2 \tag{79}$$

or

$$\mathbf{E}R(\lambda) = \mathbf{E}\|(I - A_0(\lambda))\mathbf{y}_0^M\|^2 + \|\boldsymbol{\varepsilon}^t A_0(\lambda)\|^2. \tag{80}$$

Thus an unbiased estimate of $\mathbf{E}R(\lambda)$ is $\hat{R}_0(\lambda)$, given by

$$\hat{R}_0(\lambda) = p + \sum_{j=1}^{p} r_{0;i}(\lambda)^2 y_{0;i}^{M^2} - 2r_{0;i}(\lambda) \tag{81}$$

where

$$\mathbf{y}_0^M = U_{K0}\mathbf{y}^M, \tag{82}$$

$$r_{0;i}(\lambda) = \frac{p\lambda}{\lambda_{0;i} + p\lambda}. \tag{83}$$

The minimizer of $\hat{R}_0(\lambda)$ is not as easily found as the minimizer of (75) because $\lambda_{0;i}$ and $\mathbf{y}_0^M$ are functions of the active set that can vary with λ.

The most common case in aerosol data inversion is $C = I$, $\mathbf{b} = \mathbf{0}$. Here the addition of a constraint corresponds to deleting the variable f_i and the ith column of K_D from (2), and we can write $K_D = [K_a, K_0]$, where K_a corresponds to the constrained variables. Observe that both $\hat{R}_0(\lambda)$ and $\hat{R}(\lambda)$ tend to $\|\mathbf{y}^M\|/p$ as $\lambda \to \infty$, and if there is a nonnegative solution to (2), $\boldsymbol{\varepsilon}' = 0$, then $\hat{R}_0(\lambda)$ and $\hat{R}(\lambda)$ both tend to one as λ tends to zero. Additionally, we can show that

$$\lambda_i \geqq \lambda_{0;i}, \qquad i = 1, \cdots, p \tag{84}$$

where strict inequality holds if the rank of K_a equals p. Equation (84) implies that for small λ, we can expect $\hat{R}(\lambda) > \hat{R}_0(\lambda)$. The relationship between $\hat{R}_0(\lambda)$ and $\hat{R}(\lambda)$ is more clear if the left-hand singular eigenvectors of K_a and K_D are approximately equal; in this case we can show that

$$\hat{R}(\lambda) - \hat{R}_0(\lambda) = \sum_{i=1}^{p} 2p\lambda(1-\alpha_i) - (p\lambda \mathbf{y}_U^M)^2(1-\alpha_i^2) \tag{85}$$

where

$$\alpha_i = \frac{\lambda_{0;i} + p\lambda}{\lambda_{0;i} + \hat{\lambda}_i + p\lambda} \tag{86}$$

and $\hat{\lambda}_i$ is an eigenvalue of $K_a K_a^T$. The difference defined by (85) is positive for small λ and negative for large λ, and because of this we might expect the minimizer of $\hat{R}_0(\lambda)$ to be less than the minimizer of $\hat{R}(\lambda)$.

In Fig. 1 we compare $\hat{R}_0(\lambda)$ and $\hat{R}(\lambda)$ as functions of λ to $R(\lambda)$, $\|f_\lambda(x) - f^t(x)\|_1$, and $\|f^t(x) - f_\lambda(x)\|_{TV}$, where

$$\|f(x)\|_1 = \int_0^1 |f(x)|\, dx, \tag{87}$$

$$\|f(x)\|_{TV} = \min\{f(x)\} + \int_0^1 |f'(x)|\, dx. \tag{88}$$

Data with a signal-to-noise ratio of 20 (S/N = 20) were artificially generated for the differential mobility analyzer (DMA) [2] at 54 different inner rod voltages varying between 45 and 4,500 volts for the distribution shown in Fig. 1(a). Since the test distribution is smooth, the inequality constraints add little information to the problem; thus, each $f_\lambda(x)$ has only a few active constraints, and the difference between $\hat{R}(\lambda)$ and $\hat{R}_0(\lambda)$ should be negligible as shown in Fig. 1(b). The difference between $R(\lambda)$ and the estimator $\hat{R}_0(\lambda)$ is noticeable, but the minimizers are nearly equal. Also we can note that the minimizer of $R(\lambda)$ is approximately the same as the minimizer of

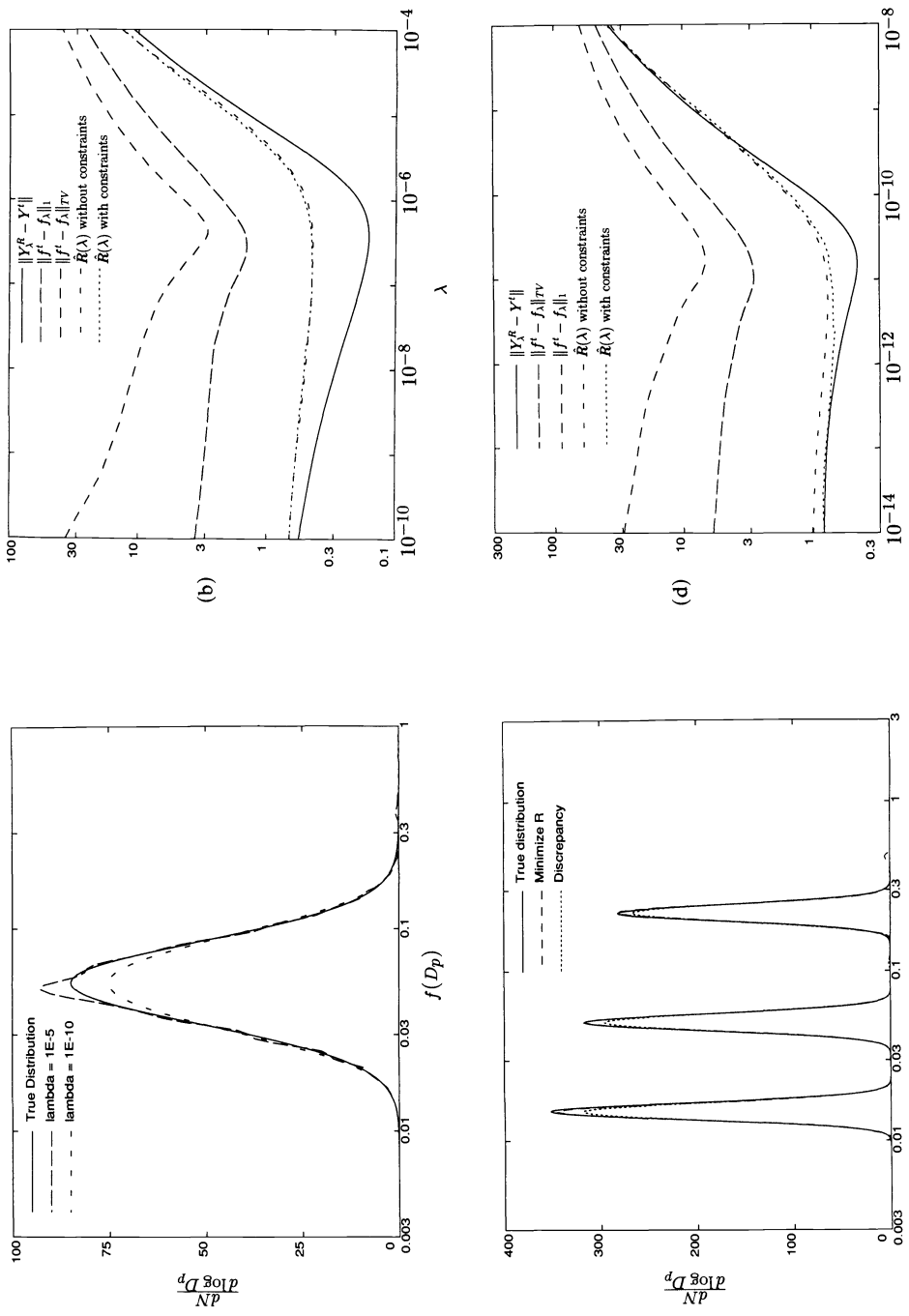


FIG. 1. *Regularized solution properties over a range of smoothing parameters.* (b) *corresponds to the test distribution shown in* (a), *and* (d) *corresponds to the test distribution shown in* (c).

$\|f_\lambda(x)-f^t(x)\|_1$ and $\|f^t(x)-f_\lambda(x)\|_{TV}$. Some meaning for these norms is provided by Fig. 1(a) that shows $f_\lambda(x)$ for $\lambda = 1\times 10^{-5}$ (oversmoothed) and for $\lambda = 1\times 10^{-10}$ (undersmoothed).

Similar plots are shown in Fig. 1(d) where we attempt to reconstruct the trimodal distribution shown in Fig. 1(c). Data were generated as in the previous example with S/N = 20. In this case, the inequality constraints are an important part of the solution; at the optimal λ we found that 50 percent of the constraints were active. However, even here we find only a slight difference between $\hat{R}(\lambda)$ and $\hat{R}_0(\lambda)$. Although $\hat{R}_0(\lambda)$ is a better estimator of $R(\lambda)$ in this case for most λ, it does not provide a better estimate of the minimizer of $R(\lambda)$. We do see the expected $\hat{R}(\lambda)<\hat{R}_0(\lambda)$ at large λ and $\hat{R}(\lambda)>\hat{R}_0(\lambda)$ at small λ. Here, $\lambda_D = 2\times 10^{-10}$, and the corresponding solution is shown along with the solution corresponding to the minimizer of $\hat{R}_0(\lambda)$. Here we see that the method of discrepancy causes a significant amount of oversmoothing.

In a large number of realistic test aerosol inversions, we find only a small difference in the minimizers of $\hat{R}(\lambda)$ and $\hat{R}_0(\lambda)$, and often the minimizer of $\hat{R}_0(\lambda)$ can be found with only a few iterations of a successive substitution scheme. Here, given λ^i, an estimate of the minimizer of $\hat{R}_0(\lambda)$, we

1) Identify the set of active constraints $C_a(\lambda^i)$;

2) Find λ^{i+1} by minimizing $\tilde{R}$, where $\tilde{R}$ approximates $\hat{R}_0(\lambda)$ by assuming the set of active constraints $C_a(\lambda^i)$ is fixed.

This procedure will not always converge, and in some cases we are forced to use a more robust minimization algorithm to choose successive λ's.

4. Generalized cross validation. In the case of aerosol measurements, we often do not have reliable estimates of the magnitude of the errors available. Here, the analysis in the previous section leads to trouble because λ depends on the unknown or poorly known σ.

If $C = 0$, then we can predict σ and choose λ from the data by minimizing $V(\lambda)$ [4],

$$V(\lambda) = \sum_{i=1}^{p} (y^R_{[i]}(\lambda) - y^M_i)^2 w_i(\lambda) \tag{89}$$

where $y^R_{[i]}(\lambda)$ is the ith datum predicted by the regularized solution when the ith measurement is omitted from (2). The weights $w_i(\lambda)$ are needed to ensure that $V(\lambda)$ reflects the rotation invariant nature of (2). The idea behind choosing λ to minimize (89) is simple: a good choice of λ is the one that best enables the data to predict an omitted datum. As the number of data increases, this technique for choosing λ becomes optimal as described in [15] and [14].

We can show that [4]

$$y^R_{[i]}(\lambda) - y^M_i = (y^R_i(\lambda) - y^M_i)/(1 - a_{i,i}(\lambda)) \tag{90}$$

where $a_{i,i}(\lambda)$ is the ith diagonal element of $A(\lambda)$. From this and the observation that $w_i(\lambda)$ should equal one if $A(\lambda)$ is circulant, we can show that an appropriate expression for $w_i(\lambda)$ is [9]

$$w_i(\lambda) = \frac{(1 - a_{i,i}(\lambda))^2}{(p - \mathbf{tr}\,[A(\lambda)])^2}. \tag{91}$$

This leads to

$$V(\lambda) = \frac{\|(I - A(\lambda))\|^2}{(\mathbf{tr}\,[I - A(\lambda)])^2}. \tag{92}$$

Equation (92) can be combined with (74) to yield

$$V(\lambda)=\frac{\sum_{i=1}^{p} r_i(\lambda)^2 y_{U;i}^{M2}}{\left(\sum_{i=1}^{p} r_i(\lambda)\right)^2} \tag{93}$$

where $r_i(\lambda)$ and $y_{U;i}^{M}$ are as defined in (76). Note that

$$\begin{aligned} &V(\lambda)\to\|\mathbf{y}^M\|^2 \quad \text{as } \lambda\to\infty, \\ &V(\lambda)\to\frac{\sum_{i=1}^{p} y_{U;i}^{M2}/\lambda_i^2}{\left(\sum_{i=1}^{p} 1/\lambda_i\right)^2} \quad \text{as } \lambda\to 0. \end{aligned} \tag{94}$$

It is not uncommon for $\lambda=0$ to minimize $V(\lambda)$.

The analysis for the constrained case is similar if we assume the set of constraints that are active at $\mathbf{f}_\lambda$ is not affected by the removal of one of the data. Here we choose the λ that minimizes $V_0(\lambda)$,

$$V_0(\lambda)=\sum_{i=1} (y_{0[i]}^{R}(\lambda)-y_{0i}^{M})^2 w_{0i}(\lambda) \tag{95}$$

where $y_{0[i]}^{R}(\lambda)$ is the constrained analogue of $y_{[i]}^{R}(\lambda)$, and y_{0i}^{M} is defined by (66). Since we assumed that the constraints are not affected by the removal of a single datum, we find

$$y_{0[i]}^{R}(\lambda)-y_{0i}^{M}=(y_{0i}^{R}(\lambda)-y_{0i}^{M})/(1-a_{0i,i}(\lambda)). \tag{96}$$

Similarly, an appropriate expression for the weights is given by

$$w_{0i}(\lambda)=\frac{(1-a_{0i,i}(\lambda))^2}{(\mathbf{tr}\,[I-A_0(\lambda)])^2} \tag{97}$$

and

$$V_0(\lambda)=\frac{\|(A_0(\lambda)-I)\mathbf{y}_0^M\|^2}{(\mathbf{tr}\,[I-A_0(\lambda)])^2}. \tag{98}$$

In the constrained case, the calculation can become lengthy if the set of active constraints depends on λ.

Again we consider in more detail the case $C=I$, $\mathbf{b}=\mathbf{0}$. The difference between $V_0(\lambda)$ and $V(\lambda)$ is not as transparent as the difference between $R(\lambda)$ and $R_0(\lambda)$. We only note that for large λ we can expect that $f_\lambda(x)$ will be very smooth; thus, there will be few active constraints, if any, and here $V_0(\lambda)=V(\lambda)$.

We have examined $V_0(\lambda)$ and $V(\lambda)$ under the same conditions as those of the examples presented in the previous section. In Fig. 2(a) we see $V_0(\lambda)$ and $V(\lambda)$ in the case of the smooth distribution where the inequality constraints are unimportant, and as expected little difference between the two is observed, as was the case in the previous section for $R(\lambda)$ and $R_0(\lambda)$. Also, we note that the minimizer of $V(\lambda)$ is a good estimate of the minimizer of $R(\lambda)$.

In Fig. 2(b) we show the $V_0(\lambda)$ and $V(\lambda)$ that are obtained for the reconstruction of the trimodal distribution shown in Fig. 1(c). The numerical conditions are the same as those used to generate Fig. 1(c). Note that the constraints provide critical information in choosing λ. Here, $\lambda=0$ is the minimizer of $V(\lambda)$, and the plot of $\|\mathbf{y}_\lambda^R-\mathbf{y}^M\|^2$ and the solution difference norms show that this would result in an unacceptable solution. Similar results were obtained for S/N = 10 and S/N = 100.

It is clear from Fig. 2 that the derivative of $V_0(\lambda)$ need not be continuous, and this reflects the discrete addition of information as constraints are added or deleted. For very small λ, the set of constraints can be sensitive to small changes in λ, and we

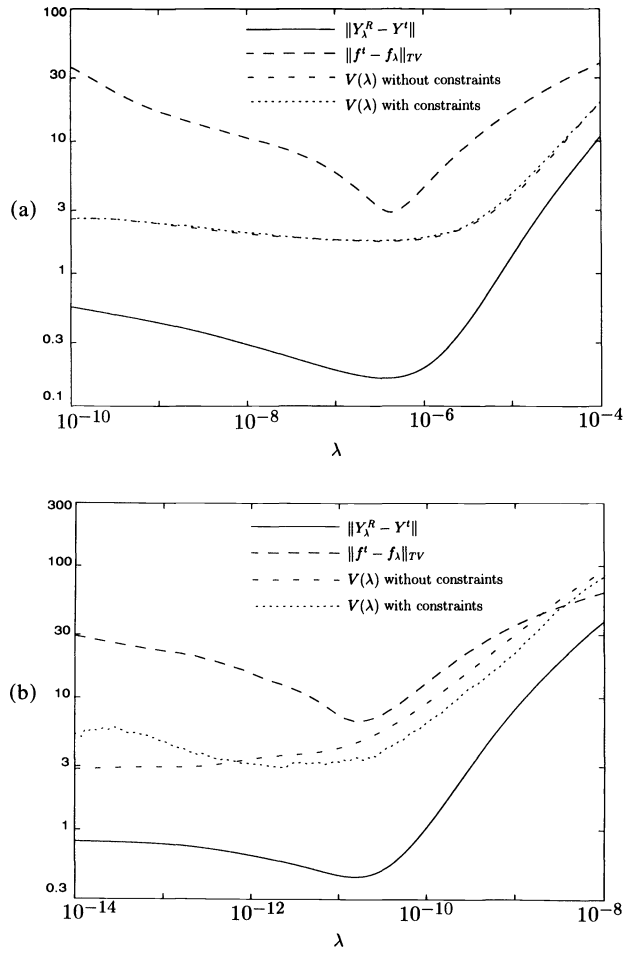


FIG. 2. GCV *function over a range of smoothing parameters.* (a) *corresponds to the test distribution shown in Fig.* 1(a), *and* (b) *corresponds to the test distribution shown in Fig.* 1(c).

may find a minimum of $V_0(\lambda)$ in this region of λ. This minimum is not desirable because of the instability of the minimum with respect to changes in the data and the constraints; additionally in this region the assumption that the set of constraints does not change as a datum is deleted is most likely invalid.

If the minimizers of $V_0(\lambda)$ and $V(\lambda)$ are very different, then the technique used to minimize $R_0(\lambda)$ fails when trying to minimize $V_0(\lambda)$; here, minimizing $V_0(\lambda)$ is difficult, and we do not have a simple technique that applies to all cases.

5. Numerical comparisons of λ_D, λ_V, and λ_R. In this section we present two examples that compare the regularized solutions obtained with the smoothing parameters λ_D, λ_R, and λ_V. We will ignore the smoothing parameters obtained when the constraints are omitted; if generalized cross validation is being used then, as shown in the previous section, the results can be catastrophic, and if $\mathbf{E}R(\lambda)$ is being minimized then usually the results do not change significantly if the constraints are removed.

In the first example we compare the average difference between the regularized solution and the true solution over a range of signal-to-noise ratios for the two true distributions shown in Figs. 3(a) and 3(b). In all cases we assume data from the DMA

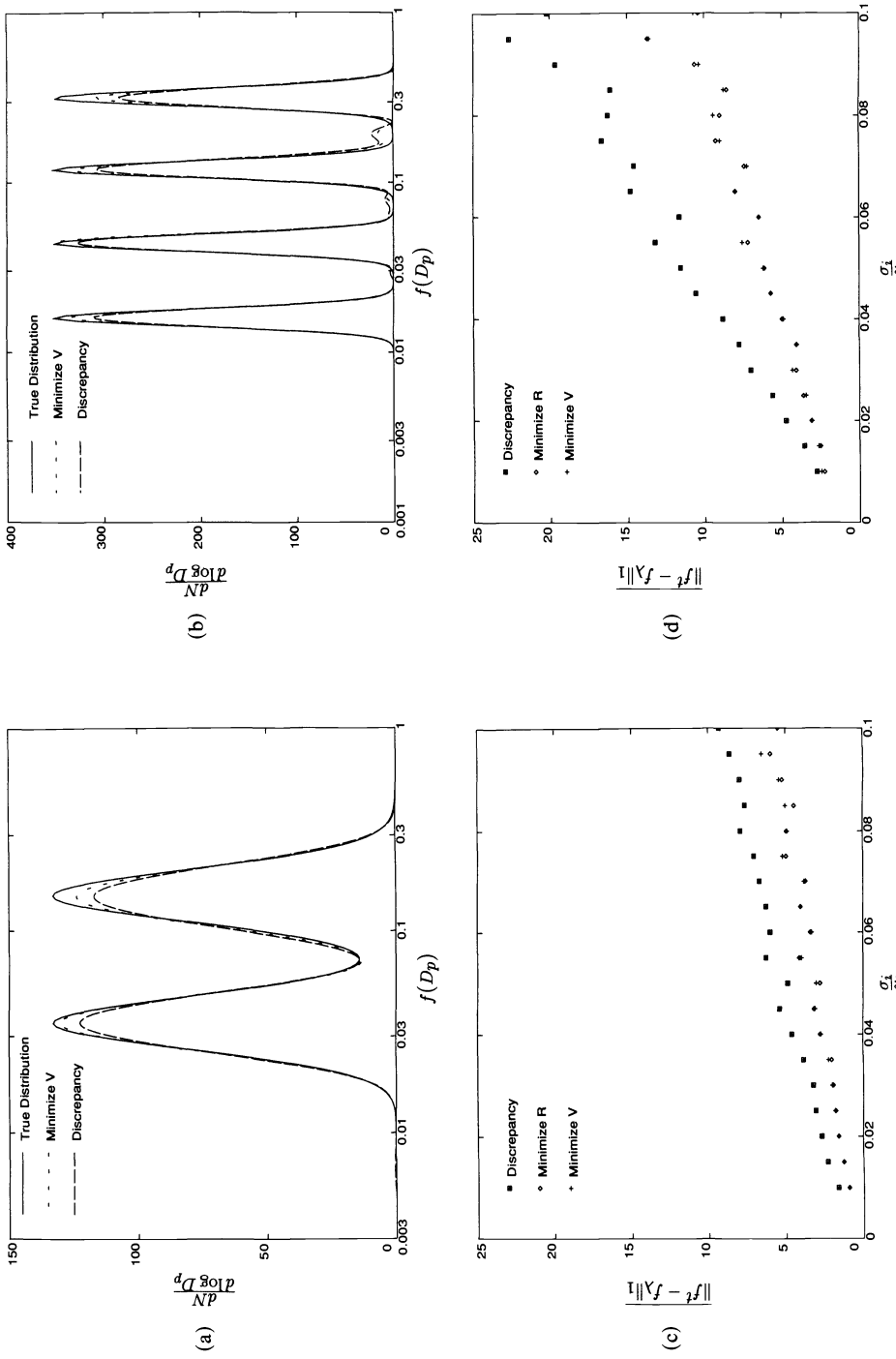


FIG. 3. *Average simulated differences between the true solution and the regularized solution over a range of signal-to-noise ratios.* (c) *and* (e) *correspond to the bimodal distribution shown in* (a); (d) *and* (f) *correspond to the 4-modal distribution shown in* (b).

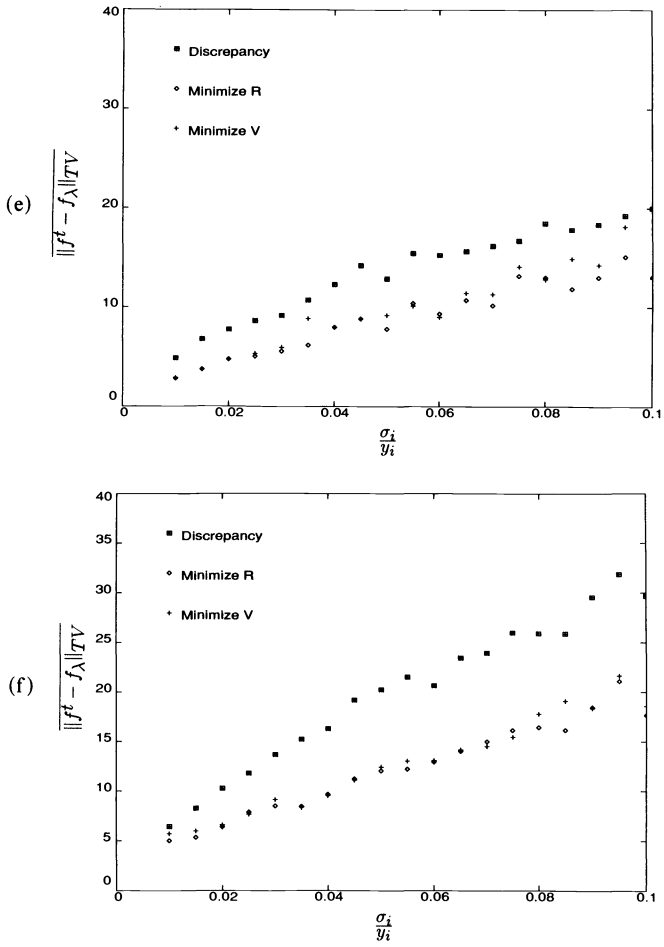


FIG. 3—continued

described in the previous section were available, along with data from the diffusion battery described in [3]. We also assumed that the size distribution was to be zero outside the interval (0.005 μm, 0.5 μm) and that the solution can be represented by 200 linear splines. Many sets of noisy data were generated with the signal-to-noise ratio ranging from 4 and 100, and the regularized solutions were calculated for the smoothing parameters λ_R, λ_D, and λ_V. The average values of $\|f'(x)-f_\lambda(x)\|_{TV}$ and $\|f_\lambda(x)-f'(x)\|_1$ are shown in Figs. 3(c)-3(f). Also, in Figs. 3(a) and 3(b) we present representative solutions for S/N = 20.

In this example, we can see a clear advantage in using λ_R or λ_V as opposed to λ_D for the range of conditions tested. As the signal-to-noise ratio increases, the accuracy of the inverted solution must decay on the average; however, this decay is accelerated when λ_D is used to define $\mathbf{f}_\lambda$. Also note that the difference between the solutions obtained from λ_R and λ_V is negligible in most cases, and this lends more credence to the claim that generalized cross validation is able to provide a good choice of the smoothing parameter and predict the magnitude of the errors in the data *from the data alone.*

Generalized cross validation works well in this example because the amount of information provided by the data is sufficient to justify minimizing (89); generalized

cross validation cannot be expected to perform well in cases where the information content of the data is small relative to the amount of structure in the solution. For example, in the case of only a few impactor data, it is not clear if the minimizer of (92) is desirable.

In the second example we investigate the effects of inaccurately estimating the signal-to-noise ratio. This is important because often the magnitude of the error in aerosol measurement data cannot be accurately estimated, and here we expect, as the estimate becomes worse, that GCV will become more attractive. In this example, we assumed data were available from the same sources that were used in the previous example. Many sets of erroneous data were generated for unimodal and trimodal analogues of the test distributions shown in Fig. 3 while the signal-to-noise ratio varied between 4 and 100. Regularized solutions were calculated for λ_V, λ_R, and λ_D, while the estimated signal-to-noise ratio remained at 20. The difference between the regularized solution and the true solution is shown in Fig. 4 over the range of true signal-to-noise ratios. These results clearly demonstrate the power of using λ_V in the regularized solution. The main point here is that generalized cross validation was able to consistently predict an acceptable amount of smoothing in the test cases without being affected by the poorly estimated signal-to-noise ratio; this inaccurate estimation of the true S/N, however, had a disastrous affect on the $\mathbf{f}_\lambda$ corresponding to λ_R and λ_D. We can also see that of the three methods for choosing the smoothing parameter, λ_D is most sensitive to overestimating the signal-to-noise ratio.

The advantages of using λ_V can be further highlighted by examining, for example, the average $\|f^t(x)-f_\lambda(x)\|_{TV}$ that we would obtain if the estimated signal-to-noise ratio were 20 while the logarithm of the true signal-to-noise ratio was a uniform random variable varying between $20c$ and $20/c$, with $1<c\leqq 5$. Thus we seek $\bar{N}(c)$ defined by

$$\bar{N}(c)=\frac{1}{2\log c}\int_{(20/c)}^{20c} N(x)\,d\log x \tag{99}$$

where N is the average value of the norm of interest evaluated at a signal-to-noise level of x given in Fig. 4.

The results are shown in Fig. 5 and emphasize the average cost of a poor estimate of the signal-to-noise ratio. We see in many cases that we are better off using λ_V, even if the estimated signal-to-noise ratio is exact. For the two distributions having the most structure, there may be some advantage to using λ_R if our estimate of the true signal-to-noise ratio is within a factor of 1.5; however, the difference is slight. For distributions with large amounts of structure relative to the amount of data, the idea behind λ_V is lacking, since it may not be possible for the set of data to predict a key omitted datum as suggested by (89).

The results in Fig. 5 point to an advantage of this work over that presented in [19], where the analysis was based on using λ_D. In most realistic situations involving aerosol data, the signal-to-noise ratio cannot be estimated accurately enough to justify using the method of discrepancy.

6. Conclusions. Regularization parameters, λ_D, λ_R, and λ_V, based on the method of discrepancy, minimization of an unbiased estimate of the expected errors, and generalized cross validation, respectively, have been investigated for the regularized solution to the constrained linear inverse problem arising in aerosol data inversion. A computationally feasible method of finding λ_R and λ_V when constraints are important has been presented. An example was presented where consideration of the constraints is crucial in defining λ_V. The results of numerical experiments suggest that λ_D is too

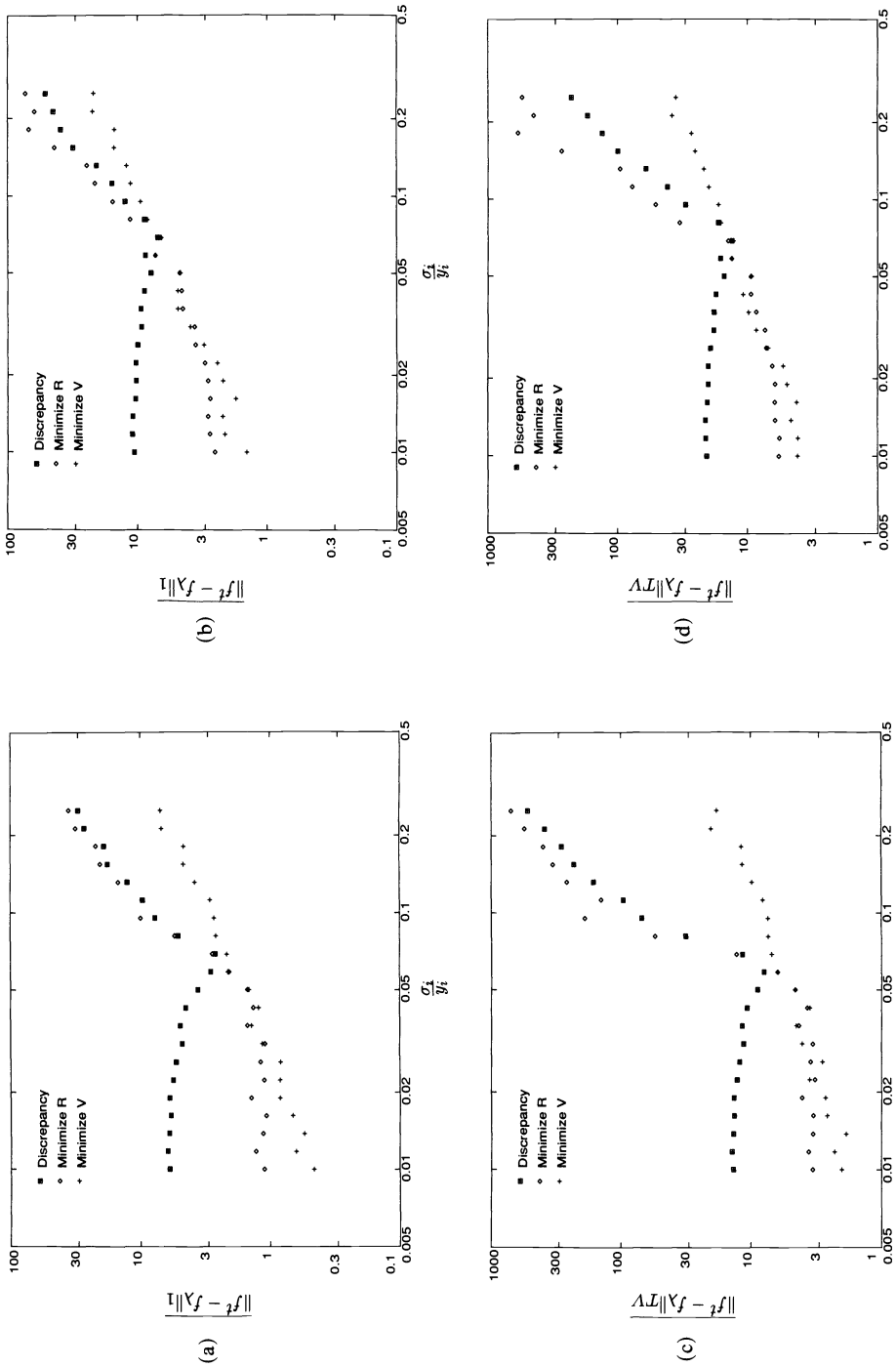


FIG. 4. *Average simulated differences between the true solution and the regularized solution when the signal-to-noise ratio is (incorrectly) estimated to equal* 20. (a) *and* (c) *correspond to a unimodal test distribution*; (b) *and* (d) *correspond to a trimodal test distribution.*

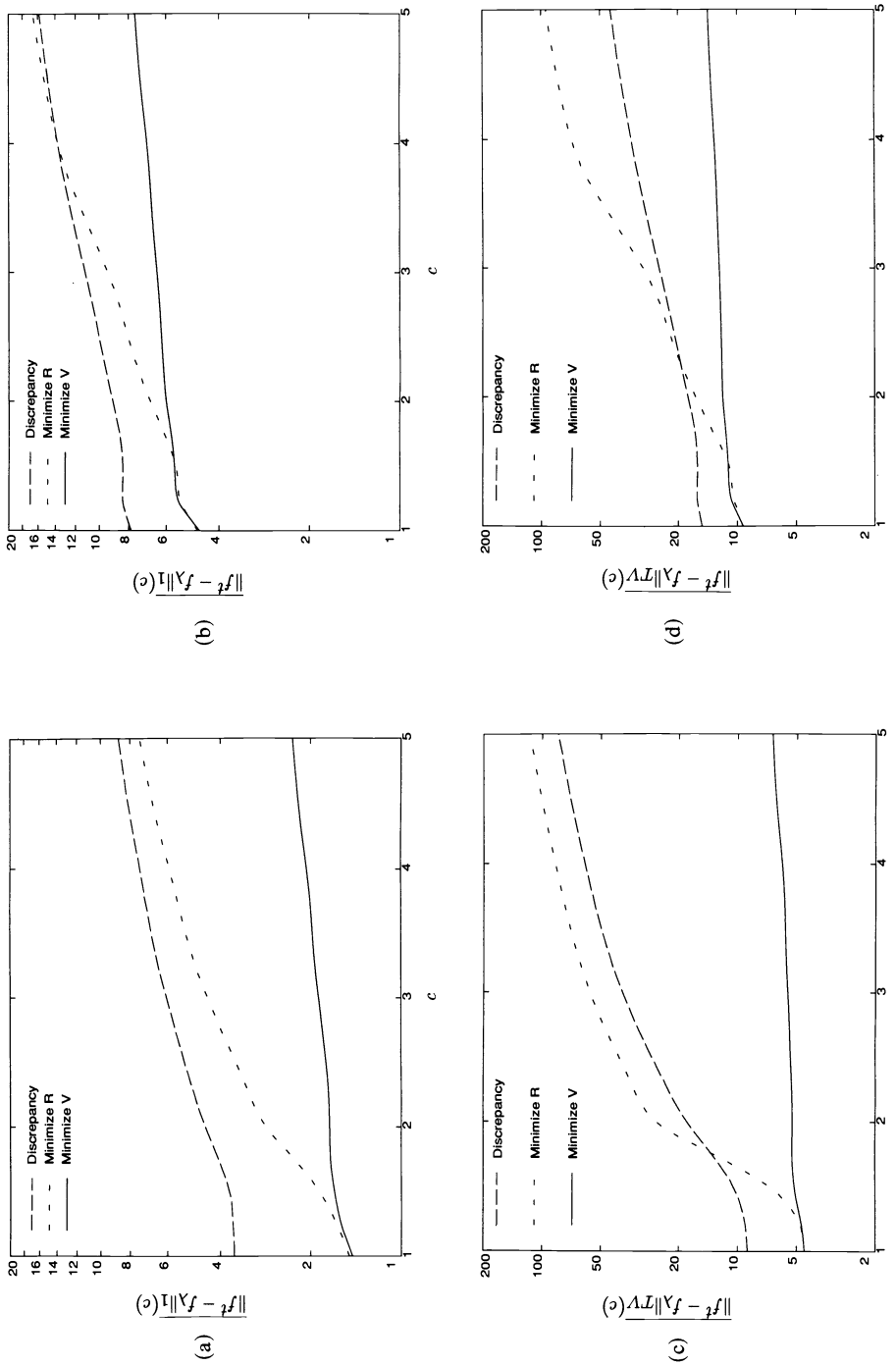


FIG. 5. *Average simulated differences between the true solution and the regularized solution when the logarithm of the signal-to-noise ratio is a uniform random variable and the estimated signal-to-noise ratio is* 20. (a) *and* (c) *correspond to a unimodal distribution;* (b) *and* (d) *correspond to a trimodal distribution.*

large and point out the importance of using generalized cross validation if the signal-to-noise ratio cannot be accurately estimated.

REFERENCES

[1] R. A. ADAMS, *Sobolev Spaces*, Academic Press, New York, 1975.

[2] D. J. ALOFS AND P. BALAKUMAR, *Inversion to obtain aerosol size distributions from measurements with a differential mobility analyzer*, J. Aerosol Sci., 13 (1982), pp. 512–527.

[3] Y. S. CHENG AND H. C. YEH, *Theory of a screen-type diffusion battery*, J. Aerosol Sci., 11 (1980), pp. 313–320.

[4] P. CRAVEN AND G. WAHBA, *Smoothing noisy data with spline functions*, Numer. Math., 31 (1979), pp. 377–403.

[5] J. G. CRUMP AND J. H. SEINFELD, *Further results on inversion of aerosol size distribution data: higher-order Sobolev spaces and constraints*, Aerosol Sci. Tech., 1 (1982), pp. 363–369.

[6] E. DE DONCKER AND I. ROBINSON, TRIEX: *Integration over a triangle using non-linear extrapolation*, ACM Trans. Math. Software, 10 (1984), pp. 17–22.

[7] P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND M. H. WRIGHT, *Procedures for optimization problems with a mixture of bounds and general linear constraints*, ACM Trans. Math. Software, 10 (1984), pp. 282–298.

[8] P. E. GILL, W. MURRAY, S. J. HAMMARLING, M. A. SAUNDERS, AND M. H. WRIGHT, *User's guide for* LSSOL (*Version* 1.0): *A Fortran package for constrained linear least-squares and convex quadratic programming*, Tech. Report SOL 86-1, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA, 1986.

[9] G. H. GOLUB, M. HEATH, AND G. WAHBA, *Generalized cross-validation as a method for choosing a good ridge parameter*, Technometrics, 21 (1979), pp. 215–223.

[10] C. GU AND G. WAHBA, *Minimizing* GCV/GML *scores with multiple smoothing parameters via the Newton method*, Tech. Report 847, Department of Statistics, University of Wisconsin, Madison, WI, 1988.

[11] D. K. KAHANER AND O. W. RECHARD, TWODQD: *an adaptive routine for two-dimensional integration*, J. Comput. Appl. Math., 17 (1987), pp. 215–234.

[12] G. KIMELDORF AND G. WAHBA, *Some results on Tchebycheffian spline functions*, J. Math. Anal. Appl., 33 (1971), pp. 82–95.

[13] D. P. LAURIE, CUBTRI, *Automatic cubature over a triangle*, ACM Trans. Math. Software, 8 (1982), pp. 210–218.

[14] K. LI, *Asymptotic optimality for* c_p, c_L, *cross validation and generalized cross-validation: discrete index set*, Ann. Statist., 15 (1987), pp. 958–975.

[15] ———, *Asymptotic optimality of* c_L *and generalized cross-validation in ridge regression with application to spline smoothing*, Ann. Statist., 14 (1986), pp. 1101–1112.

[16] D. L. PHILLIPS, *A technique for the numerical solution of certain integral equations of the first kind*, J. Assoc. Comput. Mach., 9 (1962), pp. 84–97.

[17] M. VILLALOBOS AND G. WAHBA, *Inequality-constrained multivariate smoothing splines with application to the estimation of posterior probabilities*, J. Amer. Statist. Assoc., 82 (1987), pp. 239–248.

[18] G. WAHBA, *Smoothing noisy data with spline functions*, Numer. Math., 24 (1975), pp. 383–393.

[19] J. K. WOLFENBARGER AND J. H. SEINFELD, *Inversion of aerosol size distribution data*, J. Aerosol Sci., 21 (1990), pp. 227–247.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 362-382, March 1991
© 1991 Society for Industrial and Applied Mathematics
008

A MODIFIED PRONY ALGORITHM FOR FITTING FUNCTIONS DEFINED BY DIFFERENCE EQUATIONS*

M. R. OSBORNE† AND G. K. SMYTH‡

Abstract. This paper reformulates, generalizes, and investigates the stability of the modified Prony algorithm introduced by Osborne [*SIAM J. Numer. Anal.*, 12 (1975), pp. 571-592], with special reference to rational and exponential fitting. The algorithm, originally for exponential functions, is generalized to the least squares fitting of any function which satisfies a linear homogeneous difference equation. Using the difference equation formulation, the problem is expressed as a separable regression, and hence as a nonlinear eigenproblem in terms of the coefficients of the difference equation. The eigenproblem involves finding the null space of a matrix of data differences B, and is solved using a variant of inverse iteration. Stability of the algorithm is shown to depend on the fact that B closely approximates the Hessian of the sum of squares. The expectations of B and the Hessian are evaluated. In the case of rational fitting, the relative difference between B and the Hessian is shown to converge to zero almost surely. Some details of the implementation of the algorithm are given. A simulation study compares the modified Prony algorithm with the Levenberg algorithm on a rational fitting problem, and supports the theoretical results.

Key words. Prony's method, differential equations, nonlinear least squares, rational fitting, nonlinear eigenproblem, inverse iteration, asymptotic stability, Gauss-Newton algorithm

AMS(MOS) subject classifications. 62J02, 65D10

1. Introduction.

This paper reformulates, generalizes, and investigates the stability of the modified Prony algorithm introduced by Osborne [13]. The algorithm, originally for exponential functions, is generalized to provide least squares fitting of any function which satisfies a linear homogeneous difference equation (e.g., to fit compartment models). The main interest is in functions for which the difference equation is an exact discrete analogue of the limiting differential equation, and especially in the specific examples of rational and exponential function fitting.

In a classic paper, Prony [17] proposed a method for interpolating a sum of exponential functions through a series of data values at equally spaced points. He solved a system of linear equations for the coefficients of a difference equation satisfied by the exponential functions, and obtained the exponential functions from the roots of the polynomial with those coefficients. Prony's method was extended to the least squares context by, for example, Householder [6], and is actively used in systems engineering and time series (for example, Marple [9], Mulholland, Cruz, and Hill [10]). But a formulation which returned the actual least squares estimators was not produced until Osborne [13], who proposed a modified Prony algorithm in which a nonlinear eigenproblem is solved for the coefficients of the difference equation. Osborne [13] showed, by way of a simulation study, that the algorithm converged with surprising speed, but did not provide a theoretical justification. He also indicated an extension to rational function fitting, without verifying the numerical properties.

This paper is directed towards proving that the algorithm is asymptotically stable, i.e., that it converges at a rapid rate from good starting values for sufficiently large data sets. (This is analogous to the result proved by Jennrich [7] for the Gauss-Newton algorithm.) Although the general form of the proof is shown, the proof is completed here only for rational functions. The rational fitting case involves a minimum of special

* Received by the editors June 27, 1985; accepted for publication (in revised form) December 1, 1989.
† Statistics Section, School of Mathematics, Australian National University, Canberra 2601, Australia.
‡ Department of Mathematics, University of Queensland, St. Lucia 4067, Australia.

complication, and least obscures the general case. The corresponding result for exponential functions was proved by Smyth [19], and will be given in a separate paper to be devoted to the special features of the exponential fitting case.

The problem considered is that of fitting a given continuous function μ to the data pairs (t_i, y_i), $i = 1, \cdots, n$ by least squares. Let $\mu_i = \mu(t_i)$ and $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_n)^T$. The errors $y_i - \mu_i$ are assumed to be independent and to have equal and finite variances. We show that if the μ_i satisfy a difference equation, linear, and homogeneous in a parameter vector $\boldsymbol{\gamma}$, then the least squares problem may be formulated as a nonlinear eigenproblem, which can be solved using a variant of inverse iteration. Section 2 shows that the difference equation can be expressed in matrix terms as

$$X(\boldsymbol{\gamma})^T \boldsymbol{\mu} = 0$$

where X is a banded linear matrix function of $\boldsymbol{\gamma}$. Section 3 shows that the least squares problem, concentrated in terms of the nonlinear parameters, may be solved by minimizing

$$\psi(\boldsymbol{\gamma}) = \mathbf{y}^T P_X \mathbf{y}$$

where P_X is the projection onto the space $\mathcal{R}(X)$ spanned by the columns of X. Furthermore, the gradient of ψ is given by

$$\dot{\psi}(\boldsymbol{\gamma}) = 2B(\boldsymbol{\gamma})\boldsymbol{\gamma}$$

where B is a symmetric matrix function of $\boldsymbol{\gamma}$. Section 4 relates the results to rational and exponential fitting. Section 5 describes in detail the modified Prony algorithm, and briefly discusses the effect of the constraint used to standardize $\boldsymbol{\gamma}$. Section 6 shows that a sufficient condition for convergence is that B should closely approximate $\ddot{\psi}/2$, and § 7 evaluates the expectations of these quantities. Sections 8–10 discuss the asymptotic eigenstructure of X and B, and show that for rational fitting $B^+(\ddot{\psi}/2 - B) \to 0$ with probability one, thus establishing the stability of the algorithm.

Rational fitting is usually accomplished using the Gauss–Newton algorithm or its Levenberg or Marquardt modifications (Ratkowsky [18] and Bates and Watts [1]), so § 11 compares the modified Prony algorithm with the Levenberg algorithm on a rational fitting problem using a small simulation study.

2. The difference equation. We assume that $\mu(t)$ satisfies a difference equation of the form

$$\sum_{k=1}^{p+1} d_k(t; \boldsymbol{\gamma}) \Delta^{k-1} \mu(t) = 0 \tag{1}$$

and that the coefficients d_k are continuous in t and linear and homogeneous in some unknown $\boldsymbol{\gamma} \in \mathbb{R}^{q+1}$. We call $\boldsymbol{\gamma}$ the *Prony parameters*. For simplicity, we will assume that the times t_i at which μ is sampled are equally spaced over a fixed interval, although the results of this paper require only that the empirical distribution of the t_i converges to a nontrivial probability distribution. Without loss of generality we take that interval to be $[0, 1]$ so that $t_i = 1/n$, and define Δ to be the forward difference operator

$$\Delta f(t) = n[f(t + 1/n) - f(t)].$$

Generally $\boldsymbol{\gamma}$ and $\mathbf{d} = (d_1, \cdots, d_{p+1})^T$ will depend on n, but we will usually not make this explicit in our notation. Since the scales of $\boldsymbol{\gamma}$ and $\mathbf{d}$ are disposable, we will adjoin the condition $\gamma_{q+1} = 1$ or $\|\boldsymbol{\gamma}\| = 1$ with respect to some inner product, whatever is most convenient, and similarly for $\mathbf{d}$.

By assumption, the d_k may be expanded into

$$d_k(t;\boldsymbol{\gamma}) = \sum_{j=1}^{q+1} \gamma_j d_{kj}(t)$$

where the d_{kj} are known continuous functions, so (1) may be rewritten as

$$\sum_{j=1}^{q+1} \gamma_j \left(\sum_{k=1}^{p+1} d_{kj}(t) \Delta^{k-1} \right) \mu(t) = 0.$$

This may be translated into matrix terms in the following way. Let $\boldsymbol{\Pi}$ be the $n \times n$ circulant forward shift operator

$$\boldsymbol{\Pi} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ & & \vdots & & \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix},$$

and let $\boldsymbol{\Delta}$ be the circulant difference matrix $\boldsymbol{\Delta} = n(\boldsymbol{\Pi} - I)$. Then $\boldsymbol{\mu}$ satisfies

$$P \sum_{j=1}^{q+1} \gamma_j \sum_{k=1}^{p+1} \langle d_{kj}(\mathbf{t}) \rangle \boldsymbol{\Delta}^{k-1} \boldsymbol{\mu} = 0$$

where $\langle d_{kj}(\mathbf{t}) \rangle$ denotes the diagonal matrix with diagonal elements $d_{kj}(t_1)$ to $d_{kj}(t_n)$, and $P = (I_{n-p} 0)$ selects the leading $(n-p)$ rows. (Throughout this paper, scalar functions are taken to act componentwise on vectors, and $\langle \cdot \rangle$ denotes a diagonal matrix with the components of the vector argument down the diagonal.) Write also

$$C_j^T = \sum_{k=1}^{p+1} \langle d_{kj}(\mathbf{t}) \rangle \boldsymbol{\Delta}^{k-1} \quad \text{and} \quad C = \sum_{j=1}^{q+1} C_j \gamma_j.$$

Let $X_j = C_j P^T$, the leading $n-p$ columns of C_j, and let

$$X = \sum_{j=1}^{q+1} \gamma_j X_j.$$

Then (1) may be re-expressed in matrix terms as

$$PC(\boldsymbol{\gamma})^T \boldsymbol{\mu} = 0$$

or

$$X(\boldsymbol{\gamma})^T \boldsymbol{\mu} = 0. \tag{2}$$

Note that X is a $n \times (n-p)$ matrix with $p+1$ bands, and that it has full column rank if $d_{p+1} \neq 0$. Although the above definition in terms of circulant operators is often convenient, the last p columns of C are arbitrary as far as (2) is concerned; in the next section we will assume that they have been chosen to ensure that C is invertible.

The above derivation depended on $\mu(t)$ satisfying exactly a difference equation. In practice, it will often be the case that $\mu(t)$ came as the general solution of a differential equation, say of

$$\sum_{k=1}^{p+1} b_k(t;\boldsymbol{\xi}) \mathscr{D}^{k-1} \mu(t) = 0 \tag{3}$$

where $\mathscr{D}$ is the differential operator and the coefficients b_k are linear and homogeneous in $\boldsymbol{\xi}$. Until § 8 of this paper we will assume only the difference equation (1). In the remaining sections, which deal with asymptotics, we will assume that μ satisfies the differential equation (3) also, and that the difference equation converges to it, in the sense that $\boldsymbol{\gamma} \to \boldsymbol{\xi}$ and $d \to b$ as $n \to \infty$.

The most general expression for $\boldsymbol{\mu}$ which allows a characterization of the form (2) is in fact

$$\boldsymbol{\mu} = Z(\boldsymbol{\gamma})^{-T} W\boldsymbol{\alpha} \tag{4}$$

where Z is a linear function of $\boldsymbol{\gamma}$ and W is a constant $n \times p$ matrix. Then

$$X(\boldsymbol{\gamma}) = Z(\boldsymbol{\gamma})H$$

where H is any full rank $n \times (n-p)$ matrix orthogonal to W. The matrix H may be chosen to have $p+1$ bands,

$$H = \begin{pmatrix} h_{11} & & \\ \vdots & \ddots & \\ h_{1,p+1} & & h_{n-p,1} \\ & \ddots & \vdots \\ & & h_{n-p,p+1} \end{pmatrix},$$

by determining each $(h_{j,1}, \cdots, h_{j,p+1})^T$ to be orthogonal to

$$\begin{pmatrix} w_{j,1} & \cdots & w_{j,p} \\ \vdots & & \vdots \\ w_{j+p,1} & \cdots & w_{j+p,1} \end{pmatrix},$$

which comprises rows j to $j+p$ of W.

3. A nonlinear eigenproblem. We now invert the matrix difference equation (2) to obtain an explicit expression for $\boldsymbol{\mu}$ of the form (4). One way is to note that quite generally

$$C(\boldsymbol{\gamma})^T \boldsymbol{\mu} = \begin{pmatrix} 0 \\ \boldsymbol{\alpha} \end{pmatrix}$$

for some $\boldsymbol{\alpha} \in \mathbb{R}^p$. Therefore, assuming that C is invertible,

$$\begin{aligned} \boldsymbol{\mu} &= C(\boldsymbol{\gamma})^{-T} Q^T \boldsymbol{\alpha} \\ &= \mathrm{A}(\boldsymbol{\gamma})\boldsymbol{\alpha} \end{aligned} \tag{5}$$

where $A = C^{-T}Q^T$ and Q is the $p \times n$ matrix $(0 \ \ I_p)$. The p columns of A correspond to a particular set of fundamental solutions of the difference equation (1), depending on how the last p columns of C have been chosen. In any case, (5) displays $\boldsymbol{\mu}$ as a separable regression, with $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ as the linear and nonlinear parameters, respectively. For any fixed value of $\boldsymbol{\gamma}$ the sum of squares

$$\phi(\boldsymbol{\alpha}, \boldsymbol{\gamma}) = (\mathbf{y} - \boldsymbol{\mu})^T (\mathbf{y} - \boldsymbol{\mu})$$

is minimized by

$$\hat{\boldsymbol{\alpha}}(\boldsymbol{\gamma}) = (A^T A)^{-1} A^T \mathbf{y}. \tag{6}$$

Substituting this back into ϕ gives the reduced sum of squares

$$\begin{aligned} \psi(\boldsymbol{\gamma}) &= \phi(\hat{\boldsymbol{\alpha}}(\boldsymbol{\gamma}), \boldsymbol{\gamma}) \\ &= \mathbf{y}^T (I - A(A^T A)^{-1} A^T)\mathbf{y} \\ &= \mathbf{y}^T (I - P_A)\mathbf{y} \end{aligned} \tag{7}$$

where P_A is the orthogonal projection onto $\mathscr{R}(A)$. The least squares problem may be solved by minimizing ψ with respect to $\boldsymbol{\gamma}$, and recovering the least squares estimate

of $\boldsymbol{\alpha}$ from (6). Note that $X^TA=0$, so that the columns of X and A span orthogonal spaces. Therefore (7) can be written more conveniently as

$$\psi(\boldsymbol{\gamma})=\mathbf{y}^TP_X\mathbf{y}. \tag{8}$$

Since $P_X=X(X^TX)^{-1}X^T$, ψ has partial derivatives

$$\begin{aligned}\dot{\psi}_i&=2\mathbf{y}^TX_i(X^TX)^{-1}X^T\mathbf{y}-2\mathbf{y}^TX(X^TX)^{-1}X_i^TX(X^TX)^{-1}X^T\mathbf{y}\\&=2\sum_{j=1}^{q+1}\gamma_jB_{ij}\end{aligned}$$

with

$$B_{ij}=\mathbf{y}^TX_i(X^TX)^{-1}X_j^T\mathbf{y}-\mathbf{y}^TX(X^TX)^{-1}X_i^TX_j(X^TX)^{-1}X^T\mathbf{y}.$$

Therefore, gathering the B_{ij} into a symmetric matrix function of $\boldsymbol{\gamma}$, we have

$$\dot{\psi}(\boldsymbol{\gamma})=2B(\boldsymbol{\gamma})\boldsymbol{\gamma}.$$

Assuming that the constraint on $\boldsymbol{\gamma}$ takes the form $\boldsymbol{\gamma}^T\boldsymbol{\gamma}=1$, then $\psi(\boldsymbol{\gamma})$ is minimized by a solution of the nonlinear system

$$\dot{\psi}(\boldsymbol{\gamma})=2\dot{B}(\boldsymbol{\gamma})\boldsymbol{\gamma}=\lambda\boldsymbol{\gamma} \tag{9}$$

where λ is the Lagrange multiplier associated with the constraint. The interpretation and solution of this equation as a nonlinear eigenproblem is the basis of the modified Prony algorithm. Because of its fundamental role in the algorithm, we call B the *Prony matrix.*

The derivation and definition of B given above differs from Osborne [13] in its definition of $\boldsymbol{\gamma}$ and in that the components of B are displayed as quadratic functions of $\mathbf{y}$. These changes make possible asymptotic arguments and distributional calculations. The various Prony parametrizations $\boldsymbol{\gamma}$ for which the normal equations can be expressed in the form (9) are in fact linearly related to one another up to a scale factor, and the specific choice amongst them does not affect the algorithm's rate of convergence near to a stationary value (§ 6).

4. Rational and exponential fitting. One class of linear homogeneous differential equations for which there are corresponding difference equations is

$$p_a(\mathscr{D})(q(t;\boldsymbol{\gamma})\mu(t))=0$$

where $p_a(\cdot)$ is any polynomial with known coefficients, and $q(t;\boldsymbol{\gamma})$ is any continuous function of time, linear and homogeneous in $\boldsymbol{\gamma}$. The general solution of these equations has the form

$$\mu(t)=\frac{f(t)}{q(t;\boldsymbol{\gamma})}$$

where $f(t)$ is the general solution of $p_a(\mathscr{D})f(t)=0$.

Rational functions arise in the particular case that $p_a(\mathscr{D})=\mathscr{D}^p$ and $q(t;\boldsymbol{\gamma})=p_\gamma(t)=\sum_{j=1}^{q+1}\gamma_jt^{j-1}$. The general solution is then

$$\mu(t)=\frac{p_\alpha(t)}{p_\gamma(t)} \tag{10}$$

with $p_\alpha(t)=\sum_{j=1}^{p}\alpha_jt^{j-1}$. The rational function (10) satisfies the difference equation

$$\Delta^p(p_\gamma(t)\mu(t))=0 \tag{11}$$

which is of the form (1) with $d_k(t;\boldsymbol{\gamma}) = \binom{p}{k-1}\Delta^{p-k+1}p_\gamma(t)$. In matrix terms, (11) is

$$P\boldsymbol{\Delta}^p\langle p_\gamma(\mathbf{t})\rangle\boldsymbol{\mu} = 0$$

where $\boldsymbol{\Delta}$ is now the circulant difference matrix. This leads to

$$X^T = P\boldsymbol{\Delta}^p\langle p_\gamma(\mathbf{t})\rangle$$

and hence

$$X_j^T = \frac{dX^T}{d\gamma_j} = P\boldsymbol{\Delta}^p\langle \mathbf{t}^{j-1}\rangle$$

where $\langle \mathbf{t}^{j-1}\rangle$ denotes the diagonal matrix with elements t_1^{j-1} to t_n^{j-1}.

Let T be the $n \times p$ matrix with elements $T_{ij} = t_i^{j-1}$, so that $p_\alpha(t) = T\boldsymbol{\alpha}$. For rational fitting, the straightforward choice of C^T to be $\boldsymbol{\Delta}^p\langle p_\gamma(\mathbf{t})\rangle$ is not invertible. If we choose the last p rows of C^T to be instead $(T^TT)^{-1}T^T\langle p_\gamma(\mathbf{t})\rangle$, then the general formula (5) for $\boldsymbol{\mu}$ gives $A = \langle p_\gamma(\mathbf{t})\rangle^{-1}T$, and recovers the parametrization given above in (10).

Comparing rational functions with the general form (4), the denominator matrix $Z(\boldsymbol{\gamma})$ in this case is the diagonal matrix $\langle p_\gamma(\mathbf{t})\rangle$. This greatly simplifies the convergence proofs in §§ 9 and 10, and is the reason for focusing on rational functions in this paper. The matrix H is the difference operator $\boldsymbol{\Delta}^pP^T$ which has the polynomials as its null space. Extension of rational fitting to nonequally spaced t_i requires only a suitable choice of H. This is relatively straightforward; a direct formula is given in Osborne [13].

Suppose now that $\mu(t)$ satisfies the constant coefficient differential equation

$$\sum_{k=1}^{p+1} \xi_k \mathscr{D}^{k-1}\mu(t) = 0,$$

and that the polynomial $p_\xi(z) = \sum_{k=1}^{p+1}\xi_k z^{k-1}$ has real, distinct roots $-\beta_j$. Then the general solution for $\mu(t)$ may be expressed as a sum of exponential functions

$$\mu(t) = \sum_{j=1}^{p} \alpha_j e^{-\beta_j t}, \tag{12}$$

and the corresponding difference equation is

$$\sum_{k=1}^{p+1} \gamma_k \Delta^{k-1}\mu(t) = 0$$

where the polynomial $\sum_{k=1}^{p+1}\gamma_k z^{k-1}$ has roots $-\zeta_j = n(1-e^{-\beta_j/n})$. (Notice that $\zeta_j \to \beta_j$ as $n \to \infty$.) In matrix terms the difference equation is $PC^T\boldsymbol{\mu} = 0$ with

$$C^T = \sum_{k=1}^{p+1} \gamma_k \boldsymbol{\Delta}^{k-1}.$$

For exponential fitting, the C_j^T are simply $\boldsymbol{\Delta}^{j-1}$, and C is a (generally invertible) circulant matrix.

Since C has $p+1$ bands, it has $p+1$ distinct nonzero elements, $\mathbf{c}^T = (c_1, \cdots, c_{p+1})$, say. In Osborne [13], the algorithm was parametrized directly in terms of $\mathbf{c}$ rather than $\boldsymbol{\gamma}$. These two parametrizations will be compared in a separate paper on exponential fitting.

For exponential fitting, (5) expresses $\boldsymbol{\mu}$ as a linear combination of periodic solutions of the difference equation, rather than the more obvious definition with $A_{ij} = e^{-\beta_j t_i}$. Factoring C here, using the complex eigendecomposition available for circulant matrices, shows that rational and exponential fitting are, in a sense, dual problems.

Let $C = F\Lambda F^*$ where F is the Fourier matrix (Davis [3]) and Λ is the diagonal matrix of complex eigenvalues. Then

$$F^*\boldsymbol{\mu} = \Lambda^{-1}F^*Q^T\boldsymbol{\alpha},$$

which expresses the discrete Fourier transform of $\boldsymbol{\mu}$ as a rational function of the complex frequency $e^{2\pi\sqrt{-1}/n}$. In other words, exponential fitting corresponds to rational fitting in the frequency domain.

5. A modified Prony algorithm. We now consider the solution of (9). Henceforth we will scale $\boldsymbol{\gamma}$ to have norm one, so the problem becomes that of minimizing the reduced sum of squares $\psi(\boldsymbol{\gamma})$ subject to this constraint. Let

$$F(\boldsymbol{\gamma}, \lambda) = \psi(\boldsymbol{\gamma}) + \lambda(1 - \boldsymbol{\gamma}^T\boldsymbol{\gamma})$$

where λ is a Lagrange multipler. Then

$$\dot{F}_\gamma = 2B(\boldsymbol{\gamma})\boldsymbol{\gamma} - 2\lambda\boldsymbol{\gamma} \quad \text{and} \quad \dot{F}_\lambda = 1 - \boldsymbol{\gamma}^T\boldsymbol{\gamma},$$

so the necessary conditions for a stationary point are

$$(B(\boldsymbol{\gamma}) - \lambda I)\boldsymbol{\gamma} = 0 \tag{13}$$

and

$$\boldsymbol{\gamma}^T\boldsymbol{\gamma} = 1.$$

We now show that λ must be zero at a solution of (13). It can be seen from (8) that $\psi(\boldsymbol{\gamma})$ does not depend on $\|\boldsymbol{\gamma}\|$. Thus $\dot{\psi}$ must be in a direction orthogonal to $\boldsymbol{\gamma}$, so that

$$\boldsymbol{\gamma}^T\dot{\psi}(\boldsymbol{\gamma}) = 2\boldsymbol{\gamma}^TB(\boldsymbol{\gamma})\boldsymbol{\gamma} = 0. \tag{14}$$

Premultiplying $\dot{F}_\gamma = 0$ by $\boldsymbol{\gamma}^T$, and using (14) gives the result that $\lambda = 0$. This suggests the following iteration. Given a current estimate $\boldsymbol{\gamma}^k$, solve the linear eigenproblem

$$\begin{aligned}(B(\boldsymbol{\gamma}^k) - \lambda^{k+1}I)\boldsymbol{\gamma}^{k+1} &= 0\\ \boldsymbol{\gamma}^{k+1^T}\boldsymbol{\gamma}^{k+1} &= 1\end{aligned} \tag{15}$$

for $\boldsymbol{\gamma}^{k+1}$ and λ^{k+1}, with λ^{k+1} the nearest to zero of such solutions. Convergence can be accepted when λ^{k+1} is small compared with $\|B\|$. This iterative scheme is the *modified Prony algorithm.*

5.1. Constraint invariance. The constraint $\boldsymbol{\gamma}^T\boldsymbol{\gamma} = 1$, which we use here and in the remainder of this paper, is of course to some extent arbitrary. In particular, the result that the Lagrange multiplier is zero at a solution holds for any constraint $G(\boldsymbol{\gamma}) = 0$ such that $\hat{\boldsymbol{\gamma}}^T\dot{G}(\hat{\boldsymbol{\gamma}}) \neq 0$, for example, for any linear constraint $\boldsymbol{\kappa}^T\boldsymbol{\gamma} = 0$ such that $\boldsymbol{\kappa}$ is not orthogonal to $\hat{\boldsymbol{\gamma}}$. Quadratic norm constraints are of special interest because they correspond to linear reparametrizations of the Prony parameters. To see this, suppose we had chosen $\|\boldsymbol{\gamma}\| = 1$ with respect to some other inner product, say $\boldsymbol{\gamma}^TA\boldsymbol{\gamma} = 1$ with A a positive-definite matrix. The objective function would have been

$$F(\boldsymbol{\gamma}, \lambda) = \psi(\boldsymbol{\gamma}) + \lambda(1 - \boldsymbol{\gamma}^TA\boldsymbol{\gamma})$$

leading to the generalized nonlinear eigenproblem

$$(B(\boldsymbol{\gamma}) - \lambda A)\boldsymbol{\gamma} = 0$$

and to the sequence of linear problems

$$(B(\boldsymbol{\gamma}^k) - \lambda^{k+1}A)\boldsymbol{\gamma}^{k+1} = 0.$$

This is equivalent to

$$(A^{-1/2}B(\boldsymbol{\gamma}^k)A^{-T/2} - \lambda^{k+1}I)\boldsymbol{\delta}^{k+1} = 0$$

where $\boldsymbol{\delta} = A^{T/2}\boldsymbol{\gamma}$ and $A^{1/2}$ is any square root satisfying $A^{1/2}A^{T/2} = A$. If B has a zero at $\hat{\boldsymbol{\gamma}}$, then $A^{-1/2}BA^{-T/2}$ has a zero at $\hat{\boldsymbol{\delta}} = A^{T/2}\hat{\boldsymbol{\gamma}}$, so using A as the inner product matrix is equivalent to the linear reparametrization from $\boldsymbol{\gamma}$ to $\boldsymbol{\delta}$. This leads to a different modified Prony sequence, but one which has the same ultimate rate of convergence, as we show in § 6.

5.2. Implementation. When implemented efficiently, an iteration of the modified Prony algorithm requires computer time roughly similar to that of an iteration of a Gauss–Newton-type algorithm. The linear eigenproblem (15) can be solved extremely rapidly by inverse iteration (Wilkinson [23]). When testing for convergence, only an order of magnitude estimate of $\|B\|$ is required: we have found $1/(q+1)\sum_{ij}|B_{ij}|$ to be adequate. Our version of the modified Prony algorithm is given below:

$\boldsymbol{\gamma}^0 :=$ starting value
$\lambda^0 := 0$
$k := 0$
repeat
  $\varepsilon := \tau\,(1/(q+1))\sum_{ij}|B_{ij}|$
  $\mathbf{v}^0 := \boldsymbol{\gamma}^k$
  $\eta^0 := \lambda^k$
  $l := 0$
  {find eigenvalue of B closest λ^k by inverse iteration}
  repeat
    $\mathbf{w}^{l+1} := (B - \eta^l I)^{-1}\mathbf{v}^l$
    {normalize $\mathbf{w}^{l+1}$ for numerical stability}
    $\mathbf{v}^{l+1} := \mathbf{w}^{l+1}/\|\mathbf{w}^{l+1}\|_\infty$
    $\mathbf{w}^{l+2} := (B - \eta^l I)^{-1}\mathbf{v}^{l+1}$
    $\eta^{l+2} := \eta^l + \mathbf{w}^{l+2^T}\mathbf{v}^{l+1}/\mathbf{w}^{l+2^T}\mathbf{w}^{l+2}$
    $\mathbf{v}^{l+2} := \mathbf{w}^{l+2}/\|\mathbf{w}^{l+2}\|_2$
    $l := l+2$
  until $|\eta^l - \eta^{l-2}| < \varepsilon$
  $\boldsymbol{\gamma}^{k+1} := \mathbf{v}^l$
  $\lambda^{k+1} := \eta^l$
  $k := k+1$
until $|\lambda^k| < \varepsilon$

In the above algorithm, the symmetric system

$$(B - \eta^l I)\mathbf{w}^{l+1} = \mathbf{v}^l \tag{16}$$

may be solved by Gaussian elimination, or by a method specially adapted to symmetric systems such as diagonal pivoting (Bunch and Kaufman [2]). Inverse iteration is implemented in a similar way to that suggested by Osborne [15], with the feature that $B - \eta^l I$ is factored once to solve two linear systems. The choice of the constant τ (tolerance) reflects the precision required and the maximum condition number of a matrix that can be stored in finite arithmetic. For example, in double precision on a 36-bit machine, $\tau = 10^{-15}$ gives precise solutions.

Empirical evidence with exponential fitting suggests that the modified Prony algorithm has an impressive ability to converge to some stationary value even from quite poor starting values. For exponential fitting the algorithm may be started from

$\gamma = \mathbf{e}_1$ (the first coordinate vector), for which $\mu(t)$ is constant. For rational fitting, in contrast, good starting values are often required and no general recipe can be given.

5.3. Calculation of B. The efficient calculation of the Prony matrix B is the key to the efficient implementation of the algorithm. This may be done in $O(nq^2)$ flops, for which the banded structure of X is crucial. The following scheme is suggested for computation.

$Y := (X_1^T\mathbf{y} \cdots X_{q+1}^T\mathbf{y})$	An $(n-p)\times(q+1)$ matrix which does not depend on γ.
$M := (X^TX)^{1/2}$	The Choleski factor, a lower triangular $(n-p)^2$ matrix with $q+1$ bands.
$Y := M^{-1}Y$	$(X^TX)^{-1/2}X_j^T\mathbf{y}$, $j=1,\cdots,q+1$.
$\mathbf{v} := M^{-T}Y\gamma$	$(X^TX)^{-1}X^T\mathbf{y}$, an $n-p$ vector.
$V := (X_1\mathbf{v} \cdots X_{q+1}\mathbf{v})$	$X_j(X^TX)^{-1}X^T\mathbf{y}$, $j=1,\cdots,q+1$, and $n\times(q+1)$ matrix.
$B := Y^TY - V^TV$	

If desired, the fitted means may be extracted easily by

$$\hat{\boldsymbol{\mu}} = P_X\mathbf{y} = V\gamma. \tag{17}$$

For rational fitting the $X_j^T\mathbf{y}$ are $P\Delta^p\langle\mathbf{t}^{j-1}\rangle\mathbf{y}$, which can be calculated by repeatedly differencing the $\langle\mathbf{t}^{j-1}\rangle\mathbf{y}$. Also X^T is $P\Delta^p\langle p_\gamma(\mathbf{t})\rangle$. For rational fitting, it is efficient to obtain V^TV from $\mathbf{w} = \Delta^{pT}P^T\mathbf{v}$ and

$$(V^TV)_{ij} = \mathbf{w}^T\langle\mathbf{t}^{i+j-2}\rangle\mathbf{w}$$

without first calculating and storing V.

For exponential fitting, $X_j^T\mathbf{y} = P\Delta^{j-1}\mathbf{y}$, which can be calculated by recursive differencing, and similarly for $X_j\mathbf{v} = \Delta^{j-1T}P^T\mathbf{v}$. In the exponential fitting case, the matrix X is Toeplitz as well as banded, so only the $p+1$ nonzero components of the first column need to be stored. These components, $\mathbf{c}^T = (c_1, \cdots, c_{p+1})$ say, are related to γ through $\mathbf{c} = U\gamma$ with

$$U = \begin{pmatrix} 1 & -1 & 1 & \cdots & & (-1)^p \\ & 1 & -2 & & & \\ & & 1 & & & \\ & & & \ddots & & \vdots \\ & & & & 1 & -\binom{p}{1} \\ & & & & & 1 \end{pmatrix} \begin{pmatrix} 1 & & & \\ & n & & \\ & & \ddots & \\ & & & n^p \end{pmatrix},$$

and again this can be calculated recursively.

6. The convergence matrix. In this section we derive sufficient conditions for local convergence of the modified Prony algorithm. Let us express the modified Prony iteration formally as

$$\gamma^{k+1} = F(\gamma^k)$$

where F is defined implicitly by (15). The iteration has a fixed point at the least squares estimate. It has a point of attraction there also if it converges to $\hat{\gamma}$ from any point in a surrounding neighbourhood. In that case we say it is *stable* at $\hat{\gamma}$. A sufficient condition for stability is given by Ostrowski's theorem (Ortega and Rheinboldt [12]), namely, that F is Fréchet differentiable and

$$\rho(\dot{F}(\hat{\gamma})) < 1$$

where $\rho(\cdot)$ denotes spectral radius. Following Smyth [20] and Kass and Smyth [8], we call

$$\dot{F}(\hat{\boldsymbol{\gamma}}) = \frac{d\boldsymbol{\gamma}^{k+1}}{d\boldsymbol{\gamma}^{k}}(\hat{\boldsymbol{\gamma}})$$

the *convergence matrix* and $\rho(\dot{F}(\hat{\boldsymbol{\gamma}}))$ the *convergence factor.*

We now obtain the convergence matrix of the modified Prony algorithm. The theorem assumes that the null space of $B(\hat{\boldsymbol{\gamma}})$ has dimension one, so that it is spanned by $\hat{\boldsymbol{\gamma}}$. That this is asymptotically so, is proved in § 9. The derivative $\dot{B}$ of B with respect to $\boldsymbol{\gamma}$ is a three-dimensional tensor, to be thought of as an array of partial derivatives $\partial B/\partial \gamma_k$, $k = 1, \cdots, q+1$. Therefore $\dot{B}(\boldsymbol{\gamma})\boldsymbol{\gamma}$ denotes the $(q+1)\times(q+1)$ matrix with kth column $(\partial B/\partial \gamma_k)\boldsymbol{\gamma}$.

LEMMA 1. *For all* $\boldsymbol{\gamma}$, $\boldsymbol{\gamma}^T\dot{\boldsymbol{\gamma}} = 0$ *and* $\boldsymbol{\gamma}^T B(\boldsymbol{\gamma})\boldsymbol{\gamma} = 0$. *At the least squares estimate* $\hat{\boldsymbol{\gamma}}^T\dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}} = 0$.

Proof. The first result follows from differentiating $\boldsymbol{\gamma}^T\boldsymbol{\gamma} = 1$. The second has already been mentioned in § 5, and follows from a geometric argument. The third follows from it by differentiating, since $B(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}} = 0$. □

THEOREM 1. *The convergence matrix of the modified Prony algorithm is*

$$B(\hat{\boldsymbol{\gamma}})^+\dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}}$$

where $B(\hat{\boldsymbol{\gamma}})^+$ *is the Moore–Penrose inverse of* $B(\hat{\boldsymbol{\gamma}})$.

Proof. Differentiating (15) with respect to $\boldsymbol{\gamma}^k$, and evaluating at $\hat{\boldsymbol{\gamma}}$ gives

$$(\dot{B}(\hat{\boldsymbol{\gamma}}) - \dot{\lambda}(\hat{\boldsymbol{\gamma}})I)\hat{\boldsymbol{\gamma}} + (B(\hat{\boldsymbol{\gamma}}) - \lambda(\hat{\boldsymbol{\gamma}})I)\dot{F}(\hat{\boldsymbol{\gamma}}) = 0. \tag{18}$$

It was shown in § 5 that $\lambda(\hat{\boldsymbol{\gamma}}) = 0$. Premultiplying (18) by $\hat{\boldsymbol{\gamma}}^T$ and applying Lemma 1 and $B(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}} = 0$, shows that $\dot{\lambda}(\hat{\boldsymbol{\gamma}}) = 0$ also. So we have

$$\dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}} + B(\hat{\boldsymbol{\gamma}})\dot{F}(\hat{\boldsymbol{\gamma}}) = 0.$$

But

$$B\dot{\boldsymbol{\gamma}} = (B + \boldsymbol{\gamma}\boldsymbol{\gamma}^T)\dot{\boldsymbol{\gamma}}$$

since $\boldsymbol{\gamma}^T\dot{\boldsymbol{\gamma}} = 0$, so

$$\begin{aligned}\dot{F}(\hat{\boldsymbol{\gamma}}) &= -(B(\hat{\boldsymbol{\gamma}}) + \hat{\boldsymbol{\gamma}}\hat{\boldsymbol{\gamma}}^T)^{-1}\dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}} \\ &= -(B(\hat{\boldsymbol{\gamma}})^+ + \hat{\boldsymbol{\gamma}}\hat{\boldsymbol{\gamma}}^T)\dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}} \\ &= -B(\hat{\boldsymbol{\gamma}})^+\dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}}\end{aligned}$$

applying Lemma 1 again. □

Differentiating (9) gives

$$\ddot{\psi}(\boldsymbol{\gamma}) = 2B(\boldsymbol{\gamma}) + 2\dot{B}(\boldsymbol{\gamma})\boldsymbol{\gamma},$$

which shows that the convergence matrix measures the relative difference between B and $\frac{1}{2}\ddot{\psi}$. This result is analogous to that for the Gauss–Newton algorithm, which converges when the Fisher information matrix $\dot{\boldsymbol{\mu}}^T\dot{\boldsymbol{\mu}}$ is a good approximation to $\frac{1}{2}\ddot{\phi}$.

The following lemma and theorem show that all linearly related Prony parametrizations have similar convergence matrices.

LEMMA 2. *If* B *is a symmetric matrix with null space spanned by* $\boldsymbol{\gamma}$ $(\boldsymbol{\gamma}^T\boldsymbol{\gamma} = 1)$, *then*

$$B^+ = \lim_{\varepsilon \to 0} (I - \boldsymbol{\gamma}\boldsymbol{\gamma}^T)(B + \varepsilon \mathbf{r}\mathbf{r}^T)^{-1}(I - \boldsymbol{\gamma}\boldsymbol{\gamma}^T)$$

for any $\mathbf{r}$ *for which* $\mathbf{r}^T\boldsymbol{\gamma} \neq 0$.

Proof. Let

$$B = (Z \quad \boldsymbol{\gamma}) \begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} Z^T \\ \boldsymbol{\gamma}^T \end{pmatrix}$$

be an eigendecomposition of B, so that

$$\begin{aligned}(B+\varepsilon \mathbf{r}\mathbf{r}^T) &= (Z \quad \boldsymbol{\gamma}) \left[\begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix} + \varepsilon \begin{pmatrix} Z^T \\ \boldsymbol{\gamma}^T \end{pmatrix} \mathbf{r}\mathbf{r}^T (Z \quad \boldsymbol{\gamma}) \right] \begin{pmatrix} Z^T \\ \boldsymbol{\gamma}^T \end{pmatrix} \\ &= (Z \quad \boldsymbol{\gamma}) \left[\begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix} + \varepsilon \begin{pmatrix} Z^T\mathbf{r}\mathbf{r}^TZ & Z^T\mathbf{r}\mathbf{r}^T\boldsymbol{\gamma} \\ \boldsymbol{\gamma}^T\mathbf{r}\mathbf{r}^TZ & (\boldsymbol{\gamma}^T\mathbf{r})^2 \end{pmatrix} \right] \begin{pmatrix} Z^T \\ \boldsymbol{\gamma}^T \end{pmatrix}\end{aligned}$$

and

$$(B+\varepsilon \mathbf{r}\mathbf{r}^T)^{-1} = (Z \quad \boldsymbol{\gamma}) \begin{bmatrix} (\Lambda + \varepsilon Z^T\mathbf{r}\mathbf{r}^TZ + \varepsilon \mathbf{r}^TZZ^T\mathbf{r})^{-1} & \cdot \\ & \cdot \end{bmatrix} \begin{pmatrix} Z^T \\ \boldsymbol{\gamma}^T \end{pmatrix}$$

since $\boldsymbol{\gamma}^T\mathbf{r} \neq 0$. Hence

$$\begin{aligned}(I-\boldsymbol{\gamma}\boldsymbol{\gamma}^T)(B+\varepsilon \mathbf{r}\mathbf{r}^T)^{-1}(I-\boldsymbol{\gamma}\boldsymbol{\gamma}^T) &= (Z \quad 0) \begin{bmatrix} \Lambda^{-1}+o(\varepsilon) & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{pmatrix} Z^T \\ 0 \end{pmatrix} \\ &= Z(\Lambda^{-1}+o(\varepsilon))Z^T \\ &= B^+ + o(\varepsilon). \end{aligned}$$

□

THEOREM 2. *The eigenvalues of $B^+\dot{B}\boldsymbol{\gamma}$ are invariant under linear reparametrization.*

Proof. Let $\boldsymbol{\delta} = U^{-1}\boldsymbol{\gamma}$ with U a nonsingular known matrix, and let B_δ and B_γ be the Prony matrices for $\boldsymbol{\delta}$ and $\boldsymbol{\gamma}$, respectively. Now

$$\dot{\psi}_\delta = U^T\dot{\psi}_\gamma,$$

so

$$B_\delta\boldsymbol{\delta} = U^TB_\gamma\boldsymbol{\gamma} = U^TB_\gamma U\boldsymbol{\delta}$$

and

$$B_\delta = U^TB_\gamma U.$$

Differentiating again

$$\begin{aligned}\dot{B}_\delta\boldsymbol{\delta} &= U^T(\dot{B}_\gamma U\boldsymbol{\delta})U \\ &= U^T(\dot{B}_\gamma\boldsymbol{\gamma})U.\end{aligned}$$

Hence

$$\begin{aligned}B_\delta^+\dot{B}_\delta\boldsymbol{\delta} &= \lim_{\varepsilon\to 0}(I-\boldsymbol{\delta}(\boldsymbol{\delta}^T\boldsymbol{\delta})^{-1}\boldsymbol{\delta}^T)(B_\delta+\varepsilon\boldsymbol{\delta}\boldsymbol{\delta}^T)^{-1}\dot{B}_\delta\boldsymbol{\delta} \\ &= \lim_{\varepsilon\to 0}(I-\boldsymbol{\delta}(\boldsymbol{\delta}^T\boldsymbol{\delta})^{-1}\boldsymbol{\delta}^T)(U^TB_\gamma U+\varepsilon\boldsymbol{\delta}\boldsymbol{\delta}^T)^{-1}U^T(\dot{B}_\gamma\boldsymbol{\gamma})U \\ &= \lim_{\varepsilon\to 0}U^{-1}U(I-\boldsymbol{\delta}(\boldsymbol{\delta}^T\boldsymbol{\delta})^{-1}\boldsymbol{\delta}^T)U^{-1}(B_\gamma+\varepsilon U^{-T}\boldsymbol{\delta}\boldsymbol{\delta}^TU^{-1})^{-1}\dot{B}_\gamma\boldsymbol{\gamma}U \\ &= \lim_{\varepsilon\to 0}U^{-1}(I-\boldsymbol{\gamma}(\boldsymbol{\gamma}^T\boldsymbol{\gamma})^{-1}\boldsymbol{\gamma}^T)(B_\gamma+\varepsilon U^{-T}\boldsymbol{\delta}\boldsymbol{\delta}^TU^{-1})^{-1}\dot{B}_\gamma\boldsymbol{\gamma}U \\ &= U^{-1}B_\gamma^+\dot{B}_\gamma\boldsymbol{\gamma}U,\end{aligned}$$

since $\boldsymbol{\delta}^TU^{-1}\boldsymbol{\gamma} = \boldsymbol{\delta}^T\boldsymbol{\delta} \neq 0$, and this is similar to

$$B_\gamma^+\dot{B}_\gamma\boldsymbol{\gamma}.$$

□

7. Expectations. Calculating expectations for B and $\dot{B}\boldsymbol{\gamma}$ shows that the convergence factor converges to zero as the variance of the y_i, $\sigma^2 \to 0$. (Sections 9 and 10 show that it also converges to zero for fixed σ^2 and $n \to \infty$.)

THEOREM 3.

$$\mathbb{E}(B(\boldsymbol{\gamma}_0)) = V_0$$

where

$$V_{0ij} = \boldsymbol{\mu}^T X_i (X^T X)^{-1} X_j^T \boldsymbol{\mu}$$

and

$$\mathbb{E}(\dot{B}(\boldsymbol{\gamma}_0)\boldsymbol{\gamma}_0) = 0.$$

Proof. All expressions in this proof are to be evaluated at the true value $\boldsymbol{\gamma}_0$. Using $X^T\mathbf{y} = X^T(\mathbf{y}-\boldsymbol{\mu})$, and the standard identities for rearranging terms in matrix traces, we can write

$$\begin{aligned} B_{ij} = &\operatorname{tr} X_i (X^T X)^{-1} X_j^T \mathbf{y}\mathbf{y}^T \\ &- \operatorname{tr} X (X^T X)^{-1} X_i^T X_j (X^T X)^{-1} X^T (\mathbf{y}-\boldsymbol{\mu})(\mathbf{y}-\boldsymbol{\mu})^T \end{aligned}$$

which has expectation

$$\begin{aligned} &\operatorname{tr} X_i (X^T X)^{-1} X_j^T (I\sigma^2 + \boldsymbol{\mu}\boldsymbol{\mu}^T) - \operatorname{tr} X (X^T X)^{-1} X_i^T X_j (X^T X)^{-1} X^T I\sigma^2 \\ &= \boldsymbol{\mu}^T X_i (X^T X)^{-1} X_j^T \boldsymbol{\mu}. \end{aligned}$$

To do the same for $\dot{B}\boldsymbol{\gamma}$ we need an explicit expression. The simplest way is to differentiate

$$\psi = \mathbf{y}^T P_X \mathbf{y}$$

twice with respect to $\boldsymbol{\gamma}$, and to use

$$\dot{B}\boldsymbol{\gamma} = \tfrac{1}{2}\ddot{\psi} - B.$$

This gives

$$\begin{aligned} (\dot{B}\boldsymbol{\gamma})_{ij} = &-\mathbf{y}^T X_i (X^T X)^{-1} X_j^T X (X^T X)^{-1} X^T \mathbf{y} - \text{symmetry term} \\ &- \mathbf{y}^T X_i (X^T X)^{-1} X^T X_j (X^T X)^{-1} X^T \mathbf{y} - \text{symmetry term} \\ &+ \mathbf{y}^T X (X^T X)^{-1} X_i^T X (X^T X)^{-1} X_j^T X (X^T X)^{-1} X^T \mathbf{y} \\ &+ \text{symmetry term} \\ &+ \mathbf{y}^T X (X^T X)^{-1} X^T X_i (X^T X)^{-1} X_j^T X (X^T X)^{-1} X^T \mathbf{y} \\ &+ \mathbf{y}^T X (X^T X)^{-1} X_i^T X (X^T X)^{-1} X^T X_j (X^T X)^{-1} X^T \mathbf{y} \end{aligned}$$

where "symmetry term" means as for the last term but with i and j interchanged. The theorem follows by writing $X^T\mathbf{y} = X^T(\mathbf{y}-\boldsymbol{\mu})$, applying the trace identities, and canceling out like terms. □

COROLLARY 1. $B(\hat{\boldsymbol{\gamma}})^+ \dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}} \to 0$ *as* $\sigma^2 \to 0$.

Proof. As in the proof of Theorem 1, we use Lemma 1 to write

$$B(\hat{\boldsymbol{\gamma}})^+ \dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}} = (B(\hat{\boldsymbol{\gamma}}) + \hat{\boldsymbol{\gamma}}\hat{\boldsymbol{\gamma}}^T)^{-1} \dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}}. \tag{19}$$

Now $B(\boldsymbol{\gamma}_0) \to V_0$, $\dot{B}(\boldsymbol{\gamma}_0)\boldsymbol{\gamma}_0 \to 0$ and $\hat{\boldsymbol{\gamma}} \to \boldsymbol{\gamma}_0$ as $\sigma^2 \to 0$, so (19) has the same limit as

$$(B(\boldsymbol{\gamma}_0) + \boldsymbol{\gamma}_0\boldsymbol{\gamma}_0^T)^{-1} \dot{B}(\boldsymbol{\gamma}_0)\boldsymbol{\gamma}_0,$$

namely,

$$(V_0 + \boldsymbol{\gamma}_0\boldsymbol{\gamma}_0^T)^{-1} 0 = 0.$$

□

Note that

$$V_0 = \dot{\boldsymbol{\mu}}_\gamma^T P_X \dot{\boldsymbol{\mu}}_\gamma$$

since $X^T \boldsymbol{\mu} = 0$ implies that

$$X^T \dot{\boldsymbol{\mu}}_\gamma = -\dot{X}_\gamma^T \boldsymbol{\mu}.$$

One interpretation of V_0 is that $\sigma^{-2} V_0$ is the Fisher information matrix for $\boldsymbol{\gamma}$, conditional on the linear parameters $\boldsymbol{\alpha}$.

8. Asymptotic assumptions. Most of the remainder of this paper will deal with asymptotic results, so we now need to make specific assumptions about our asymptotic framework. These are basically that y and μ satisfy regularity conditions, and that μ satisfies a differential equation like (3) and its discrete analogue (1). The parameters $\boldsymbol{\xi}$ in (3) parametrize p special solutions of the differential equation, and the function μ can be expressed as a linear combination of these, with coefficients $\alpha_1, \cdots, \alpha_p$ say. Let $\boldsymbol{\theta} = (\boldsymbol{\alpha}^T, \boldsymbol{\xi}^T)^T$. Assume the following:

(a) $\boldsymbol{\theta} \in \Theta$, where Θ is a compact subset of $\mathbb{R}^p \times S$ with S the unit sphere in $\mathbb{R}^{q+1}$, which contains the true value $\boldsymbol{\theta}_0$ as an interior point. The errors $\varepsilon_{n,i} = y_i - \mu_i(\boldsymbol{\theta}_0)$ form a triangular array of independent deviates, which are identically distributed with mean 0 and finite variance σ^2. The time points t_i are equally spaced on the unit interval, so that $t_i = i/n$.

(b) The function

$$\int_0^1 (\mu(t; \boldsymbol{\theta}_0) - \mu(t; \boldsymbol{\theta}))^2 \, dt$$

has a unique minimum in Θ at $\boldsymbol{\theta}_0$.

(c) The function μ is twice continuously differentiable with respect to $\boldsymbol{\xi}$, and p-times continuously differentiable with respect to t. All these derivatives are jointly continuous in t and $\boldsymbol{\theta}$.

(d) The information matrix $\mathcal{I}$, defined by

$$\mathcal{I}_{ij} = \int_0^1 \frac{\partial \mu}{\partial \theta_i}(t; \boldsymbol{\theta}_0) \frac{\partial \mu}{\partial \theta_j}(t; \boldsymbol{\theta}_0) \, dt$$

is positive semidefinite. Its null space has dimension one, and is spanned by $(0, \boldsymbol{\xi}_0^T)^T$.

(e) The difference equation (1) converges to the differential equation (3) in the sense that the d_k converge to the b_k and $\boldsymbol{\gamma}$ converges to $\boldsymbol{\xi}$.

Conditions (b)–(d) are similar to Jennrich [7]. Condition (b) asserts that μ is parametrized uniquely at $\boldsymbol{\theta}_0$, while (d) guarantees that $\boldsymbol{\theta}$ is estimable up to a scale factor for $\boldsymbol{\xi}$.

We need the following form of the law of large numbers.

THEOREM 4 (Law of large numbers). *If $f(t; \boldsymbol{\theta})$ is jointly continuous on $[0, 1] \times \Theta$, then, with probability* 1

$$\frac{1}{n} \sum_{i=1}^{n} f(t_i; \boldsymbol{\theta}) \varepsilon_{n,i} \to 0$$

uniformly in $\boldsymbol{\theta}$.

Proof. This result is not proved in detail because it is of a standard type. For fixed $\boldsymbol{\theta}$, the result follows from Theorem 4.1.3 of Stout [22] or from Corollary 1 of Stout [21], and from their generalizations to triangular arrays of errors as described in Stout [21]. The proof that the result holds uniformly in $\boldsymbol{\theta}$ follows closely that of Theorem 4 of Jennrich [7]. □

Under the above conditions, the least squares estimator $\hat{\boldsymbol{\theta}}$ can be shown to be strongly consistent and asymptotically normal. Proofs would be similar to those of Jennrich [7].

The above conditions generally hold for exponential fitting, and for rational fitting provided that the denominator polynomial $p_\gamma(t)$ is nonzero on $[0, 1]$.

9. Eigenstructure of B.

THEOREM 5. *The least and greatest singular values of X are $O(1)$ and $O(n^p)$, respectively.*

Proof. The vectors which correspond to small singular values of X are obtained from realizations of smooth functions at the time points t_i. Let f be a p times continuously differentiable function on $[0, 1]$ that is not a solution of the differential equation (3). Write

$$\|f\|^2 = \int_0^1 f(t)^2\, dt$$

and let

$$\mathbf{z} = \|f(\mathbf{t})\|^{-1} f(\mathbf{t}).$$

Now let $i \to \infty$ in such a way that $t_i = i/n \to t$. Then

$$(X^T\mathbf{z})_i = \|f(\mathbf{t})\|^{-1} \sum_{k=1}^{p+1} d_k(t_i; \boldsymbol{\gamma}) \Delta^k f(t_i)$$

$$\to \|f\|^{-1} \sum_{k=1}^{p+1} b_k(t; \boldsymbol{\xi}) \mathscr{D}^k f(t),$$

which is a nonzero constant. On the other hand, the elements of X are $O(n^p)$ through the contribution of $\boldsymbol{\Delta}^p$, so the largest singular value is $O(n^p)$. □

Theorem 5 implies that the condition number of X will be large, and appears to have consequences for the numerical accuracy of $(X^TX)^{1/2}$, B and $\hat{\boldsymbol{\mu}}$. Experience with the exponential and rational fitting examples suggests, though, that this problem is mitigated by the fact that X possesses a compact analytic specification in terms of the Prony parameters.

The condition number of B itself is $O(1)$, as is shown by the following theorem. The form of the proof given here is quite general, although it is completed only for rational functions. Rational functions offer the least complication because the operators D_k (see proof) are already diagonal. For exponential fitting the proof must be prepared by a series of rather long lemmas which diagonalize the D_k using discrete Fourier transform methods; these were proved by Smyth [19] and, because of their length, will appear separately.

THEOREM 6. *In the rational fitting case,*

$$\frac{1}{n} B(\hat{\boldsymbol{\gamma}}) \xrightarrow{\text{a.s.}} \mathscr{J}_0$$

where

$$\mathscr{J}_0 = \lim_{n\to\infty} \frac{1}{n} \dot{\boldsymbol{\mu}}_\gamma^T P_X \dot{\boldsymbol{\mu}}_\gamma(\boldsymbol{\gamma}_0).$$

Proof. We need to establish that $B(\boldsymbol{\gamma})$ converges almost surely, uniformly in a neighbourhood of $\boldsymbol{\gamma}_0$, to a continuous function $\mathcal{J}(\boldsymbol{\gamma})$, which is equal to $\mathcal{J}_0$ at $\boldsymbol{\gamma}_0$. It will be convenient to write

$$D_k = C_k C^{-1}$$

for $k = 1, \cdots, q+1$. In the rational fitting case, the D_k are the diagonal matrices

$$\langle p_\gamma(\mathbf{t})^{-1} \mathbf{t}^{k-1} \rangle.$$

Let $P = (I_{n-p}\ 0)$. Substituting

$$X_i^T = PC_i^T = PC^T C^{-T} C_i^T = X^T D_i^T$$

into the expression for B gives

$$B_{ij} = \mathbf{y}^T D_i P_X D_j^T \mathbf{y} - \mathbf{y}^T P_X D_i^T D_j P_X \mathbf{y}.$$

Expanding P_X as $I - P_A$ (as in § 3) and $\mathbf{y}$ as $\boldsymbol{\mu}_0 + \boldsymbol{\varepsilon}$ gives

$$\frac{1}{n} B_{ij} = \frac{1}{n} (\boldsymbol{\mu}_0 + \boldsymbol{\varepsilon})^T D_i (I - P_A) D_j^T (\boldsymbol{\mu}_0 + \boldsymbol{\varepsilon}) - \frac{1}{n} (\boldsymbol{\mu}_0 + \boldsymbol{\varepsilon})^T (I - P_A) D_i^T D_j (I - P_A)(\boldsymbol{\mu}_0 + \boldsymbol{\varepsilon}),$$

and we consider the terms appearing in this expansion individually. The two terms

$$\frac{1}{n} \boldsymbol{\varepsilon}^T D_i D_j^T \boldsymbol{\varepsilon} - \frac{1}{n} \boldsymbol{\varepsilon}^T D_i^T D_j \boldsymbol{\varepsilon}$$

cancel out, since D_i and D_j are diagonal matrices. Repeated application of Theorem 4 shows that all other terms which involve $\boldsymbol{\varepsilon}$ converge to zero. Consider, for example, the term

$$\frac{1}{n} \boldsymbol{\mu}_0^T D_i P_A D_j^T \boldsymbol{\varepsilon} = \left(\frac{1}{n} \boldsymbol{\mu}_0^T D_i A \right) \left(\frac{1}{n} A^T A \right)^{-1} \left(\frac{1}{n} A^T D_j^T \boldsymbol{\varepsilon} \right).$$

The kth element of $(1/n) A^T D_j^T \boldsymbol{\varepsilon}$ is

$$\frac{1}{n} \sum_{i=1}^{n} \frac{t_i^{j+k-2}}{p_\gamma(t_i)^2} \varepsilon_{n,i},$$

which converges to zero almost surely by Theorem 4. The whole term converges to zero almost surely, since $(1/n) \boldsymbol{\mu}_0^T D_i A$ converges to a constant vector with elements

$$\int_0^1 \mu_0(t) \frac{t^{i+k-2}}{p_\gamma(t)^2} \, dt$$

for $k = 1, \cdots, p$, while $(1/n) A^T A$ converges to a constant positive-definite matrix with elements

$$\int_0^1 \frac{t^{k+l}}{p_\gamma(t)^2} \, dt$$

for $k, l = 1, \cdots, p$. Moreover, the convergence is uniform for $\boldsymbol{\gamma}$ in a compact set. Similarly, the term

$$\boldsymbol{\varepsilon}^T P_A D_i^T D_j P_A \boldsymbol{\varepsilon} = \left(\frac{1}{n} \boldsymbol{\varepsilon}^T A \right) \left(\frac{1}{n} A^T A \right)^{-1} \left(\frac{1}{n} A^T D_i^T D_j A \right) \left(\frac{1}{n} A^T A \right)^{-1} \left(\frac{1}{n} A^T \boldsymbol{\varepsilon} \right)$$

causes no problems, since $(1/n)A^T D_i^T D_j A$ converges to a constant $p \times p$ matrix, and Theorem 4 is applied to show that each of the $(1/n)A^T \boldsymbol{\varepsilon}$ converge to zero. This term is in fact of smaller order than the first.

The only terms in the expansion for B_{ij}/n which remain are

$$\frac{1}{n}\boldsymbol{\mu}_0^T D_i (I - P_A) D_j^T \boldsymbol{\mu}_0 - \frac{1}{n}\boldsymbol{\mu}_0^T (I - P_A) D_i^T D_j (I - P_A)\boldsymbol{\mu}_0,$$

and these converge to a constant, $\mathcal{J}_{ij}$ say. Again the convergence is uniform for $\boldsymbol{\gamma}$ in a compact set. The proof is completed by gathering the $\mathcal{J}_{ij}$ into a matrix function $\mathcal{J}(\boldsymbol{\gamma})$ of $\boldsymbol{\gamma}$, and observing that $\mathcal{J}(\boldsymbol{\gamma}_0) = \mathcal{J}_0$. □

COROLLARY 2. *With probability one, $n^{-1}B(\hat{\boldsymbol{\gamma}})$ has a positive-semidefinite limit, the Moore–Penrose inverse of which is the asymptotic covariance matrix of $n^{1/2}\sigma^{-1}\hat{\boldsymbol{\gamma}}$.*

COROLLARY 3. *With probability one, the zero eigenvalue of $B(\hat{\boldsymbol{\gamma}})$ is asymptotically isolated with multiplicity one.*

Proof. $n^{-1}B(\hat{\boldsymbol{\gamma}})$ asymptotically approximates $\mathcal{J}_0$, which by the assumptions of § 8 has one zero eigenvalue with eigenvector $\boldsymbol{\gamma}_0$. □

10. Asymptotic stability.

THEOREM 7. *The modified Prony iteration applied to rational fitting is almost surely asymptotically stable.*

Proof. Theorem 1 gives an expression for the convergence matrix, which can be rewritten using Lemma 1 and Corollary 3 of Theorem 6 as

$$\left(\frac{1}{n}B(\hat{\boldsymbol{\gamma}}) + \hat{\boldsymbol{\gamma}}\hat{\boldsymbol{\gamma}}^T\right)^{-1} \frac{1}{n}\dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}}. \tag{20}$$

It is sufficient to prove that (20) converges to zero almost surely. Theorem 6 shows that $(1/n)B(\hat{\boldsymbol{\gamma}}) + \hat{\boldsymbol{\gamma}}\hat{\boldsymbol{\gamma}}^T$ converges to the positive definite matrix $\mathcal{J}_0 + \boldsymbol{\gamma}_0\boldsymbol{\gamma}_0^T$. It can also be proved that

$$\frac{1}{n}\dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}} \xrightarrow{\text{a.s.}} 0,$$

although the proof is not given here because it is tedious and follows the same form as that of Theorem 6 (it uses the expansion for $\dot{B}(\boldsymbol{\gamma})\boldsymbol{\gamma}$ given in § 7). This establishes the proof: the matrix ratio (20) converges to zero because its numerator converges to zero, while its denominator converges to a positive-definite matrix. □

The asymptotic stability of the modified Prony algorithm is actually closely related to the convergence of the usual Hessian of the sum of squares $\ddot{\phi}/n$. Consider the rational function parametrization in terms of $\boldsymbol{\beta}^T = (\gamma_2 \cdots \gamma_{q+1})/\gamma_1$. We can write $\boldsymbol{\mu} = A(\boldsymbol{\beta})\boldsymbol{\alpha}$ with

$$A_{ij} = \frac{t_i^{j-1}}{1 + \sum_{k=1}^{q} \beta_k t^k}.$$

The Jacobian of the transformation $\dot{\boldsymbol{\gamma}}_\beta$ is a rectangular matrix with full column rank, and null space spanned by $\boldsymbol{\gamma}$. Therefore, $(\dot{\boldsymbol{\gamma}}_\beta(\boldsymbol{\gamma})\ \boldsymbol{\gamma})$ is a square nonsingular matrix. The, now standard, results of Jennrich [7] can be applied to show that

$$\lim_{n\to\infty} \frac{1}{n}\ddot{\phi}_\theta(\hat{\boldsymbol{\theta}}) \overset{\text{a.s.}}{=} \lim_{n\to\infty} \frac{2}{n}\dot{\boldsymbol{\mu}}_\theta(\boldsymbol{\theta}_0)^T \dot{\boldsymbol{\mu}}_\theta(\boldsymbol{\theta}_0)$$

which is positive definite. So

$$\frac{1}{n}\ddot{\psi}_\beta(\hat{\boldsymbol{\beta}}) = \frac{1}{n}(\ddot{\phi}_\beta - \ddot{\phi}_{\beta\alpha}\ddot{\phi}_\alpha^{-1}\ddot{\phi}_{\alpha\beta})(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}})$$

almost surely has the same limit as

$$\frac{2}{n}\dot{\boldsymbol{\mu}}_\beta(I-P_A)\dot{\boldsymbol{\mu}}_\beta(\boldsymbol{\theta}_0)=\frac{2}{n}\dot{\boldsymbol{\mu}}_\beta P_X\dot{\boldsymbol{\mu}}_\beta(\boldsymbol{\theta}_0),$$

which again is positive definite. Therefore

$$\frac{1}{2n}\ddot{\psi}_\gamma(\hat{\boldsymbol{\gamma}})=\frac{1}{n}(\dot{\boldsymbol{\gamma}}_\beta(\hat{\boldsymbol{\beta}}) \quad \hat{\boldsymbol{\gamma}})^{-T}\begin{pmatrix}\ddot{\psi}_\beta(\hat{\boldsymbol{\beta}}) & 0\\ 0 & 0\end{pmatrix}(\dot{\boldsymbol{\gamma}}_\beta(\hat{\boldsymbol{\beta}}) \quad \hat{\boldsymbol{\gamma}})^{-1}$$

almost surely has the same limit as

$$\frac{2}{n}\dot{\boldsymbol{\mu}}_\gamma^T P_X\dot{\boldsymbol{\mu}}(\boldsymbol{\gamma}_0)=\frac{1}{n}(\dot{\boldsymbol{\gamma}}_\beta(\boldsymbol{\beta}_0) \quad \boldsymbol{\gamma}_0)^{-T}\begin{pmatrix}\dot{\boldsymbol{\mu}}_\beta^T P_X\dot{\boldsymbol{\mu}}_\beta(\boldsymbol{\beta}_0) & 0\\ 0 & 0\end{pmatrix}(\dot{\boldsymbol{\gamma}}_\beta(\boldsymbol{\beta}_0) \quad \boldsymbol{\gamma}_0)^{-1}$$

which is positive semidefinite, with null space spanned by $\boldsymbol{\gamma}_0$. This limit may be recognized to be $\mathscr{J}_0$. Therefore, the fact that

$$\frac{1}{n}\ddot{\psi}(\hat{\boldsymbol{\gamma}})=\frac{1}{n}(B(\hat{\boldsymbol{\gamma}})+\dot{B}(\hat{\boldsymbol{\gamma}})\hat{\boldsymbol{\gamma}})\xrightarrow{\text{a.s.}}\mathscr{J}_0$$

can be seen to follow from the usual convergence results. (This approach was taken by Osborne and Smyth [16].)

11. A numerical experiment. The purpose of this section is to compare the modified Prony algorithm with a good general purpose nonlinear squares procedure, namely, the Levenberg modification of the Gauss–Newton algorithm (Fletcher [5]). Osborne [13] reported some results for exponential fitting (very favorable for the Prony algorithm), so the results here are for rational fitting. The modified Prony algorithm was implemented as described in § 5. The symmetric linear system appearing in the inverse iteration sequence was solved by diagonal pivoting, as implemented by Bunch in LINPACK (Dongarra et al. [4]). The Levenberg algorithm was implemented essentially as described by Osborne [14], the Levenberg parameter having expansion factor 2, contraction factor 10 and initial value 1.

The convergence criterion used by the Levenberg algorithm was

$$(ssf^{1/2}-ssr^{1/2})/(1+ssf^{1/2})<\tau, \tag{21}$$

where *ssf* is the actual sum of squares and *ssr* is the sum of squares based on a linearization of the problem, and the tolerance τ was set to 10^{-7}. Although the Prony and Levenberg convergence criteria are not strictly comparable, the Prony tolerance parameter was adjusted to 10^{-10} so that the two algorithms returned estimates that were on average of the same precision.

All calculations were performed in double precision in Fortran 77 on a Sperry Univac 1100/82 computer.

11.1. Test problem. Data was simulated using the mean function

$$\mu(t)=(\alpha_1+\alpha_2 t)/(1+\beta_1 t+\beta_2 t^2)=(.5+.5t)/(1-.5t+.1t^2).$$

Data sets were constructed to have standard deviations $\sigma=.03$, .01, .003, .001 and sample sizes $n=32, 64, 128, 256, 512$. Random deviates were associated with the means in such a way that each data set comprised every second point of the next larger data set. This was done so that sample sizes could be compared along a common sequence of random deviates.

Ten replicates were generated for each of the four distributions: the normal, student's t on 3 distribution functions (infinite third moments), lognormal (skew) and Pareto's distribution with $k = 1$ and $\alpha = 3$ (skew and infinite third moments). Random deviates were generated by inverting the distribution functions, i.e.,

$$\varepsilon = \sigma F^{-1}(U)$$

with U a pseudorandom number generated by the NAG subroutine G05CAF (Numerical Algorithms Group [11]) with seed equal to 1984. The first 10,000 values generated by G05CAF were ignored (most were used for exponential fitting simulations), while the next 5120 were used for the simulations reported here.

In order to make an objective choice, the true parameter values themselves were used as starting values. These were quite far from the least squares estimates for small n and large σ, less so for large n and small σ, as can be seen from Table 3.

11.2. Results. As it turned out, the convergence results were similar for all four distributions; for example, Table 1 gives the iteration counts for the modified Prony algorithm for the first replicate. It appears that the convergence rates achieved by the algorithms are little affected by skewness or by the third and higher moments of the error distribution (although the actual least squares estimates returned will be affected). So only the results for the normal distribution are reported below.

Iteration counts for the normal simulations are given in Table 2. As expected from the form of the convergence matrices, and also from the proximity of the starting values to the final estimates, both algorithms converged more rapidly for large n and small σ. (See Smyth [20] and Kass and Smyth [8] for a discussion of the convergence matrix for the Gauss–Newton algorithm.) The pattern for the Levenberg algorithm

TABLE 1

Iteration counts for the modified Prony algorithm for one replicate of four distributions.

σ	n	Normal	Student	Log-normal	Pareto
.030	32	5	5	5	4
	64	4	5	4	4
	128	4	4	4	4
	256	5	4	4	4
	512	4	4	4	4
.10	32	4	4	3	3
	64	4	4	3	3
	128	3	3	3	3
	256	3	3	3	3
	512	3	3	3	3
.003	32	3	3	3	3
	64	3	3	3	3
	128	2	2	2	2
	256	2	2	2	2
	512	3	3	2	2
.001	32	2	2	2	2
	64	2	2	2	2
	128	2	2	2	2
	256	2	2	2	2
	512	2	2	2	2

TABLE 2

Median and maximum iteration counts for normal simulations. Results for the Prony algorithm are given above those for the Levenberg algorithm.

$n\backslash\sigma$	.030		.010		.003		.001	
32	5	17	4	6	3	3	2	3
	6	10	5	7	5	6	5	5
64	5	12	4	5	3	3	2	2
	6	8	5.5	6	5	5	5	5
128	6	7	4	4	2.5	3	2	3
	6	7	5	6	5	5	5	5
256	4.5	6	3	4	2	3	2	2
	6	6	5	6	5	5	5	5
512	4	5	3	3	3	3	2	2
	6	6	5	6	5	5	5	5

was obscured somewhat by its careful convergence criterion (21), which caused it to produce relatively more precise estimates when the sum of squares was small. This is discussed further below. In most cases the Prony algorithm required slightly fewer iterations than the Levenberg, although both algorithms were finished within six iterations most of the time even for $n = 32$ and $\sigma = .03$. Computer time was not strictly recorded, but the two algorithms took similar amounts of elapsed time. Table 3 gives the means and standard deviations of $\hat{\beta}_1$ and $\hat{\beta}_2$. For two samples with $n = 32$ and $\sigma = .03$, Prony converged to a stationary value which was not the least squares estimate; except for these, the programs returned effectively identical estimates. The two samples for which Prony failed gave rise to final Prony matrices $B(\hat{\gamma})$ which were indefinite, while all other cases gave positive-semidefinite matrices.

Two changes to the Levenberg algorithm, specifically tuning it to this problem, allowed it to converge rather more rapidly, as shown in Table 4. The first was to set

TABLE 3

Means and standard deviations, over 10 replicates, of the least squares estimates of β_1 and β_2. True values are $\beta_1 = -.5$ and $\beta_2 = .1$.

$n\backslash\sigma$	.03	.01	.003	.001
32	−.2910 (.24)	−.4474 (.055)	−.4853 (.015)	−.4952 (.005)
	−0.238 (.14)	.0692 (.034)	.0914 (.095)	.0972 (.003)
64	−.5042 (.21)	−.5025 (.066)	−.5099 (.020)	−.5003 (.007)
	.1033 (.12)	.1017 (.038)	.1006 (.011)	.1002 (.004)
128	−.4694 (.14)	−.4918 (.046)	−.4977 (.013)	−.4993 (.005)
	.0816 (.08)	.0950 (.027)	.0986 (.008)	.9954 (.003)
256	−.4979 (.13)	−.5002 (.041)	−.5002 (.012)	−.5001 (.004)
	.0966 (.08)	.0994 (.025)	.0999 (.007)	.1000 (.002)
512	−.5180 (.16)	−.5080 (.051)	−.5026 (0.15)	−.5009 (.005)
	.1097 (.09)	.1044 (.030)	.1014 (.009)	.1005 (.003)

TABLE 4
Median and maximum iteration counts for unmodified Gauss-Newton with a simplified convergence criterion.

$n\backslash\sigma$	.030		.010		.003		.001	
32	4	5	3	3	3	3	2	3
64	3	5	3	4	3	3	2.5	3
128	3	5	3	3	2.5	3	2	3
256	3	3	2	3	2	3	2	2
512	3	3	2	3	2	3	2	2

the Levenberg parameter to zero, so that the algorithm reduced to the unmodified Gauss-Newton algorithm. The second was to remove the constant from the denominator of the convergence criterion (21). This constant is a standard safety device in general purpose programs, but when *ssf* is small it causes the program to seek a small absolute rather than relative different between $ssf^{1/2}$ and $ssr^{1/2}$. Without the constant, the number of iterations required decreases consistently with n, as expected. Although the precision of estimation accepted by the new criterion is no longer equivalent to that of Prony, the differences between the parameter estimates returned are very small.

Experimentation showed that both Prony and Gauss-Newton algorithms were sensitive to the starting values for small n and large σ. This is in contrast with results for exponential fitting, for which the modified Prony algorithm is remarkably robust with respect to the starting value.

We conclude that, for this rational fitting problem, the modified Prony algorithm is intermediate in performance between a general purpose Levenberg algorithm and a Gauss-Newton type algorithm specially tuned to the problem.

Simulations for exponential fitting will be given in a separate paper, which will focus on the special problems of exponential fitting. Curiously, the modified Prony algorithm actually appears to perform better for exponential fitting than it does for rational fitting, despite the well-known fact that other algorithms find exponential fitting very much more difficult (as did the Levenberg algorithm in our simulations).

Acknowledgment. The authors are grateful to Professor E. J. Hannan for valuable discussions.

REFERENCES

[1] D. M. BATES AND D. G. WATTS, *Nonlinear Regression Analysis and Its Applications*, John Wiley, New York, 1988.

[2] J. R. BUNCH AND L. KAUFMANN, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comp., 31 (1977), pp. 162-179.

[3] P. J. DAVIS, *Circulant Matrices*, John Wiley, New York, 1979.

[4] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER, AND G. W. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[5] R. FLETCHER, *Practical Methods of Optimization Vol.* 1: *Unconstrained Optimization*, Wiley, Chichester, U.K., 1980.

[6] A. S. HOUSEHOLDER, *On Prony's method of fitting exponential decay curves and multiple-hit survival curves*, Report ORNL-455, Oak Ridge National Laboratory, Oak Ridge, TN, 1950.

[7] R. I. JENNRICH, *Asymptotic properties of non-linear least squares estimators*, Ann. Math. Statist., 40 (1969), pp. 633-643.

[8] R. KASS AND G. K. SMYTH, *Convergence of the scoring algorithm: a geometric interpretation*, Tech. Report, Department of Mathematics, University of Queensland, St. Lucia, Australia, 1989.

[9] S. L. MARPLE, *Digital Spectral Analysis with Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[10] R. J. MULHOLLAND, J. R. CRUZ, AND J. HILL, *State-variable canonical forms for Prony's method*, Internat. J. Systems Sci., 17 (1986), pp. 55–64.

[11] Numerical Algorithms Group, NAG FORTRAN *Library Manual Mark* 10, Numerical Algorithms Group, Oxford, U.K., 1983.

[12] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[13] M. R. OSBORNE, *Some special nonlinear least squares problems*, SIAM J. Numer. Anal., 12 (1975), pp. 571–592.

[14] ———, *Nonlinear least squares—the Levenberg algorithm revisited*, J. Austral. Math. Soc. Ser. B, 19 (1976), pp. 343–357.

[15] ———, *Inverse iteration, Newton's method, and nonlinear eigenvalue problems*, in Proc. Conference in Honour of J. H. Wilkinson, Institute of Mathematics and Its Applications, Royal Society, London. Pub., 19 (1978), pp. 21–53.

[16] M. R. OSBORNE AND G. K. SMYTH, *An algorithm for exponential fitting revisited*, Contribution to Essays in Time Series and Allied Processes: Papers in Honour of E. J. Hannan, J. Gani and M. B. Priestley, eds., Applied Probability Trust, Sheffield, 1986, pp. 419–430.

[17] R. Prony, *Essai expérimental et analytique: sur les lois de la dilatabilité de fluides élastiques et sur celles de la force expansive de la vapeur de l'eau et de la vapeur de l'alcool, à différentes températures*, J. de L'École Polytechnique, 1 (1795), pp. 24–76.

[18] D. A. RATKOWSKY, *Nonlinear Regression Modelling: A Unified Practical Approach*, Marcel Dekker, New York, 1983.

[19] G. K. SMYTH, *Coupled and separable iterations in nonlinear estimation*, Ph.D. thesis, Australian National University, Canberra, Australia, 1985.

[20] ———, *Curvature and convergence*, in Proc. Statistical Computing Section, American Statistical Society, Alexandria, VA, 1987, pp. 278–283.

[21] W. F. STOUT, *Some results on the complete and almost sure convergences of linear combinations of independent random variables and martingale differences*, Ann. Math. Statist., 5 (1968), pp. 1549–1562.

[22] ———, *Almost Sure Convergence*, Academic Press, New York, 1974.

[23] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 383-398, March 1991
© 1991 Society for Industrial and Applied Mathematics
009

MINIMIZING GCV/GML SCORES WITH MULTIPLE SMOOTHING PARAMETERS VIA THE NEWTON METHOD*

CHONG GU† AND GRACE WAHBA‡

Abstract. The (modified) Newton method is adapted to optimize generalized cross validation (GCV) and generalized maximum likelihood (GML) scores with multiple smoothing parameters. The main concerns in solving the optimization problem are the speed and the reliability of the algorithm, as well as the invariance of the algorithm under transformations under which the problem itself is invariant. The proposed algorithm is believed to be highly efficient for the problem, though it is still rather expensive for large data sets, since its operational counts are $(2/3)kn^3+O(n^2)$, with k the number of smoothing parameters and n the number of observations. Sensible procedures for computing good starting values are also proposed, which should help in keeping the execution load to the minimum possible. The algorithm is implemented in Rkpack [RKPACK *and its applications: Fitting smoothing spline models*, Tech. Report 857, Department of Statistics, University of Wisconsin, Madison, WI, 1989] and illustrated by examples of fitting additive and interaction spline models. It is noted that the algorithm can also be applied to the maximum likelihood (ML) and the restricted maximum likelihood (REML) estimation of the variance component models.

Key words. additive/interaction spline models, gradient, Hessian, invariance, Newton method, smoothing parameters, starting values

AMS(MOS) subject classifications. 65D07, 65D10, 65D15, 65K10, 65V05

1. Introduction.

Suppose we observe

$$y_j = L_j f + \varepsilon_j \qquad j = 1, \cdots, n,$$

where the L_j's are bounded linear functionals in a Hilbert space $\mathcal{H}$, and the ε_j's are independently and identically distributed Gaussian noise with possibly unknown variance σ^2. Such setups encompass a broad range of smoothing and indirect sensing problems. The solution f_λ to the variational problem

$$\min \frac{1}{n}\sum_{j=1}^{n}(y_j - L_j f)^2 + \lambda \|P_1 f\|^2 \tag{1}$$

is called a spline in a general sense (see [14]), where P_1 is a projection operator to a subspace $\mathcal{H}_1$ with codimension M, $\|\cdot\|$ is the norm in $\mathcal{H}$, and λ is the so-called smoothing parameter. The parameter λ controls the trade-off between the residual sum of squares $\sum_{j=1}^{n}(y_j - L_j f_\lambda)^2$ and the penalty $\|P_1 f_\lambda\|^2$. f_λ is called a regularized estimate of f in the literature of ill-posed problems, and λ the regularization parameter. See, e.g., [17] and references cited therein.

Let ξ_j denote the representer of L_j. It is derived by Kimeldorf and Wahba [14] that $f_\lambda = \sum_{j=1}^{n} c_j(P_1\xi_j) + \sum_{\nu=1}^{M} d_\nu \phi_\nu$, where $\{\phi_\nu\}_{\nu=1}^{M}$ span $\mathcal{H}_0$, the null space of P_1. $\mathbf{c}=(c_1, \cdots, c_n)$; $\mathbf{d}=(d_1, \cdots, d_M)$ are solutions to the minimization problem

$$\min \frac{1}{n}\|\mathbf{y} - S\mathbf{d} - \tilde{Q}\mathbf{c}\|^2 + \lambda \mathbf{c}^T \tilde{Q}\mathbf{c}, \tag{2}$$

* Received by the editors December 28, 1988; accepted for publication (in revised form) February 7, 1990. This research was supported in part by Air Force Office of Scientific Research grant AFOSR-87-0171 and by National Aeronautics and Space Administration contract NAG5-316.

† Department of Statistics, Purdue University, West Lafayette, Indiana 47907. The work of this author was also supported by the Natural Sciences and Engineering Research Council of Canada while he was at the University of British Columbia.

‡ Department of Statistics, University of Wisconsin-Madison, 1210 West Dayton St., Madison, Wisconsin 53706.

where $\tilde{Q} = (\langle P_1 \xi_{j_1}, P_1 \xi_{j_2} \rangle)$ and $S = (L_j \phi_\nu)$, with $\langle \cdot, \cdot \rangle$ indicating the inner product in $\mathcal{H}$. It can be shown (see [21]) that the solution to the linear system

$$\begin{aligned} (\tilde{Q} + n\lambda I)\mathbf{c} + S\mathbf{d} &= \mathbf{y}, \\ S^T\mathbf{c} &= 0 \end{aligned} \tag{3}$$

is a minimizer of (2), and when $\tilde{Q}$ is of full rank, it is the unique minimizer.

Consider an orthogonal decomposition of $\mathcal{H}$ into more than two components, $\mathcal{H} = \bigoplus_{i=0}^{k} \mathcal{H}_i$. A direct generalization of (1) is

$$\min \frac{1}{n} \sum_{j=1}^{n} (y_j - L_j f)^2 + \sum_{i=1}^{k} \lambda_i \| P_i f \|^2, \tag{4}$$

where the λ_i's are a set of smoothing parameters and P_i is the orthogonal projection operator onto $\mathcal{H}_i$. Writing $\lambda_i = \lambda / \theta_i$, we can rewrite (4) as

$$\min \frac{1}{n} \sum_{j=1}^{n} (y_j - L_j f)^2 + \lambda \| P_* f \|_{\boldsymbol{\theta}}^2, \tag{5}$$

where $P_* = \sum_{i=1}^{k} P_i$ is the projection operator onto $\mathcal{H}_* = \bigoplus_{i=1}^{k} \mathcal{H}_i$, and $\| f \|_{\boldsymbol{\theta}}^2 = \| P_0 f \|^2 + \sum_{i=1}^{k} \theta_i^{-1} \| P_i f \|^2$ is a modified norm indexed by $\boldsymbol{\theta}$, where $\| \cdot \|$ is the original norm. It can be shown that the representer of L_j under the norm $\| \cdot \|_{\boldsymbol{\theta}}$ is $\xi_j^{\boldsymbol{\theta}} = G(P_0 \xi_j) + \sum_{i=1}^{k} \theta_i (P_i \xi_j)$, where ξ_j is its representer under the norm $\| \cdot \|$. Denoting $\langle \cdot, \cdot \rangle$, $\langle \cdot, \cdot \rangle_{\boldsymbol{\theta}}$ as the inner products corresponding to the norms $\| \cdot \|$, $\| \cdot \|_{\boldsymbol{\theta}}$, respectively, we have $\tilde{Q}_*^{\boldsymbol{\theta}} = (\langle P_* \xi_{j_1}^{\boldsymbol{\theta}}, P_* \xi_{j_2}^{\boldsymbol{\theta}} \rangle_{\boldsymbol{\theta}}) = \sum_{i=1}^{k} \theta_i \tilde{Q}_i$, where $\tilde{Q}_i = (\langle P_i \xi_{j_1}, P_i \xi_{j_2} \rangle)$. Thus the solution to (4) can be written as $f_{\boldsymbol{\lambda}} = \sum_{j=1}^{n} c_j \xi_j^{\boldsymbol{\theta}} + \sum_{\nu=1}^{M} d_\nu \phi_\nu$, with $\mathbf{c}$, $\mathbf{d}$ determined by

$$\min \frac{1}{n} \| \mathbf{y} - S\mathbf{d} - \tilde{Q}_*^{\boldsymbol{\theta}} \mathbf{c} \|^2 + \lambda \mathbf{c}^T \tilde{Q}_*^{\boldsymbol{\theta}} \mathbf{c}. \tag{6}$$

Choosing appropriate smoothing parameters $\boldsymbol{\lambda}$ is crucial for effectively recovering truth functions from the data by fitting spline models. Two of the commonly recognized data-driven methods for choosing smoothing parameters are the generalized cross-validation (GCV) and the generalized maximum likelihood (GML) methods. Writing

$$\tilde{Q}_*^{\boldsymbol{\theta}} \mathbf{c} + S\mathbf{d} = (L_1 f_{\boldsymbol{\lambda}}, \cdots, L_n f_{\boldsymbol{\lambda}})^T = A(\boldsymbol{\lambda})\mathbf{y},$$

the GCV method seeks $\boldsymbol{\lambda}$ to minimize

$$V(\lambda) = \frac{(1/n) \| (I - A(\boldsymbol{\lambda}))\mathbf{y} \|^2}{[(1/n) \operatorname{tr} (I - A(\boldsymbol{\lambda}))]^2}.$$

$A(\boldsymbol{\lambda})$ is the so-called influence matrix. Letting

$$S = FR = (F_1, F_2) \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

be the QR-decomposition of S, it can be shown (see [21]) that $I - A(\boldsymbol{\lambda}) = n\lambda F_2 (F_2^T \tilde{Q}_*^{\boldsymbol{\theta}} F_2 + n\lambda I)^{-1} F_2^T$, hence

$$V(\boldsymbol{\lambda}) = V(\lambda, \boldsymbol{\theta}) = n \frac{\mathbf{z}^T (Q_*^{\boldsymbol{\theta}} + n\lambda I)^{-2} \mathbf{z}}{[\operatorname{tr} (Q_*^{\boldsymbol{\theta}} + n\lambda I)^{-1}]^2}, \tag{7}$$

where $\mathbf{z} = F_2^T \mathbf{y}$ and $Q_*^{\boldsymbol{\theta}} = F_2^T \tilde{Q}_*^{\boldsymbol{\theta}} F_2 = \sum_{i=1}^{k} \theta_i (F_2^T \tilde{Q}_i F_2) = \sum_{i=1}^{k} \theta_i Q_i$, where $Q_i = F_2^T \tilde{Q}_i F_2$. The GCV method is proposed by Craven and Wahba [6] and is shown to be asymptotically optimum for minimizing predictive mean square error ([6] and [15]). See also [24], where it is shown that this method is also good for minimizing solution mean square error in a variety of circumstances.

Based on the Bayesian interpretation of the smoothing spline models [20], Wahba [22] also derives the GML method which seeks the minimizer of

$$M(\boldsymbol{\lambda}) = \frac{\mathbf{y}^T(I - A(\boldsymbol{\lambda}))\mathbf{y}/n}{[\det{}^+(I - A(\boldsymbol{\lambda}))]^{1/n_1}},$$

where $\det^+(I-A)$ is the product of the $n_1 = n - M$ nonzero eigenvalues of $I - A$. In parallel to (7), we have

$$M(\boldsymbol{\lambda}) = M(\lambda, \boldsymbol{\theta}) = \frac{\mathbf{z}^T(Q_*^{\boldsymbol{\theta}} + n\lambda I)^{-1}\mathbf{z}/n}{[\det(Q_*^{\boldsymbol{\theta}} + n\lambda I)^{-1}]^{1/n_1}}. \tag{8}$$

A theoretical comparison of the GCV and GML methods can be found in [22]. The purpose of this article is to present a numerical algorithm for the computation of selecting the smoothing parameters according to the GCV or GML criterion. Our main concern is to deal with problems with $k > 1$. The method is developed on the basis of the $k = 1$ algorithm of Gu et al. [11].

2. Preliminaries. To minimize the functions $V(\lambda, \boldsymbol{\theta})$ or $M(\lambda, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ and $n\lambda$, we wish to iterate on the following cycle:

1) For fixed $\boldsymbol{\theta}$, minimize $V(\lambda|\boldsymbol{\theta})$ or $M(\lambda|\boldsymbol{\theta})$ with respect to $n\lambda$.

2) Update $\boldsymbol{\theta}$ using information from the current estimates.

Step 1) above can be implemented through the single smoothing parameter GCV/GML algorithm based on the Householder tridiagonalization, as proposed by Gu et al. [11]. To carry out step 2), we will evaluate the gradient and the Hessian of $V(\boldsymbol{\theta}|\lambda)$ or $M(\boldsymbol{\theta}|\lambda)$ with respect to $\boldsymbol{\eta} = \log(\boldsymbol{\theta})$, then apply the modified Newton method (see [8]) to update the $\boldsymbol{\eta}$. In this section, we first discuss the choices for the scaling of the variables ($\boldsymbol{\eta}$) and the scaling of the objective functions ($V(\cdot)$ of $M(\cdot)$). Then we present the expressions of the gradient and the Hessian for later use.

We choose the variables $\boldsymbol{\eta}$ instead of $\boldsymbol{\theta}$ mainly for their *invariance*. Looking at the formulas of $V(\cdot)$ and $M(\cdot)$ ((7) and (8)), we can see that what really matter are the matrices $\theta_i Q_i$'s, while the "face values" of the θ_i's are subject to rescaling when the matrices Q_i's are multiplied by some positive constants. Notice that the problem itself is not changed by such transformations. Invariance under this kind of transformation is hence essential to any sensible algorithm. It is easy to see that the derivatives of $V(\cdot)$ and $M(\cdot)$ with respect to $\boldsymbol{\eta}$ are invariant, and hence the methods based on these derivatives are invariant. Standard calculations show that the derivatives of $V(\cdot)$ and $M(\cdot)$ with respect to $\boldsymbol{\theta}$ are in general *not* invariant in this sense, which disqualifies them for serving as the basic variables. Another immediate numerical merit from adopting $\boldsymbol{\eta}$ instead of $\boldsymbol{\theta}$ is that we change a constrained optimization problem ($\boldsymbol{\theta} \geqq 0$) to an unconstrained one, which allows much simpler treatment.

The objective functions can always be rescaled by monotone transformations without changing the optima of the problems. In our problem, it is observed that $\log V(\cdot)$ and $\log M(\cdot)$ have simpler derivative formulas. However, this transformation is not adopted for the following reason. To *minimize* the objective function efficiently, we want the objective function to be as *convex* as possible, since most optimization methods are modeled after convex functions. When the convexity is violated, we have to modify the methods and to suffer lower efficiency, and the methods may even fail to converge. Since $\log(\cdot)$ is concave, we can expect $\log f(\cdot)$ to be "less convex" than $f(\cdot)$ for general $f(\cdot) > 0$. Actually, it can be shown that a positive-definite Hessian of $\log f(\cdot)$ implies a positive-definite Hessian of $f(\cdot)$, but the reverse is not true.

Having chosen the objective functions and the basic variables, we now collect the formulas of the gradient and the Hessian. Define $D=\sum_{i=1}^{k}(e^{\eta_i}Q_i)+n\lambda I=Q_*^{\theta}+n\lambda I$. We write $V(\boldsymbol{\eta}|\lambda)=n\mathbf{z}^TD^{-2}\mathbf{z}/(\operatorname{tr}D^{-1})^2$ and $M(\boldsymbol{\eta}|\lambda)=\mathbf{z}^TD^{-1}\mathbf{z}/n(\det D^{-1})^{1/n_1}$. We have the following lemmas.

LEMMA 2.1.

$$\frac{\partial V}{\partial\eta_i}=n\left(\frac{\dot r_i}{t^2}-2\frac{r\dot t_i}{t^3}\right), \tag{9}$$

$$\frac{\partial^2 V}{\partial\eta_i\,\partial\eta_j}=n\left(\frac{\ddot r_{ij}}{t^2}-2\frac{\dot r_i\dot t_j}{t^3}-2\frac{\dot t_i\dot r_j}{t^3}-2\frac{r\ddot t_{ij}}{t^3}+6\frac{r\dot t_i\dot t_j}{t^4}\right), \tag{10}$$

where

$$\dot r_i=\frac{\partial r}{\partial\eta_i}=\frac{\partial}{\partial\eta_i}(\mathbf{z}^TD^{-2}\mathbf{z})=-2\mathbf{z}^TD^{-2}(e^{\eta_i}Q_i)D^{-1}\mathbf{z}, \tag{11}$$

$$\dot t_i=\frac{\partial t}{\partial\eta_i}=\frac{\partial}{\partial\eta_i}(\operatorname{tr}D^{-1})=-\operatorname{tr}(D^{-1}(e^{\eta_i}Q_i)D^{-1}), \tag{12}$$

$$\begin{aligned}\ddot r_{ij}=2[&\mathbf{z}^TD^{-2}(e^{\eta_i}Q_i)D^{-1}(e^{\eta_j}Q_j)D^{-1}\mathbf{z}+\mathbf{z}^TD^{-1}(e^{\eta_i}Q_i)D^{-2}(e^{\eta_j}Q_j)D^{-1}\mathbf{z}\\&+\mathbf{z}^TD^{-1}(e^{\eta_i}Q_i)D^{-1}(e^{\eta_j}Q_j)D^{-2}\mathbf{z}]+\delta_{|i-j|}\dot r_i,\end{aligned} \tag{13}$$

$$\ddot t_{ij}=2\operatorname{tr}(D^{-1}(e^{\eta_i}Q_i)D^{-1}(e^{\eta_j}Q_j)D^{-1})+\delta_{|i-j|}\dot t_i, \tag{14}$$

where δ_i is Kronecker's delta.

LEMMA 2.2.

$$\frac{\partial M}{\partial\eta_i}=\frac{1}{n}\left(\frac{\dot r_i}{t^{1/n}}-\frac{1}{n}\frac{r\dot t_i}{t^{1+1/n}}\right), \tag{15}$$

$$\frac{\partial^2 M}{\partial\eta_i\,\partial\eta_j}=\frac{1}{n}\left(\frac{\ddot r_{ij}}{t^{1/n_1}}-\frac{1}{n_1}\frac{\dot r_i\dot t_j}{t^{1+1/n_1}}-\frac{1}{n_1}\frac{\dot t_i\dot r_j}{t^{1+1/n_1}}-\frac{1}{n_1}\frac{r\ddot t_{ij}}{t^{1+1/n_1}}+\frac{1}{n_1}\frac{n_1+1}{n_1}\frac{r\dot t_i\dot t_j}{t^{2+1/n_1}}\right), \tag{16}$$

where

$$\dot r_i=\frac{\partial r}{\partial\eta_i}=\frac{\partial}{\partial\eta_i}(\mathbf{z}^TD^{-1}\mathbf{z})=-\mathbf{z}^TD^{-1}(e^{\eta_i}Q_i)D^{-1}\mathbf{z}, \tag{17}$$

$$\dot t_i=\frac{\partial t}{\partial\eta_i}=\frac{\partial}{\partial\eta_i}(\det D^{-1})=-\det(D^{-1})\operatorname{tr}(e^{\eta_i}Q_iD^{-1}), \tag{18}$$

$$\ddot r_{ij}=2\mathbf{z}^TD^{-1}(e^{\eta_i}Q_i)D^{-1}(e^{\eta_j}Q_j)D^{-1}\mathbf{z}+\delta_{|i-j|}\dot r_i, \tag{19}$$

$$\begin{aligned}\ddot t_{ij}=&\det(D^{-1})\operatorname{tr}((e^{\eta_i}Q_i)D^{-1}(e^{\eta_j}Q_j)D^{-1})\\&+\det(D^{-1})\operatorname{tr}(e^{\eta_i}Q_iD^{-1})\operatorname{tr}(e^{\eta_j}Q_jD^{-1})+\delta_{|i-j|}\dot t_i,\end{aligned} \tag{20}$$

where δ_i is Kronecker's delta.

The proofs to the above two lemmas are straightforward and tedious. We omit them here.

3. Algorithms. In this section, we will specify the main algorithm and discuss its various aspects. More discussions will be collected in § 6 as remarks.

ALGORITHM 3.1. Assuming inputs of the null-space matrix S, the response vector $\mathbf{y}$, the matrices $\tilde{Q}_i$, $i=1,\cdots,k$, and the starting values $\boldsymbol{\eta}_0$, we propose performing the following:

1. Initialization:
   (a) Compute the QR-decomposition of $S=FR=(F_1, F_2)\binom{R_1}{0}$.
   (b) Compute $\mathbf{z}=F_2^T\mathbf{y}$ and $Q_i=F_2^T\tilde{Q}_iF_2$.
   (c) Set $\Delta\boldsymbol{\eta}=0$, $\boldsymbol{\eta}_-=\boldsymbol{\eta}_0$, $V_-=\infty$ (or $M_-=\infty$).
2. Iteration:
   (a) For the current trial value $\boldsymbol{\eta}=\boldsymbol{\eta}_-+\Delta\boldsymbol{\eta}$, collect $\tilde{D}=Q_*^{\boldsymbol{\theta}}=\sum_{i=1}^k\theta_iQ_i$.
   (b) Compute $\tilde{D}=U\tilde{T}U^T$, where U is orthogonal and $\tilde{T}$ is tridiagonal. Compute $\mathbf{x}=U^T\mathbf{z}$.
   (c) Minimize

$$V(\lambda|\boldsymbol{\eta})=\frac{n\mathbf{x}^T(\tilde{T}+n\lambda I)^{-2}\mathbf{x}}{[\operatorname{tr}(\tilde{T}+n\lambda I)^{-1}]^2} \tag{21}$$

   or

$$M(\lambda|\boldsymbol{\eta})=\frac{\mathbf{x}^T(\tilde{T}+n\lambda I)^{-1}\mathbf{x}/n}{\det(\tilde{T}+n\lambda I)^{-1/n1}}. \tag{22}$$

   If $V>V_-$ (or $M>M_-$), set $\Delta\boldsymbol{\eta}=\Delta\boldsymbol{\eta}/2$, goto (a); else proceed.
   (d) Evaluate the gradient $\mathbf{g}=(\partial/\partial\boldsymbol{\eta})V(\boldsymbol{\eta}|\lambda)$ (or $(\partial/\partial\boldsymbol{\eta})M(\boldsymbol{\eta}|\lambda)$) and the Hessian $H=(\partial^2/\partial\boldsymbol{\eta}\,\partial\boldsymbol{\eta}^T)V(\boldsymbol{\eta}|\lambda)$ (or $(\partial^2/\partial\boldsymbol{\eta}\,\partial\boldsymbol{\eta}^T)M(\boldsymbol{\eta}|\lambda)$). Calculate the increment $\Delta\boldsymbol{\eta}=-\tilde{H}^{-1}\mathbf{g}$, where $\tilde{H}=H+\operatorname{diag}(\mathbf{e})$ is positive definite. If H itself is positive definite "enough," $\mathbf{e}$ is simply 0.
   (e) Check convergence conditions. If the conditions fail, set $\boldsymbol{\eta}_-=\boldsymbol{\eta}$, $V_-=V$ (or $M_-=M$), goto (a); else proceed.
3. Calculate the optimal model:
   (a) If $\Delta\eta_i<-\gamma$, set $\eta_i=-\infty$, where γ is a "large" number, say, $\gamma\in(.5, .9)$.
   (b) Collect $\tilde{D}=\sum_{i=1}^k\theta_iQ_i$. Calculate the model minimizing $V(\lambda|\boldsymbol{\eta})$ (or $M(\lambda|\boldsymbol{\eta})$).

In the above specification, most items clearly explain themselves, except steps 2(d), 2(e) and 3(a), which we will explain in turn. Step 2(d) is the major part of this algorithm. Define $D=\tilde{D}+n\lambda I$ (this definition is consistent with the one in § 2), $T=\tilde{T}+n\lambda I$, and $K_i=e^{\eta_i}U^TQ_iU$. We can write (11)-(14) as

$$\dot{r}_i=-2\mathbf{x}^TT^{-2}K_iT^{-1}\mathbf{x}=-2(T^{-1}\mathbf{x})^T(T^{-1}K_i)(T^{-1}\mathbf{x}), \tag{23}$$

$$\dot{t}_i=-\operatorname{tr}(T^{-2}K_i), \tag{24}$$

$$\begin{aligned}\ddot{r}_{ij}=2[&(T^{-1}\mathbf{x})^T(T^{-1}K_i)(T^{-1}K_j)(T^{-1}\mathbf{x})+(T^{-1}\mathbf{x})^T(T^{-1}K_i)^T(T^{-1}K_j)(T^{-1}\mathbf{x})\\&+(T^{-1}\mathbf{x})^T(T^{-1}K_i)^T(T^{-1}K_j)^T(T^{-1}\mathbf{x})]+\delta_{|i-j|}\dot{r}_i,\end{aligned} \tag{25}$$

$$\ddot{t}_{ij}=2\operatorname{tr}((T^{-2}K_i)(T^{-1}K_j))+\delta_{|i-j|}\dot{t}_i. \tag{26}$$

Similarly, (17)-(20) become

$$\dot{r}_i=-\mathbf{x}^T(T^{-1}K_i)(T^{-1}\mathbf{x}), \tag{27}$$

$$\dot{t}_i=-\det(T^{-1})\operatorname{tr}(T^{-1}K_i), \tag{28}$$

$$\ddot{r}_{ij}=2\mathbf{x}^T(T^{-1}K_i)(T^{-1}K_j)(T^{-1}\mathbf{x})+\delta_{|i-j|}\dot{r}_i, \tag{29}$$

$$\ddot{t}_{ij} = \det(T^{-1})\,\mathrm{tr}((T^{-1}K_i)(T^{-1}K_j)) + \det(T^{-1})\,\mathrm{tr}(T^{-1}K_i)\,\mathrm{tr}(T^{-1}K_j) + \delta_{|i-j|}\dot{t}_i. \tag{30}$$

The gradients and the Hessians presented in Lemmas 2.1 and 2.2 will be calculated via (23)-(26) and (27)-(30). The modified Newton method based on the modified Cholesky decomposition, as described in [8], is adopted to calculate the update direction $\Delta\boldsymbol{\eta} = -\tilde{H}^{-1}\mathbf{g}$. Since the score evaluation is expensive, we choose not to perform a step length search, but simply to pick unity as the default step length. We execute an Armijo-type procedure of halving the step to insure an improved score in each iteration, where improvement is guaranteed by the fact that $-\tilde{H}^{-1}\mathbf{g}$ is a descent direction.

Given a user-supplied precision requirement ε_A, the algorithm is thought to have converged if at least one of the following criteria is satisfied:

1) (a) $V_- - V < \varepsilon_A(1+V)$, and (b) $\|\mathbf{g}\|_\infty < \sqrt{\varepsilon_A}\,(1+V)$.

2) $\|\mathbf{g}\|_\infty < \varepsilon_A$.

The criteria are modifications of the suggestions by Gill et al. [8, pp. 306-307]. Our rule 1(a) is identical to their U1, while our rule 1(b) is expected to do what their U2-U3 do. In our problem some optimal η_i could be at $-\infty$. In such a situation, the algorithm will keep wanting to move towards the optimum with big steps even when the score values are well within the required precision of the optimal value. This is a typical ill-conditioned situation where their U2 will never be satisfied. By adopting a more stringent version of their U3 (our rule 1(b)) instead of their U2-U3 combination, we can avoid such endless iteration and yet deliver qualified termination, since our rule 1(b) always implies the satisfaction of their U2 when the problem is well conditioned (see [8, p. 307]). Our rule 2 is simply their U4. Their U5 is discarded for the same reason discussed above. At this point, the motivation of step 3(a) of the algorithm should have become clear.

We now briefly discuss the operational counts of the algorithm. Steps 1(a), 1(b), and 2(a) can be executed with $O(n^2)$ operations, provided M (the rank of the null space matrix S) and k are both of constant order. Step 2(b) can be implemented via the Householder tridiagonalization algorithm, which in general takes about $(2/3)n_1^3$ operations, while some time-saving is possible through the distributed truncation proposed by Gu et al. [11]. Step 2(c) is usually performed by a golden section search on $\log(n\lambda)$; each evaluation of the score functions $V(\cdot)$ or $M(\cdot)$ via formulas (21) or (22) requires only $O(n)$ operations (see [11]). As we mentioned earlier, step 2(d) is the major burden on the algorithm. To calculate each of the $K_i = e^{\eta_i} U^T Q_i U$, noticing that U is available from step 2(b) in a factored form as the product of a series of Householder transforms, we could successively apply the Householder transforms from both sides of Q_i, which need approximately the same number of operations as step 2(b). Making use of the identity $\sum_{i=1}^{k} K_i = \tilde{T}$, we need a total of $(2/3)(k-1)n_1^3$ operations for the K_i's. Since the linear system $T\mathbf{x} = b$ can be solved with $O(n)$ operations, $T^{-1}K_i$'s and $T^{-2}K_i$'s can be obtained with $O(n^2)$ operations. Hence the total number of operations needed for each iteration is in general $(2/3)kn_1^3 + O(n^2)$. For each failure trial with $V > V_-$ or $M > M_-$, we have to spend $(2/3)n_1^3$ operations (step 2(b)) before discovering it. Step 3(b) needs another $(2/3)n_1^3$ to calculate the final results. The above operational counts are based on related discussions in [7] and [9]. See also [11].

Good starting values are important to Newton-type iterative optimization methods. They are even more crucial to our algorithm since our iteration is extremely expensive for large n. Deriving good starting values is not a mere numerical problem but something

to do with the original setup of the model. Below, we will propose two sensible approaches for obtaining good starting values for the algorithm. Both of them are based on the background problem formulation presented in § 1.

The first proposal is built on the assumption that the optimal smoothing parameters λ_i's share approximately the same decreasing rate as the number of observations increases. We can then randomly select a subset of the observations available, calculate the optimal $\boldsymbol{\theta}$ for the subset, and use these as the starting value for some bigger subset or the complete data set. The idea here is to perform a "crude" search with a smaller problem size to save execution time. If this assumption is strongly believed, then a large size iteration can be avoided by simply adopting the $\boldsymbol{\theta}$ obtained from a subset run and going ahead to perform only step 3(b) of the algorithm on the complete data set.

The second proposal is more involved. Suppose we knew the underlying truth function f_*, and it could be decomposed as the sum of projections onto orthogonal subspaces $\mathscr{H}_i$'s, $f_* = \sum_{i=0}^{k} P_i f_*$, where P_i denotes the projection operator onto subspace $\mathscr{H}_i$. When using the smoothing spline model to retrieve the truth function from data, it is sensible to adjust the face values of the norms $\|P_i f\|$ by the "strength" of the corresponding components of the truth function, $\|P_i f_*\|$, to balance the roughness penalties put on different components. Specifically, when $\|P_1 f_*\| = .07$ and $\|P_2 f_*\| = 7$, say, we might regard $P_1 f$ with $\|P_1 f\| = .05$ as rough as $P_2 f$ with $\|P_2 f\| = 5$. This heuristic leads to the following formulation with one smoothing parameter:

$$\min \frac{1}{n} \sum_{j=1}^{n} (y_j - L_j f)^2 + \lambda \sum_{i=1}^{k} \frac{\|P_i f\|^2}{\|P_i f_*\|^2},$$

or equivalently, $\theta_i = \|P_i f_*\|^2$. $\boldsymbol{\theta}$ chosen this way should be close to optimal, and hence be a good starting value for the iteration. Of course in practice we will never know the truth function f_*, and in turn the factors $\|P_i f_*\|^2$. However, all we need here is just a set of starting values for the iterative algorithm, and some estimates of the factors $\|P_i f_*\|^2$ should suffice. The resulting starting value procedure, which is made default in the implementation, is thus Algorithm 3.2.

ALGORITHM 3.2. If no starting values are specified, we calculate the default starting values as follows:

1. Set $\theta_i = (\mathrm{tr}\,(Q_i))^{-1}$, fit the one smoothing parameter spline model by minimizing $V(\lambda|\boldsymbol{\theta})$ or $M(\lambda|\boldsymbol{\theta})$, calculate the parameters $\mathbf{c}$.
2. Estimate $\|P_i f_*\|^2$ by $\theta_{i0} = \theta_i^2 \mathbf{c}^T \tilde{Q}_i \mathbf{c}$, and set the starting values of Algorithm 3.1 to be $\eta_{i0} = \log(\theta_{i0})$.

The choice of θ_i's in step 1 above is arbitrary but *invariant* in the sense we discussed in § 2; other invariant selections might be equally appropriate. We estimate $\|P_i f_*\|^2$'s simply by replacing $P_i f_*$ with $P_i f_\lambda = \sum_j c_j \theta_i (P_i \xi_j)$, where ξ_j is the representer of the linear functional L_i under the original norm $\|\cdot\|$. Remember that $\langle P_i \xi_{j_1}, P_i \xi_{j_2} \rangle = (\tilde{Q}_i)_{j_1 j_2}$. Algorithm 3.2 takes about $(2/3)n^3 + O(n^2)$ operations for execution.

4. Examples: Additive/interaction spline models. To test the algorithms presented in the previous section, we apply them to fit the additive/interaction spline models, which were proposed by Barry [2], [3] and Wahba [23], and were first illustrated by Gu et al. [11] in the $k = 2$ case where a grid search on the one-dimensional θ was conducted. In this section, we will report some of our numerical experiments with the iterative multiple smoothing parameter algorithm proposed in this article. Various statistical aspects of the additive/interaction spline models are currently under study. The findings of the study will be presented elsewhere. For a review of the additive

models via the backfitting algorithm without automatic smoothing parameter selection, see [13] and [4].

The formulation of the additive/interaction spline models is based on the tensor product Hilbert space formulation. Take the component Hilbert spaces (on [0, 1]) as the reproducing kernel Hilbert space $W_2^m = \{f: f^{(\nu)}$ abs cont., $\nu = 0, \cdots, m-1$, $\int (f^{(m)})^2 < \infty\}$ with norm $\|f\|^2 = \sum_{\nu=0}^{m-1} (\int_0^1 f^{(\nu)})^2 + \int_0^1 (f^{(m)})^2$ (see [6]). We let $\mathcal{H}$ be the tensor product Hilbert space $\mathcal{H} = \bigotimes_{l=1}^d \mathcal{H}^l$, with $\mathcal{H}^l = W_2^m$. Readers are referred to [1] for technical details on tensor products of reproducing kernel Hilbert spaces. We note that the component spaces of the tensor product space need not to be of the same form. For example, we might specify different m's for different $\mathcal{H}^l$'s for our formulation here, though we choose not to do so to keep the notation simple. Write $W_2^m = \mathcal{N} \oplus \mathcal{P}_{m-1} \oplus \mathcal{S}_m$, where $\mathcal{N}$ is the space of constant functions with square norm $(\int_0^1 f)^2$, $\mathcal{P}_{m-1}$ is the space of all polynomials with degrees less than m which integrate to zero, with square norm $\sum_{\nu=1}^{m-1} (\int_0^1 f^{(\nu)})^2$, and $\mathcal{S}_m$ is the space of functions with square integrable mth derivative and satisfy $\int_0^1 f^{(\nu)} = 0$, $\nu = 0, \cdots, m-1$, with square norm $\int_0^1 (f^{(m)})^2$. When $m = 1$, the space $\mathcal{P}_0$ vanishes. (Further decomposition of $\mathcal{P}_{m-1}$ for $m > 2$ is sometimes useful (see [11]), though we will not give much detail here.) From the direct sum decompositions of the component spaces formulated above, the space $\mathcal{H} = \bigotimes_{l=1}^d \mathcal{H}^l = \bigotimes_{l=1}^d (\mathcal{N}^l \oplus \mathcal{P}_{m-1}^l \oplus \mathcal{S}_m^l)$ can be represented as the direct sum of 3^d orthogonal subspaces when $m > 1$, and 2^d subspaces when $m = 1$. Various statistical model formulations can be specified from this structure. For example, the additive models are obtained by eliminating the subspaces with more than one nonconstant tensor component (i.e., with fewer than $d-1$ $\mathcal{N}$'s as components).

We first present a simulated additive spline example on $[0, 1]^4$. The truth function is specified by

$$f(\mathbf{x}) = f(x_1, x_2, x_3, x_4) = 10 \sin(\pi x_2) + \exp(3x_3) + 10^6 x_4^{11}(1-x_4)^6 + 10^4 x_4^3 (1-x_4)^{10}.$$

The bounded linear functionals L_j's are chosen as the evaluation functionals $[\mathbf{x}_j]f = f(\mathbf{x}_j)$. We generated $n = 100$ sampling points in $[0, 1]^4$ randomly, and computed the observations by $y_j = f(\mathbf{x}_j) + \varepsilon_j$, $j = 1, \cdots, n$, where ε_j's are independent Gaussian pseudorandom numbers with mean 0 and variance 1. The sampling points were generated using the Fortran routine uni of the Core Mathematics Library (Cmlib) from the National Bureau of Standards, with mdig = 32 and seed 2,375; the first 400 (after a null call to pass the seed) random variables were cut into four segments of length 100, and formed the first to fourth coordinates of the sampling points in the natural order. The ε_j's were generated by the routine rnor of Cmlib with mdig = 32, and we took the 100 outcomes after the null call which passed the seed 5,732 to the routine. The scatter plot matrix of $\mathbf{x}$ and y is shown in Fig. 1. We fitted models with $m = 2$, 1, respectively. For $m = 2$, we selected $\mathcal{H}_0 = \text{span}\{1, x_1 - .5, x_2 - .5, x_3 - .5, x_4 - .5\}$, where 1 spans $\bigotimes_{l=1}^4 \mathcal{N}^l$, $x_i - .5$ span $((\bigotimes_{l \neq i} \mathcal{N}^l) \otimes \mathcal{P}_1^i)$, $i = 1, \cdots, 4$. The penalized spaces were $\mathcal{H}_i = (\bigotimes_{l \neq i} \mathcal{N}^l) \otimes \mathcal{S}_2^i$, $i = 1, \cdots, 4$. For $m = 1$, we chose $\mathcal{H}_0 = \{1\}$ and $\mathcal{H}_i = (\bigotimes_{l \neq i} \mathcal{N}^l) \otimes \mathcal{S}_1^i$, $i = 1, \cdots, 4$. Other subspaces were deleted. The evaluation formulas for the matrices S, $\tilde{Q}_i$, $i = 1, \cdots, 4$, and for the representers $P_i \xi_j$ can be found from [11] for the $m = 2$ setting directly, and the $m = 1$ formulas can be similarly derived. Figures 2 and 3 illustrate the fitted additive components (solid lines) compared to the true additive components (dotted lines) for $m = 2$ (respectively, 1) with the vertical positions of the solid lines being adjusted to make the solid and dotted lines of the same frames integrate to the same number, as they should. The $m = 1$ curves (broken lines) look more wiggly than the $m = 2$ curves (cubic splines), although the main features are the same. The criterion for selecting the smoothing parameters was the

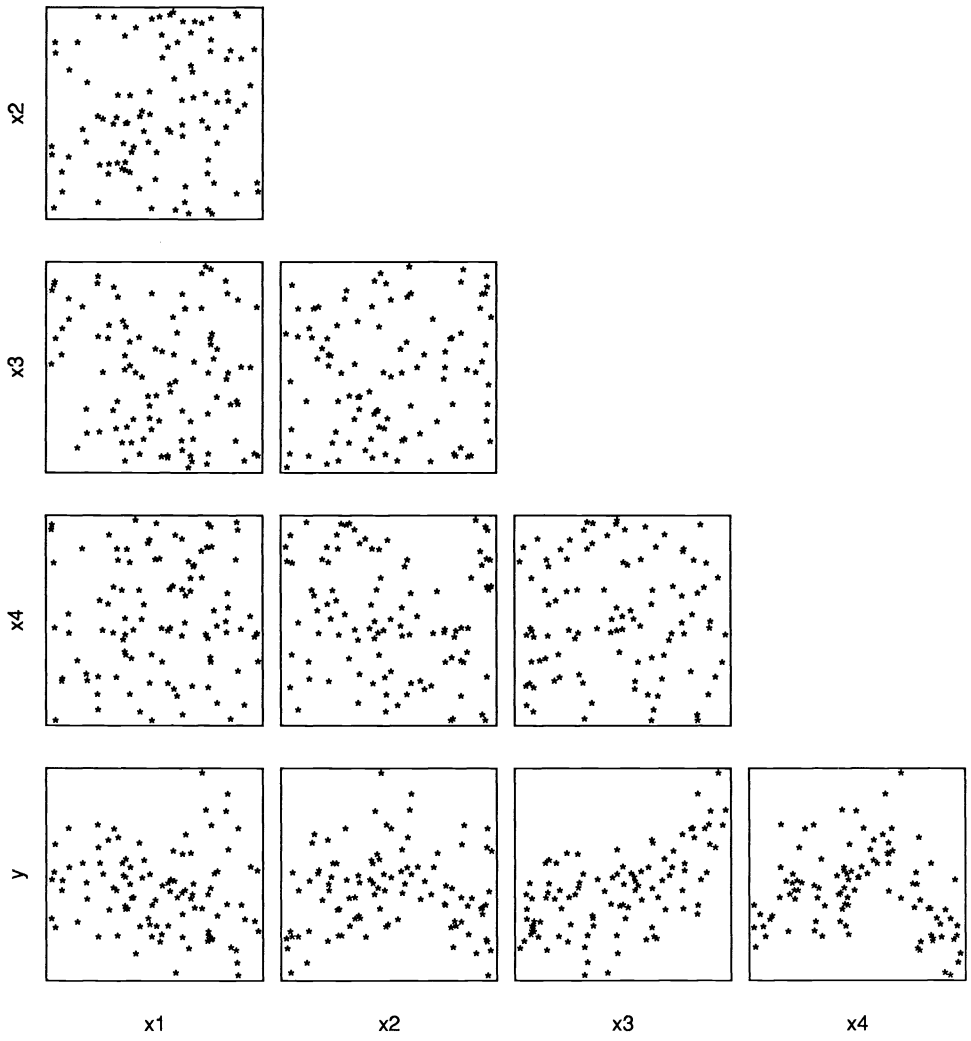


FIG. 1. *Scatter plot matrix: Additive model.*

GCV score. The GCV scores and the corresponding mean square errors (evaluated at the sampling points) of each iteration are listed in Table 1. In Table 1, the iterations numbered 0 correspond to the starting values, and the last iterations (number 5 for $m = 2$ and number 3 for $m = 1$) in the two cases correspond to the models illustrated in Fig. 2 and Fig. 3, respectively. It appears that the starting value procedure (Algorithm 3.2) worked pretty well, and the convergence of the iteration was very fast. It can also be seen that both V and mean square error (MSE) are smaller for $m = 2$, indicating that $m = 2$ should have been chosen as a better model according to the GCV score, and actually it *is* better according to the MSE. An execution for the $k = 4$, $n = 100$ case with five iterations (first column of Table 1; the amount of work is that of six main loop executions) ran 724 cpu seconds on a Sun-3/280 (without a floating point accelerator) in the Department of Statistics, University of Wisconsin-Madison. We have also tried $n = 200$ and $n = 400$ for the same model. An $n = 200$ execution with four iterations ran 3,628 cpu seconds on the same machine, and an $n = 400$ run with three iterations took 16,371 cpu seconds.

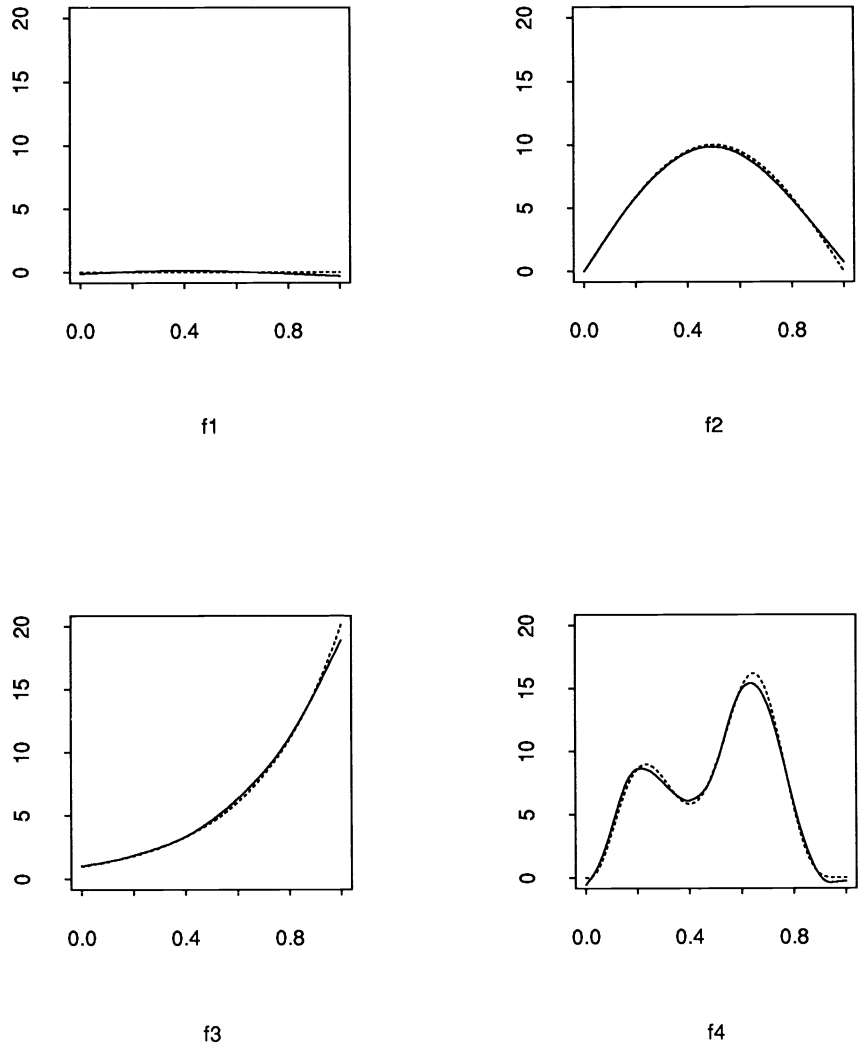


FIG. 2. *Additive model*: $m = 2$.

The second example has a nonnull interaction component. This time we worked on the unit cube $[0, 1]^3$, with truth function

$$f(\mathbf{x}) = f(x_1, x_2, x_3) = 10 \sin(\pi x_2) + \exp(3x_3) + 5 \cos(2\pi(x_1 - x_2)),$$

and $y_j = f(\mathbf{x}_j) + \varepsilon_j$. We used similar procedures and the same seeds to generate the sampling points and noise with the same mean and variance as in the additive model example, with $n = 400$ and $d = 3$. The scatter plot matrix of x_i's and y is omitted since it is no more informative than Fig. 1. We chose $m = 1$ for the interaction model example for its simpler formulation, though we would expect the plots of the estimation to look rather wrinkled. Eliminating the 3-factor interaction, we were left with seven subspaces of the tensor product Hilbert space $\mathscr{H} = \otimes_{l=1}^{3} \mathscr{H}^l$, namely, $\mathscr{H}_0 = \{1\} = \otimes_{l=1}^{3} \mathscr{N}^l$, $\mathscr{H}_i = (\otimes_{l \neq i} \mathscr{N}^l) \otimes \mathscr{S}_1^i$, $i = 1, 2, 3$, and $\mathscr{H}_{(i)} = (\otimes_{l \neq i} \mathscr{S}_1^l) \otimes \mathscr{N}^i$, $i = 1, 2, 3$, where $\mathscr{H}_0$ was the null space and the other $k = 6$ subspaces were penalized. We know that the truth function has null projections in $\mathscr{H}_1$, $\mathscr{H}_{(2)}$, and $\mathscr{H}_{(1)}$. Projecting the estimated function to the

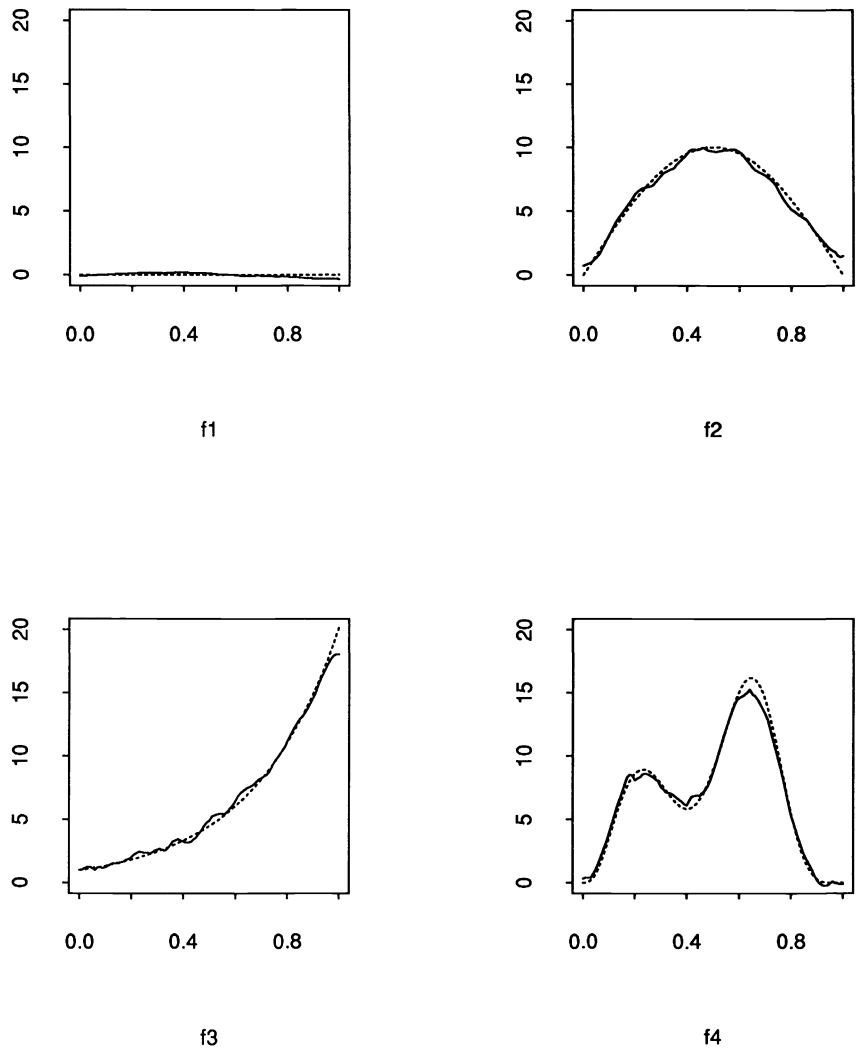


FIG. 3. *Additive model*: $m = 1$.

TABLE 1
Iterations of the additive models.

$m = 2$			$m = 1$		
Iter. No.	V	MSE	Iter. No.	V	MSE
0	1.50409	.291176	0	1.80939	.519140
1	1.50180	.232195	1	1.75559	.462696
2	1.45412	.273181	2	1.74514	.446254
3	1.41040	.243224	3	1.74504	.444939
4	1.40893	.234954			
5	1.40893	.234726			

subspaces, we can compare the truth and the estimated components individually. For the subspace $\mathcal{H}_1$, the estimated component $f_1(x_1)$ ranged between $[-.14, .12]$ on grids $x_1 = 0(.01)1$. On $\mathcal{H}_{(2)}$, the estimated component $f_{1,3}(x_1, x_3)$ ranged between $[-.38, .38]$ on the tensor product mesh $x_1, x_3 = 0(.033)1$. And on $\mathcal{H}_{(1)}$, $f_{2,3}(x_2, x_3)$ ranged between $[-.18, .15]$ on the tensor product mesh $x_2, x_3 = 0(.033)1$. All three null component estimates were well below the standard deviation of the ε_j's. The other two additive components $f_2 \in \mathcal{H}_2$ and $f_3 \in \mathcal{H}_3$ are plotted in Fig. 4 in the same manner as in Fig. 3, and the truth and the estimation of the interaction component $f_{1,2} \in \mathcal{H}_{(3)}$ are plotted in Fig. 5. The iteration sequence of this example is shown in Table 2. The iteration number 0 is again the starting value model. The algorithm converged at the third iteration in the sense that the optimal GCV value was within .1 percent of the achieved one. The total execution time was 25,859 cpu seconds.

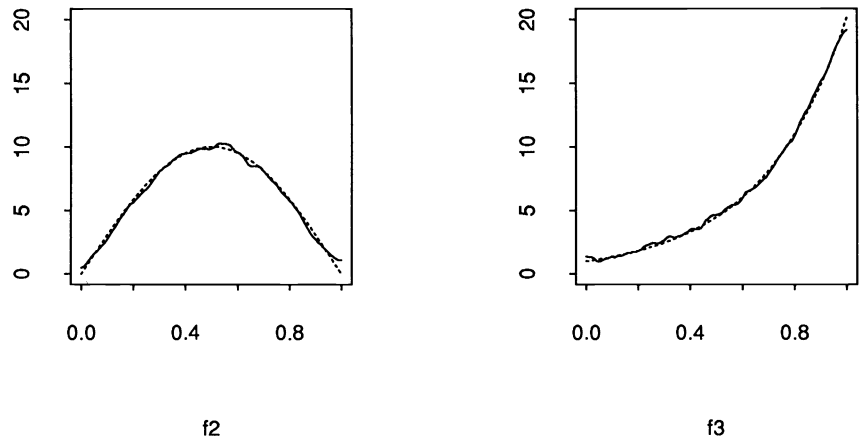


FIG. 4. *Interaction model: Additive components.*

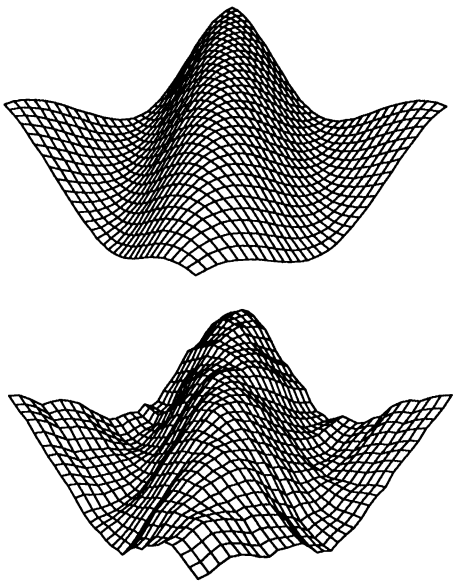


FIG. 5. *Interaction model: Interaction component.*

TABLE 2
Iterations of the interaction model.

Iter. No.	V	MSE	Iter. No.	V	MSE
0	1.40266	.376505	2	1.29839	.291201
1	1.31241	.303801	3	1.29585	.285957

5. Variance component models. Consider the variance component model (see [18, § 4j])

$$\mathbf{y} = S\boldsymbol{\beta} + \sum_{i=1}^{k} B_i \boldsymbol{\xi}_i + \boldsymbol{\varepsilon},$$

where $S\boldsymbol{\beta}$ is the fixed effect of dimension M, the B_i's are known matrices with $B_i B_i^T = \tilde{Q}_i$, $\boldsymbol{\xi}_i \sim N(0, b_i I)$, $\boldsymbol{\varepsilon} \sim N(0, \sigma^2 I)$, and $\boldsymbol{\xi}_i$'s and $\boldsymbol{\varepsilon}$ are mutually independent. Let the QR-decomposition of S be

$$S = FR = (F_1, F_2)\begin{pmatrix} R_1 \\ 0 \end{pmatrix}.$$

Since $\mathbf{z} = F_2^T \mathbf{y}$ are $n - M$ linearly independent contrasts of $\mathbf{y}$, the restricted maximum likelihood (REML) estimates of b_i's and σ^2 are the minimizers of the restricted log likelihood

$$l_*(\mathbf{b}, \sigma^2 | \mathbf{z}) = -\tfrac{1}{2} \log [\det (\textstyle\sum b_i Q_i + \sigma^2 I)] - \tfrac{1}{2} \mathbf{z}^T (\textstyle\sum b_i Q_i + \sigma^2 I)^{-1} \mathbf{z} + C_1,$$

where $Q_i = F_2^T \tilde{Q}_i F_2$ are as defined in earlier sections, and C_1 is a constant. See also [12]. Reparameterizing the problem by $b_i = b\theta_i$, $\sigma^2 = b(n\lambda)$, and maximizing $l_*(\cdot)$ with respect to b, we get $\hat{b} = \mathbf{z}^T (\sum \theta_i Q_i + n\lambda I)^{-1} \mathbf{z}/(n - M)$, and the profile restricted log likelihood is

$$l_*(\boldsymbol{\theta}, \lambda | \hat{b}) = -\frac{1}{2} \log [\det (\textstyle\sum \theta_i Q_i + n\lambda I)] - \frac{n - M}{2} \log (\hat{b}) - C_2,$$

where C_2 is another constant. (By profile likelihood we mean the likelihood function where some of the parameters have already been optimized.) This leads to the equivalent criterion of minimizing

$$M(\boldsymbol{\theta}, \lambda | \mathbf{z}) = \frac{\mathbf{z}^T (\sum \theta_i Q_i + n\lambda I)^{-1} \mathbf{z}/n}{[\det (\sum \theta_i Q_i + n\lambda I)^{-1}]^{1/(n-M)}}.$$

Hence, our algorithm is directly applicable to the REML estimation of the usual variance component models.

For the maximum likelihood (ML) estimation, writing $\mathbf{r} = \mathbf{y} - S\boldsymbol{\beta}$, we have the log likelihood

$$l(\mathbf{b}, \sigma^2 | \boldsymbol{\beta}) = -\tfrac{1}{2} \log [\det (\textstyle\sum b_i \tilde{Q}_i + \sigma^2 I)] - \frac{1}{2} \mathbf{r}^T (\textstyle\sum b_i \tilde{Q}_i + \sigma^2 I)^{-1} \mathbf{r} + C_3,$$

where C_3 is a constant. After the reparameterization $b_i = b\theta_i$, $\sigma^2 = b(n\lambda)$, we can again solve for b explicitly, leading to the minimization of a log likelihood $l(\lambda, \boldsymbol{\theta}, \boldsymbol{\beta})$. We can then compute the ML estimation by alternating parameters as

1) Maximize $l(\lambda, \boldsymbol{\beta} | \boldsymbol{\theta})$: Tridiagonalization.
2) Maximize $l(\boldsymbol{\theta} | \lambda, \boldsymbol{\beta})$: Newton update.

In step 1), we can perform an inner iteration loop to optimize λ by grid search and $\boldsymbol{\beta}$ by Gauss–Markov estimation. Specifically, we perform a tridiagonalization $\sum \theta_i \tilde{Q}_i = UTU^T$, and compute $\mathbf{x} = U^T\mathbf{y}$, $W = U^TS$. The optimal λ given $\boldsymbol{\beta}$ is the minimizer of the equivalent score

$$M(\lambda|\boldsymbol{\beta}, \boldsymbol{\theta}) = \frac{(\mathbf{x} - W\boldsymbol{\beta})^T(T + n\lambda I)^{-1}(\mathbf{x} - W\boldsymbol{\beta})/n}{\det(T + n\lambda I)^{-1/n}}.$$

The Gauss–Markov estimator of $\boldsymbol{\beta}$ given λ can be computed through

$$\boldsymbol{\beta} = [W^T(T + n\lambda I)^{-1}W]^{-1}W^T(T + n\lambda I)^{-1}\mathbf{x}.$$

Overall, step 1) is a $(2/3)n^3 + O(n^2)$ operation. Step 2) is as before a $(2/3)(k-1)n^3$ operation. The corresponding ML algorithm is trivial to specify.

Various properties of the ML and REML estimation of variance component models and their relation with other estimation methods are discussed in [12]. See also [19]. Surprisingly, the simple variable transformation $\eta = \log \theta$, which results in the invariant and constraint-free numerical procedures for the REML and ML estimation, is not recognized in previous works summarized by Harville [12] and Rao and Kleffe [19]. In a closely related context, Lindstrom and Bates [16] illustrate how to apply the Newton method to compute the REML and ML estimates of parameters including a covariance matrix, where they employ a Cholesky decomposition as the variable transformation to ensure the nonnegative-definiteness of the covariance matrix. Numerically, we should admit that our method is too general to make use of possible special structures of B_i (hence Q_i) to reduce computational load, as well as too restrictive to handle models where the variance-covariance matrix of $\mathbf{y}$ depends nonlinearly on the unknown parameters, though in the later case the model is no longer a variance component model literally.

6. Remarks. We now remark on some further points of interest about the algorithms.

(1) *Methodology.* Our iterative method is different from standard derivative-based optimization methods. It does not operate on all parameters simultaneously. Instead, it cycles between the two sets of parameters ($\boldsymbol{\theta}$ and λ), conditioning one on the latest version of the other, and uses different strategies for updating the two sets of parameters. Loosely speaking, each iteration targets on the profile score with respect to $\boldsymbol{\theta}$ where λ is being optimized, though the updates of $\boldsymbol{\theta}$ are not based on the derivatives of the profile score function since they are not available. Hence the method may be considered as a hybrid method adapted from the Newton methods on either the profile score or the score itself. In general, the idea of alternating parameters illustrated in this article can be expected to prove useful in tackling other statistical computing problems similar to the ones considered here. It is particularly appropriate in the context of GCV/GML functions since it is our experience, and seems to be part of the folklore in the related area of components of variance, that the value of these functions near their minima is substantially more sensitive to the value of λ than to the relative sizes of the θ's. The idea of cycling between parameters appeared some time ago in the psychometric literature in a different context (see, for example, [5]).

(2) *Convergence.* The goal of the algorithm is to locate a set of smoothing parameters which offer minimum GCV/GML scores and in turn a near-optimal smoothing. If the (true, but unknown) predictive mean square error of the estimate of f as a function of the smoothing parameters has a flat bottom near its minimizer, then the GCV score will also tend to have a flat bottom. (See [6], for example.) When that is

the case we would be glad to pick any one point from the bottom, and the exact optimal θ_i's are not particularly important. In this sense the convergence property of our algorithm is at least as good as the standard Newton method for minimizing the GCV/GML scores with respect to $\log \lambda_i$'s, since for fixed λ $\log \theta_i$'s are only a constant shift from the $\log \lambda_i$'s and the $\boldsymbol{\eta} = \log \boldsymbol{\theta}$ updating is via the standard Newton method.

(3) *Optimal parameter of infinity.* A solution with some $\theta_i = 0$ (i.e., $\lambda_i = \infty$) is of special interest. It indicates the absence of a certain "effect" and hence gives a more parsimonious statistical model which is often desirable. In the current algorithm we *set* an optimal parameter of infinity $\eta_i = -\infty$ after discovering that the score function is close enough to the minimum (i.e., converged), but the algorithm still drives η_i with a large step towards $-\infty$, indicating a flat bottom extending to $\eta_i = -\infty$. In all of the examples we tried where such an optimal parameter of infinity was set (not presented here), the scores with the parameter set to infinity did reduce further compared to the scores at the point where convergence was declared. From a purely statistical point of view, there is a certain probability that, even though a certain component of f is absent, and hence its theoretically optimal η_i is $-\infty$, the exact η_i minimizing the GCV/GML function will be greater than $-\infty$. Other methods for appropriately setting $\lambda_i = \infty$ (which are not necessarily GCV/GML optimal) but have appropriate statistical properties, are under study.

(4) *Invariance.* The invariance property of concern in this article is the invariance of the algorithm. It is different from the invariance properties encountered in most of the statistical literature which enforce the invariance of the end results of statistical procedures. In contrast, our concern is to enforce the invariance of numerical iteration sequences which we hope will converge to the invariantly defined end results.

(5) *Numerical efficiency.* Our iterative algorithm is very expensive for large n, but it is believed to be highly efficient for the problem. The algorithm is efficient in the sense that its operational count, $(2/3)kn^3 + O(kn^2)$ flops per iteration, is pretty much the least that can be reasonably expected, given the fact that the single evaluation of the profile score is a $(2/3)n^3 + O(n^2)$ flop operation, which is itself the best available in general smoothing spline settings (see [11]). Some Monte Carlo approximations to the derivatives (read the trace terms in (12), (14), (18), and (20)), which require only $O(n^2)$ extra operations beyond the profile score evaluation, have also been tried. They were eventually discarded because the derivative-based methods are very sensitive to the errors in the derivative evaluations [8], and the approximations will lose most or all significant digits due to cancellation as the optimum is being approached.

(6) *Other applications and the starting values.* It is obvious that our iterative algorithm applies to all settings which result in minimization problems with similar form to equation (7) or (8). However, our starting value procedures are derived from the setting of (4), and hence may not be appropriate for other settings such as the variance component models.

(7) *Step length search.* There is no step length search included in the current version of the algorithm except a safeguard procedure of Armijo type in step 2(c) of Algorithm 3.1. As the associate editor suggested, including a step length search will lead to a more reliable algorithm. However, since the cost for score evaluation is comparable to derivative computation in our setting, a step length search will consume considerable execution time, and the overall procedure may not be as efficient. On the other hand, since we have good starting values and since the λ-step of the algorithm keeps us close to the optimal area where the scores behave reasonably well (see the plots in [11]), we found the current algorithm quite satisfactory on the examples we tried. (As a matter of fact, the λ-step of the algorithm is an exact line search, though

it is on a different line.) It will be worthwhile to investigate the pros and cons of including a step length search in further study.

Acknowledgments. The original idea of the starting value procedure Algorithm 2 was inspired by a conversation with Zehua Chen, to whom we owe thanks. We also wish to acknowledge the effort of our Computer Committee in the Department of Statistics, University of Wisconsin-Madison, led by Doug Bates, for maintaining a very comfortable computing environment in the department.

REFERENCES

[1] N. ARONSZAJN, *Theory of reproducing kernels*, Trans. Amer. Math. Soc., 68 (1950), pp. 337-404.
[2] D. BARRY, *Nonparametric Bayesian regression*, Ph.D. thesis, Department of Statistics, Yale University, New Haven, CT, 1983.
[3] ———, *Nonparametric Bayesian regression*, Ann. Statist., 14 (1986), pp. 934-953.
[4] A. BUJA, T. HASTIE, AND R. TIBSHIRANI, *Linear smoothers and additive models*, Ann. Statist., 17 (1989), pp. 453-555 (with discussion).
[5] J. CARROLL AND J. CHANG, *Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition*, Psychometrika, 35 (1970), pp. 283-319.
[6] P. CRAVEN AND G. WAHBA, *Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation*, Numer. Math., 31 (1979), pp. 377-403.
[7] J. DONGARRA, C. MOLER, J. BUNCH, AND G. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.
[8] P. GILL, W. MURRAY, AND M. WRIGHT, *Practical Optimization*, Academic Press, New York, 1981.
[9] G. GOLUB AND C. VAN LOAN, *Matrix Computation*, The Johns Hopkins University Press, Baltimore, MD, 1983.
[10] C. GU, RKPACK *and its applications: Fitting smoothing spline models*, Tech. Report 857, Department of Statistics, University of Wisconsin, Madison, WI, 1989.
[11] C. GU, D. BATES, Z. CHEN, AND G. WAHBA, *The computation of* GCV *functions through Householder tridiagonalization with application to the fitting of interaction spline models*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 457-480.
[12] D. A. HARVILLE, *Maximum likelihood approaches to variance component estimation and to related problems*, J. Amer. Statist. Assoc., 72 (1977), pp. 320-340 (with discussion).
[13] T. HASTIE AND R. TIBSHIRANI, *Generalized additive models*, Statist. Sci., 1 (1986), pp. 297-318 (with discussion).
[14] G. KIMELDORF AND G. WAHBA, *Some results on Tchebycheffian spline functions*, J. Math. Anal. Appl., 33 (1971), pp. 82-85.
[15] K. LI, *Asymptotic optimality of* C_L *and generalized cross-validation in ridge regression with application to spline smoothing*, Ann. Statist., 14 (1986), pp. 1101-1112.
[16] M. J. LINDSTROM AND D. M. BATES, *Newton-Raphson and* EM *algorithms for linear mixed-effects models for repeated-measures data*, J. Amer. Statist. Assoc., 83 (1988), pp. 1014-1022.
[17] F. O'SULLIVAN, *A statistical perspective on ill-posed inverse problems*, Statist. Sci., 1 (1986), pp. 502-527 (with discussion).
[18] C. R. RAO, *Linear Statistical Inference and Its Applications*, John Wiley, New York, 1973.
[19] C. R. RAO AND J. KLEFFE, *Estimation of Variance Components and Applications*, North-Holland, Amsterdam, 1988.
[20] G. WAHBA, *Improper priors, spline smoothing and the problem of guarding against model errors in regression*, J. Roy. Statist. Soc., Ser. B, 40 (1978), pp. 364-372.
[21] ———, *Surface fitting with scattered noisy data on Euclidean d-space and on the sphere*, Rocky Mountain J. Math., 14 (1984), pp. 281-299.
[22] ———, *A comparison of* GCV *and* GML *for choosing the smoothing parameter in the generalized spline smoothing problem*, Ann. Statist., 13 (1985), pp. 1378-1402.
[23] ———, *Partial and interaction splines for the semiparametric estimation of functions of several variables*, in Computer Science and Statistics: Proceedings of the 18th Symposium on the Interface, T. J. Boardman, ed., American Statistical Association, Washington, DC, 1986, pp. 75-80.
[24] G. WAHBA AND Y. WANG, *When is the optimal regularization parameter insensitive to the choice of the loss function*, Tech. Report 809, Department of Statistics, University of Wisconsin, Madison, WI, 1987.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 399-422, March 1991
© 1991 Society for Industrial and Applied Mathematics
010

A COMPARISON OF SOME NUMERICAL CONFORMAL MAPPING METHODS FOR EXTERIOR REGIONS*

THOMAS K. DELILLO† AND ALAN R. ELCRAT†‡

Abstract. Several methods for conformally mapping the exterior of the unit disk onto the exterior of a smooth curve are compared. These methods, which are known as the Timman, Friberg, Wegmann, and Theodorsen methods, are based on Fourier series and conjugation and are implemented using Fast Fourier Transforms. The computations reported here indicate that Wegmann's method converges faster and is more robust than the others.

Key words. numerical conformal mapping, conjugation, the Theodorsen method, the Timman method, the Friberg method, the Wegmann method

AMS(MOS) subject classifications. 30C30, 65E05

1. Introduction. This paper is concerned with a comparison among several methods for computing the conformal mapping of the exterior of the unit disk onto the exterior of a simple closed curve in the complex plane. The methods studied are often associated with the names Timman, Friberg, Wegmann, and Theodorsen [3], [10]. Timman's method is also known as James' method and appears to be the method of choice in the aerodynamics community [1], [8], [9], [12]. Wegmann's method is based on solving a sequence of Riemann–Hilbert problems for the approximate mapping functions; it has been shown theoretically to converge quadratically [15] under certain hypotheses, and the suggestion has been made that it is superior to all known methods for a large class of problems [10, p. 422]. Theodorsen's method is the oldest and most thoroughly studied of all these methods; we include it for completeness and as a bridge to earlier work. Friberg's method may be thought of as a hybrid of that of Timman and Theodorsen. The methods of Theodorsen, Timman, and Friberg converge linearly.

Theodorsen's method requires a star-shaped curve, but the others can be formulated for a general curve with arbitrary parameterizations. In many of the previous studies arclength has been used; we use both arclength and other parameters and in some of our examples this has made a significant difference in the observed convergence behavior. All of the methods considered here may be thought of as approximating the Fourier series of the desired mapping, and they make use of Fast Fourier Transforms (FFTs) to perform conjugation of a function defined in the unit circle. For Timman and Friberg the equation being solved iteratively is an integrodifferential equation and each step requires integration of a function defined on the circle. Henrici [10] has suggested using Fourier series to do this integration, and we have used this idea. To our knowledge this has not been done in earlier computational studies.

The part of our work which compares Timman and Theodorsen expands on Halsey's work [9], and is made in light of the crowding phenomenon [16]. For all of the methods we have given a more careful study of the discretization error than has been presented before. For Timman and Friberg, Kaiser [13], [7] has given a theoretical analysis which implies convergence for curves near to a circle. Our experiments confirm these results. Our general conclusion is that Wegmann's method compares favorably

* Received by the editors August 7, 1989; accepted for publication (in revised form) January 16, 1990.
† Department of Mathematics and Statistics, Wichita State University, Wichita, Kansas 67208-1595.
‡ The research of this author was partially supported by U.S. Air Force grant AFOSR-86-0274.

with the others in many situations. We have not included relaxation techniques in our study. It is possible that they could significantly improve the range of performance of both Theodorsen [6] and Timman/James [12, p. 118].

An outline of the paper is as follows. Section 2 gives a derivation of the methods based on function conjugation, as in [7], and specifies the iterations. Section 3 treats six examples.

2. Methods for exterior regions. Suppose that Γ is a smooth, simple, closed curve in the z-plane with the origin in its interior, and that $z = z(\sigma)$, $0 \leqq \sigma \leqq L$ is a smooth parameterization of Γ with $z'(\sigma) \neq 0$. We seek a conformal mapping g from the exterior of the unit disk in the w-plane to the exterior of Γ in the z-plane. g is normalized by $g(\infty) = \infty$, and either $g'(\infty) > 0$ or $g(1) = z(\sigma_0)$ for some fixed $\sigma_0 \in [0, L]$. (The second condition makes sense by virtue of the Osgood-Carathéodory theorem [10] and the smoothness of Γ.) We note that g has the Laurent expansion

$$g(w) = cw + a_0 + a_1 w^{-1} + \cdots$$

with

$$c = \gamma e^{i\alpha},$$

where the positive number γ is the "capacity" of the curve Γ [10, Chap. 16]. We may write

$$g(e^{i\theta}) = z(\sigma(\theta))$$

and, since g is determined by its boundary values, our problem reduces to finding $\sigma(\theta)$.

The unit disk in the w-plane is denoted by D, the exterior by D^c, and the circle by S.

Each of our methods is associated with an auxiliary function h of g defined on D^c [3], [7], [10]. We need the following proposition.

PROPOSITION. *For* $h \in H^1(D^c)$:

(a) $\operatorname{Im} h(e^{i\theta}) - \operatorname{Im} h(\infty) = -K\,[\operatorname{Re} h(e^{i\theta})]$,

(b) $\operatorname{Re} h(e^{i\theta}) - \operatorname{Re} h(\infty) = K\,[\operatorname{Im} h(e^{i\theta})]$.

Here K is the conjugation operator,

$$K = F^{-1}\hat{K}F$$

where F takes a function to its Fourier coefficients, a_n and

$$\hat{K} : a_n \to -i \operatorname{sgn}(n) a_n.$$

The discrete version of K is

$$K_N = F_N^{-1}\hat{K}F_N$$

where F_N is the discrete Fourier transform, which can be realized in $O(N \log N)$ multiplications using the FFT algorithm (see [5]). $H^1(D^c)$ is the Hardy space associated with $L^1(S)$. Note that K maps constant functions to zero.

The methods to be discussed are now derived from the above proposition using the appropriate auxiliary function.

(i) Theodorsen [3], [6], [7], [9]. We assume here that Γ is given by

$$z(\sigma) = \rho(\sigma)\, e^{i\sigma}, \qquad 0 \leqq \sigma < 2\pi$$

for a smooth positive function ρ. Then

$$h(w) = \log\left(\frac{g(w)}{w}\right)$$

and

$$h(e^{i\theta}) = \log|z(\sigma(\theta))| + i(\arg z(\sigma(\theta)) - \theta)$$
$$= \log \rho(\sigma(\theta)) + i(\sigma(\theta) - \theta).$$

Using the normalization $g'(\infty) > 0$ ($\alpha = 0$), Proposition (a) implies

$$\sigma(\theta) - \theta = -K[\log \rho(\sigma(\theta))].$$

For Theodorsen's method

$$\sigma^{n+1}(\theta) - \theta = -K[\log \rho(\sigma^n(\theta))].$$

It is known that the "ε-condition"

$$\varepsilon = \max \left|\frac{\rho'}{\rho}\right| < 1$$

implies that $\sigma^n(\theta)$ converges linearly to $\sigma(\theta)$ in L^2; there is a corresponding theory for the discrete problem and a bridge between the discrete and continuous.

(ii) Timman [3], [7], [9], [10], [13]. We define

$$h(w) = \log g'(w).$$

Since $g'(e^{i\theta}) = -z'(\sigma(\theta))\sigma'(\theta) i e^{-i\theta}$, we have

$$h(e^{i\theta}) = \log \sigma'(\theta) + \log|z'(\sigma(\theta))| + i\left(\arg z'(\sigma(\theta)) - \theta - \frac{\pi}{2}\right).$$

Proposition (b) implies

$$\log \sigma'(\theta) + \log|z'(\sigma(\theta))| - \log \gamma = K\left[\arg z'(\sigma(\theta)) - \theta - \frac{\pi}{2}\right].$$

Suppose we use $g(1) = z(\sigma_0)$ as our normalization. Then

$$\alpha = \operatorname{Im} h(\infty) = \frac{1}{2\pi}\int_0^{2\pi} (\arg z'(\sigma(\theta)) - \theta)\, d\theta - \frac{\pi}{2}$$

since $\operatorname{Im} h(w)$ is harmonic. The iteration is

$$\sigma^{n+1}(\theta) = \gamma_n \int_0^{\theta} \exp\left(K[\arg z'(\sigma^n(\theta)) - \theta]\right)/|z'(\sigma^n(\theta))|\, d\theta$$

where $L\gamma_n^{-1}$ is given by the integral over $[0, 2\pi]$. We perform the integration by integrating the Fourier series of the integrand term by term as suggested by Henrici [10]. Since

$$h(w) = \log(c - O(w^{-2})) = O(w^{-2}) + \log c$$

at infinity, we set the first two Fourier coefficients of $K[\arg z'(\sigma^n(\theta)) - \theta]$ equal to zero [10], [1].

Kaiser [13] has shown that σ^n converges linearly to σ if Γ satisfies certain hypotheses. In particular, if σ is arclength and Γ is close to a circle the convergence factor is .5.

(iii) Friberg [3], [7], [13]. Here

$$h(w) = \log g'(w) - \log \frac{g(w)}{w}.$$

Note that $h(w) = O(w^{-1})$ at infinity and that $h(\infty) = 0$. Proposition (b) implies

$$\log \sigma'(\theta) = K[\arg z'(\sigma(\theta)) - \arg z(\sigma(\theta))] + \log |z(\sigma(\theta))| - \log |z'(\sigma(\theta))|;$$

the iteration is

$$\hat{\sigma}^{n+1}(\theta) = \int_0^{\theta} |z'(\sigma^n(\theta))|^{-1} |z(\sigma^n(\theta))| \exp\left(K[\arg z'(\sigma^n(\theta)) - \arg z(\sigma^n(\theta))]\right) d\theta$$

and $\sigma^{n+1}(\theta) = \hat{\sigma}^{n+1}(\theta) L / \hat{\sigma}^{n+1}(2\pi)$. We note that α, γ can be found as a part of this iteration using the Timman auxiliary function.

Kaiser [13] also proved a convergence theorem for this method; in this case the convergence factor for a curve close to a circle is nearly zero.

(iv) Wegmann [10], [15]. This method is described by giving a correction to a current approximation to $\sigma(\theta)$. We want to compute $\tau(\theta)$ so that

$$g(e^{i\theta}) = z(\sigma(\theta) + \tau(\theta));$$

linearizing

$$z(\sigma(\theta)) + z'(\sigma(\theta))\tau(\theta) = g^{-}(e^{i\theta})$$

where $g^{-}(e^{i\theta})$ are the boundary values of a map to a nearby curve. We normalize by assuming $g'(\infty)$ is real. Since $\tau(\theta)$ is real $g^{-}(e^{i\theta})$ must be a solution of the Riemann–Hilbert problem

$$\operatorname{Re}(i\bar{z}'(s) g^{-}(s)) = \operatorname{Re}(i\bar{z}'(s) z(s))$$

where $s = e^{i\theta}$, $z(s) = z(\sigma(\theta))$, $z'(s) = z'(\sigma(\theta))$. It can be shown that

$$g^{-}(w) = [e^{-i\hat{\delta}} k_1(w) - \hat{\lambda} / \sin \hat{\delta}] w\, e^{l(w)}$$

where (note the correction of a misprint in [10] in the first of the following and the use of a general parameterization)

$$l(w) = -\frac{1}{\pi} \int_{|s|=1} \frac{\arg z'(s) - \theta}{s - w}\, ds,$$

$$\lambda(s) = \exp(-K[\arg z'(s) - \theta]) \operatorname{Im}(z'(s)\overline{z(s)}) |z'(s)|^{-1},$$

$$k_1(w) = \frac{1}{\pi} \int_{|s|=1} \frac{\lambda(s)}{s - w}\, ds,$$

$$\hat{\lambda} = \frac{1}{2\pi} \int_0^{2\pi} \lambda(e^{i\theta})\, d\theta$$

$$\hat{\delta} = -\frac{1}{2\pi} \int_0^{2\pi} [\arg(z'(\sigma(\theta)) - \theta)]\, d\theta.$$

Then

$$\begin{aligned} g^{-}(s) &= -\exp(i \arg z'(s) + K[\arg z'(s) - \theta]) \{K[\lambda(s)] + \hat{\lambda} \cot \hat{\delta} + i\lambda(s)\} \\ &= z(\sigma(\theta)) + z'(\sigma(\theta))\tau(\theta). \end{aligned}$$

The iteration is given by

$$\tau^n(\theta) = \frac{g^{-}(e^{i\theta})}{z'(\sigma^n(\theta))} - \frac{z(\sigma^n(\theta))}{z'(\sigma^n(\theta))}$$

and

$$\sigma^{n+1}(\theta) = \sigma^n(\theta) + \tau^n(\theta).$$

Wegmann [15] has shown that for smooth curves and a good enough initial guess, σ^n converges to σ quadratically in $W^{1,2}(S)$.

3. Numerical experiments. We have done numerical experiments with maps to the exterior of the following regions: (a) ellipse, (b) sports ground, (c) inverted ellipse, (d) cosine airfoil, (e) a perturbed circle, and (f) a general spline curve. Our implementation uses the radix-2 N point FFT routine listed in [2, p. 416] where $N = 2^M$.

There are two errors associated with the iterations:

$$\max_{1 \leqq j \leqq N} |\sigma^{(n)}(\theta_j) - \sigma^{(n-1)}(\theta_j)|$$

and

$$\max_{1 \leqq j \leqq N} |\sigma(\theta_j) - \sigma^{(n)}(\theta_j)|$$

where $\theta_j = 2\pi(j-1)/N$ are the Fourier points, σ is the exact boundary correspondence, and $\sigma^{(n)}(\theta_j)$ are the values at the nth iterate. The first error can generally be iterated to zero in machine precision. Exceptions to this occur for Wegmann's method, where it may cease to decrease much below the level of the discretization error and often exhibits a convergence/divergence behavior, to be discussed below. The second error is our computational measure of the discretization error. We have $\sigma(\theta)$ only in the

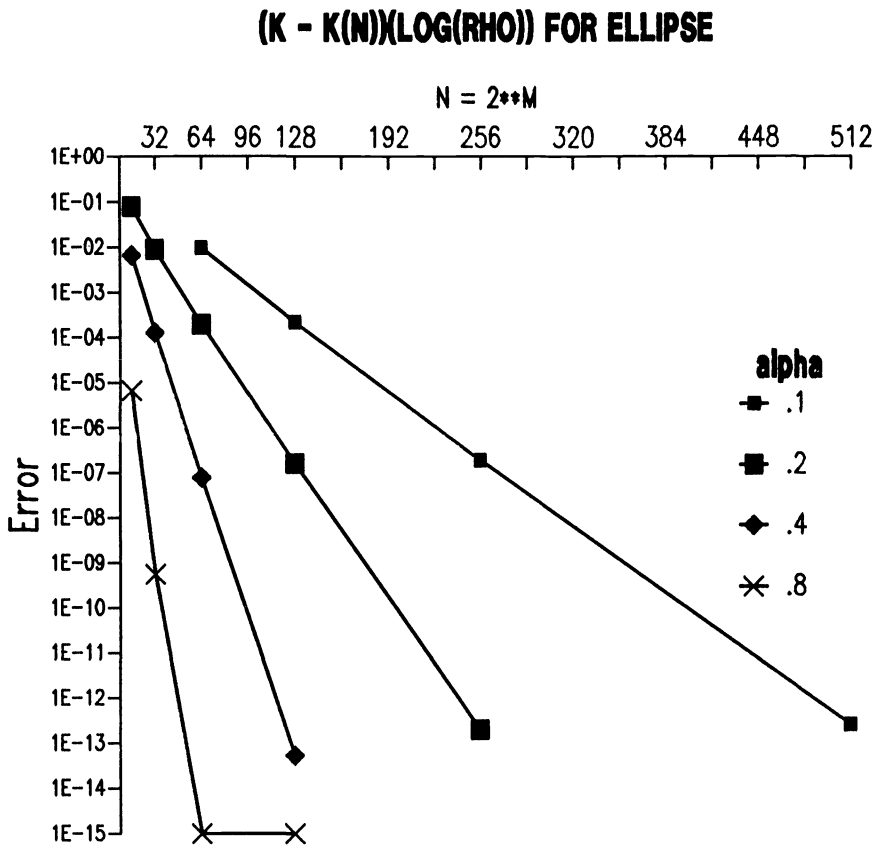


FIG. 1. $\max_{1 \leqq j \leqq N} |(K - K_N) \log \rho(\sigma(\theta_j))|$ *for ellipse.*

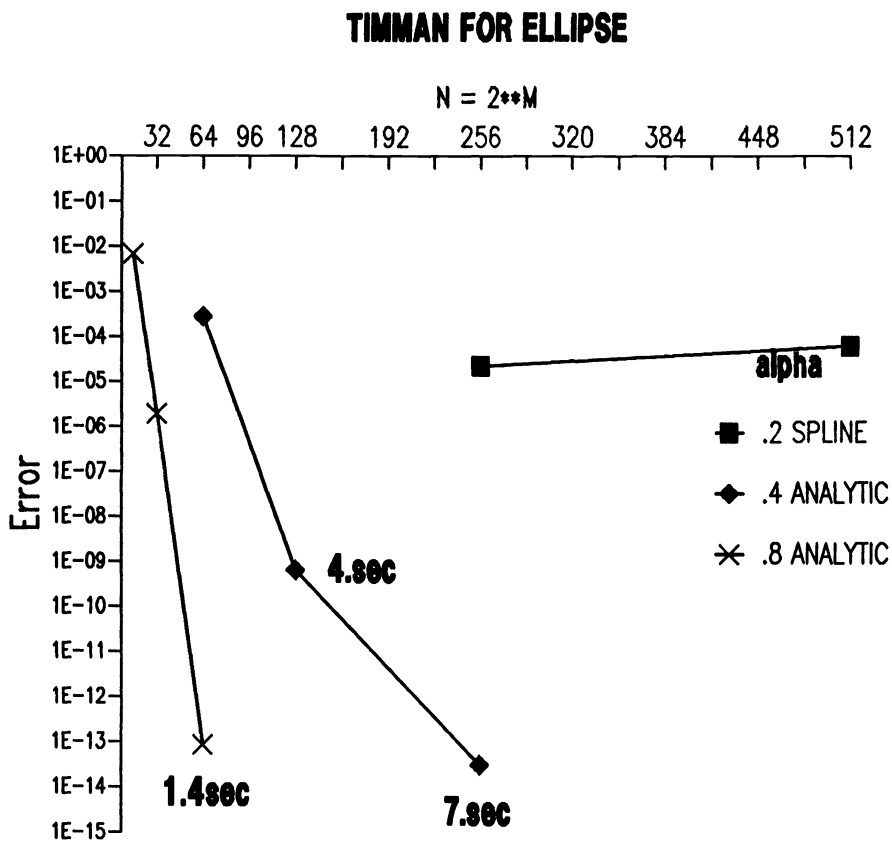


FIG. 2(a). *Discretization error for Timman for ellipse.*

case (a), so in the other examples we estimate the discretization error by computing an approximation to

$$\max_{0\leq\theta\leq 2\pi} \operatorname{dist}(g^{(n)}(e^{i\theta}), \Gamma),$$

the Hausdorff distance between $g^{(n)}(S)$ and Γ. Here $g^{(n)}$ is the approximation to the Taylor series of g found using $\sigma^{(n)}(\theta)$ and $N' > N$ Fourier points; the values of $g^{(n)}(e^{i\theta_j})$ at these N' points are found using the N'-point FFT and the Taylor series of $g^{(n)}$ zero padded from N to N' points. (For example, (b) $\operatorname{dist}(g^{(n)}(e^{i\theta_j}), \Gamma)$ can be found exactly and it is approximated by

$$|g^{(n)}(e^{i\theta_j}) - z(\arg g^{(n)}(e^{i\theta_j}))|$$

for (c) and (d).) We do not compute the discretization error for (e) and (f).

The curves (a), (b), (c), (d) are all given by piecewise analytic expressions. We also consider periodic cubic splines parameterized by the chordal distance [11] fitted to points on these curves. The chordal distance approximates the arclength so $|z'(\sigma)| \approx 1$. Usually we use $NN = 1{,}000$ knots evenly distributed in the original parameterization. In this case the spline approximates the analytic curves to within about 10^{-7} to 10^{-10} in most of our examples, so the discretization error can never go below that. However, the spline is needed in example (d) where a parameterization is needed for the near circular region generated by the Karman-Trefftz map. (To treat this case with Theodorsen's method we would have to use a spline parameterized by polar angle;

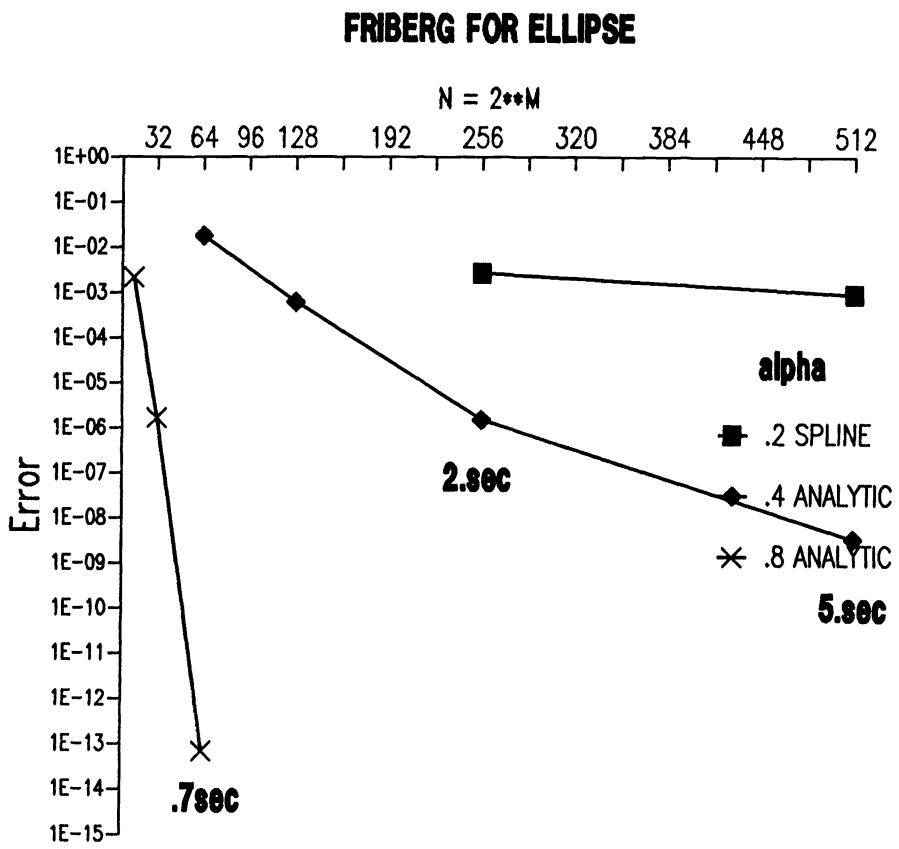


FIG. 2(b). *Discretization error for Friberg for ellipse.*

we have chosen to omit this case.) In practice, we usually have a finite number of points to define a boundary. This can be done conveniently with a spline as illustrated in example (f).

The initial guess for the examples below is always $\sigma^{(0)}(\theta) = L\theta/2\pi$, where $\theta \in [0, 2\pi]$.

The numbers near the data points on the figures are rough timings of CPU second (IBM 3081, FORTVS compiler, double precision arithmetic).

(a) Ellipse [10, p. 412]. Γ is given by

$$z(\theta) = \cos\theta + i\alpha \sin\theta, \qquad 0 \leqq \theta \leqq 2\pi.$$

σ is a "thinness" parameter, $0 < \alpha \leqq 1$. The exact solution, if $g'(\infty) > 0$ or $g(1) = 1$ is imposed, is

$$g(w) = \tfrac{1}{2}((1+\alpha)w + (1-\alpha)w^{-1}).$$

We can also represent Γ in polar coordinates, $z(\sigma) = \rho(\sigma)e^{i\sigma}$, $0 \leqq \sigma \leqq 2\pi$, with

$$\rho(\sigma) = \alpha/\sqrt{1-(1-\alpha^2)\cos^2\sigma}.$$

In this representation the exact boundary correspondence is

$$\sigma(\theta) = \arctan(\alpha \tan\theta).$$

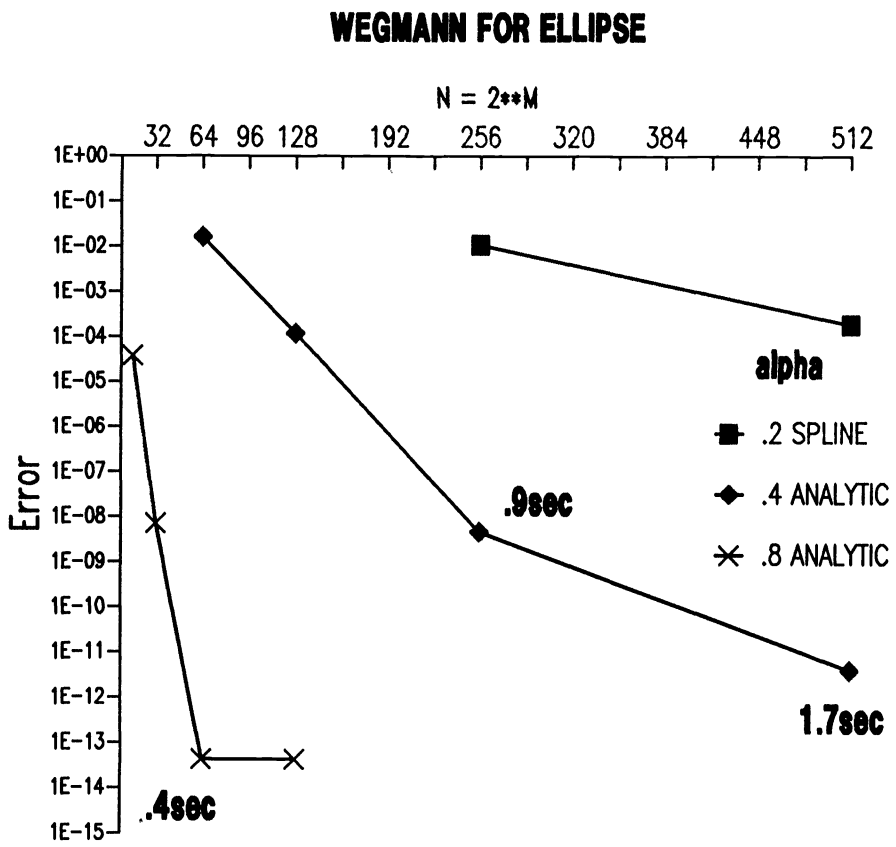


FIG. 2(c). *Discretization error for Wegmann for ellipse.*

We note that the Laurent series of g converges (in fact has only two terms) for all $w \neq 0$ but that $g'(w) = 0$ at

$$w = \pm\sqrt{\frac{1-\alpha}{1+\alpha}}.$$

From the expression for $g(w)$ we derive the Fourier coefficients of the auxiliary function for Theodorsen's method:

$$\begin{aligned} h(e^{i\theta}) &= \log\left(g(e^{i\theta})/e^{i\theta}\right) \\ &= \log \rho(\sigma(\theta)) + i(\sigma(\theta) - \theta) \\ &= \log\left(\frac{1+\alpha}{2}\right) + \beta\cos(2\theta) - \frac{\beta^2}{2}\cos(4\theta) + \cdots \\ &\quad + i\left(-\beta\sin(2\theta) + \frac{\beta^2}{2}\sin(4\theta) - \cdots\right) \end{aligned}$$

where $\beta = (1-\alpha)/(1+\alpha)$. The N-point FFT gives an approximation to the first $N/2$ terms of the sin/cos series (see [5]). Thus for the above series we have

$$\log \rho(\sigma(\theta)) \approx \log\left(\frac{1+\alpha}{2}\right) + \beta\cos(2\theta) - \frac{\beta^2}{2}\cos(4\theta) + - \cdots \pm \frac{\beta^{N/4}}{N/4}\cos\frac{N}{2}\theta.$$

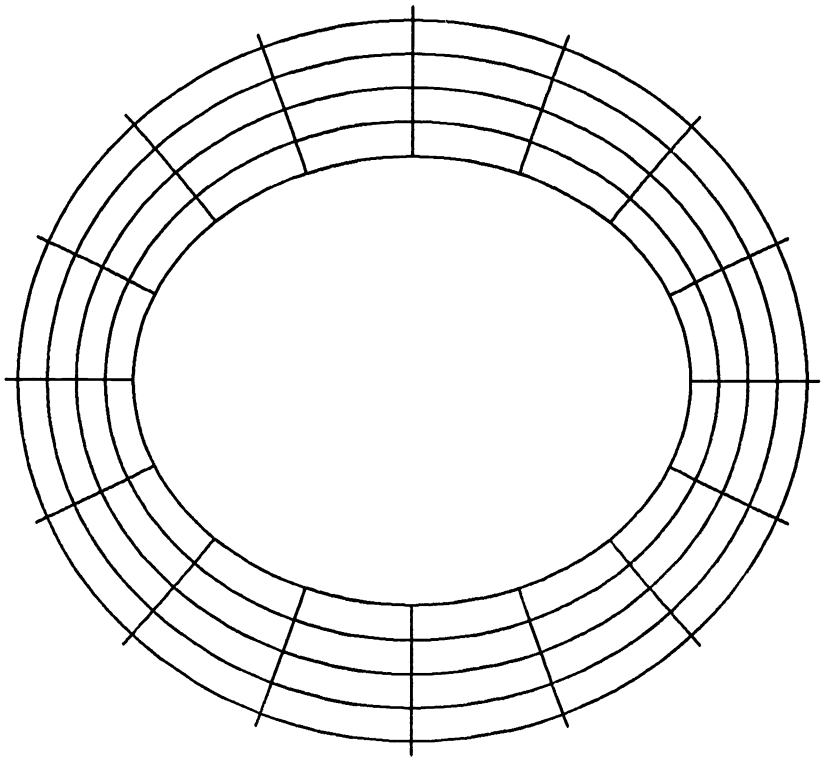


FIG. 3(a). *Ellipse* $\alpha = .8$ *with Theodorsen* $N = 64$.

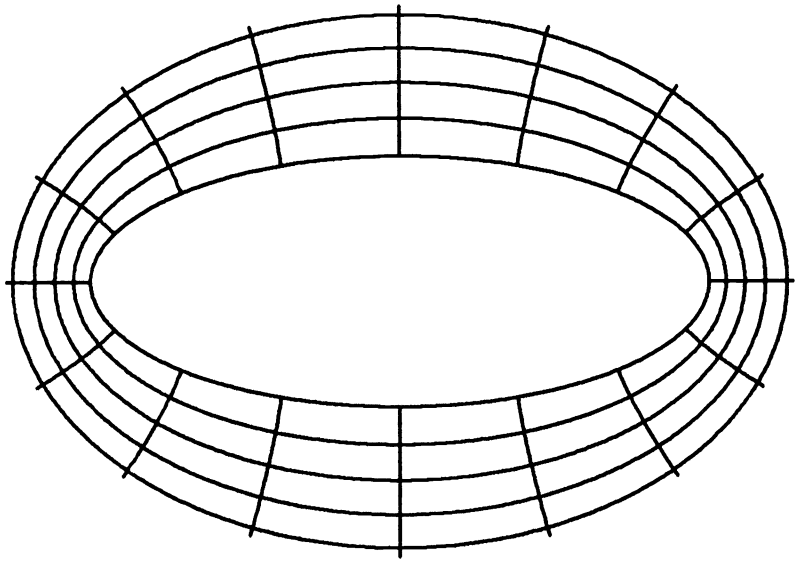


FIG. 3(b). *Ellipse* $\alpha = .4$ *with Friberg* $N = 256$.

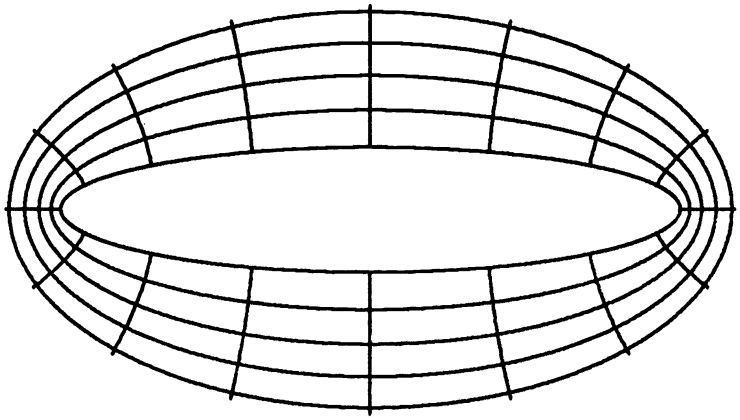


FIG. 3(c). *Ellipse* $\alpha = .2$ *with Wegmann* $N = 512$, *spline parameterization.*

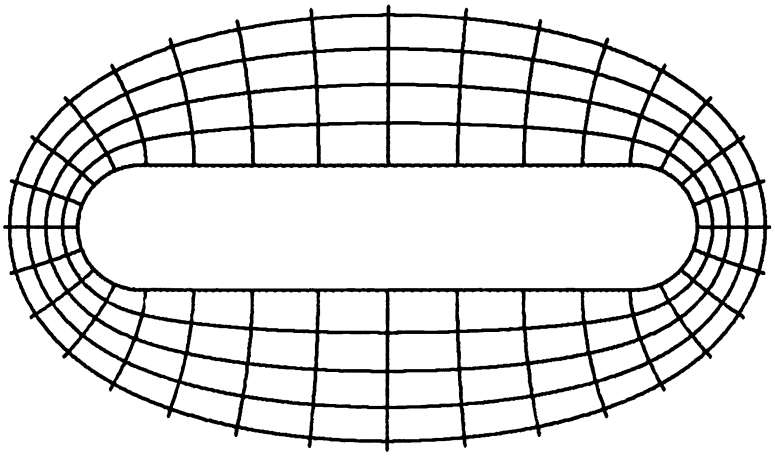


FIG. 4. *Sports ground* $\alpha = .2$ *with Timman* $N = 512$.

We see that the error committed in truncating the series is

$$(K - K_N)[\log \rho(\sigma(\theta))] = O\left(\frac{\beta^{N/4}}{N}\right)$$

in agreement with Gaier [4]. This is also in agreement with Wegmann [15, Ex. b, § 9]; there, bounds of order $R^{-N/2}$ are exhibited for the interior problem where $R > 1$ is the modulus of the nearest singularity or zero of the derivative. Here $R^2 = \beta < 1$.

Similarly for Timman's method,

$$\begin{aligned} h(e^{i\theta}) &= \log\left(g'(e^{i\theta})\right) \\ &= \log\left(\frac{1+\alpha}{2}\right) - \left(\beta\cos(2\theta) + \frac{\beta^2}{2}\cos(4\theta) + \cdots\right) \\ &\quad + i\left(\beta\sin(2\theta) + \frac{\beta^2}{2}\sin(4\theta) + \cdots\right). \end{aligned}$$

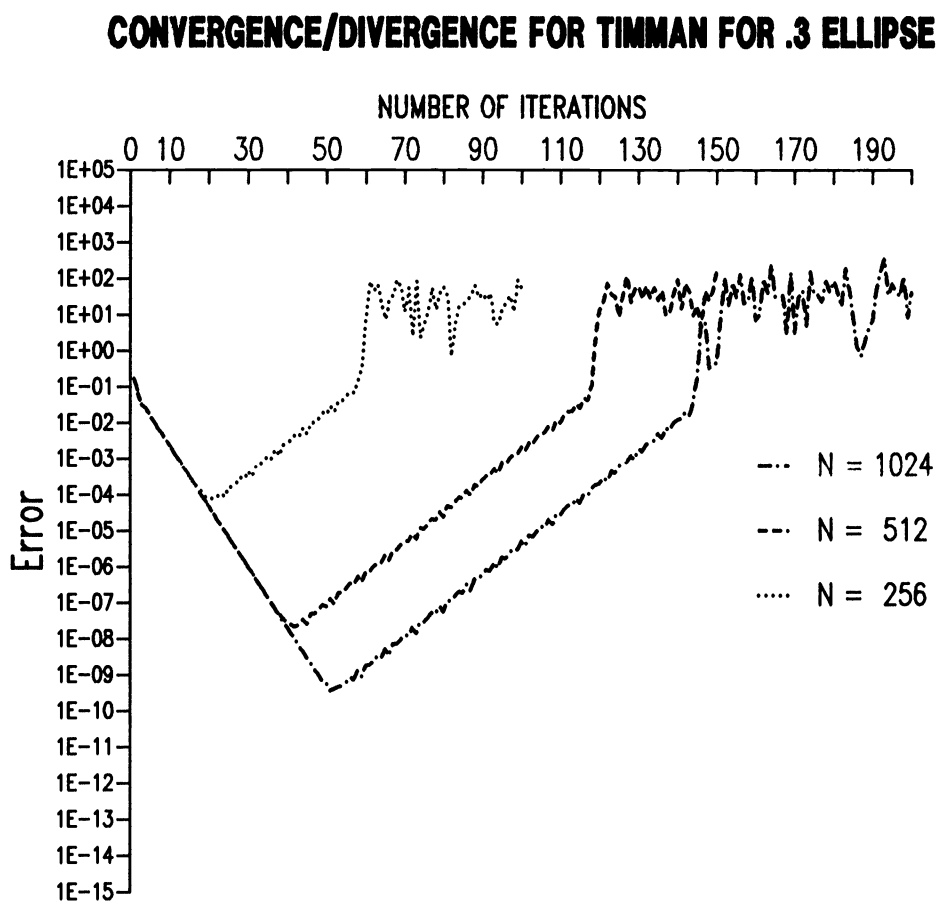


FIG. 5(a). *Convergence/divergence for Timman for .3 ellipse.*

As observed previously,

$$\log(\sigma'(\theta)) + \log|z'(\sigma(\theta))| = \operatorname{Re} h(e^{i\theta})$$

and

$$\arg(z'(\sigma(\theta))) - \theta - \frac{\pi}{2} = \operatorname{Im} h(e^{i\theta}),$$

so that the errors here exhibit the same order as above.

We also recall that the ε-condition in this example can be computed exactly, and $\varepsilon < 1$ if and only if $\alpha > \sqrt{2} - 1 = .4142\cdots$.

Our computations of the discretization error are given in Fig. 1. Gaier [4] shows that the aliasing error of the conjugated trigonometric interpolant is of the same order as the error in truncating the Fourier series of the conjugate function, as exhibited above. The errors shown here follow the trend predicted by the theory. The discretization error for Theodorsen follows Fig. 1 for $\alpha > .41\cdots$. The discretization errors for Timman, Friberg, and Wegmann for the ellipse are given in Figs. 2(a), 2(b), and 2(c). The data there are for analytic parameterization for $\alpha = .8$, .4, and for $\alpha = .2$ a spline was used. Often the spline gave a smaller discretization error. Here for $\alpha = .2$, larger N was required for convergence in the analytic case. These errors follow Fig. 1 for

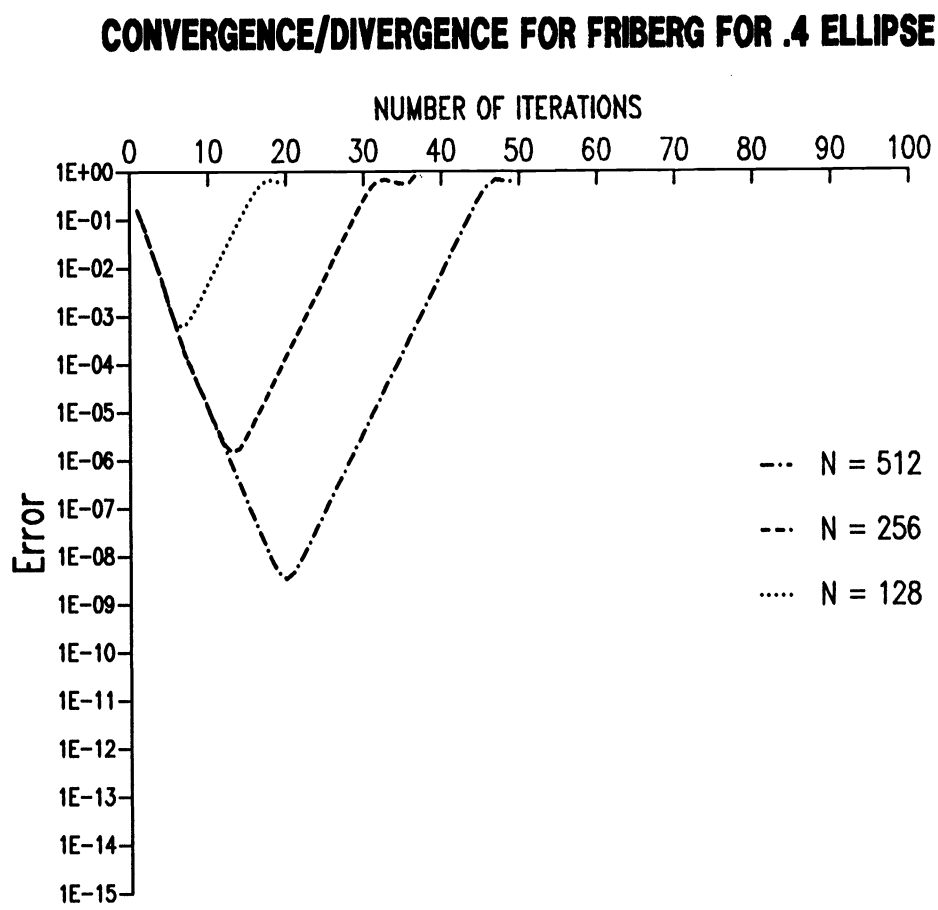


FIG. 5(b). *Convergence/divergence for Friberg for .4 ellipse.*

larger α, but degrade for smaller α (say $\alpha < .4$). Some computed regions are shown in Fig. 3.

(b) Sports ground. This region of length two is bounded by two semicircles of radius α connected by horizontal line segments. See Fig. 4. We use arclength as a parameter; $z'(\sigma)$ is not differentiable at the points connecting the semicircles and lines. Here $|z'(\sigma)| = 1$ for all σ, in contrast to example (a) where $|z'(\sigma)|$ varies between α and 1.

We have observed a convergence/divergence phenomenon (further addressed in [16]) in which errors decrease up to a certain point and then increase, for Timman, Friberg, and Wegmann. This occurs only in certain situations. For Timman it never occurs for a spline, never for the sports ground, and only for small α for an ellipse with analytic parameterization. For Friberg it occurs for an ellipse with small α for both the spline and the analytic parameterizations. For Wegmann it occurs to some extent in all cases.

These results indicate that the division by $|z'(\sigma(\theta))|$ in Timman or Wegmann, and also the multiplication by $|z'(\sigma(\theta))|$ in Friberg has some effect on the convergence behavior, and may trigger this phenomenon. It may also be associated with discretization error. It does not occur for Theodorsen where the discrete equations can be solved to machine accuracy for $\varepsilon < 1$. Wegmann [15, p. 457] noticed the start of this behavior for the interior problem and attributed it to roundoff error. Examples of

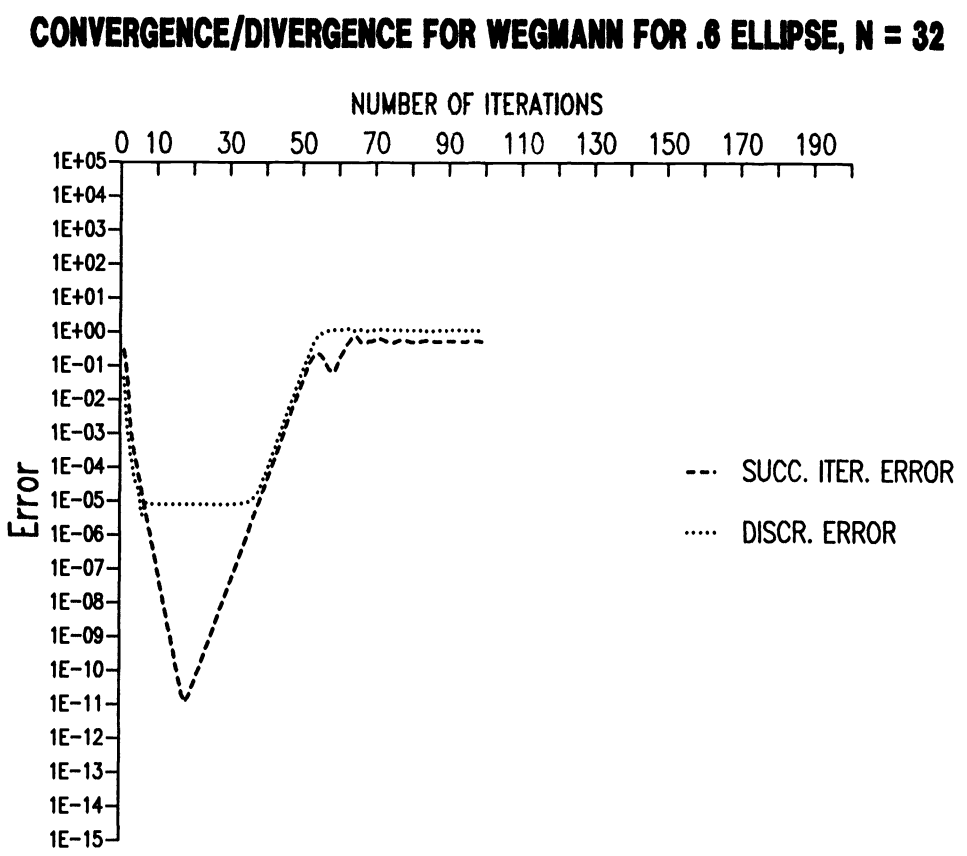


FIG. 5(c). *Convergence/divergence for Wegmann for .6 ellipse* $N = 32$.

convergence/divergence are shown in Fig. 5 for the ellipse; Figs. 5(a) and 5(b) give discretization error versus number of iterations for Timman and Friberg, respectively (error between successive iterates is roughly the same); Fig. 5(c) shows discretization error and that between successive iterates for Wegmann. It is interesting to observe that convergence rates are independent of N.

(c) Inverted ellipse. If we write $z(\sigma) = \rho(\sigma)e^{i\sigma}$, then

$$\rho(\sigma) = \sqrt{1-(1-\alpha^2)\cos^2(\sigma)}.$$

The ε-condition is the same as in (a). (See Figs. 7(a)-(c).)

The discretization error for Timman's method with this parameterization is given in Fig. 6(a). Friberg yields almost indistinguishable results. Using a spline gives results which are often an order of magnitude more accurate. The timings given in Fig. 6(a) clearly indicate that Friberg is faster than Timman for larger α. This confirms Kaiser's analysis [13], [7, p. 62], which says that the linear convergence rate for Friberg's method is nearly zero for nearly circular regions. We did not succeed in getting converged solutions for α smaller than .5 with Friberg or Timman.

Figure 6(b) gives the discretization error for Wegmann's method. The results for Theodorsen's method were nearly the same, except that Theodorsen's method failed to converge for $\alpha < \sqrt{2}-1 \approx .41$ ($\varepsilon > 1$). The use of a spline fit produced nearly identical results down to the level of accuracy of the spline. Some timings are again indicated as above.

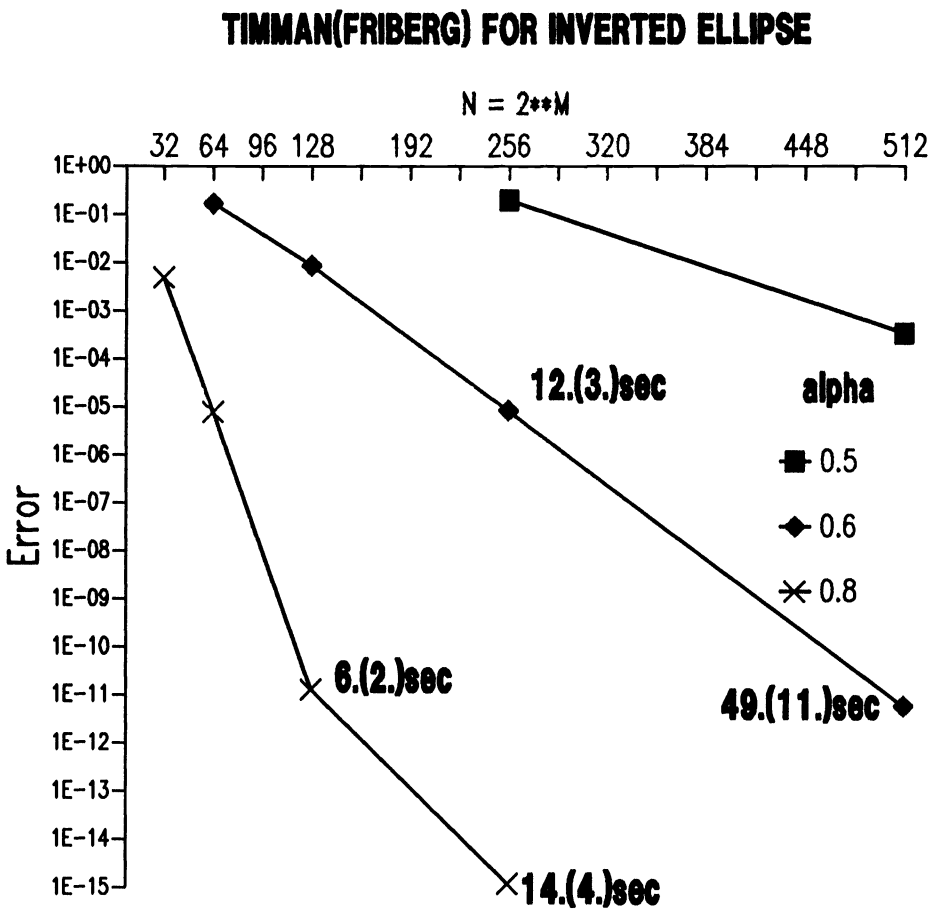


FIG. 6(a). *Timman (Friberg) for inverted ellipse.*

Wegmann's method converged for $N = 2{,}048$, $\alpha = .3$, also (see Fig. 7(c)). It is clearly seen to be the best of the four methods here both in timing and in range of α. Since it converges quadratically Wegmann's method usually only required three or four iterations to reach minimum dicretization error. For $\alpha = .6$, $N = 512$, for instance, Theodorsen required 49 iterations. See [14] for similar observations.

Note that the exterior problem for the inverted ellipse is, like the interior problem for the ellipse, a highly ill-conditioned problem as $\alpha \downarrow 0$. (The inverse map exhibits severe crowding; see [6] and [17].) We may expect such difficulty for any method which has an elongation or fingering of a portion of the boundary between the canonical and the target regions. Examples (a) and (b) are easier in this sense, i.e., maps to the interior of slender regions are harder than maps to the exterior of slender regions.

(d) cosine airfoil. We use the following curve to illustrate the map to the exterior of a curve with a corner:

$$z(\tau) = -\cos(k\tau)\, e^{i\tau}, \qquad \frac{-\pi}{2k} \leqq \tau \leqq \frac{\pi}{2k}.$$

The exterior angle $2\pi - \lambda$ is removed by the Karman-Trefftz transformation

$$\frac{\zeta - \zeta_1}{\zeta - \zeta_2} = \left(\frac{z - z_1}{z - z_2}\right)^{\delta}$$

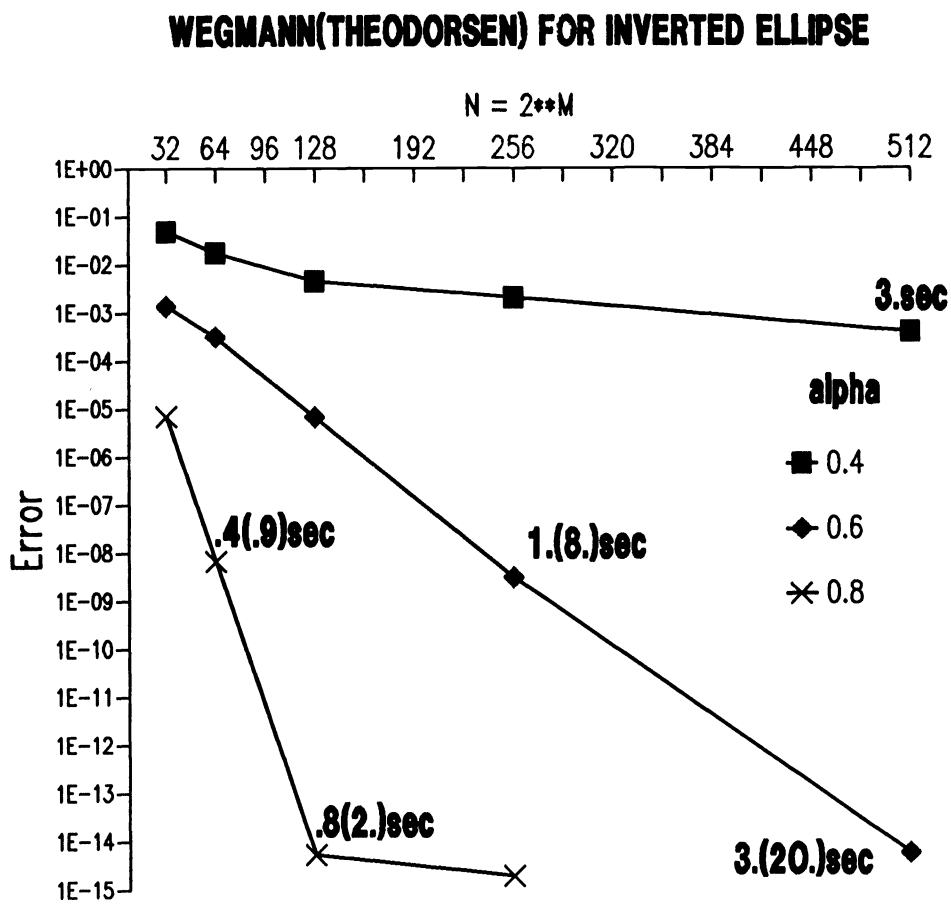


FIG. 6(b). *Wegmann (Theodorsen) for inverted ellipse.*

where z_1, z_2 map to ζ_1, ζ_2, and

$$\delta = \frac{\pi}{2\pi - \lambda}.$$

We map $NN = 1{,}000$ points to the ζ-plane and interpolate with a periodic cubic spline; this yields a smooth curve which is more nearly circular for a good choice of z_2. (This curve is not parameterized by argument, so we omitted Theodorsen's method.) Timman, Friberg, and Wegmann give the map to the region in the ζ-plane. This is composed with the inverted Karman-Trefftz map. Below $z_1 = \zeta_1 = 0$, $z_2 = \zeta_2 < 0$ for Timman and Wegmann. (Since Friberg requires the curve to surround the origin we translated the spline to the right by $-\zeta_2/2$ and then back by $-g(1)$ after the iteration.) Plots of the image in the z and ζ planes are shown in Fig. 8.

Figures 8(a)-8(d) show the maps for two cosine curves and their images under the Karman-Trefftz transformation. Figure 9 shows the discretization error in the plane of the cosine curve plotted against M for Wegmann with $z_2 = -.9$ and $k = 2$ (Figs. 8(a) and (b)). An error of 10^{-9} was reached in successive iterates of the map to the spline curve. The discretization errors for Timman and Friberg were similar but somewhat erratic; however, the errors in the successive iterates could be decreased to 10^{-15}. Timings for $N = 128$ were: Wegmann, 1.3 sec; Friberg, 1.8 sec; and Timman, 4.0 sec.

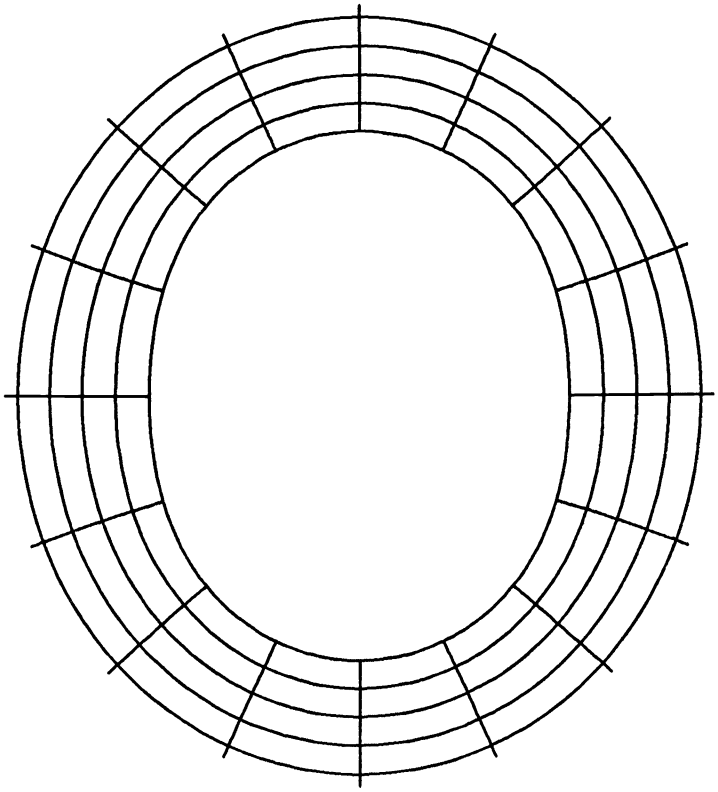


FIG. 7(a). *Inverted ellipse* $\alpha = .8$ *with Timman* $N = 64$.

(e) Perturbed circle. Here $z(\sigma) = (1 + .03 \sin(16\sigma))\, e^{i\sigma}$. The methods of Theodorsen and Friberg worked best here converging to machine precision in only two iterations (see Fig. 10(a)). For $N = 128$ Wegmann reached an error of 10^{-11} in the successive iterates. For $N = 64$ Timman's method converged to machine precision to a map with loops exhibited in Fig. 10(b). In this case the approximate map must have zeros of its derivative just outside the unit circle. For $N = 256$ Timman converged to the correct map. Similar loops are shown in [6] for Theodorsen's method for maps to the interior of a thin ellipse and to regions with corners.

(f) Spline curves. Here our boundary curves were generated by selecting a few points in the plane and interpolating them with our periodic cubic spline routine. For the region in Fig. 11(a), Timman, Friberg, and Wegmann all converged. Machine precision was achieved in the successive iteration error for Timman and Friberg and most N tried. Wegmann achieved the successive iteration errors given in Fig. 11(c) in a few iterations and then diverged. This convergence/divergence behavior may be somewhat disadvantageous in practice without a good procedure to stop the iterations. However, Wegmann again is more robust. It converged for Fig. 11(b) where Timman and Friberg failed. The effect of the curve geometry is evident again: as in the comparison of inverted ellipse and the ellipse, the fingering in Fig. 11(b) causes more inaccuracy than the slenderness in Fig. 11(a). Note also that the errors in Fig. 11(c) are roughly $O(N^{-3})$ as expected with cubic spline boundary, $z(\sigma) \in W^{3,\infty}$. Sample timings for Fig. 11(b) are 3.7 sec for $N = 1{,}024$ and 7.5 sec for $N = 2{,}048$. Thus

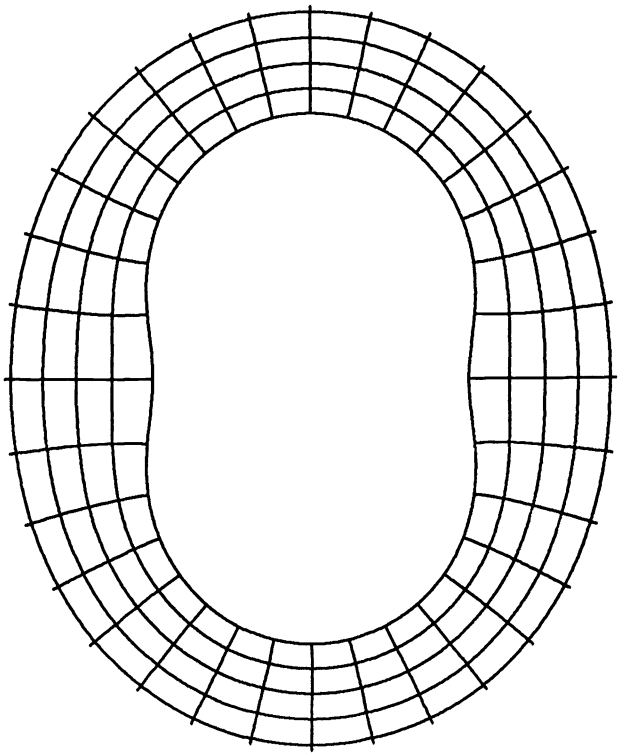


FIG. 7(b). *Inverted ellipse* $\alpha = .6$ *with Friberg* $N = 512$.

Wegmann's method seems quite promising in this practical case. In future work we hope to report on computations where the spline curve is generated by explicit maps, such as the osculation maps in [10].

4. Conclusion. In comparing the methods of Wegmann, Timman, Friberg, and Theodorsen our general conclusion is that Wegmann is the most efficient and most robust. Friberg is superior to Timman for nearly circular domains as the theory suggests; for extreme regions Timman is preferable if the arclength is used as the parameter.

We have observed a curious convergence/divergence phenomenon. For Timman it can be correlated with the use of a general parameterization; it did not occur for Timman when the arclength was used. It may also be associated with the relation between the discrete and continuous problems; it does not occur for Theodorsen with $\varepsilon < 1$ where the discrete equation can be solved to machine precision.

A careful analysis has been made of discretization error; for extreme regions the error is worse than the theory indicates for Timman, Friberg, or Wegmann. If a remedy for this could be found, the range of usefulness of Fourier methods might be extended significantly.

Acknowledgments. We thank Professor Mike Papadakis of our Aeronautical Engineering Department for assistance with the graphics and one of the referees for suggesting some corrections and improvements.

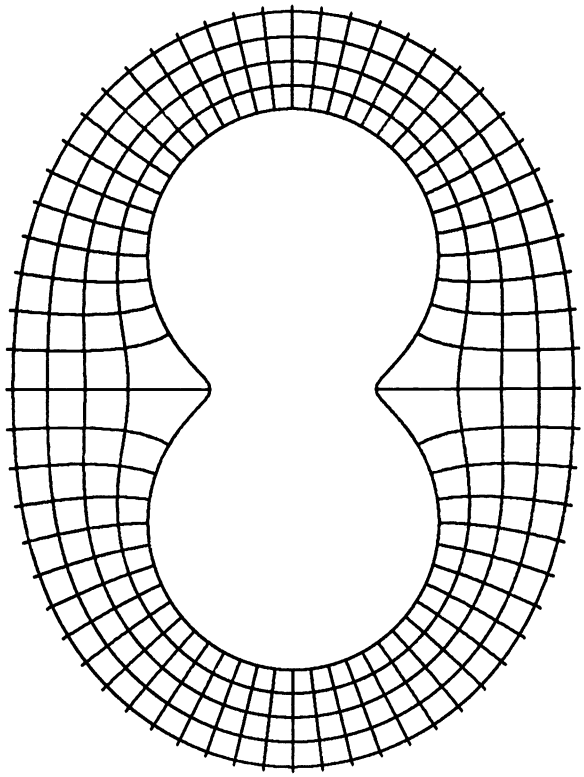


FIG. 7(c). *Inverted ellipse* $\alpha = .3$ *with Wegmann* $N = 2{,}048$.

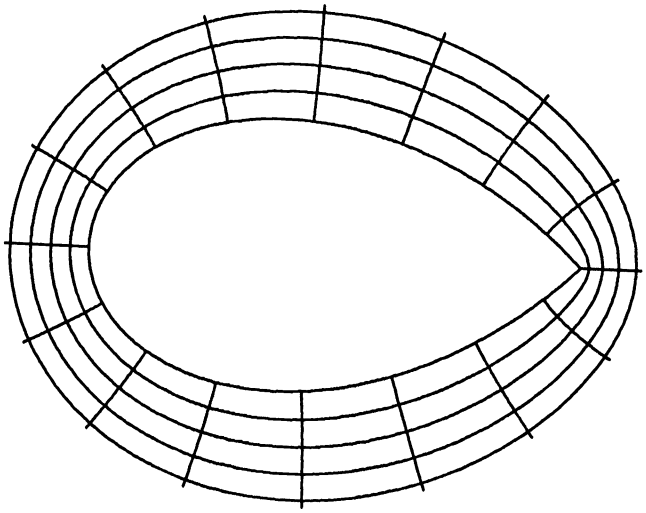


FIG. 8(a). *Cosine curve* $k = 2$ *with Wegmann* $N = 128$.

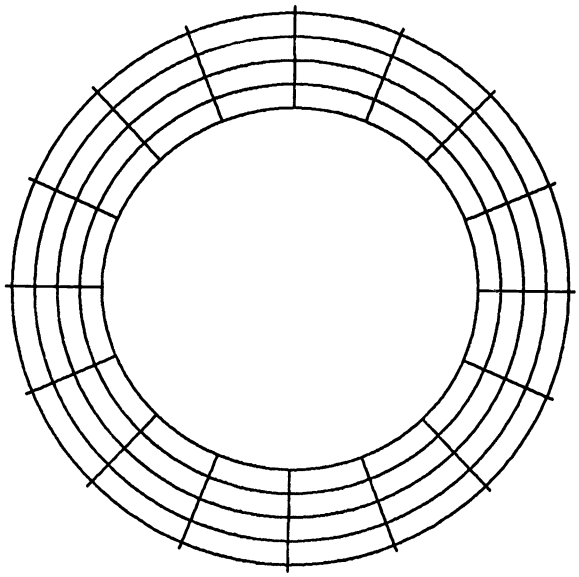


FIG. 8(b). *Image of Fig.* 8(a) *under Karman-Trefftz with* $z_2 = -.9$.

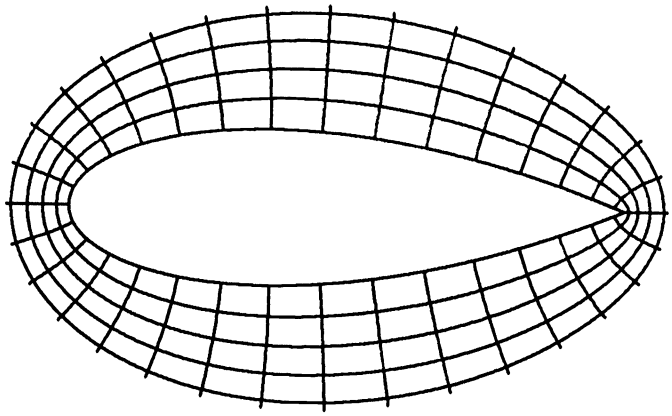


FIG. 8(c). *Cosine curve* $k = 4$ *with Timman* $N = 512$.

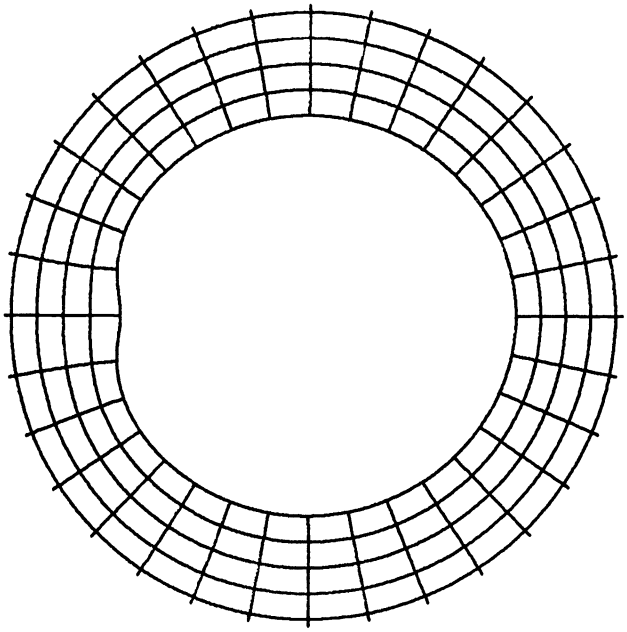


FIG. 8(d). *Image of Fig.* 8(c) *under Karman-Treffitz with* $z_2 = -.99$.

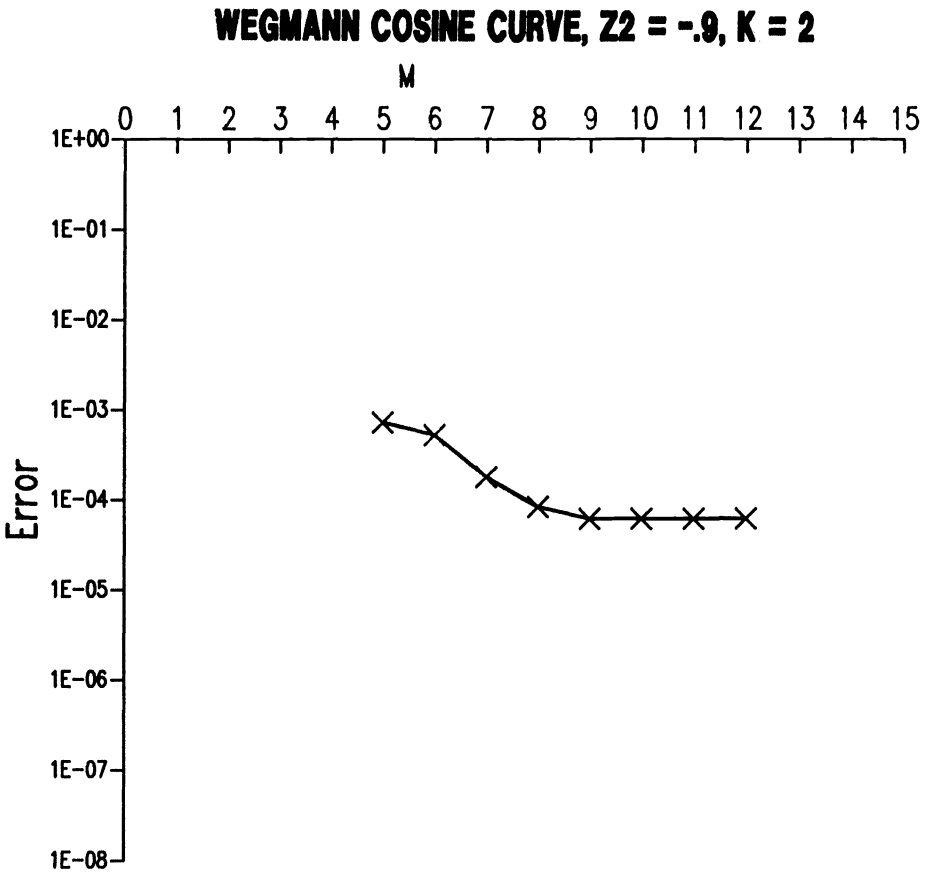


FIG. 9. *Discretization error for Wegmann for cosine curve,* $z_2 = -.9$, $k = 2$.

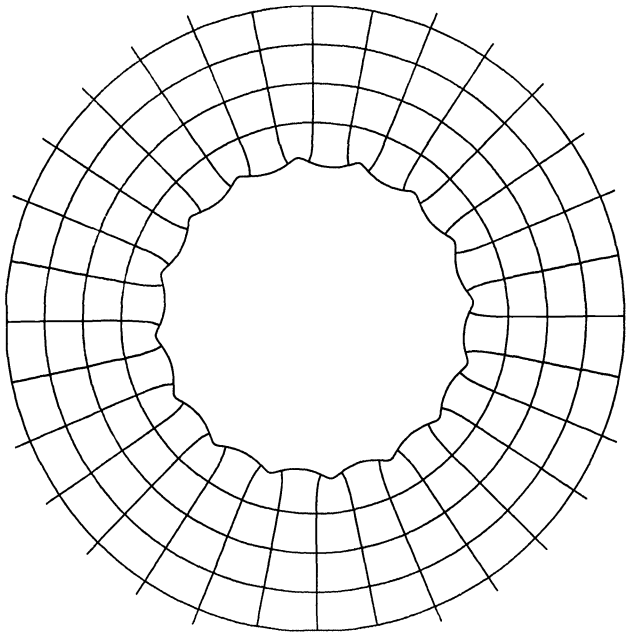


FIG. 10(a). *Perturbed circle with Theodorsen* $N = 64$.

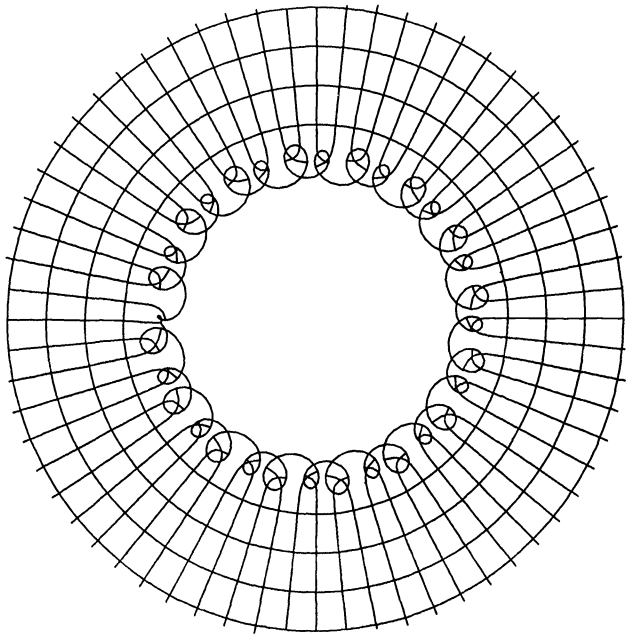


FIG. 10(b). *Loops produced by perturbed circle with Timman* $N = 64$.

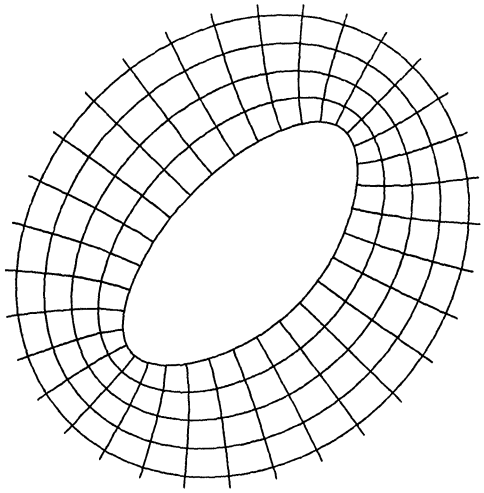


FIG. 11(a). *Spline curve with Friberg* $N = 128$.

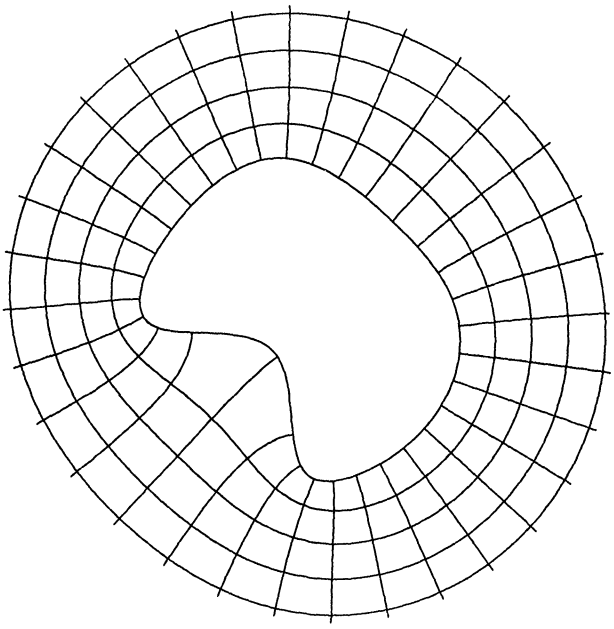


FIG. 11(b). *Spline curve with Wegmann* $N = 1{,}024$.

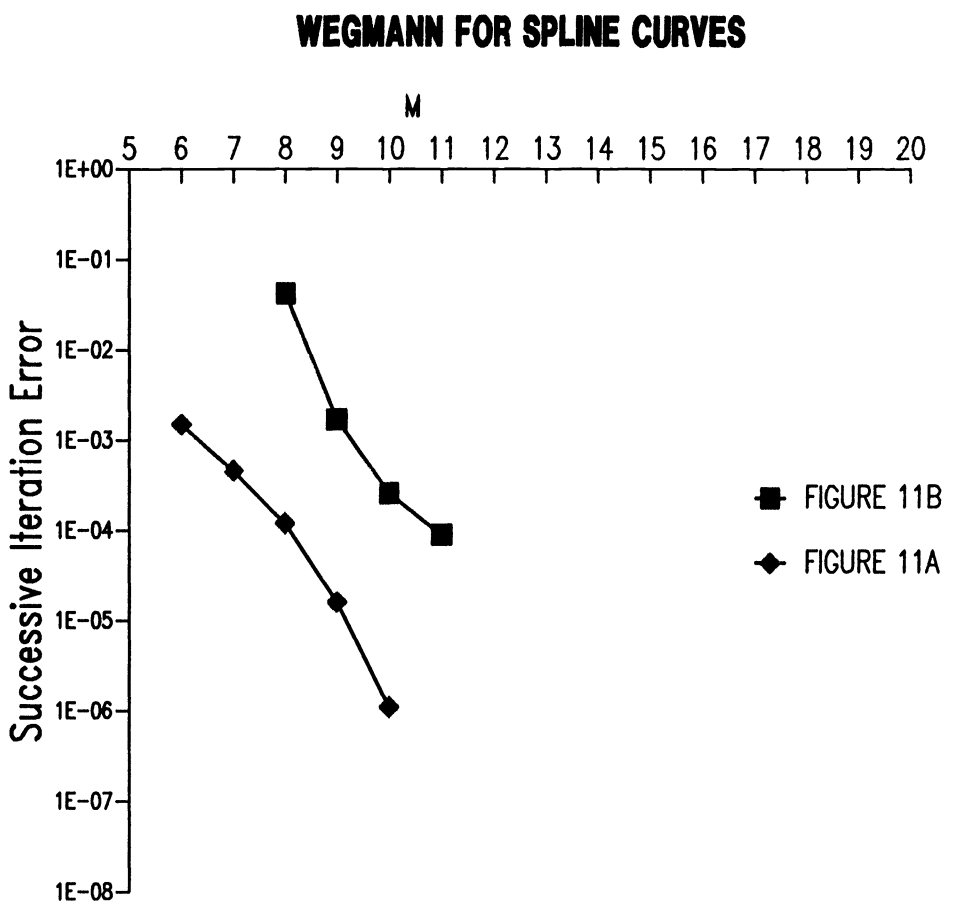


FIG. 11(c). *Successive iteration error for Wegmann for spline curves in Figs.* 11(a) (◆) *and* 11(b) (■).

REFERENCES

[1] F. BAUER, P. GARABEDIAN, D. KORN, AND A. JAMESON, *Supercritical Wing Sections* II, Lecture Notes in Econ. Math. Syst., Vol. 108, Springer-Verlag, New York, 1975.

[2] G. DAHLQUIST AND A. BJÖRCK, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[3] D. GAIER, *Konstruktive Methoden der konformen Abbildung*, Springer-Verlag, Berlin, 1964.

[4] ———, *Ableitungsfreie Abschätzuugen bei trigonometrischer Interpolation und Konjugiertenbestimmung*, Computing, 12 (1974), pp. 145-148.

[5] M. H. GUTKNECHT, *Fast algorithms for the conjugate periodic function*, Computing, 22 (1979), pp. 79-91.

[6] ———, *Numerical experiments on solving Theodorsen's integral equation for conformal maps with the* FFT *and various nonlinear iterative methods*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 1-30.

[7] ———, *Numerical conformal mapping methods based on function conjugation*, in Numerical Conformal Mapping, L. N. Trefethen, ed., North-Holland, Amsterdam, 1986, pp. 31-78.

[8] N. D. HALSEY, *Potential flow analysis of multielement airfoils using conformal mapping*, AIAA J., 17 (1979), pp. 1281-1288.

[9] ———, *Comparison of convergence characteristics of two conformal mapping methods*, AIAA J., 20 (1982), pp. 724-726.

[10] P. HENRICI, *Applied and Computational Complex Analysis, Vol.* III, John Wiley, New York, 1986.

[11] W. D. HOSKINS AND P. R. KING, *Periodic cubic spline interpolation using parametric splines*, Algorithm 73, Computer J., 15 (1972), pp. 282-283.

[12] D. C. IVES, *Conformal grid generation*, in Numerical Grid Generation, J. F. Thompson, ed., North-Holland, Amsterdam, 1982, pp. 107-135.

[13] A. KAISER, *L^∞-Konvergenz verschiedener Verfahren der sukzessiven Konjugation zur Berechnung konformer Abbildungen*, Ph.D. thesis, Mathematics Department, ETH, Zurich, Switzerland, 1986.

[14] P. LUCHINI AND F. MANZO, *Flow around simply and multiply connected bodies: a new iterative scheme for conformal mapping*, AIAA J., 27 (1989), pp. 345–351.

[15] R. WEGMANN, *Convergence proofs and error estimates for an iterative method for conformal mapping*, Numer. Math., 44 (1984), pp. 435–461.

[16] ———, *Discretized versions of Newton type iterative methods for conformal mapping*, J. Comput. Appl. Math., 29 (1990), pp. 207–244.

[17] C. ZEMACH, *A conformal map formula for difficult cases*, in Numerical Conformal Mapping, L. N. Trefethen, ed., North-Holland, Amsterdam, 1986, pp. 207–216.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 423–442, March 1991

ALGORITHMS FOR SPECIAL TRIDIAGONAL SYSTEMS*

RONALD F. BOISVERT†

Abstract. Algorithms for the solution of symmetric diagonally dominant tridiagonal systems of linear equations with constant diagonals are considered. Such systems occur, for example, when solving certain constant-coefficient elliptic partial differential equations by the Fourier method. In particular, the specialized *LU* factorization method of Malcolm and Palmer (SpLU), the cyclic reduction method of Hockney (CR), and the reversed triangular factorization method of Evans (RTF) are considered. An interesting property of the first two algorithms is that they may be terminated early for highly diagonally dominant systems. A new implementation of RTF that also has this property is presented, significantly reducing its operation count in many cases. The slightly perturbed systems that arise from problems with Neumann or periodic boundary conditions are also considered, with extensions given for SpLU to the periodic case and RTF to the Neumann case. Floating-point operation counts are given for each method, and the results of experiments on a single scalar processor are reported.

Key words. diagonally dominant matrices, Fourier method, *LU* factorization, Neumann boundary conditions, periodic boundary conditions, Poisson equation, reversed triangular factorization, symmetric matrices, Toeplitz matrices, tridiagonal matrices

AMS(MOS) subject classifications. 65F05, 65N20

1. Introduction.

Consider the $n \times n$ system of linear equations

$$T\mathbf{x} = \mathbf{g}, \tag{1}$$

where

$$T = \begin{bmatrix} a/\mu & b & & & & & \rho b \\ b & a & b & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & b & a & b \\ \rho b & & & & & b & a/\nu \end{bmatrix},$$

$b \neq 0$, $|a/b| > 2$, ρ is either zero or one, μ and ν are either one or two, and $\rho = 1$ implies $\mu = \nu = 1$. The coefficient matrix T has a very special form: it is tridiagonal (for $\rho = 0$), symmetric, diagonally dominant, and a Toeplitz matrix (for $\mu = \nu = 1$, i.e., has constant diagonals). The need to solve such systems arises, for example, when using finite difference methods to solve linear constant-coefficient boundary-value problems in various contexts [9], [13], [17]. In these cases, the constants μ, ν, and ρ are determined from the boundary conditions along opposite sides of the domain; see Table 1.

Our interest in these methods arises from the Fourier–tridiagonal method for the modified Helmholtz equation [2]. In the case of an $m \times n$ grid, for example, after preprocessing with fast Fourier transforms (FFTs), one obtains a set of m distinct systems of the form (1). One promising method of solving such systems on vector processors is to apply a serial algorithm to the m systems in parallel. This leads

* Received by the editors June 27, 1988; accepted for publication (in revised form) February 5, 1990. This article is a contribution of the National Institute of Standards and Technology, and is therefore not subject to copyright in the United States.

† Center for Computing and Applied Mathematics, National Institute of Standards and Technology, Gaithersburg, Maryland 20899.

TABLE 1
Relationship of μ, ν, and ρ to boundary conditions.

Boundary Conditions	μ	ν	ρ
Dirichlet–Dirichlet	1	1	0
Dirichlet–Neumann	1	2	0
Neumann–Dirichlet	2	1	0
Neumann–Neumann	2	2	0
Periodic	1	1	1

TABLE 2
Floating-point operation counts for Dirichlet problems.

Method	Operations	Reference
Malcolm–Palmer	$5n + 2k$	[15]
Cyclic reduction	$7n$	[19]
Special LU factorization	$7n$	[19]
Gaussian elimination	$8n$	[3]
Reversed triangular factorization	$9n$	[4]
Toeplitz factorization	$9n$	[8]

naturally to the question: what is the best serial algorithm? In what follows we will be concerned with the performance of algorithms on a single scalar processor without further reference to vector or multiprocessor considerations.

A number of methods have been proposed for solving such systems in the Dirichlet case (most admit variants for other boundary conditions): Toeplitz factorization [1], [7], [8], [16]; cyclic reduction [12], [17], [19]; reversed triangular factorization [4], [5], [6]; and specialized variants of Gaussian elimination [15], [19]. Each of these requires $O(n)$ work. Such systems may also be solved using Fourier transforms, but this leads to an $O(n \log n)$ algorithm. Table 2 gives operation counts which have been cited for these methods.

In Toeplitz factorization methods one seeks an LU factorization of T in which L is a lower bidiagonal Toeplitz matrix and U is an upper bidiagonal Toeplitz matrix. This factorization does not exist in general, but suitable L and U may be found such that LU differs from T in only the upper left corner element. The Sherman–Morrison formula [10] may then be applied to adjust the solution obtained from these LU factors. Factorizations of the form $T = QQ^T$, where Q is an $n \times (n+1)$ bidiagonal Toeplitz matrix, also exist for suitably scaled T. The reversed triangular factorization technique is based upon the application of such factorizations. Both of these methods have the advantage that they require only $O(1)$ divisions and no additional vector storage.

The basis for cyclic reduction is the observation of Golub and Hockney [12] that, when $n+1$ is a power of 2, one can eliminate all the odd-numbered unknowns in the even-numbered equations in a suitably scaled version of (1) by applying simple row operations. The resulting system for the even-numbered unknowns is of the same form as the original, and hence the process may be applied recursively until there are only a few unknowns left, in which case Gaussian elimination may be trivially applied. The remaining unknowns are then determined by a back-substitution process.

Gaussian elimination without pivoting for a symmetric tridiagonal matrix (called "Gaussian elimination" in Table 2) can be improved upon by noting that a suitably scaled T has a special LU factorization, where U has ones on the off-diagonal, L has

ones on the diagonal, and the other elements of L and U are simply related. The method of Malcolm and Palmer is based upon this factorization.

It is interesting to note that if $|a/b| \gg 2$, i.e., the diagonal dominance of T is very strong, additional economies may be introduced in these methods when they are implemented in finite precision arithmetic. The method of Malcolm and Palmer relies upon the fact that the diagonal elements of U form a sequence which quickly (in k steps) becomes constant. This allows the factorization to be terminated early. Another form of early convergence has been observed for cyclic reduction [11], [12]. Here the diagonals of the successive tridiagonal systems are roughly squared in magnitude at each stage; when the ratio of off-diagonal to diagonal elements is less than the machine precision, then the system can be considered diagonal and back substitution can be started immediately. In this paper we show how similar economies may be introduced into the reversed triangular factorization method.

We begin by stating some pertinent facts about cyclic reduction. We then describe the method of Malcolm and Palmer in some detail, and give an extension to the periodic case. We then present new variants of the reversed triangular factorization method. Finally, we compare each of these techniques and present the results of computational experiments.

2. Cyclic reduction. In this section we summarize some of the properties of the cyclic reduction method as applied to the system 1 in order to facilitate comparison with the methods described in the latter sections.

We first consider the Dirichlet case. Scaling each equation in 1 by $1/b$ we arrive at a tridiagonal system with $\alpha \equiv a/b$ ($\alpha \geq 2$) on the diagonal and ones on the off-diagonals. For simplicity we first assume that $n = m - 1$, where $m = 2^k$. The first step in cyclic reduction is to multiply equations $2, 4, \cdots, m-2$ by $-\alpha$ and add the two adjacent equations to it. This decouples the even-numbered equations from the odd ones, yielding a reduced system of order $m/2 - 1$. Once the reduced system is solved the odd-numbered equations may be used to determine the odd-numbered unknowns by back substitution. The reduced equations have the same form as the original ones, except with diagonal elements $2 - \alpha^2$. Thus, the reduction process may be applied recursively until a system of order 1 remains. All unknowns may then be determined by applying back substitution recursively.

The algorithm may be summarized as follows.

$$\alpha_{j+1} \leftarrow 2 - \alpha_j^2, \quad j = 1(1)k-1$$
$$g_i \leftarrow g_{i-2^{j-1}} - \alpha_j g_i + g_{i+2^{j-1}}, \quad i = 2^j(2^j)m - 2^j, \quad j = 1(1)k-1$$
$$g_{m/2} \leftarrow g_{m/2}/\alpha_k$$
$$g_i \leftarrow (g_i - g_{i-2^{j-1}} - g_{i+2^{j-1}})/\alpha_j, \quad i = 2^{j-1}(2^j)m - 2^{j-1}, \quad j = k-1(-1)1.$$

(Here we have taken $\alpha_1 = \alpha$ and $g_0 = g_m = 0$.) It is straightforward to show that this algorithm requires $7n$ floating-point operations (including the n operations required to initially scale the system). The cases of 1 with $\mu = 2$ and/or $\nu = 2$ may be handled similarly so that the operation counts remain the same in these cases.

We need not have that m be a power of 2 in order to apply cyclic reduction. A simple modification is necessary to generalize this algorithm to any n, requiring only that we allow the diagonal element in the last equation to differ from the others at each step of the reduction. The asymptotic operation count of $7n$ is unchanged. If we also allow the diagonal element in the first equation to differ, we may generalize further to the case of equation 1 with $\rho = 0$. We state this algorithm in detail

below. Here, n_j is the number of equations left after the jth reduction step, $\alpha_j^{(1)}$ is the diagonal element in the first equation before the jth reduction step, and finally $\alpha_j^{(n)}$ is the diagonal element in the last equation before the jth reduction step.

ALGORITHM 2.1. **Cyclic Reduction.**

0. *Preliminaries*

$$
\begin{aligned}
k &\leftarrow \lfloor \log_2 n \rfloor \\
\alpha_1 &\leftarrow \alpha \\
\alpha_{j+1} &\leftarrow 2 - \alpha_j^2, \quad j = 1, 2, \cdots, k-1 \\
\alpha_1^{(1)} &\leftarrow \alpha/\mu \\
\alpha_1^{(n)} &\leftarrow \alpha/\nu \\
n_0 &\leftarrow n \\
n_j &\leftarrow \lfloor n_{j-1} \rfloor, \quad j = 1, 2, \cdots, k-1
\end{aligned}
$$

1. *Rescale*

$$g_i \leftarrow g_i/b, \quad i = 1, 2, \cdots, n$$

2. *Reduction*

For $j = 1, \cdots, k-1$ *do:*

$$
\begin{aligned}
g_{2^j} &\leftarrow \alpha_j (g_{2^j - 2^{j-1}}/\alpha_j^{(1)} - g_{2^j}) + g_{2^j + 2^{j-1}} \\
\alpha_{j+1}^{(1)} &\leftarrow 1 + \alpha_j/\alpha_j^{(1)} - \alpha_j^2 \\
g_i &\leftarrow g_{i-2^{j-1}} - \alpha_j g_i + g_{i+2^{j-1}}, \quad i = 2 \cdot 2^j, 3 \cdot 2^j, \cdots, (n_j - 1)2^j \\
i &\leftarrow n_j 2^j \\
g_i &\leftarrow \begin{cases} g_{i-2^{j-1}} - \alpha_j g_i + \alpha_j g_{i+2^{j-1}}/\alpha_j^{(n)}, & \text{if } i + 2^{j-1} \le n \\ g_{i-2^{j-1}} - \alpha_j g_i, & \text{otherwise} \end{cases} \\
\alpha_{j+1}^{(n)} &\leftarrow \begin{cases} 1 - \alpha_j^2 + \alpha_j/\alpha_j^{(n)}, & \text{if } i + 2^{j-1} \le n \\ 1 - \alpha_j \alpha_j^{(n)}, & \text{otherwise} \end{cases}
\end{aligned}
$$

3. *Explicit solve*

If $n_{k-1} = 3$ *then:* $(3 \times 3$ *case)*

$$
\begin{aligned}
g_{2^k} &\leftarrow (g_{2^k} - g_{2^k - 2^{k-1}}/\alpha_k^{(1)} - g_{2^k + 2^{k-1}}/\alpha_k^{(n)})/(\alpha_k - 1/\alpha_k^{(1)} - 1/\alpha_k^{(n)}) \\
g_{2^k - 2^{k-1}} &\leftarrow (g_{2^k - 2^{k-1}} - g_{2^k})/\alpha_k^{(1)} \\
g_{2^k + 2^{k-1}} &\leftarrow (g_{2^k + 2^{k-1}} - g_{2^k})/\alpha_k^{(n)}
\end{aligned}
$$

Else: $(2 \times 2$ *case)*

$$
\begin{aligned}
g_{2^k} &\leftarrow (g_{2^k} - g_{2^k - 2^{k-1}}/\alpha_k^{(1)})/(\alpha_k^{(n)} - 1/\alpha_k^{(1)}) \\
g_{2^k - 2^{k-1}} &\leftarrow (g_{2^k - 2^{k-1}} - g_{2^k})/\alpha_k^{(1)}
\end{aligned}
$$

Endif

4. *Back substitution*

For $j = k-1, k-2, \cdots, 1$ *do:*

$$
\begin{aligned}
g_{2^{j-1}} &\leftarrow (g_{2^{j-1}} - g_{2^j})/\alpha_j^{(1)} \\
l &\leftarrow 2^{j-1} + (\lfloor (n_j + 1)/2 \rfloor - 2)2^j \\
g_i &\leftarrow (g_i - g_{i-2^{j-1}} - g_{i+2^{j-1}})/\alpha_j, \quad i = 2^{j-1} + 2^j, 2^{j-1} + 2 \cdot 2^j, \cdots, l \\
i &\leftarrow \max(2^{j-1}, l) + 2^j \\
g_i &\leftarrow \begin{cases} (g_i - g_{i-2^{j-1}} - g_{i+2^{j-1}})/\alpha_j, & \text{if } i + 2^{j-1} \le n \\ (g_i - g_{i-2^{j-1}})/\alpha_j^{(n)}, & \text{otherwise.} \end{cases}
\end{aligned}
$$

For $|\alpha| \gg 2$ we see that $|\alpha_j| \approx \alpha^{2^{j-1}}$, and hence the algorithm, as stated, presents the danger of floating-point overflow. However, as pointed out by Hockney [12], this can be turned to our advantage. If $|\alpha_j| > 1/\epsilon$, where ϵ is the machine epsilon, then the reduced system is diagonal to machine precision. At this point, the reduced system may be solved with $m/2^j$ multiplications and back substitution may be begun immediately. If the algorithm is stopped at step $j = k_c$, then the operation count is

TABLE 3
Effect of α on cyclic reduction operation count for $\epsilon = 2^{1-t}$.

	$t = 47$		$t = 23$	
α	k_c	Ops	k_c	Ops
2.001	10	$7.0n$	9	$7.0n$
2.010	9	$7.0n$	8	$7.0n$
2.100	7	$7.0n$	6	$6.9n$
2.500	6	$6.9n$	5	$6.8n$
4.000	5	$6.8n$	4	$6.7n$
10.000	4	$6.7n$	3	$6.4n$
50.000	4	$6.7n$	2	$5.8n$

reduced to $(7 - 5/2^{k_c})n$. Table 3 shows the values of k_c and the resulting operation count for various values of α and ϵ. From the table it is clear that significant savings are obtained only when α is so large that the reduction may stop after the second step. The reason for this is that, since the work halved at each step of the reduction, most of the work is done in the first few steps.

We next consider the periodic case. Scaling (1) as before we obtain the same tridiagonal system modified only by the addition of ones in the (1,n) and (n,1) positions. Here, we assume that $n = m - 1$, where $m = 2^k + 1$. The basic reduction process is exactly as before, except that we reduce the odd equations first and use the last equation to help reduce the first one. After $k - 1$ reduction steps we are left with a system of order 2. After this system is solved the remaining unknowns are again determined by back substitution.

The algorithm may be summarized as follows.

$$\begin{aligned}
\alpha_{j+1} &\leftarrow 2 - \alpha_j^2, \quad j = 1(1)k-1 \\
g_i &\leftarrow g_{i-2^{j-1}} - \alpha_j g_i + g_{i+2^{j-1}}, \quad i = 1(2^j)m - 2^j, \quad j = 1(1)k-1 \\
g_{1+2^{k-1}} &\leftarrow (\alpha_k g_{1+2^{k-1}} - 2g_1)/(\alpha_k^2 - 4) \\
g_1 &\leftarrow (g_1 - 2g_{1+2^{k-1}})/\alpha_k \\
g_i &\leftarrow (g_i - g_{i-2^{j-1}} - g_{i+2^{j-1}})/\alpha_j, \quad i = 1 + 2^{j-1}(2^j)m - 2^{j-1}, \\
&\qquad\qquad\qquad\qquad\qquad\qquad j = k - 1(-1)1.
\end{aligned}$$

(Here we have taken $\alpha_1 = \alpha$, $g_m = g_1$, and $g_{1-2^{j-1}} = g_{m-2^{j-1}}$.) Once again the algorithm is seen to require $7n$ floating-point operations. The diagonals of the successive systems are the same as in the Dirichlet case, and hence the same early stopping criterion may be applied, yielding an effective operation count of $(7 - 5/2^{k_c})n$.

The modification to this algorithm in the case where n is not a power of 2 does not appear to be as straightforward as the $\rho = 0$ case. In [18] Sweet presents a generalization analogous to that of the fast Fourier transform; unfortunately his generalization can show significant performance degradation when m has factors other than 2. We will not consider this case further.

In the next sections we will show how the diagonal dominance of (1) also leads to a reduction of operations in the special *LU* factorization and the reversed triangular factorization methods.

3. Special *LU* factorization. In [15], Malcolm and Palmer described how to reduce the number of operations of Gaussian elimination for symmetric diagonally dominant tridiagonal systems with constant diagonals. Their results also indicate how

Neumann-type boundary conditions can be handled. In this section we first outline their results, and then extend them to the case of periodic boundary conditions.

3.1. Dirichlet and Neumann cases. We first consider (1) in the case $\rho = 0$. After scaling by $1/b$ we arrive at the equivalent problem

$$T\mathbf{x} = \mathbf{g}, \tag{2}$$

where

$$T = \begin{bmatrix} \alpha/\mu & 1 & & & & & \\ 1 & \alpha & 1 & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & 1 & \alpha & 1 \\ & & & & & 1 & \alpha/\nu \end{bmatrix},$$

$\alpha = a/b$, with $|\alpha| > 2$. We may then write $T = LDL^T$, where

$$L = \begin{bmatrix} 1 & & & & & \\ l_1 & 1 & & & & \\ & \cdot & \cdot & & & \\ & & \cdot & \cdot & & \\ & & & l_{n-2} & 1 & \\ & & & & l_{n-1} & 1 \end{bmatrix}, \quad D = \begin{bmatrix} u_1 & & & & & \\ & u_2 & & & & \\ & & \cdot & & & \\ & & & \cdot & & \\ & & & & u_{n-1} & \\ & & & & & u_n \end{bmatrix}. \tag{3}$$

The u_i and l_i are defined by the recursion

$$\begin{aligned} u_1 &= \alpha/\mu \\ u_i &= \alpha - 1/u_{i-1}, \qquad i = 2, \cdots, n-1 \\ u_n &= \alpha/\nu - 1/u_{n-1} \\ l_i &= 1, \qquad i = 1, \cdots, n. \end{aligned}$$

After determining L and D the solution vector $\mathbf{x}$ may be found by solving the triangular systems $L\mathbf{y} = \mathbf{g}$ and $L^T\mathbf{x} = D^{-1}\mathbf{y}$ in turn. Note that the decomposition is uniquely represented by the n values l_i, $i = 1, 2, \cdots, n$.

In [15] it was observed that the sequence u_i converges to $u = [\alpha + \operatorname{sgn}(\alpha)(\alpha^2 - 4)^{1/2}]/2$ (provided $\mu \neq \alpha^2$ and $\mu \neq \alpha/v$, where $v = [\alpha - \operatorname{sgn}(\alpha)(\alpha^2 - 4)^{1/2}]/2$). Furthermore, it was shown that on a computer with precision[1] ϵ, the sequence u_i becomes constant for $i \geq k_u$, where

$$k_u = \left\lceil \frac{-\ln \epsilon}{2 \ln |u|} \right\rceil.$$

This leads to the following algorithm for the solution to (1) with $\rho = 0$ and $|\alpha| > 2$. (The solution overwrites the input vector $\mathbf{g}$.)

[1] A computer with t-digit, base β arithmetic has precision β^{1-t}.

TABLE 4
Values of the convergence index k_u for $\epsilon = 2^{1-t}$.

α	k_u	
	$t = 47$	$t = 24$
2.001	505	253
2.010	160	80
2.100	51	26
2.500	24	12
4.000	13	7
10.000	7	4
50.000	5	3

ALGORITHM 3.1. **Special *LU* Factorization.**

0. *Preliminaries*
   $l \leftarrow 2/[\alpha + \mathrm{sgn}(\alpha)(\alpha^2 - 4)^{1/2}]$
   $k \leftarrow \min(\lfloor \ln \epsilon / 2 \ln |l| \rfloor, n)$
1. *Rescale*
   $g_i \leftarrow g_i/b, \quad i = 1, 2, \cdots, n$
2. *Factorization*
   $l_1 \leftarrow \mu/\alpha$
   $l_i \leftarrow 1/[\alpha - l_{i-1}], \quad i = 2, 3, \cdots, \min(k, n-1)$
   $l_i \leftarrow l, \quad i = k+1, k+2, \cdots, n-1$
   $l_n \leftarrow 1/[\alpha/\nu - l_{n-1}]$
3. *Forward substitution*
   $g_i \leftarrow g_i - l_{i-1} g_{i-1}, \quad i = 2, 3, \cdots, n$
4. *Backward substitution*
   $g_n \leftarrow l_n g_n$
   $g_i \leftarrow l_i(g_i - g_{i+1}), \quad i = n-1, n-2, \cdots, 1$

Algorithm 3.1 requires $5n + 2\tilde{k}_u$ floating-point operations, where $\tilde{k}_u = \max(k_u, n)$, and a working storage of $\tilde{k}_u$ words. The ratio k_u/n becomes small quickly as $|\alpha|$ moves away from 2 (see Table 4), and hence the savings in using this algorithm can be significant in many cases. When $|\alpha|$ is near 2, or the early convergence of l_i is ignored, then $7n$ operations are required. In this case the algorithm is equivalent to Gaussian elimination with a straightforward modification to account for constant diagonals.

This algorithm also has certain advantages when the system (1) must be solved for many different right-hand side vectors **g**. In this case, step 2 (the factorization), may be eliminated in all but the first solve. Each additional solve requires $5n$ operations. This amounts to a significant savings for weakly diagonally dominant systems. In the strongly diagonally dominant case the factorization step is nearly eliminated anyway, and the savings in repeat solves is only $2\tilde{k}_u$ operations.

The degenerate cases with $|\alpha| = 2$ may be handled as follows:

- $\mu = \nu = 1$: Set $k = n$ and apply steps 1–4 of the algorithm described above.
- $\mu = 2$, $\nu = 1$: Here $l_i = \mathrm{sgn}(\alpha)$ for $i = 1, 2, \cdots, n-1$; thus we may set $k = 1$ and apply steps 1–4 of the algorithm above. (Multiplications may be saved by separately considering the cases $l = 1$ and $l = -1$.)
- $\mu = 1$, $\nu = 2$: We may reduce this to the case $\mu = 2$, $\nu = 1$ by reordering the equations and unknowns (i.e., by processing the g_i in reverse order).
- $\mu = \nu = 2$: The matrix is singular.

3.2. Periodic case. We next consider (1) in the case $\mu = \nu = \rho = 1$. After scaling by $1/b$ this yields the problem

$$\left[\begin{array}{c|c} T & \mathbf{e} \\ \hline \mathbf{e}^T & \alpha \end{array}\right] \left[\begin{array}{c} \mathbf{x} \\ \hline x_n \end{array}\right] = \left[\begin{array}{c} \mathbf{g} \\ \hline g_n \end{array}\right], \tag{4}$$

where T is the matrix of (2) with $\mu = \nu = 1$, and $\mathbf{e}$ is the vector

$$\mathbf{e}^T = [\ 1 \quad 0 \quad \cdot \quad \cdot \quad \cdot \quad 0 \quad 1\ ].$$

T is $(n-1) \times (n-1)$, and $\mathbf{e}$, $\mathbf{x}$, and $\mathbf{g}$ are of length $n-1$. Let L and D be matrices of the form (3) with $\mu = \nu = 1$. We then have the decomposition

$$\left[\begin{array}{c|c} T & \mathbf{e} \\ \hline \mathbf{e}^T & \alpha \end{array}\right] = \left[\begin{array}{c|c} L & 0 \\ \hline \mathbf{w}^T & 1 \end{array}\right] \left[\begin{array}{c|c} D & 0 \\ \hline 0 & u_n \end{array}\right] \left[\begin{array}{c|c} L^T & \mathbf{w} \\ \hline 0 & 1 \end{array}\right].$$

The relations which determine $\mathbf{w}$ and u_n are then

$$LD\mathbf{w} = \mathbf{e},$$
$$u_n + \mathbf{w}^T D\mathbf{w} = \alpha,$$

from which we get

$$\begin{array}{lcl} w_1 & = & l_1 \\ w_i & = & -w_{i-1} l_i, \quad i = 2, 3, \cdots, n-2 \\ w_{n-1} & = & (1 - w_{n-2}) l_{n-1} \end{array}$$

and

$$u_n = \alpha - \sum_{j=1}^{n-1} u_j w_j^2 = \alpha - \left[w_1 - \sum_{j=2}^{n-2} w_{j-1} w_j + (1 - w_{n-2}) w_{n-1} \right].$$

After determining L, D, and $\mathbf{w}$, the solution to (4) may be found by solving

$$L\mathbf{y} = \mathbf{g},$$

then setting

$$x_n = (g_n - \mathbf{w}^T \mathbf{y})/u_n,$$

and finally solving

$$L^T \mathbf{x} = D^{-1}\mathbf{y} - x_n \mathbf{w}.$$

Note that the decomposition is uniquely determined by the $2n-1$ values $l_n \equiv 1/u_n$ and l_i, w_i, $i = 1, 2, \cdots, n-1$.

THEOREM 3.2. *The sequence $w_1 = l_1$, $w_i = -w_{i-1} l_i$, $i > 1$ converges to zero for $|\alpha| > 2$.*

Proof. In [15] it is shown that, if $|\alpha| > 2$, the sequence $|l_j|$ is strictly increasing and converges to $|l| < 1$. Thus,

$$|w_i| = \left| (-1)^{i+1} \prod_{j=1}^{i} l_j \right| = \prod_{j=1}^{i} |l_j| < |l|^i,$$

from which convergence to zero follows. □

The proof of the theorem gives a method of determining an upper bound for the value of i after which w_i is zero to machine precision. This occurs when $|l|^i = \epsilon$. Taking logarithms yields

$$i = k_w \equiv \left\lceil \frac{\ln \epsilon}{\ln |l|} \right\rceil = 2k_u,$$

where k_u is the convergence index of the sequence u_i. This, along with the convergence results for the sequence u_i, leads to the following algorithm for the solution to (1) with $\mu = \nu = \rho = 1$ and $|\alpha| > 2$. (The solution overwrites the input vector $\mathbf{g}$.)

ALGORITHM 3.3. **Special *LU* Factorization (Periodic).**

0. *Preliminaries*
   $l \leftarrow 2/[\alpha + \operatorname{sgn}(\alpha)(\alpha^2 - 4)^{1/2}]$
   $k_u \leftarrow \min(\lfloor \ln \epsilon / 2 \ln |l| \rfloor, n-1)$
   $k_w \leftarrow \min(2k_u, n-2)$
1. *Rescale*
   $g_i \leftarrow g_i/b, \quad i = 1, 2, \cdots, n$
2. *Factorization*
   $l_1 \leftarrow 1/\alpha$
   $l_i \leftarrow 1/[\alpha - l_{i-1}], \quad i = 2, 3, \cdots, k_u$
   $l_i \leftarrow l, \quad i = k_u + 1, k_u + 2, \cdots, n-1$
   $w_1 \leftarrow l_1$
   $w_i \leftarrow -w_{i-1} l_i, \quad i = 2, 3 \cdots, k_w$
   $w_i \leftarrow 0, \quad i = k_w + 1, k_w + 2, \cdots, n-2$
   $w_{n-1} \leftarrow (1 - w_{n-2}) l_{n-1}$
   $l_n \leftarrow 1/[\alpha - w_1 + \sum_{j=2}^{k_w} (w_j w_{j-1}) - (1 - w_{n-2}) w_{n-1}]$
3. *Forward substitution*
   $g_i \leftarrow g_i - l_{i-1} g_{i-1}, \quad i = 2, 3, \cdots, n-1$
   $g_n \leftarrow l_n (g_n - \sum_{j=1}^{k_w} w_j g_j - w_{n-1} g_{n-1})$
4. *Backward substitution*
   $g_{n-1} \leftarrow l_{n-1}(g_{n-1} - (1 - w_{n-2}) g_n)$
   $g_i \leftarrow l_i (g_i - g_{i+1}) - w_i g_n, \quad i = n-2, n-3, \cdots, 1$

This algorithm requires $7n + 6\tilde{k}_w + 2\tilde{k}_u$ floating-point operations, where $\tilde{k}_w = \max(k_w, n)$. A working storage of length $2n$ is required (actually, only $\tilde{k}_u + \tilde{k}_w$ words are necessary). When $k_u \geq n$ the algorithm requires $15n$ operations.

In the case of multiple right-hand sides we may again eliminate step 2 in all but the first solve. Thus, subsequent systems may be solved with only $7n + 2\tilde{k}_w$ operations. For weakly diagonally dominant systems this is equivalent to $9n$ operations, a significant savings over the $15n$ required for the first solve. In the strongly diagonally dominant case there is only a savings of $12\tilde{k}_w$ operations, which is insignificant except for very small n.

When $|\alpha| = 2$ the system is singular unless $\alpha = 2$ and n is odd. In the latter case we may set $k_u = k_w = n - 1$ and use steps 1–4 above.

4. Reversed triangular factorization. Each of the algorithms of §3 requires working storage of length $O(k_u)$. In contrast, a method proposed by Evans [4], [5], [6] requires no auxiliary vector storage and only $O(1)$ divisions. In this section we describe this method, extending it to the cases where $\mu = 2$ and/or $\nu = 2$, showing how the resulting algorithms may be implemented with the same types of economies as in the Malcolm–Palmer method.

4.1. Dirichlet and Neumann cases. We first consider (1) in the case $\rho = 0$. Choosing β such that $b/a = -\beta/(1+\beta^2)$ with $0 < |\beta| < 1$ and scaling (1) by $(1+\beta^2)/a$ yields the equivalent problem

$$T\mathbf{x} = \mathbf{g}, \tag{5}$$

where

$$T = \begin{bmatrix} (1+\beta^2)/\mu & -\beta & & & & & & \\ -\beta & (1+\beta^2) & -\beta & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & -\beta & (1+\beta^2) & -\beta & \\ & & & & & -\beta & (1+\beta^2)/\nu \end{bmatrix}.$$

T has the factorization $T = QQ^T$, where Q is the $n \times (n+1)$ matrix

$$Q = \begin{bmatrix} \sigma & -\beta & & & & & & \\ & 1 & -\beta & & & & & \\ & & 1 & -\beta & & & & \\ & & & \ddots & \ddots & & & \\ & & & & \ddots & \ddots & & \\ & & & & & 1 & -\beta & \\ & & & & & & 1 & -\gamma \end{bmatrix},$$

$\sigma^2 = (1+\beta^2)/\mu - \beta^2$, and $\gamma^2 = (1+\beta^2)/\nu - 1$. Thus, a solution to (5) may be determined by solving $Q\mathbf{y} = \mathbf{g}$ and $Q^T\mathbf{x} = \mathbf{y}$ in turn. Note that the factorization is determined simply by the scalars β, σ, and γ, and hence no auxiliary vector storage is required. (The vectors $\mathbf{y}$ and $\mathbf{x}$ may occupy the same storage.)

We first show how the element y_{n+1} may be computed. The remaining y_i are then determined by backward substitution and the x_i follow from a forward substitution. From $Q\mathbf{y} = \mathbf{g}$ we have

$$\begin{aligned} \sigma y_1 \quad - \quad \beta^{n-1}\gamma y_{n+1} &= \textstyle\sum_{j=1}^{n} \beta^{j-1} g_j, \\ y_{n-k} \quad - \quad \beta^k \gamma y_{n+1} &= \textstyle\sum_{j=n-k}^{n} \beta^{j-(n-k)} g_j, \quad 0 \le k \le n-2, \end{aligned} \tag{6}$$

while from $Q^T\mathbf{x} = \mathbf{y}$ we have

$$x_n = \sum_{k=0}^{n-2} \beta^k y_{n-k} + \beta^{n-1} y_1/\sigma, \tag{7}$$

$$-\gamma x_n = y_{n+1}. \tag{8}$$

Putting the expressions for y_{n-k} from (6) into (7), using (8), and simplifying gives

$$\left[1 + \frac{\gamma^2\beta^{2(n-1)}}{\sigma^2} + \gamma^2 \sum_{k=0}^{n-2} \beta^{2k}\right] x_n = \frac{1}{\sigma^2} \sum_{j=0}^{n-1} \beta^{2(n-1-j)} \beta^j g_{n-j} \tag{9}$$

$$+\sum_{j=0}^{n-2}\left(\sum_{k=0}^{n-2-j}\beta^{2k}\right)\beta^j g_{n-j}.$$

Using the summation formula for β^{2k} for $|\beta| < 1$ and simplifying further yields

$$\delta x_n = \sum_{j=0}^{n-2}\beta^j g_{n-j} + (c+1)\beta^{n-1}g_1 + c\sum_{j=n}^{2n-2}\beta^j g_{j-n+2}, \tag{10}$$

where $\delta = 1-\beta^2+\gamma^2(1+c\beta^{2n-2})$, and $c = (1-\beta^2-\sigma^2)/\sigma^2$. This expression for x_n is a polynomial of degree $2n-2$ in β which may be stably and efficiently evaluated using Horner's rule.

Since $|\beta| < 1$ we have that $\beta^j \to 0$. If $|\beta|$ is sufficiently small, then many of the high-order terms in (10) will be zero to machine precision. This can be used to reduce the operation count for the computation of x_n. To determine when we can discard terms, first rewrite (10) as

$$x_n = \sum_{j=0}^{2n-2} c_j\beta^j,$$

where

$$\delta c_j = \begin{cases} g_{n-j}, & 0 \le j \le n-2 \\ (c+1)g_1, & j = n-1 \\ cg_{j-n+2}, & n \le j \le 2n-2. \end{cases}$$

We will discard terms that are small relative to the size of the result vector $\mathbf{x}$, i.e., for

$$\frac{|c_j\beta^j|}{\|\mathbf{x}\|} < \epsilon, \tag{11}$$

where $\|\cdot\|$ denotes the infinity norm. The following will be used to bound this quantity:

$$|c_j| \le \frac{2\|\mathbf{g}\|}{|\delta|},$$
$$\|T\| = (1+|\beta|)^2,$$
$$|\delta| \ge \frac{(1-\beta^2)^2}{2}.$$

The first inequality follows from $|c| \le 1$. Each requires that μ and ν be in the set $\{1,2\}$. Using these in turn we have

$$\frac{|c_j\beta^j|}{\|\mathbf{x}\|} \le \frac{2\|\mathbf{g}\||\beta|^j}{|\delta|\|\mathbf{x}\|} \le \frac{2\|T\|\|\beta|^j}{|\delta|} = \frac{2(1+|\beta|)^2|\beta|^j}{|\delta|} \le \frac{4|\beta|^j}{(1-|\beta|)^2}.$$

From this we see that a sufficient condition for (11) to hold is

$$|\beta|^j < (1-|\beta|)^2\epsilon/4.$$

Taking logs we find that this occurs when

$$j > 2k_\beta \equiv \log_{|\beta|}[(1-|\beta|)^2\epsilon/4].$$

TABLE 5
Values of the convergence index k_β for $\epsilon = 2^{1-t}$.

α	k_β	
	$t = 47$	$t = 24$
2.001	636	384
2.010	190	111
2.100	57	32
2.500	25	14
4.000	13	7
10.000	8	4
50.000	5	3

Table 5 displays sample values of k_β; significant savings in computation time are possible when $k_\beta << n-1$.

This leads to the following algorithm for the solution to (5) with $\rho = 0$ and $|\alpha| > 2$. (The solution overwrites the input vector **g**.)

ALGORITHM 4.1. **Reversed Triangular Factorization.**

0. *Preliminaries*
   $\beta \leftarrow -\alpha(1 + [1 - 4/\alpha^2]^{1/2})/2$
   $\beta \leftarrow 1/\beta$, *if* $|\beta| > 1$
   $\sigma \leftarrow [(1+\beta^2)/\mu - \beta^2]^{1/2}$
   $\gamma^2 \leftarrow (1+\beta^2)/\nu - 1$
   $c \leftarrow (1 - \beta^2 - \sigma^2)/\sigma^2$
   $\delta \leftarrow 1 - \beta^2 + \gamma^2(1 + c\beta^{2n-2})$
   $k \leftarrow \min(\lceil \log_{|\beta|}(1-|\beta|)^2\epsilon/4 \rceil, 2n-2)$
1. *Rescale*
   $s \leftarrow (1+\beta^2)/a$
   $g_i \leftarrow sg_i, \quad i = 1, 2, \cdots, n$
2. *Factorization*
   $x_n \leftarrow 0$
   $m \leftarrow k - n + 2$
   $x_n \leftarrow \beta x_n + g_i, \quad i = m, m-1, \cdots, 2$
   $x_n \leftarrow cx_n$
   $x_n \leftarrow \beta x_n + (c+1)g_1, \quad$ *if* $k \geq n-1$
   $m \leftarrow \max(n-k, 2)$
   $x_n \leftarrow \beta x_n + g_i, \quad i = m, m+1, \cdots, n$
   $x_n \leftarrow x_n/\delta$
3. *Back substitution*
   $g_n \leftarrow g_n - \gamma^2 x_n$
   $g_i \leftarrow g_i + \beta g_{i+1}, \quad i = n-1, n-2, \cdots, 1$
   $g_1 \leftarrow g_1/\sigma^2$
4. *Forward substitution*
   $g_i \leftarrow g_i + \beta g_{i-1}, \quad i = 2, 3, \cdots, n-1$
   $g_n \leftarrow x_n$

This algorithm uses $5n + 4\tilde{k}_\beta$ floating-point operations, where $\tilde{k}_\beta = \max(k_\beta, n)$. If the early convergence of the series for x_n is not accounted for, or $k_\beta > n-1$, then this reduces to $9n$ floating-point operations.

The degenerate cases for $|\beta| = 1$ may be handled as follows:

- $\mu = \nu = 1$: In this case (9) reduces to

$$(n+1)x_n = \sum_{i=0}^{n-1} (n-i)\beta^i g_{n-i},$$

from which x_n may be computed using Horner's rule. (Further multiplications may be saved by separately considering the cases $\beta = 1$ and $\beta = -1$.) Steps 1, 3, and 4 of the algorithm may then be applied with $\sigma^2 = \gamma^2 = 1$.

- $\mu = 2$, $\nu = 1$: Here $\sigma = 0$, $\gamma = 1$, and the first column of Q drops out, leaving only forward and backward substitution phases. The algorithm is as follows. (Further multiplications may be saved by separately considering the cases $\beta = 1$ and $\beta = -1$.)

$$\begin{aligned}
&s \leftarrow 2/a \\
&g_i \leftarrow sg_i, \quad i = 1, 2, \cdots, n \\
&g_1 \leftarrow -\beta g_1 \\
&g_i \leftarrow -\beta(g_i - g_{i-1}), \quad i = 2, 3, \cdots, n-1 \\
&g_n \leftarrow g_n - g_{n-1} \\
&g_i \leftarrow -\beta(g_i - g_{i+1}), \quad i = n-1, n-2, \cdots, 1
\end{aligned}$$

- $\mu = 1$, $\nu = 2$: Here $\sigma = 1$, $\gamma = 0$, and the last column of Q drops out, leaving only backward and forward substitution phases. The algorithm is as follows. (Further multiplications may be saved by separately considering the cases $\beta = 1$ and $\beta = -1$.)

$$\begin{aligned}
&s \leftarrow 2/a \\
&g_i \leftarrow sg_i, \quad i = 1, 2, \cdots, n \\
&g_i \leftarrow g_i + \beta g_{i+1}, \quad i = n-1, n-2, \cdots, 1 \\
&g_i \leftarrow g_i + \beta g_{i-1}, \quad i = 2, 3, \cdots, n
\end{aligned}$$

- $\mu = \nu = 2$: The matrix is singular.

4.2. Periodic case. We next consider (1) in the case $\mu = \nu = \rho = 1$. Scaling the system as before we then have the equivalent problem

$$T\mathbf{x} = \mathbf{g}, \tag{12}$$

where

$$T = \begin{bmatrix}
(1+\beta^2) & -\beta & & & & & -\beta \\
-\beta & (1+\beta^2) & -\beta & & & & \\
 & \cdot & \cdot & \cdot & & & \\
 & & \cdot & \cdot & \cdot & & \\
 & & & \cdot & \cdot & \cdot & \\
 & & & & -\beta & (1+\beta^2) & -\beta \\
-\beta & & & & & -\beta & (1+\beta^2)
\end{bmatrix}.$$

T has the factorization $T = QQ^T$, where Q is the $n \times n$ matrix

$$Q = \begin{bmatrix}
1 & & & & & & -\beta \\
-\beta & 1 & & & & & \\
 & -\beta & 1 & & & & \\
 & & \cdot & \cdot & \cdot & & \\
 & & & \cdot & \cdot & \cdot & \\
 & & & & \cdot & \cdot & \cdot \\
 & & & & & -\beta & 1
\end{bmatrix}.$$

TABLE 6
Values of the convergence index k_p for $\epsilon = 2^{1-t}$.

α	k_p	
	$t = 47$	$t = 24$
2.001	1119	614
2.010	343	184
2.100	106	55
2.500	48	25
4.000	25	13
10.000	14	8
50.000	9	5

Note that the factorization is determined simply by the scalar β and hence no auxiliary vector storage is required. A solution to (12) may be determined by solving $Q\mathbf{y} = \mathbf{g}$ and $Q^T\mathbf{x} = \mathbf{y}$ in turn. The former system may be solved by forward substitution once y_1 is known, while the latter may be solved by backward substitution once x_n is known.

It is straightforward to show that

$$(1 - \beta^n)y_1 = g_1 + \sum_{j=1}^{n-1} \beta^j g_{n+1-j}, \tag{13}$$

$$(1 - \beta^n)x_n = y_n + \sum_{j=1}^{n-1} \beta^j y_j. \tag{14}$$

When $|\beta| << 1$ many of the high-order terms in β will be zero to machine precision, which can be used to reduce the operation count for the computation of y_1 and x_n as before. Let d_j and e_j be the coefficients of β^j in (13) and (14), respectively. We then have

$$\frac{|d_j\beta^j|}{\|\mathbf{y}\|} \le \frac{\|\mathbf{g}\||\beta|^j}{(1-\beta^n)\|\mathbf{y}\|} \le \frac{\|Q\||\beta|^j}{1-\beta^n},$$

and

$$\frac{|e_j\beta^j|}{\|\mathbf{x}\|} \le \frac{\|\mathbf{y}\||\beta|^j}{(1-\beta^n)\|\mathbf{x}\|} \le \frac{\|Q^T\||\beta|^j}{1-\beta^n}.$$

Using $\|Q\| = \|Q^T\| = 1 + |\beta|$ we find that both of these quantities are bounded by

$$\frac{(1+|\beta|)|\beta|^j}{1-\beta^n} \le \frac{(1+|\beta|)|\beta|^j}{1-\beta^2} = \frac{|\beta|^j}{1-|\beta|}.$$

Thus, terms in 13 and (14) may be safely discarded for

$$j > k_p \equiv \log_{|\beta|}[(1-|\beta|)\epsilon].$$

Sample values of the convergence index k_p are given in Table 6.

This leads to the following algorithm for the solution to (1) with $\mu = \nu = \rho = 1$ and $|\alpha| > 2$. (The solution overwrites the input vector $\mathbf{g}$.)

TABLE 7
Floating-point operation counts.

Method	$\rho = 0$	$\rho = 1$
Gaussian elimination	$8n$	$18n$
Cyclic reduction	$7n-5n/2^{k_c}$	$7n-5n/2^{k_c}$ †
Special LU factorization	$5n+2\tilde{k}_u$	$7n+6\tilde{k}_w+2\tilde{k}_u$
Reversed triangular factorization	$5n+4\tilde{k}_\beta$	$5n+4\tilde{k}_p$

† For $n = 2^k$ only.

ALGORITHM 4.2. **Reversed Triangular Factorization (Periodic).**

0. *Preliminaries*
   $\beta \leftarrow -\alpha(1+[1-4/\alpha^2]^{1/2})/2$
   $\beta \leftarrow 1/\beta, \quad$ *if* $|\beta| > 1$
   $\delta \leftarrow 1-\beta^n$
   $k \leftarrow \min(\lceil \log_{|\beta|}(1-|\beta|)\epsilon \rceil, n-1)$
1. *Rescale*
   $s \leftarrow (1+\beta^2)/a$
   $g_i \leftarrow sg_i, \quad i = 1,2,\cdots,n$
2. *Determine* y_1
   $y_1 \leftarrow g_{n-k+1}$
   $y_1 \leftarrow \beta y_1 + g_i, \quad i = n-k+2, n-k+3, \cdots, n$
   $g_1 \leftarrow (\beta y_1 + g_1)/\delta$
3. *Forward substitution*
   $g_i \leftarrow g_i + \beta g_{i-1}, \quad i = 2,3,\cdots,n$
4. *Determine* x_n
   $x_n \leftarrow g_k$
   $x_n \leftarrow \beta x_n + g_i, \quad i = k-1, k-2, \cdots, 1$
   $g_n \leftarrow (\beta x_n + g_n)/\delta$
5. *Backward substitution*
   $g_i \leftarrow g_i + \beta g_{i+1}, \quad i = n-1, n-2, \cdots, 1$

This algorithm requires $5n+4\tilde{k}_p$ floating-point operations, where $\tilde{k}_p = \max(k_p, n)$. If the early convergence of the series for y_1 and x_n are not accounted for, or $k_p > n$, then this reduces to $9n$ operations, which is larger than for the algorithm of §3.1. However, when $k_p << n$, the operation count is considerably better.

When $|\alpha| = 2$ the system is singular unless $\alpha = 2$ and n is odd. In the latter case we may set $k_p = n-1$ and use steps 1–5 above. (Further multiplications may be saved by separately considering the cases $\beta = 1$ and $\beta = -1$.)

5. Comparison of methods. In this section we compare the methods described in the previous sections and present the results of some computational experiments.

Table 7 summarizes the floating-point operation counts for the methods of §§2, 3, and 4. Also included are operation counts for Gaussian elimination without pivoting for a symmetric tridiagonal system (e.g., the algorithm used by LINPACK's SPTSL [3]). In the periodic case the algorithm is modified in a straightforward way to account for the extra nonzeros in the last row and column of the matrix.

We first consider the case $\rho = 0$. For highly diagonally dominant systems ($|\alpha| \gg 2$) we have that $\tilde{k}_u$ and $\tilde{k}_\beta$ are $O(1)$, in which case the special LU factorization and reversed triangular factorization appear best. Each method deteriorates in performance as $|\alpha| \to 2$, with cyclic reduction being the least affected. In the worst case $k_c = \log_2 n$, and $\tilde{k}_u = \tilde{k}_\beta = n$, in which case cyclic reduction and the special LU factorization appear to be best.

TABLE 8
Floating-point operation counts for repeat solves.

Method	$\rho = 0$	$\rho = 1$
Gaussian elimination	$5n$	$9n$
Special LU factorization	$5n$	$7n + 2\tilde{k}_w$

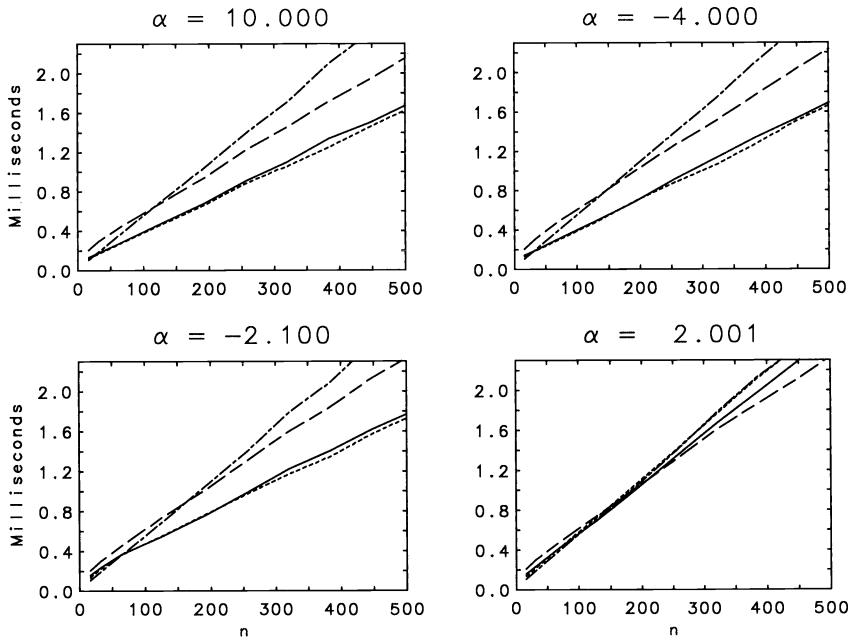


FIG. 1. *Timing of algorithms in Dirichlet case for small n (solid line is* RTF, *short dash is* SpLU, *long dash is* CR, *long-short dash is* GE).

We next consider the periodic case $\rho = 1$. For highly diagonally dominant systems we have that $\tilde{k}_p$, $\tilde{k}_w$, and $\tilde{k}_u$ are $O(1)$, in which case the reversed triangular factorization is most efficient. Once again, each method deteriorates in performance as $|\alpha| \to 2$, with cyclic reduction being the least affected. In the worst case $k_c = \log_2 n$ and $k_p = k_u = \tilde{k}_w = n$, in which case cyclic reduction appears to be best.

We reemphasize that these conclusions are only valid for computations on a scalar processor. Cyclic reduction is preferred for solving single tridiagonal systems on vector computers because vector operations may be carried out during most of the computation [14]. The other algorithms described here are inherently recursive.

Of the methods considered, only the special LU factorization and Gaussian elimination can take advantage of the case when many systems of the form (1) must be solved, each differing only in the right-hand-side vector $\mathbf{g}$. After the first solve, the factorization portion of these algorithms may be eliminated. Doing this we arrive at methods with operation counts given in Table 8. We first consider the case $\rho = 0$. The methods of Table 8 appear to be best when $|\alpha| \approx 2$, although they do not appear to be significantly better than a full solve using reversed triangular factorization in the highly diagonally dominant case. In the periodic case reversed triangular factorization always seems preferable.

The operation counts that we have given are valid only for large n. In addition, they ignore all overhead required to implement the methods. The following numerical experiment is designed to assess the performance of these methods in practice.

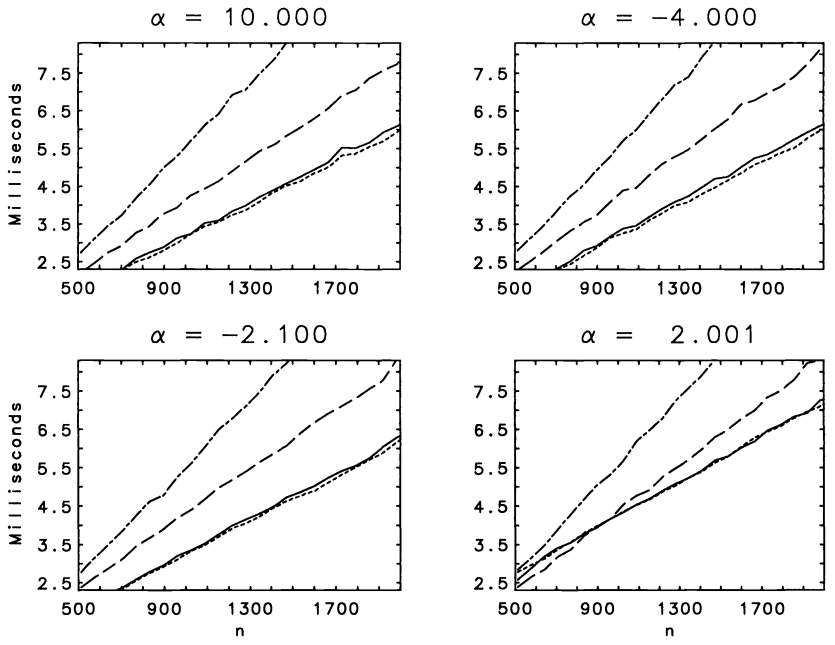


FIG. 2. *Timing of algorithms in Dirichlet case for large n (solid line is* RTF, *short dash is* SpLU, *long dash is* CR, *long-short dash is* GE).

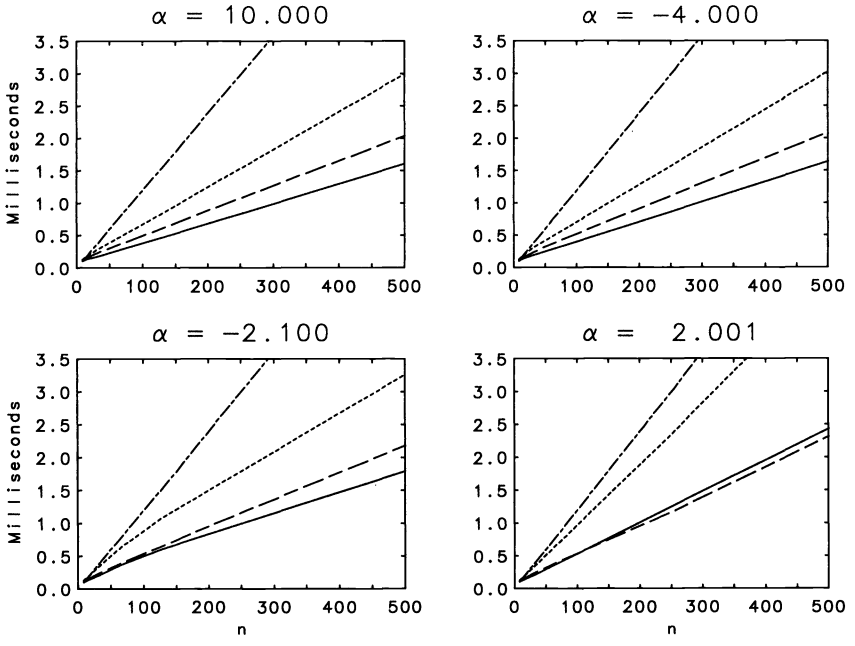


FIG. 3. *Timing of algorithms in periodic case for small n (solid line is* RTF, *short dash is* SpLU, *long dash is* CR, *long-short dash is* GE).

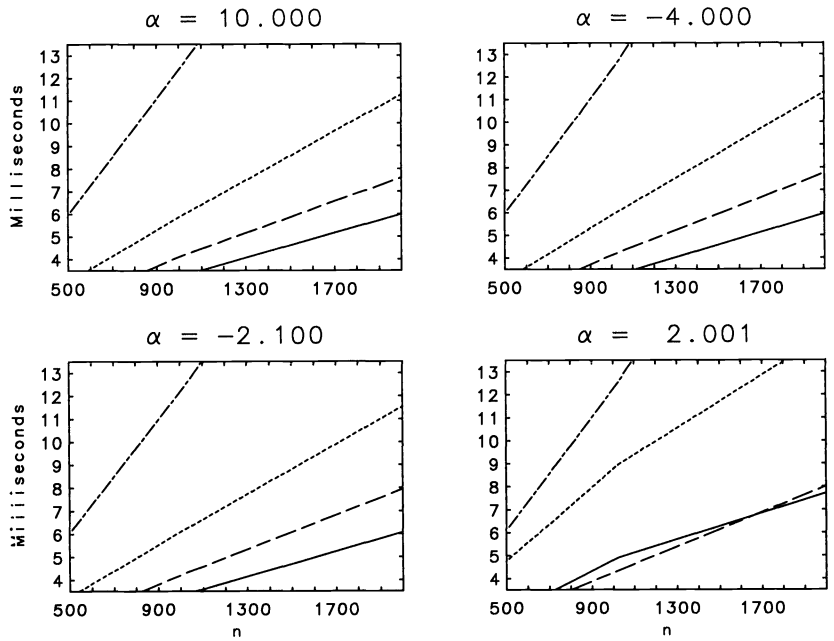


FIG. 4. *Timing of algorithms in periodic case for large n (solid line is* RTF, *short dash is* SpLU, *long dash is* CR, *long-short dash is* GE).

In Figs. 1–4 we present timing data for cyclic reduction (CR, long dash), the special *LU* factorization method (SpLU, short dash), the reversed triangular factorization method (RTF, solid line), and Gaussian elimination (GE, long-short dash) as a function of n for various values of α in the Dirichlet and periodic cases. (The results for other boundary conditions are similar to the Dirichlet results.) Times are given in milliseconds on a Cyber 180/855 (NOS). Each algorithm was written in standard Fortran and compiled with the highest level of optimization (FTN5,OPT=3). The problems were generated with solutions $\mathbf{x} = 1$; the accuracy of the computed solutions for each method were comparable. All tests are for a single right-hand side. Data were gathered for $n = 16, 32, 64(64)2048$ in the Dirichlet case and for $n = 2^k$, for $k = 4, 5, \cdots, 11$ in the periodic case. A subset of these data is presented in Table 9, where entries are given in the form *millisec : megaflops*. The megaflop entries provide a check on the operation counts; since the computations are done on a scalar processor, these rates should converge with n and be roughly the same for each method.

The data are generally consistent with the timing formulas of Table 7. SpLU and RTF are the most sensitive to the value of $|\alpha|$, while CR is the least: the degradation of SpLU and RTF from the largest to smallest $|\alpha|$ for $n = 2048$ was about 15 percent in the Dirichlet case and 23 percent in the periodic case; the degradation of CR was only 5 percent in both cases. In the Dirichlet case the best methods were SpLU and RTF, whose performance was nearly identical. As $|\alpha|$ approached 2 their superiority to CR decreased: for $n = 2048$ they were 26 percent faster at the largest $|\alpha|$, and only 15 percent faster at the smallest $|\alpha|$. At $\alpha = 2.001$ CR was actually slightly faster for n in the range 200–900. In the periodic case RTF performed best; however, its superiority to cyclic reduction decreased as $|\alpha|$ approached 2: for $n = 2048$ it was 22 percent faster at the largest $|\alpha|$, and only 4 percent faster at the smallest $|\alpha|$. Once again, at $\alpha = 2.001$ CR was actually slightly faster for intermediate n, i.e., 128, 256,

TABLE 9
Timing of algorithms on Cyber 855 (NOS). *Entries are* milliseconds : megaflops.

$\alpha = 10.0$								
	Dirichlet				Periodic			
n	CR	SpLU	RTF	GE	CR†	SpLU	RTF	GE
16	0.2 : 0.5	0.1 : 0.8	0.1 : 0.8	0.1 : 1.2	0.2 : 0.7	0.2 : 1.0	0.1 : 1.0	0.2 : 1.5
64	0.4 : 1.0	0.3 : 1.2	0.3 : 1.3	0.4 : 1.4	0.4 : 1.2	0.5 : 1.1	0.3 : 1.4	0.8 : 1.5
256	1.2 : 1.4	0.9 : 1.5	0.9 : 1.4	1.4 : 1.5	1.1 : 1.6	1.6 : 1.2	0.8 : 1.6	3.1 : 1.5
1024	4.2 : 1.6	3.2 : 1.6	3.2 : 1.6	5.7 : 1.4	4.2 : 1.6	6.0 : 1.2	3.3 : 1.6	12.7 : 1.5
2048	8.2 : 1.7	6.1 : 1.7	6.3 : 1.6	11.6 : 1.4	7.8 : 1.8	11.5 : 1.3	6.1 : 1.7	26.9 : 1.4
$\alpha = -4.0$								
	Dirichlet				Periodic			
n	CR	SpLU	RTF	GE	CR†	SpLU	RTF	GE
16	0.2 : 0.5	0.1 : 0.8	0.1 : 0.9	0.1 : 1.2	0.2 : 0.7	0.2 : 1.2	0.1 : 1.1	0.2 : 1.5
64	0.5 : 1.0	0.3 : 1.2	0.3 : 1.3	0.4 : 1.4	0.4 : 1.1	0.5 : 1.2	0.3 : 1.4	0.8 : 1.5
256	1.3 : 1.4	0.9 : 1.5	0.9 : 1.5	1.4 : 1.5	1.1 : 1.6	1.6 : 1.2	0.9 : 1.6	3.0 : 1.5
1024	4.4 : 1.6	3.3 : 1.6	3.4 : 1.5	5.7 : 1.4	4.2 : 1.7	6.0 : 1.2	3.3 : 1.6	12.6 : 1.5
2048	8.1 : 1.7	6.1 : 1.7	6.2 : 1.7	11.5 : 1.4	7.9 : 1.8	11.6 : 1.3	6.1 : 1.7	26.9 : 1.4
$\alpha = -2.1$								
	Dirichlet				Periodic			
n	CR	SpLU	RTF	GE	CR†	SpLU	RTF	GE
16	0.2 : 0.5	0.1 : 0.8	0.2 : 0.9	0.1 : 1.2	0.2 : 0.7	0.2 : 1.2	0.1 : 1.1	0.2 : 1.5
64	0.5 : 1.0	0.4 : 1.2	0.4 : 1.5	0.4 : 1.4	0.4 : 1.2	0.6 : 1.5	0.4 : 1.6	0.8 : 1.5
256	1.3 : 1.4	1.0 : 1.4	1.0 : 1.5	1.4 : 1.5	1.2 : 1.5	1.8 : 1.4	1.0 : 1.7	3.1 : 1.5
1024	4.4 : 1.6	3.3 : 1.6	3.3 : 1.6	5.6 : 1.5	4.3 : 1.7	6.2 : 1.3	3.4 : 1.6	12.5 : 1.5
2048	8.6 : 1.7	6.4 : 1.6	6.5 : 1.6	11.8 : 1.4	8.1 : 1.8	11.8 : 1.3	6.2 : 1.7	26.8 : 1.4
$\alpha = 2.001$								
	Dirichlet				Periodic			
n	CR	SpLU	RTF	GE	CR†	SpLU	RTF	GE
16	0.2 : 0.5	0.1 : 0.8	0.2 : 0.9	0.1 : 1.2	0.2 : 0.7	0.2 : 1.2	0.1 : 1.1	0.2 : 1.5
64	0.5 : 1.0	0.4 : 1.1	0.4 : 1.5	0.4 : 1.4	0.4 : 1.2	0.6 : 1.5	0.4 : 1.6	0.8 : 1.5
256	1.3 : 1.4	1.4 : 1.3	1.3 : 1.7	1.4 : 1.5	1.2 : 1.5	2.4 : 1.6	1.3 : 1.8	3.1 : 1.5
1024	4.5 : 1.6	4.4 : 1.4	4.3 : 1.8	5.7 : 1.4	4.4 : 1.6	9.0 : 1.6	4.9 : 1.9	12.5 : 1.5
2048	8.6 : 1.7	7.3 : 1.5	7.3 : 1.7	11.6 : 1.4	8.2 : 1.7	14.9 : 1.4	7.9 : 1.9	26.9 : 1.4

† implementation restricted to $n = 2^k$.

512, and 1024. Note that an implementation of CR for arbitrary n would execute more slowly than the one tested here.

We conclude that, on a single scalar processor, SpLU and RTF are best for Dirichlet/Neumann problems and RTF is best for periodic problems, although their superiority over CR decreases as the system becomes less diagonally dominant. The difference between the methods is most apparent in the periodic case. In the nonperiodic, weakly diagonally dominant case, the SpLU is preferred when multiple solves for different right-hand sides are required.

REFERENCES

[1] M. D. BAKES, *An alternative method of solution of certain tri-diagonal systems of linear equations*, Comput. J., 7 (1964), pp. 135–136.

[2] R. F. BOISVERT, *A fourth-order accurate Fourier method for the Helmholtz equation in three dimensions*, ACM Trans. Math. Software, 13 (1987), pp. 221–234.

[3] J. J. DONGARRA, C. B. MOLER, J. R. BUNCH, AND G. W. STEWART, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979.

[4] D. J. EVANS, *An algorithm for the solution of certain tridiagonal systems of linear equations*, Comput. J., 15 (1972), pp. 356–359.

[5] D. J. EVANS, *Fast* ADI *methods for the solution of linear parabolic partial differential equations involving* 2 *space dimensions*, BIT, 17 (1977), pp. 486–491.

[6] ———, *On the solution of certain Toeplitz tridiagonal linear systems*, SIAM J. Numer. Anal., 17 (1980), pp. 675–680.

[7] D. J. EVANS AND C. D. V. FORRINGTON, *Note on the solution of certain tri-diagonal systems of linear equations*, Comput. J., 5 (1963), pp. 327–328.

[8] D. FISHER, G. GOLUB, O. HALD, C. LEIVA, AND O. WIDLUND, *On Fourier–Toeplitz methods for separable elliptic problems*, Math. Comp., 28 (1974), pp. 349–368.

[9] I. GLADWELL AND I. WAIT, *A Survey of Numerical Methods for Partial Differential Equations*, Clarendon Press, Oxford, U.K., 1979.

[10] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[11] D. HELLER, *Some aspects of the cyclic reduction algorithm for block triangular linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 484–496.

[12] R. W. HOCKNEY, *A fast direct solution of Poisson's equation using Fourier analysis*, J. Assoc. Comput. Mach., 12 (1965), pp. 95–113.

[13] H. B. KELLER, *Numerical Methods for Two-Point Boundary Value Problems*, Blaisdell, London, 1963.

[14] J. J. LAMBIOTTE AND R. G. VOIGT, *The solution of tridiagonal linear systems on the* CDC *Star*-100 *computer*, ACM Trans. Math. Software, 1 (1975), pp. 308–329.

[15] M. A. MALCOLM AND J. PALMER, *A fast method for solving a class of tridiagonal linear systems*, Comm. ACM, 17 (1974), pp. 14–17.

[16] D. J. ROSE, *An algorithm for solving a special class of tridiagonal systems of linear equations*, Comm. ACM, 12 (1969), pp. 234–236.

[17] P. N. SWARZTRAUBER, *The methods of cyclic reduction, Fourier analysis, and the* FACR *algorithm for the discrete solution of Poisson's equation on a rectangle*, SIAM Rev., 19 (1977), pp. 490–501.

[18] R. A. SWEET, *A generalized cyclic reduction algorithm*, SIAM J. Numer. Anal., 11 (1974), pp. 506–520.

[19] O. B. WIDLUND, *On the use of fast methods for separable finite difference equations for the solution of general elliptic problems*, in Sparse Matrices and Their Applications, D. J. Rose and R. A. Willoughby, eds., Plenum Press, New York, 1972, pp. 121–131.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 443–453, March 1991
©1991 Society for Industrial and Applied Mathematics
012

APPROXIMATION OF INTEGRALS FOR BOUNDARY ELEMENT METHODS*

KURT GEORG†

Abstract. A new method for approximating two-dimensional integrals $\int_B f(x)\,\mu(dx)$ over surfaces $B \subset \mathbf{R}^3$ is introduced where μ is the standard measure of surface area. Such integrals typically occur in boundary element methods. The algorithm is based on triangulations $T := \bigcup T_i$ approximating B. Under the assumption that the surface B is given implicitly by an equation $H(x) = 0$, a retraction $P : U \supset B \to B$ is used to obtain a curved subdivision $B = \bigcup B_i$ via $B_i := PT_i$. Except in very special cases, this retraction is not analytically accessible, but is generated by a subroutine. Hence standard multiple integral techniques are not available. Thus, the approach given here differs from the usual panel method. It is shown how to calculate the integrals as precisely as wished. Two numerical examples are given. The first integrand $f(x) \equiv 1$ is regular, and it is shown that a very accurate extrapolation method can be used. The second integrand $f(x) \sim ||x - x_0||^{-1}$ is singular, and an adaptive refinement procedure is displayed.

Key words. surface integration, quadrature formula, boundary element method, trapezoidal rule, extrapolation method, adaptive refinement

AMS(MOS) subject classifications. 65D30, 65N35, 65R20, 45E05

1. Motivation.

Let us motivate the following discussion by considering a well-known integral equation related to Laplace's equation:

$$\begin{aligned} \Delta u &= 0 \quad \text{in} \quad D, \\ u &= g \quad \text{on} \quad B, \end{aligned} \tag{1.1}$$

where $D \subset \mathbf{R}^3$ is a bounded open nonempty domain with sufficiently regular boundary B. We introduce the fundamental solution

$$s(x,y) := \frac{1}{4\pi} \frac{1}{||x-y||},$$

and consider the following double layer potential ansatz for a solution u of (1.1):

$$u(x) = \int_B \frac{\partial s(x,y)}{\partial \nu(y)}\, \sigma(y)\, \mu(dy), \tag{1.2}$$

where $\nu(y)$ for $y \in B$ is the unit normal vector pointing out of D and σ is an unknown density function on B to be determined by the boundary integral method. Here and in the following, the symbol μ denotes the standard measure of surface area; $\mu(dy)$ is often called the surface element. It turns out, see, e.g., [9]–[12], that σ satisfies the following integral equation of the second kind:

$$\sigma(x) - 2\int_B \frac{\partial s(x,y)}{\partial \nu(y)}\, \sigma(y)\, \mu(dy) = -2g(x)\,. \tag{1.3}$$

*Received by the editors February 2, 1990; accepted for publication April 3, 1990. This work was supported by National Science Foundation grant DMS-8947761 and by Deutsche Forschungsgemeinschaft SFB 256.

†Department of Mathematics, Colorado State University, Fort Collins, Colorado 80523.

A typical discretization method for solving (1.3) is a collocation method, see, e.g., Hackbusch [11]. We introduce basis functions ϕ_j on B and corresponding collocation points $x_k \in B$ to obtain an interpolation of σ:

$$\sigma(x_k) = \sum_j \sigma_j \phi_j(x_k) .$$

The following collocation equation determines the coefficients σ_j of this interpolation:

$$\sum_j \phi_j(x_k)\, \sigma_j - 2 \sum_j \int_B \frac{\partial s(x_k, y)}{\partial \nu(y)}\, \phi_j(y)\, \mu(dy)\ \sigma_j = -2g(x_k) . \tag{1.4}$$

In order to solve the linear equation (1.4) for the unknowns σ_j, the matrix entries

$$M[k,j] := \int_B \frac{\partial s(x_k, y)}{\partial \nu(y)}\, \phi_j(y)\, \mu(dy) \tag{1.5}$$

have to be approximated. The generation of these or similar matrix entries is an essential topic in boundary element methods. Atkinson [4] discusses a Galerkin method for discretizing (1.1).

In order to find suitable basis functions for the above collocation method or for a Galerkin method, usually a triangulation T of the surface B is introduced. Since the surface is closed, i.e., has no boundary, such triangulations usually are obtained from a two-dimensional compact pseudomanifold $\mathcal{T}$ without boundary, see, e.g., [2]. More precisely, we consider a finite collection

$$\mathcal{T} = \{T_i\}_{i=1,\cdots,n}, \quad T := \bigcup_{i=1}^{n} T_i \tag{1.6}$$

of triangles T_i, each having affinely independent vertices in $\mathbf{R}^3$, such that the following two conditions hold:

(1.7) Any two triangles $T_i \neq T_j$, $i \neq j$ intersect in a common edge or vertex or not at all.

(1.8) Any edge is common to exactly two different triangles of $\mathcal{T}$.

The panel methods that are very popular in engineering replace the surface B by the triangulation T. In our example for Poisson's equation (1.1), this replacement has to be performed in all integrals of (1.2), (1.3), (1.4), (1.5). Thus, a preliminary discretization error is generated which, for the purpose of the following discussion, will be called the *panel error*. This panel error has to be considered in addition to any discretization error which is generated by a collocation or Galerkin method. Unfortunately, the numerical analysis of the panel error is not clear at all, contrary to the discretization errors of collocation or Galerkin methods which are well known, see, e.g., [11]. At best, we can hope for a panel error of the order $O(h^2)$, where h measures the mesh size of the triangulation.

The advantage of the panel method is that basis functions ϕ_j can easily be obtained, similarly to the techniques in finite element methods. Typically, the support of such basis functions intersects only very few triangles of $\mathcal{T}$, so that the integrals which have to be computed in order to generate the matrix entries (1.5) can be taken over a few triangles. However, the panel error deprives us of the possibility of making use of

smoother elements. We now outline a different approach that still uses a triangulation but avoids the panel error. In addition, it is shown that the resulting integrals can be numerically approximated as precisely as wished.

2. A subdivision of the surface. We make use of the fact that the surface B is given implicitly by an equation. To be more precise, we make the following three assumptions:

(2.1) Let $H : \mathbf{R}^3 \to \mathbf{R}$ be smooth and 0 a regular value of H.

(2.2) $B := \{x \in \mathbf{R}^3 : H(x) = 0\}$.

(2.3) $D := \{x \in \mathbf{R}^3 : H(x) \leq 0\}$ is nonempty and bounded.

These conditions could be relaxed considerably, but this is not so important for the exposition of our ideas. We consider either of the following two Newton iterations for *orthogonally* converging towards the surface B:

(2.4a) **input** $x_0 \in \mathbf{R}^3$;
for $i = 0, 1, 2, \cdots$ **do** $x_{i+1} \leftarrow x_i - ||\nabla H(x_i)||^{-2} H(x_i)\, \nabla H(x_i)$;
if the iteration converges **then output** $\lim_{i\to\infty} x_i$.

(2.4b) **input** $x_0 \in \mathbf{R}^3$;
for $i = 0, 1, 2, \cdots$ **do** $x_{i+1} \leftarrow x_i - ||\nabla H(x_0)||^{-2} H(x_i)\, \nabla H(x_0)$;
if the iteration converges **then output** $\lim_{i\to\infty} x_i$.

It is clear from the local convergence of Newton's method and the assumptions (2.1)–(2.3) that there exists an open neighborbood U of B such that either one of the above input-output devices defines a smooth map $P : U \to B$, which is a retraction since $P(x) = x$ for $x \in B$. It is easy to implement this retraction numerically. This will be used in the sequel. Furthermore, for a given $\epsilon > 0$, it is also possible to reduce the neighborhood U so that $||x - P(x)|| \leq \epsilon$ and $\left|\langle x - P(x)\,,\, \nabla H(x)\rangle\right| \geq (1-\epsilon)||x-P(x)||\,||\nabla H(x)||$ holds for all $x \in U$. Here $\langle \cdot\,,\, \cdot \rangle$ denotes the standard scalar product of $\mathbf{R}^3$.

Intuitively, it is clear that a two-dimensional compact pseudomanifold $\mathcal{T}$ without boundary, see (1.6), approximates B reasonably well if we can find a neighborhood U in the above sense and such that the following conditions hold:

(2.5) $T_i \in U$ for all $i = 1, \cdots, n$.

(2.6) $\nabla H(x)$ is not tangent to T_i for all $x \in T_i$ and $i \in \{1, \cdots, n\}$.

(2.7) The retraction P induces a topological isomorphism from T onto B.

It is possible to automatically generate triangulations via piecewise linear algorithms, see, e.g., [2], [3], [8] for a general reference. As was shown in [1], such triangulations have the above property for sufficiently small mesh sizes. In this paper, we make use of the fact that the isomorphism (2.7) generates a subdivision of B:

$$\mathcal{B} = \{B_i\}_{i=1,\cdots,n}\,, \quad \text{where} \quad B_i := P(T_i). \tag{2.8}$$

Analogously to the techniques in finite element methods, we can introduce basis functions ϕ_j on T which are adapted to the given triangle structure of $\mathcal{T}$. Via the map P, this finite element structure can easily be transported onto the surface B, in a manner conforming to the subdivision (2.8).

In order to generate the matrix entries (1.5), we typically would have to calculate integrals of the form

$$M[k,j] = \sum_{i \in I(j)} \int_{B_i} \frac{\partial s(x_k, y)}{\partial \nu(y)} \phi_j(y)\, \mu(dy)\,, \tag{2.9}$$

where j and k are the running indices of the matrix, i is the index of the subdivision (2.8), and the index set $I(j)$ in the summation is typically very small, describing a few neighboring pieces of $\mathcal{B}$ supporting the basis function ϕ_j. This basis function is typically generated by a piecewise polynomial function $\tilde{\phi}_j$ on T, i.e.,

(2.10) $\phi_j\big(P(z)\big) = \tilde{\phi}_j(z)$ for $z \in T$.

(2.11) For all $i \in I(j)$, the function $\tilde{\phi}_j(z)$, $z \in T_i$ is a polynomial in the barycentric coordinates with respect to the vertices of T_i.

(2.12) $\tilde{\phi}_j$ vanishes on T_i for $i \notin I(j)$.

We propose to numerically approximate the integrals of the form (2.9) via the transformation P. Thus, (2.9) equals

$$M[k,j] = \sum_{i \in I(j)} \int_{T_i} \frac{\partial s(x_k, P(z))}{\partial \nu(P(z))} \phi_j\big(P(z)\big)\, P_*\mu(dz)\,. \tag{2.13}$$

Here, $P_*\mu$ indicates the measure on T which is obtained from the standard surface area measure μ on B via the map P, i.e., $P_*\mu(A) := \mu(P(A))$ for all measurable sets $A \subset T$.

Hence, our interest is concentrated on the numerical approximation of integrals

$$\int_\Delta f\big(P(z)\big)\, P_*\mu(dz), \tag{2.14}$$

where Δ is a triangle of a pseudomanifold approximating $\mathcal{T}$ of B, and the integrand $f\big(P(z)\big)$ can be numerically approximated on Δ and occasionally may have a weak singularity at a point of Δ.

In principle, we could use two directional derivatives $\partial_1 P$ and $\partial_2 P$ of P with respect to an orthogonal coordinate system of the Δ-plane to obtain

$$P_*\mu(dz) = ||\partial_1 P(z) \times \partial_2 P(z)||\, \lambda(dz)\,, \tag{2.15}$$

where λ is the standard Lebesgue measure on the plane. Hence, the problem is reduced to the numerical approximation of a standard multiple integral, and there are many methods available, see, e.g., Davis and Rabinowitz [6] or Stroud [18]. Efficient methods have also been discussed for the case of spherical coordinates (see Atkinson [5] or Keast [14]).

However, in this paper we consider the case where the map P is only given via a numerical algorithm (except in very simple cases), and hence we would have to use numerical approximations of the directional derivatives. Instead, we prefer to introduce a new approach, namely, a numerical method for directly approximating the integral (2.13), i.e., without making use of the directional derivatives of P.

3. The integration method. We propose to approximate the integral (2.14) by a repeated use of the trapezoidal rule

$$\int_\Delta f\big(P(z)\big)\, P_*\mu(dz) \approx \operatorname{mean}_{i=1}^3\Big(f\big(P(v_i)\big)\Big)\, \operatorname{area}_{i=1}^3\big(P(v_i)\big), \tag{3.1}$$

where v_1, v_2, v_3 are the vertices of the triangle Δ, $\text{mean}_{i=1}^3(a_i)$ indicates the mean value of the numbers a_1, a_2, a_3, and $\text{area}_{i=1}^3(w_i)$ indicates the area of a triangle composed of the vertices w_1, w_2, w_3. Note that this rule differs from the standard trapezoidal rule

$$\int_\Delta f\big(P(z)\big)\, P_*\mu(dz) \approx \text{mean}_{i=1}^3\Big(f\big(P(v_i)\big)\gamma(v_i)\Big)\ \text{area}_{i=1}^3(v_i), \tag{3.2}$$

which is obtained from multiple integration methods via the formula (2.15) by setting

$$\gamma(v) = ||\partial_1 P(v) \times \partial_2 P(v)||.$$

We use the trapezoidal rule (3.1) in an adaptive refinement procedure as indicated in Fig. 1 below:

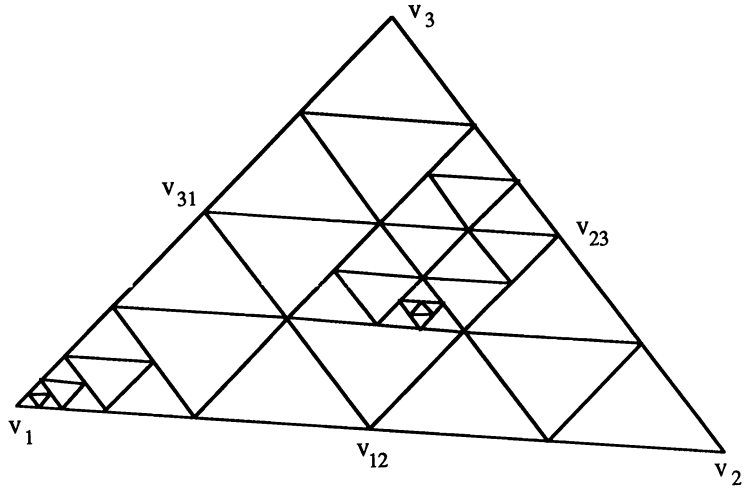


FIG. 1. *Refining the triangle.*

First we calculate a coarse approximation $\texttt{int}_1$ via (3.1), then we subdivide the triangle $[v_1, v_2, v_3]$ into four triangles

$$[v_1, v_{12}, v_{31}],\ [v_{12}, v_2, v_{23}],\ [v_{12}, v_{23}, v_{31}],\ [v_{31}, v_{23}, v_3],$$

and the trapezoidal rule (3.1) is applied to all four triangles. By summing up these values, we obtain a refined approximation $\texttt{int}_2$ of the given integral. The two approximations are compared. If they differ by less than a given tolerance "`TolInt`" (see the tolerance test below), then the algorithm stops. Otherwise, each of the four triangles is treated similarly. A recursive call of this rule enables the algorithm to locally adjust the refinement according to the given tolerance, as indicated in Fig. 1.

To be more precise, we give below a verbatim listing of the essential part of the algorithm, namely, the recursively written PASCAL procedure "`trapez`." The procedure "`newton`" calculates the retraction map P.

```
procedure trapez(var v1, v2, v3, p1, p2, p3: vector;
                 var f1, f2, f3: real; level: integer);
  var
    int1, int2: real;                          {approximate integral values}
    v12, v23, v31, p12, p23, p31: vector;                      {refinement}
    f12, f23, f31: real;                                  {integrand values}
    i: integer;

  begin
  if (level < leveldim)
  then
    begin
    writeln('level exceeded');
    writeln(myfile,'level exceeded');
    close(myfile);
    halt;
    end;
  for i:= 1 to 3 do
    begin
    v12[i]:= 0.5 * (v1[i] + v2[i]);
    v23[i]:= 0.5 * (v2[i] + v3[i]);
    v31[i]:= 0.5 * (v3[i] + v1[i]);
    end;
  newton(v12, p12); newton(v23, p23); newton(v31, p31);
  f12:= integrand(p12);
  f23:= integrand(p23);
  f31:= integrand(p31);
  int1:= area(p1,  p2,  p3 ) * mean(f1,  f2,  f3 );
  int2:= area(p1,  p12, p31) * mean(f1,  f12, f31)
       + area(p12, p2,  p23) * mean(f12, f2,  f23)
       + area(p12, p23, p31) * mean(f12, f23, f31)
       + area(p31, p23, p3 ) * mean(f31, f23, f3 );
  if (abs(int1-int2) < TolInt)                              {tolerance test}
  then
    begin
    integral:= integral + int2;
    levstat[level]:= levstat[level] + 1;          {for statistical purposes}
    end
  else
    begin
    trapez(v1,  v12, v31, p1,  p12, p31, f1,  f12, f31, level+1);
    trapez(v12, v2,  v23, p12, p2,  p23, f12, f2,  f23, level+1);
    trapez(v12, v23, v31, p12, p23, p31, f12, f23, f31, level+1);
    trapez(v31, v23, v3,  p31, p23, p3 , f31, f23, f3,  level+1);
    end;
  end;
```

4. Examples. In order to test the integration method, we use a simple example, namely, $H(x) := ||x||^2 - 1$, so that $B = \{x \in \mathbf{R}^3 : H(x) = 0\}$ is the unit sphere. The vertices of the triangle $\Delta := [e_1, e_2, e_3]$ are the standard unit basis vectors. This gives a big triangle; usually the triangles of a triangulation will be much smaller and closer to the surface. Hence this numerical example displays many more refinements than will be usual. The projection $P(\Delta)$ is a spherical triangle which cuts out $\frac{1}{8}$ of the sphere. We consider two integrals. The first just describes the area of $P(\Delta)$:

$$I_1 := \int_{P(\Delta)} 1 \, \mu(dy) = \int_{\Delta} 1 \, P_* \mu(dz) = \frac{\pi}{2}. \tag{4.1}$$

The second has a weak singularity in $x = e_1$ at one of the vertices of the triangle. By interpreting the integral as a solid angle, see, e.g., [11] or [16], an exact value is readily obtained:

$$I_2 := \int_{P(\Delta)} \frac{\partial s(x,y)}{\partial \nu(y)} \, \mu(dy) = \int_{\Delta} \frac{\partial s(x, P(z))}{\partial \nu(P(z))} \, P_* \mu(dz) = \frac{\pi}{4}. \tag{4.2}$$

Below are two tables indicating the performance of the integration method for varying tolerances "`TolInt`." It is important to note that the calculations were performed with a relative machine precision of $\approx 10^{-7}$.

TABLE 4.1

The first table refers to the area (4.1) *of the spherical triangle. As we expected, the refining of the various triangles is very uniform since no singularities occur. The global* `error` *is significantly bigger than the local errors which are* $\leq$ `TolInt` *since the latter are accumulated. We show statistics of the number of triangles which were accepted via the tolerance test at each level of refinement.*

TolInt =	0.01	0.001	0.0001	0.00001	0.000001
Num. Int. I =	1.5510229	1.5658103	1.5695473	1.5703585	1.5705895
error: $\frac{\pi}{2} - I$ =	0.0197734	0.004986	0.0012490	0.0004378	0.0002068
Level	Triangles	Triangles	Triangles	Triangles	Triangles
1	0	0	0	0	0
2	0	0	0	0	0
3	16	0	0	0	0
4	0	64	0	0	0
5	0	0	256	54	0
6	0	0	0	808	768
7	0	0	0	0	1024

In the above method, it is important to decide what has to be done about the singular values of the integrand f. Let us discuss the replacing of the singular value of f by a big number β; more precisely, we substitute for f the function

$$\tilde{f}(y) := \begin{cases} f(y) & \text{for } |f(y)| \leq \beta, \\ \beta & \text{for } f(y) > \beta, \\ -\beta & \text{for } f(y) < -\beta. \end{cases}$$

If the singular point x is a vertex point of the refinement, then it is easy to see that, due to the tolerance test, the method refines towards this point until approximately

$$\beta\alpha \approx \texttt{TolInt},$$

TABLE 4.2

The next table refers to the integral (4.2). *Since the integrand has a weak singularity at the vertex* $x = e_1$, *we observe an iterated refining process towards this point. However, the overall numerical effort is not influenced very much by this local refinement since most triangles are accepted via the tolerance test at a reasonably low level of refinement.*

TolInt =	0.01	0.001	0.0001	0.00001	0.000001
Num. Int. I =	0.77329779	0.78282577	0.78384632	0.78498906	0.78525865
error: $\frac{\pi}{4} - I =$	0.01210038	0.0025724	0.00155184	0.0004091	0.00013951
Level	Triangles	Triangles	Triangles	Triangles	Triangles
3	15	0	0	0	0
4	3	63	44	0	0
5	3	3	79	191	0
6	3	3	3	259	959
7	4	3	3	3	259
8	0	3	3	3	3
9	0	3	3	3	3
10	0	3	3	3	3
11	0	4	3	3	3
12	0	0	3	3	3
13	0	0	3	3	3
14	0	0	4	3	3
15	0	0	0	3	3
16	0	0	0	3	3
17	0	0	0	4	3
18	0	0	0	0	3
19	0	0	0	0	3
20	0	0	0	0	3
21	0	0	0	0	4

where α is the area of the triangle Δ of the refinement having vertex x. On the other hand, if the integrand f has a singularity of the form

$$f(y) \sim \frac{1}{||x-y||},$$

it it easy to see by using polar coordinates that

$$\int_{P(\Delta)} f(y)\,\mu(dy) = O(\delta),$$

where δ measures the diameter of Δ. Hence the method has been designed to neglect this integral piece if

$$\delta \approx \texttt{TolInt}.$$

By making the coarse approximation

$$\alpha \approx \delta^2,$$

it now becomes clear that it is sufficient to set

$$\beta \approx \frac{1}{\texttt{TolInt}}$$

in order to achieve the desired accuracy through the adaptive refinement. This technique was used in the numerical example (4.2). Numerical experiments confirm the

above discussion: if β is increased further, then only the number of refinements near the singularity increases without essentially improving the accuracy of the global integral (4.2).

5. Extrapolation. It is very suggestive to try an extrapolation method for the case that the integrand in (2.14) is sufficiently smooth on $P(\Delta)$. However, an asymptotic expansion of the composite trapezoidal rule as a theoretical background for an extrapolation method is lacking in our case. Below we conjecture such an expansion theorem which will be confirmed by the numerical experiments. Let the triangle Δ of a triangulation $\mathcal{T}$ of B be given. For each integer $n > 0$, we consider an equidistant subdivision of Δ into n^2 triangles $\{\Delta_i\}_{i=1,\cdots,n^2}$ by drawing $n-1$ parallel lines to each edge of Δ such that each edge is subdivided into n equidistant intervals; see Fig. 2 below for the case $n = 5$.

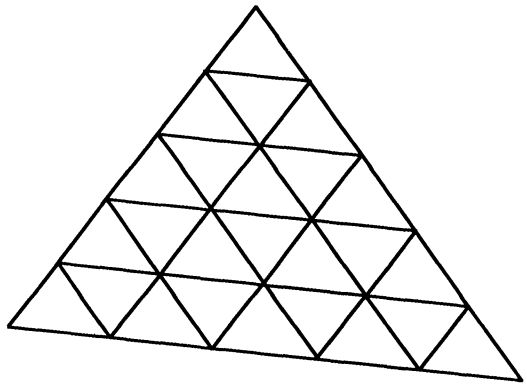


FIG. 2. *Equidistant subdivision of a triangle.*

We now subdivide the integral (2.14) correspondingly:

$$\int_{\Delta} f\big(P(z)\big)\, P_*\mu(dz) = \sum_{i=1}^{n^2} \int_{\Delta_i} f\big(P(z)\big)\, P_*\mu(dz) \tag{5.1}$$

and approximate each summand via the trapezoidal rule (3.1). Thus we obtain a composite trapezoidal rule for approximating (5.1), which we call $I(n)$. Now our conjecture can be formulated.

CONJECTURE 5.1. *Let $k > 0$ be an integer. Under the assumption that the map H and the integrand f are sufficiently smooth, there exist constants $\{c_\kappa\}_{\kappa=1,\cdots,k} \subset \mathbf{R}$ which are independent of the degree of subdivision n such that*

$$\int_{\Delta} f\big(P(z)\big)\, P_*\mu(dz) - I(n) = \sum_{\kappa=1}^{k} \frac{c_\kappa}{n^{2\kappa}} + O\Big(\frac{1}{n^{2k+2}}\Big)\,.$$

For the approximation of one-dimensional integrals

$$\int_0^1 g(t)\, dt$$

via the standard composite trapezoidal rule, such asymptotic expansion theorems are obtained from the famous Euler–Maclaurin sum formula, see, e.g., [17, §3.3]; and

similar formulas were obtained for multiple integrals in [15]; see also [18, §3.17]. Unfortunately, because of the difference between our trapezoidal rule (3.1) and the standard trapezoidal rule (3.2), this does not imply the Conjecture 5.1. This subject will be pursued elswhere.

Example 5.2. We consider again the numerical example (4.1), which we now approximate via our composite trapezoidal rule as described following (5.1). The table below gives the numerical results for increasing degree of subdivision n. The table is given in the typical fashion of the Romberg integration, see, e.g., [17, p. 133]. We note again that the relative machine error of the calculation was $\approx 10^{-7}$.

(5.2)

n	$I(n)$		
4	1.4943614		
8	1.5510229	1.5699101	
16	1.5658103	1.5707395	1.5707948

For the corresponding errors, we obtain the table

(5.3)

n	$\frac{\pi}{2} - I(n)$		
4	0.0764349		
8	0.0197734	0.0008863	
16	0.0049860	0.0000569	0.0000016

This is strong numerical evidence for the validity of Conjecture 5.1.

6. Concluding remarks. The procedure in §3 was written in a concise way in order to make the logical structure easier to understand. However, this was done at the cost of computational efficiency since for many points the retraction "`newton`" and the function evaluation "`integrand`" are calculated twice. To avoid this, a much more complex program must be written. The present paper is merely intended to underline the usefulness of our approach, which differs from standard techniques in integration theory. For planar coordinates, sophisticated and efficient adaptive integration routines have been developed; see, e.g., Friedman and Wright [7] and Kahaner and Rechard [13]. It would be desirable to adapt these or similar techniques to the case which is considered here.

The procedure in §3 is not suitable for use with an extrapolation method. A modification has to be written for this important case. Furthermore, since the computational efficiency of this method mainly depends on the number of triangles which are generated, the Bulirsch numbers $n = 1, 2, 3, 4, 6, 8, 12, \cdots$ should be used; see [17, eq. (3.4.5)(b)]. It is also pointed out there that rational interpolation is more efficient than polynomial interpolation in this context.

For the typical application (2.9) that we have in mind, it is known a priori whether the integrand is singular or not. Note that singularity may also be assumed for numerical purposes if x_k is very near to but not in the piece B_i. It is therefore advisable to make an a priori choice of the integration method: an adaptive refinement procedure as indicated in §3 for the case in which the integrand is singular; and an extrapolation method as indicated in Example 5.2 for the case in which the integrand is sufficiently smooth. Since the triangles T_i are usually tangent to B and very small, very few refinement steps should provide for sufficient accuracy in both cases. Note that our numerical examples (4.1)–(4.2) have a very large triangle. This was chosen on purpose in order to better display the performance of the integration methods.

Acknowledgments. I strongly appreciated some enlightening discussions which I had with Gene Allgower from Colorado State University and with Ron Guenther from Oregon State University.

REFERENCES

[1] E. L. ALLGOWER AND K. GEORG, *Estimates for piecewise linear approximations of implicitly defined manifolds*, Appl. Math. Lett., 1 (1989), pp. 1–7.

[2] ———, *Numerical Continuation Methods: An Introduction*, Springer-Verlag, Berlin, Heidelberg, New York, 1990.

[3] E. L. ALLGOWER AND S. GNUTZMANN, *An algorithm for piecewise linear approximation of implicitly defined two-dimensional surfaces*, SIAM J. Numer. Anal., 24 (1987), pp. 452–469.

[4] K. E. ATKINSON, *The numerical solution of Laplace's equation in three dimensions*, SIAM J. Numer. Anal., 19 (1982), pp. 263–274.

[5] ———, *Numerical integration of the sphere*, J. Austral. Math. Soc. Ser. B, 23 (1982), pp. 332–347.

[6] P. J. DAVIS AND P. RABINOWITZ, *Methods of Numerical Integration*, Second Edition, Academic Press, Orlando, FL, 1984.

[7] J. H. FRIEDMAN AND M. H. WRIGHT, *A nested partitioning procedure for numerical multiple integration*, ACM Trans. Math. Software, 7 (1981), pp. 76–92.

[8] S. GNUTZMANN, *Stückweise lineare Approximation implizit definierter Mannigfaltigkeiten*, Ph.D. thesis, Department of Mathematics, University of Hamburg, Federal Republic of Germany, 1989.

[9] R. B GUENTHER AND J. LEE, *Partial Differential Equations of Mathematical Physics and Integral Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[10] N. M. GÜNTER, *Die Potentialtheorie und ihre Anwendung auf Grundaufgaben der Mathematischen Physik*, B. G. Teubner, Leipzig, German Democratic Republic, 1957.

[11] W. HACKBUSCH, *Integralgleichungen. Theorie und Numerik*, B. G. Teubner, Stuttgart, Federal Republic of Germany, 1989.

[12] M. A. JASWON AND G. T. SYMM, *Integral Equation Methods in Potential Theory and Electrostatics*, Academic Press, New York, 1977.

[13] D. K. KAHANER AND O. W. RECHARD, TWODQD *an adaptive routine for two-dimensional integration*, J. Comput. Appl. Math., 17 (1987), pp. 215–234.

[14] P. KEAST, *Cubature formulas for the surface of the sphere*, J. Comput. Appl. Math., 17 (1987), pp. 151–172.

[15] J. N. LYNESS AND B. J. J. MCHUGH, *Integration over multidimensional hypercubes. 1: A progressive procedure*, Comput. J., 6 (1963), pp. 264–270.

[16] S. G. MIKHLIN, *Linear Integral Equations*, Hindustan Publishing Corp., Delhi, India, 1960.

[17] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, Berlin, Heidelberg, New York, 1980.

[18] A. H. STROUD, *Approximate Calculation of Multiple Integrals*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 2, pp. 454–468, March 1991
© 1991 Society for Industrial and Applied Mathematics
013

A DISCRETE VARIATIONAL GRID GENERATION METHOD*

JOSÉ E. CASTILLO†

Abstract. Variational grid generation techniques are now used to produce grids suitable for solving numerical partial differential equations in irregular geometries. The method considered here controls grid properties that are defined directly over the discrete grid: spacing between the grid points, and area or volume of the grid cells. The grid is produced by solving a large scale nonlinear minimization problem using a conjugate gradient method. The characteristics of the Discrete Variational Method (DV), along with the existence and uniqueness results for the minimization problem, are discussed.

Key words. grid generation, variational methods, conjugate gradient

AMS(MOS) subject classification. 65

1. Introduction.

In the variational methods described by Steinberg and Roache [13], similar to those described by Brackbill and Saltzman [3], three functionals are presented that provide a measure of spacing between the grid lines (smoothness), a measure of the area of the grid cells, and the orthogonality of the grid lines. The minimization problem is solved by calculating the Euler–Lagrange (E–L) equations for the variational problem; a grid is created by solving a centered finite difference approximation of these equations. In theory, a straightforward discretization of the integrals should provide a similar solution; instead, there are serious difficulties [4]. In this approach, the derivatives in the integrals are replaced by centered finite differences and the integrals are replaced by summations over the grid points. Only first derivatives of the coordinates appear in the direct minimization problem, so a centered finite difference discretization at a grid point does not involve values at that point. This produces strong decoupling problems for the direct discretization that are not created in the E–L approach (see [11]).

In the E–L formulation [13], there are certain integral constraints on the solution that are automatically satisfied. In the straightforward discretization, the analog of these constraints is not automatically satisfied. The straightforward discretization approach transforms the smoothness integral into a linear minimization problem with a linear constraint, while the area integral is transformed into a nonlinear minimization problem with a nonlinear constraint [4]. Such problems are much harder to solve than the unconstrained problems that occur in continuous cases. A direct formulation, called the "Discrete Method" (DV), of variational grid generation, is obtained when the properties to be controlled are derived directly from the discrete geometry ([1], [5]). This method will be described below. There have been other efforts in generating grids by the optimization of direct properties (see Kennon and Dulikravich [11]); however, the functionals for the method presented here, as well as its properties and the minimization procedure used, differ considerably from theirs.

As in the E–L approach, the DV method controls three properties of the grid: grid spacing, grid cell areas, and grid orthogonality. Here the grid spacing and cell area functionals will be studied in detail. The effects of adding orthogonality control are currently under study [7]. The grid spacing and area functionals will be described be-

* Received by the editors August 3, 1987; accepted for publication (in revised form) April 11, 1990.

† Department of Mathematical Sciences, San Diego State University, San Diego, California 92182.

low; more detail and analysis can be found in [1] and [5]. First an intuitive description of the functionals will be given, followed by the detailed notation and formulas.

To control the two properties, a weighted combination of both sums is to be minimized. In order to understand the behavior of the DV method, it is important to understand the behavior of the length and area functionals separately. A good understanding of the solution of each minimization problem will provide information relevant to the more general minimization problem that is being considered, i.e., a weighted combination of the two functionals.

In §2, a discrete minimization problem that gives some intuition to the more general grid generation problem is presented; in §3, the direct formulation in two dimensions is presented; in §4, the length control functional is studied; in §5, the area control functional is studied; in §6, the combination of the two properties is discussed and computational examples are given; in §7, the numerical minimization problem is described briefly; then in §8, the performance of the existing codes is discussed. It is concluded that these methods prove to be robust, fast, and capable of generating grids suitable for numerical calculations.

The following notation is used: a grid on a polygonal region Ω is a "subdivision of Ω into quadrilaterals," the vertices of the quadrilaterals are called the "points of the grid," and the quadrilaterals are called the "cells of the grid." For the sake of simplicity, we discuss the approach for rectangles; however, the approach also can be applied to more general regions. The region will have $m+2$ points in the logical "horizontal" direction (m interior points) and $n+2$ points in the logical "vertical" direction (n interior) points; hence, the grid has mn interior points (see Fig. 1). Let $P_{ij} = (x_{ij}, y_{ij})^t$, $1 \le i \le m+2$, and $1 \le j \le n+2$ be the ij point.

The points on the "horizontal" boundaries are $\{P_{r,1}\}_{r=1}^{m+2}$ and $\{P_{s,n+2}\}_{s=1}^{n+2}$. Similarly, the points on the "vertical" boundary are $\{P_{1,r}\}_{r=1}^{n+2}$ and $\{P_{m+2,s}\}_{s=1}^{n+2}$. The mn interior points are $\{P_{r,s}\}_{r,s=2}^{m+1,n+1}$. A vector z of all the $2mn$ coordinates of the mn interior points is needed; i.e., if $P_{r,s} = (x_{r,s}, y_{r,s})$, then

$$z^t = (x_{2,2}, y_{2,2}, x_{2,3}, y_{2,3}, \cdots, x_{m+1,n+1}, y_{m+1,n+1}).$$

Given a grid on Ω, it is desirable to obtain another grid with the same points on the boundary but with the interior points chosen such that the new grid has "optimal" geometrical properties, a grid with the same points on the boundary of Ω and the same number of interior points in Ω that minimize a given functional. The optimality is obtained by minimizing functionals. Two functionals, F_S and F_A, will be introduced below. The functional F_S is used to control the spacing between the grid points, while the functional F_A is used to control the area of the grid cells. To control both spacing between the grid points and areas of the grid cells, the functional,

$$F = \sigma F_S + (1-\sigma) F_A$$

is used where σ is a given number such that $0 \le \sigma \le 1$. In order to understand the

functional F, a study of F_A and F_S is presented below.

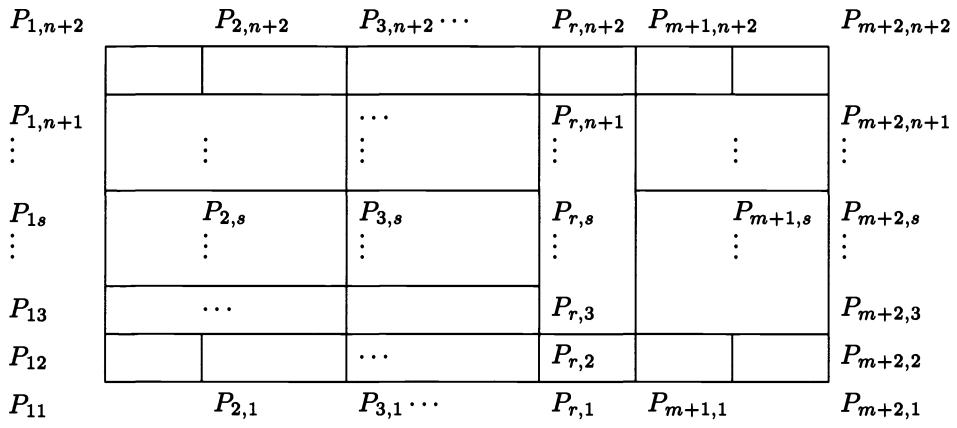


FIG. 1

2. A discrete minimization problem. The following discrete minimization problem in one dimension provides an analog for the general Discrete Variational Method. Let n be a positive integer and let the index i satisfy $1 \leq i \leq n$. Let w_i be a real positive number, then the problem is to find x_i such that $x_1 = 0$, $x_n = 1$, and

$$S = \sum_{i=1}^{n-1} \frac{(x_{i+1} - x_i)^2}{2w_i} \tag{1}$$

is a minimum, and

$$\sum_{i=1}^{n-1} (x_{i+1} - x_i) = 1. \tag{2}$$

This problem can be solved easily by the method of Lagrange multipliers. To do this, set

$$Q = S - \lambda \left(\sum_{i=1}^{n} (x_{i+1} - x_i) - 1 \right). \tag{3}$$

Then the minimum is obtained by solving the system of equations,

$$\partial Q = \frac{(x_{i+1} - x_i)}{w_i} - \lambda = 0, \qquad 1 \leq i \leq n\,,$$

along with (2). The solution of this equation is

$$(x_{i+1} - x_i) = \lambda w_i. \tag{4}$$

Substitute into (4) to get

$$\lambda = \left(\sum_{i=1}^{n} w_i \right)^{-1}. \tag{5}$$

In particular, the solution

$$x_{i+1} - x_i = \lambda w_i$$

says that the distance between the x_i points is proportional to w_i. There are several ways to use this characterization for grid generation [13]. In particular, the grid can be thought of as a set of points joined by straight lines. Two properties of a grid that can be represented by this characterization in two or three dimensions are the distance between the grid points and the area (volume) of the grid cells.

3. Discrete variational formulation in two dimensions. A discrete variational formulation of the grid generation problem is obtained if the properties to be controlled are considered directly. To have the grid points evenly spaced, the material in §2 suggests that the sum of the squares of the segment lengths between the grid points should be minimized:

$$\text{minimize } F_S = \sum s_{ij}^2 \tag{6}$$

with the constraint, see [13],

$$C_S = \sum s_{ij} = \text{constant}. \tag{7}$$

For controlling the area, the sum of the squares of the true discrete area of the quadrilateral cell should be minimized:

$$\text{minimize } F_A = \sum A_{ij}^2 \tag{8}$$

with the constraint

$$C_A = \sum A_{ij} = \text{constant}. \tag{9}$$

Note that in the DV method for spacing between the grid points, the constraint is automatically satisfied since the sum of all the segments is a telescopic sum which depends only on the values on the boundary. For the area sum, the constraint is the sum of the areas of the true quadrilateral cells, which is shown to be the total area of the region, and this solely depends on the values on the boundary. Therefore, this discrete direct variational formulation is an analog of the continuous variational formulation introduced by Steinberg and Roache [13].

4. Functional F_S for length control. In order to control the lengths of the logical "horizontal" segments between the grid points, consider the functional S_H given by

$$S_H = \sum_{i=1}^{m+1} \sum_{j=2}^{n+1} \|P_{i+1,j} - P_{i,j}\|_2^2 \,. \tag{10}$$

Similarly, consider the functional S_V given by

$$S_V = \sum_{i=2}^{m+1} \sum_{j=1}^{n+1} \|P_{i,j+1} - P_{ij}\|_2^2 \,. \tag{11}$$

S_V allows the lengths of the "vertical" segments between the points of the grid to be controlled (see Fig. 1). Hence, the functional F_S can be written

$$F_S(z) = S_H(z) + S_V(z) \,. \tag{12}$$

There are three approaches for obtaining the properties of the functional F_S ([1], [5]). One of these approaches is presented here. The following lemma allows the computation of the gradient and the Hessian of F_S.

LEMMA 1. *Let* $A^t = (x_a, y_a)$, $B^t = (x_b, y_b)$, $C^t = (x_c, y_c)$, $D^t = (x_d, y_d)$, *and* $P^t = (x_p, y_p)$*be given points in* $\mathbf{R}^2$; *and let* $\tilde{f}$ *be the the following function.*

$$\tilde{f} = \|P - A\|_2^2 + \|P - B\|_2^2 + \|P - C\|_2^2 + \|P - D\|_2^2 \,. \tag{13}$$

If

$$\frac{\partial \tilde{f}}{\partial P} = \begin{pmatrix} \frac{\partial \tilde{f}}{\partial x_p} \\ \frac{\partial \tilde{f}}{\partial y_p} \end{pmatrix},$$

$$\frac{\partial^2 \tilde{f}}{\partial A \partial P} = \begin{pmatrix} \frac{\partial^2 \tilde{f}}{\partial x_a \partial x_p} & \frac{\partial^2 \tilde{f}}{\partial y_a \partial x_p} \\ \frac{\partial^2 \tilde{f}}{\partial x_a \partial y_p} & \frac{\partial^2 \tilde{f}}{\partial y_a \partial y_p} \end{pmatrix},$$

then

$$\frac{\partial \tilde{f}}{\partial P} = 2[(P - A) + (P - B) + (P - C) + (P - D)] \,,$$

$$\frac{\partial^2 \tilde{f}}{\partial P^2} = 8 I_2 \,,$$

$$\frac{\partial^2 \tilde{f}}{\partial A \partial P} = \frac{\partial^2 \tilde{f}}{\partial B \partial P} = \frac{\partial^2 F}{\partial C \partial P} = \frac{\partial^2 F}{\partial D \partial P} = -2 I_2 \,.$$

Here I_2 *is the* 2×2 *identity matrix.*

Proof. This is a straightforward computation.

THEOREM 1. *Let* g_S *denote the gradient of* F_S *and* H_S *as its Hessian, then*

$$g_S^t = (g_1^t, g_2^t, \cdots, g_k^t, \cdots, g_{mn}^t) \tag{14}$$

where $k = n(i-2) + j - 1$ *for* $2 \le i \le m+1$, $2 \le j \le n+1$ *and*

$$g_k = \frac{\partial F_S}{\partial P_{i,j}} = 2[(P_{i,j} - P_{i-1,j}) + (P_{i,j} - P_{i,j-1}) + (P_{i,j} - P_{i+1,j}) + (P_{i,j} - P_{i,j+1})] \,.$$

Also,

$$\frac{1}{2} H_S = L_{mn} \otimes I_2, \tag{15}$$

where L_{mn} *is a block tridiagonal matrix given by*

$$L_{mn} = \underbrace{\begin{bmatrix} T_n & -I_n & 0 & & & \\ -I_n & T_n & -I_n & 0 & & \\ & & & & & 0 \\ & & & & & -I_n \\ & & & 0 & -I_n & T_n \end{bmatrix}}_{m\text{-blocks}} .$$

I_n is the identity matrix of order n, and T_n is the n-order tridiagonal matrix:

$$T_n = \begin{bmatrix} 4 & -1 & 0 & & & \\ -1 & 4 & -1 & 0 & & \\ & & & & & 0 \\ & & & & & -1 \\ & & & 0 & -1 & 4 \end{bmatrix}.$$

$L_{mn} \otimes I_2$, the Kronecker product of L_{mn} and I_2, is an $mn \times mn$ array of 2×2 blocks with the (u,v)th block being $\lambda_{uv} I_2$, where $L = [\lambda_{uv}]$ (see [2]).

Proof. Given $P_{i,j}$, there are only four vertices $P_{i-1,j}$, $P_{i,j-1}$, $P_{i+1,j}$, and $P_{i,j+1}$ connected to it ([1], [5]), so the rest follows.

COROLLARY 1. (a) *H_S is positive definite.*

(b) *F_S has a unique minimum and at the minimum, the points of the grid satisfy*

$$P_{i,j} = \frac{P_{i-1,j} + P_{i,j-1} + P_{i+1,j} + P_{i,j+1}}{4}.$$

Proof. It is well known [10] that L_{mn} is positive definite; hence L_{mn} can be written as

$$L_{mn} = U^t U; \tag{16}$$

then

$$\begin{aligned} \frac{1}{2} H_S &= U^t U \otimes I_2 = (U^t \otimes I_2)(U \otimes I_2) \\ &= (U \otimes I_2)^t (U \otimes I_2). \end{aligned}$$

This last statement is derived from the properties of the Kronecker product [2]. Because $U \otimes I_2$ is nonsingular, H_S is positive definite. The functional has positive definite Hessian, so it has a unique minimum given by the zero of the gradient. Note that L_{mn} is the usual second order discrete Laplacian; therefore, the length control functional produces the same solution as the smoothness integral from the Steinberg and Roache method.

5. Functional F_A for area control. Let F_A denote the sum of the squares of the area of the grid cells; i.e.,

$$F_A = \sum_{i=1}^{m+1} \sum_{j=1}^{n+1} A_{i,j}^2, \tag{17}$$

where $A_{i,j}$ is the area of (i,j)th cell. In order to be precise, the (i,j)th cell of the grid is the "oriented quadrilateral," $P_{i,j}$, $P_{i+1,j}$, $P_{i+1,j+1}$, $P_{i,j+1}$.

5.1. Oriented polygons. The properties of the area of oriented polygons play an important role in the results; hence, some general results for oriented polygons are given.

DEFINITION. Given the oriented triangle PQR, its area $A(PQR)$ is given by the following formula:

$$A(PQR) = \frac{1}{2} \det[Q - P, R - P]. \tag{18}$$

Here $P = (x_p, y_p)^t$, $Q = (x_q, y_q)^t$, and $R = (x_r, y_r)^t$. It is easy to show that $A(PQR)$ is positive if the triangle is oriented counterclockwise and negative if it is oriented clockwise.

DEFINITION. Given an oriented n-polygon $P_1, P_2, \cdots, P_n$, its area $A(P_1, P_2, \cdots, P_n)$ is defined as follows:

$$A(P_1, \cdots, P_n) = \sum_{i=2}^{n-1} A(P_1, P_i, P_{i+1}) \,. \tag{19}$$

The following theorem shows that the above definition is a "good" definition.

THEOREM 2. *Given an oriented n-polygon* $P_1, P_2, \cdots, P_n$,

$$A(P_1, P_2, \cdots, P_n) = \frac{1}{2} \sum_{i=1}^{n} \det(P_i, P_{i+1}), \quad \textit{where} \quad P_{n+1} = P_1 \tag{20}$$

and

$$A(P_1, P_2, \cdots, P_n) = A(P_k, P_{k+1}, \cdots, P_n, P_1, P_2, \cdots, P_{k-1}). \tag{21}$$

Proof. For the proof, see [1] and [5].

The following theorem states that the area function is additive.

THEOREM 3. *Given the oriented n-polygon* $P_1, P_2, \cdots, P_n$ *and another oriented polygon* $P_n, Q_1, Q_2, \cdots, Q_m, P_1$ *that have a common side* P_1P_n, *but with opposite orientation, the sum of the areas of the polygons is equal to the area of the polygon formed with the union of the two polygons; i.e.,*

$$A(P_1, \cdots, P_n) + A(P_n, Q_1, Q_2, \cdots, Q_m, P_1) = A(P_1, \cdots, P_n, Q_1, \cdots, Q_m) \,. \tag{22}$$

Proof. For the proof, see [1] and [5].

THEOREM 4. *Given a grid on a polygonal region* Ω,

$$\sum_{i=1}^{m+1} \sum_{j=1}^{n+1} A_{ij} = \textit{Area of } \Omega, \tag{23}$$

where $A_{ij} = A(P_{i,j}, P_{i+1,j}, P_{i+1,j+1}, P_{i,j+1})$.

Proof. The proof is a long but easy consequence of the previous theorem.

Note that this is the analogous constraint for the area problem; since the area of the region is constant, the constraint is always satisfied.

5.2. Gradient and Hessian of the area functional. The means are now available to state the properties of the functional F_A. It is important to notice that there are $(m+1)(n+1)$ areas and that there are mn interior points in the grid; that is, there are $2mn$ unknown grid coordinates. Let a be the column vector of order $(m+1)(n+1)$ whose components are the areas of the cells of the grid; i.e.,

$$a^t = (A_{1,1}, A_{1,2}, \cdots, A_{1,n+1}, A_{2,1}, A_{2,2}, \cdots, A_{2,n+1}, \cdots, A_{m+1,1}, \cdots, A_{m+1,n+1})$$

where

$$A_{ij} = A(P_{ij}, P_{i+1,j}, P_{i+1,j+1}, P_{i,j+1}) \,;$$

then, the functional can be written

$$F_A = \|a\|_2^2 . \tag{24}$$

The following lemma will make it easy to compute the gradient and the Hessian of F_A.

LEMMA 2. *Let $\alpha = A(P,Q,R,S)$ be the area of the oriented quadrilateral P,Q,R,S. Then*

(a) $\alpha = A(P,Q,R,S) = \frac{1}{2}\det(P-R, Q-S)$.

(b)

$$2\frac{\partial\alpha}{\partial P} = J_2(Q-S), \qquad 2\frac{\partial\alpha}{\partial Q} = J_2(R-P) ,$$
$$2\frac{\partial\alpha}{\partial R} = J_2(S-Q), \qquad 2\frac{\partial\alpha}{\partial S} = J_2(P-R) ,$$

where

$$J_2 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} .$$

(c)

$$\frac{\partial^2\alpha}{\partial P^2} = \frac{\partial^2\alpha}{\partial Q^2} = \frac{\partial^2\alpha}{\partial R^2} = \frac{\partial^2\alpha}{\partial S^2} = 0 ,$$
$$2\frac{\partial^2\alpha}{\partial S\partial P} = 2\frac{\partial^2\alpha}{\partial R\partial S} = -J_2$$
$$2\frac{\partial^2\alpha}{\partial Q\partial P} = 2\frac{\partial^2\alpha}{\partial R\partial Q} = J_2$$
$$\frac{\partial^2\alpha}{\partial R\partial P} = \frac{\partial^2\alpha}{\partial S\partial Q} = 0 .$$

Proof. For the proof, see [1] and [5]. The formulas for the gradient and Hessian can now be computed.

THEOREM 5. *Let g_A be the gradient of F_A and let H_A be its Hessian. Then*

$$g_A = 2G^t a, \tag{25}$$

where G^t is a $2mn$ by $(m+1)(n+1)$ matrix whose columns are the gradients of areas of the cells of the grid,

$$G^t = [g_{11}|g_{12}|\cdots|g_{i,j}|\cdots|g_{m+1,n+1}]$$

and g_{ij} = gradient of A_{ij}.

$$\frac{1}{2}H_A = G^tG + \sum_{i=1}^{m+1}\sum_{j=1}^{n+1} A_{ij}H_{ij}, \tag{26}$$

where H_{ij} is the Hessian of A_{ij}.

Proof. For the proof, see [1] and [5].

5.3. Structure of the Hessian of the area functional. Some additional structure will now be given for the second term in the formulas for the Hessian. Consider the matrix

$$W = 2\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} A_{ij}H_{ij} = (\gamma_{r,s}), \tag{27}$$

where $\gamma_{r,s}$ are 2×2 blocks. Then

$$\gamma_{rs} = 2\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} A_{ij}\frac{\partial^2 A_{ij}}{\partial P_{u,v}\partial P_{k,\ell}}.$$

Defining

$$\varphi(k,l) = n(k-2) + \ell - 1\ ,$$

note that,

$$2\frac{\partial^2 A_{i-1,j}}{\partial P_{i,j+1}\partial P_{ij}} = +J_2\ ,$$
$$2\frac{\partial^2 A_{i,j}}{\partial P_{i,j+1}\partial P_{i,j}} = -J_2,$$

and

$$2\frac{\partial^2 A_{i,j-1}}{\partial P_{i+1,j}\delta P_{ij}} = -J_2\ ,$$
$$2\frac{\partial^2 A_{ij}}{\partial P_{i+1,j}\partial P_{ij}} = J_2\ ;$$

otherwise

$$\frac{\partial^2 A_{ij}}{\partial P_{u,v}\partial P_{k,\ell}} = 0\ .$$

Now if $\varphi(i,j) = r$, then $\varphi(i,j+1) = r+1$ and $\varphi(i-1,j) = r-n$; therefore,

$$\begin{aligned}\gamma_{r,r+1} &= A_{i-1,j}2\frac{\partial^2 A_{i-1,j}}{\partial P_{i,j+1}\partial P_{ij}} + A_{i,j}2\frac{\partial^2 A_{ij}}{\partial P_{i,j+1}\partial P_{ij}}\\ &= (A_{i-1,j}J_2 - A_{ij}J_2)\\ &= J_2(A_{i-1,j} - A_{ij})\ .\end{aligned}$$

Similarly

$$\gamma_{r,r+n} = J_2(A_{ij} - A_{i,j-1})\ ,$$

but notice that

$$\gamma_{n,n+1} = \gamma_{2n,2n+1} = \cdots = \gamma_{n,3n+1} = 0$$

because $P_{k,n+1}$ and $P_{k+1,2}$ are consecutive but are not adjacent. Note that

$$\begin{aligned}\gamma_{r+1,r} &= \gamma_{r,r+1}^t = [J_2(A_{i-1,j} - A_{ij})]^t \\ &= J_2^t(A_{i-1,j} - A_{ij}) \\ &= -J_2(A_{i-1,j} - A_{ij}) = -\gamma_{r,r+1} \\ &= -\gamma_{r,r+1}\,.\end{aligned}$$

All of the above implies that W has the following block structure:

$$W = \begin{pmatrix} T_{11} & T_{12} & 0 & & \\ -T_{12} & T_{22} & T_{23} & & \\ 0 & -T_{23} & T_{33} & & \\ & & & \ddots & T_{m-1,m} \\ & & & -T_{m-1,n} & T_{m,m} \end{pmatrix} \otimes J_2$$

or

$$W = T \otimes J_2,$$

where

$$T_{r,r} = \begin{pmatrix} 0 & \tau_1^{(r)} & 0 & & \\ -\tau_1^{(r)} & 0 & \tau_2^{(r)} & & \\ 0 & -\tau_2^{(r)} & 0 & & \\ \vdots & \vdots & \ddots & & \\ & & & & -\tau_{n-1}^{(r)} \\ 0 & 0 & & -\tau_{n-1}^{(r)} & 0 \end{pmatrix},$$

and

$$T_{r,r+1} = \operatorname{diag}(\tilde{\tau}_q^{(r)}).$$

In order to find an expression for $\tau_q^{(r)}$ and $\tilde{\tau}_q^{(r)}$, it is necessary to use the fact that

$$\varphi(i,j) = n(i-2) + j - 1$$

so

$$\begin{aligned} k = n(r-1) + q \quad &\Longrightarrow \\ r = i - 1\,; \qquad q &= j - 1, \\ \tau_q^{(r)} = A_{r,q+1} - A_{r+1,q+1}, &\qquad 1 \le r \le m\,, \\ \tilde{\tau}_q^{(r)} = A_{r+1,q+1} - A_{r+1,q}, &\qquad 1 \le q \le n\,, \end{aligned}$$

so the following theorem has been proved.

THEOREM 6.

$$2\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} A_{ij}H_{ij} = T \otimes J_2, \tag{28}$$

where T is $mn \times mn$ is block tridiagonal and antisymmetric, $T_{r,r}$ is tridiagonal antisymmetric with the form given above, and $T_{r,r+1}$ is diagonal with the form,

$$T_{r,r+1} = \operatorname{diag}(\tilde{\tau}_q^{(r)}),$$

where

$$\tau_q^{(r)} = A_{r,q+1} - A_{r+1,g+1}\,, \qquad \begin{cases} 1 \le r \le m \\ 1 \le q \le n-1 \end{cases},$$

$$\tilde{\tau}_q^{(r)} = A_{r+1,q+1} - A_{r+1,q}\,, \qquad \begin{cases} 1 \le r \le m \\ 1 \le q \le n-1 \end{cases}.$$

COROLLARY 2.

$$H_A = 2G^tG + T \otimes J_2,$$
$$H_A = \frac{1}{2}(I_{mn} \otimes J_2)\tilde{G}\tilde{G}^t(I_{mn} \otimes J_2)^t + T \otimes J_2.$$

5.4. Analysis of the area problem. The formulas for the gradient and Hessian can be used to show some interesting results about minimizing the area functional. Let us look at the case where the grid subdivides the region into equal area cells. The vector with all elements equal to 1 is a solution to the critical points equation: $G^t\vec{1} = 0$ for all z. This is easily seen to be true because

$$\sum_{i=1}^{m+1}\sum_{j=1}^{n+1} A_{i,j} = \textit{Area of } \Omega = a(\Omega)$$
$$\Longrightarrow \qquad \mathbf{a}^t \cdot \vec{1} = \textit{Area of } \Omega = a(\Omega)$$

therefore

$$\frac{\partial \mathbf{a}^t}{\partial z_k}\vec{1} = 0 \quad \text{for all} \quad k \Longrightarrow G^t\vec{1} = 0\,.$$

CLAIM. *If $\tilde{z}$ is such that*

$$a(\tilde{z}) = \alpha \cdot 1,$$

then

$$\alpha = \frac{\textit{Area of } \Omega}{(n+1)(m+1)} = \frac{a(\Omega)}{(n+1)(m+1)},$$
$$g_A'(\tilde{z}) = 0,$$
$$H_A(\tilde{z}) \ge 0,$$
$$F_A(\tilde{z}) = \alpha a(\Omega);$$

i.e., $\tilde{z}$ is a minimum of F_A.

Proof. $g_A = G^t a$ so $g_A(\tilde{z}) = 0$. Also,

$$a = \alpha\vec{1} \Longrightarrow \alpha = \frac{\textit{Area of } \Omega}{(n+1)(m+1)}\,.$$

Finally, it has been shown that

$$H_A = 2G^tG + T \otimes J_2,$$

where the elements of T depend only on the difference of $A_{i,j} - A_{r,s}$, but since in this case, all the cells have the same area, $T = 0$ and

$$H_A = 2G^tG \geq 0 .$$

Thus the area functional will have a semipositive definite Hessian when the second term in its Hessian is small enough.

6. Control of the two properties. When control over the two properties is desired, the problem is to minimize the functional

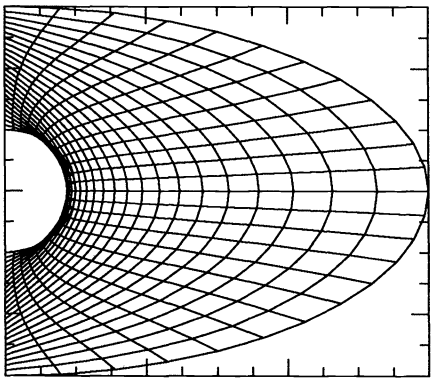
$$F = \sigma F_S + (1 - \sigma)F_A,$$

where σ is a given number such that $0 \leq \sigma \leq 1$. For this problem the Hessian can be written as

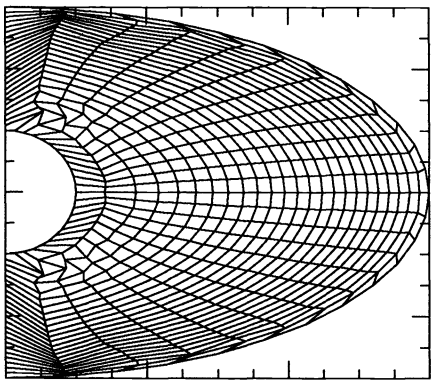
$$H = \sigma H_S + (1 - \sigma)G^tG + (1 - \sigma)W .$$

Recall that H_S is constant and positive definite, that G^tG is semipositive definite, and that W is indefinite but is a sum of products of the areas of the cells multiplied by differences of areas of neighboring cells. Thus W will be very small for fine grids or grids that have nearly equal areas, so $\sigma \neq 0$ can be chosen ([1], [8]), such that the Hessian will be positive definite and then the minimization problem will have a unique solution.

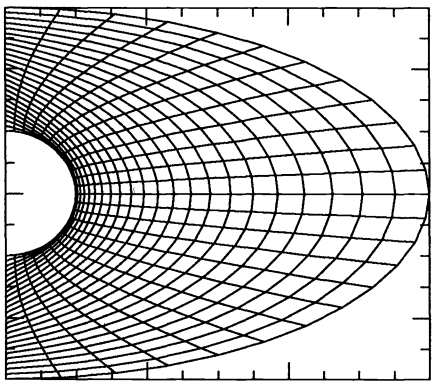
To illustrate the effects of combining the functionals, Figs. 2–3 are presented. The elliptical region used is considered difficult for all grid generators. In Figs. 2(a) to 2(c) the grid is 25×25. Note that the 25×25 grid contains 1058 unknown coordinates; this is a large scale minimization problem. The subfigures are labelled with an a, b, or c; a means that only length control is used to generate the grid, b means only area control is used; while c means that a combination of the two functionals is used; in this example, $\sigma = 0.1$. Note that pure length control tends to fold the grid near the inner ellipse and that pure area control tends to produce nonconvex cells near the inner ellipse; while the combination produces grids suitable for computation. It is believed that nonconvex cells in the pure area control grids are caused by the distribution of the points on the boundary. In Figs. 3(a) to 3(c) the region (horn) is more difficult than the elliptical one since it has no equal area solution. Here the area functional has serious difficulties and folds the grid. A combination of the two functionals produces an unfolded grid.



(a)

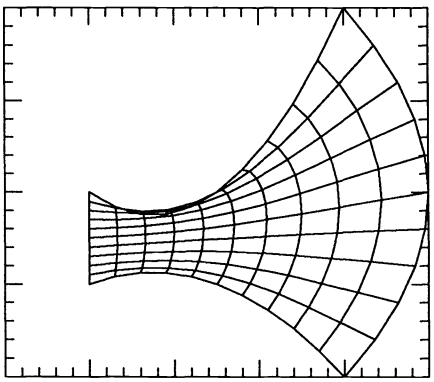


(b)

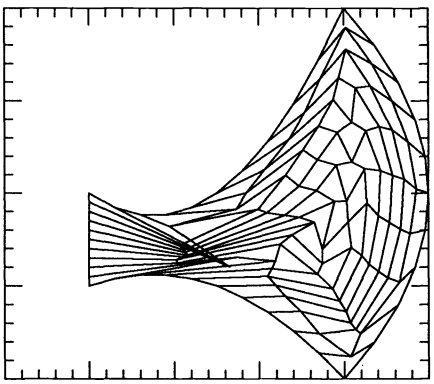


(c)

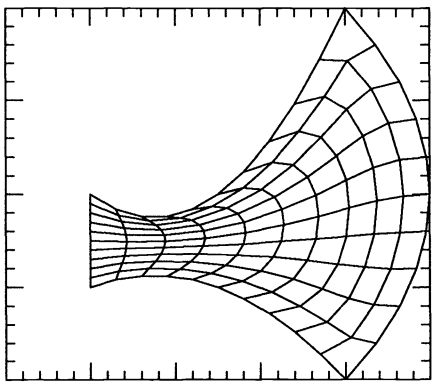
FIG. 2



(a)



(b)



(c)

FIG. 3

7. Minimization procedure. The minimization problem associated with the Discrete Variational Method can be solved by a nonlinear conjugate gradient method (see [1], [5], [9], [12]). The minimization problem can be posed as a least squares problem. For the length functional, the functional is quadratic while its gradient is linear. The number of lengths is greater than or equal to the number of unknown variables; therefore, any least squares solver can do a good job. In the case of the area control functional, the functional is quartic while the gradient is cubic; thus, this problem is truly nonlinear. The number of areas becomes almost immediately (e.g., for a 5×5 grid) smaller than the number of variables. Consequently, the Jacobian of the functional is rank deficient. For both functionals or for a combination, the number of variables becomes large very rapidly. Consequently, most standard solvers will have difficulty with these problems. In addition, any solver that stores the full gradient or Hessian will have serious storage limitations.

It is important to note here that once the value of either the length or area functional have been computed, all of the information needed to compute the gradient and the Hessian have been obtained. In addition, the Hessian is very sparse and the nonzero entries are easily located using the results in this paper. These are key factors for a preconditioned conjugate gradient implementation. Our current codes use a conjugate gradient with inexact line search ([1], [5], [12]). Codes are in the process of being developed for generating two- and three-dimensional grids that take full advantage of the theory presented in this paper. These codes will be competitive with other grid generation codes.

8. Performance. Many examples have been generated using the code described in the previous section (see Fig. 2(a)–3(c)). The performance has been excellent, giving a substantial check on the theory presented in this paper. In addition, the generated grids have been compared to those computed by Steinberg and Roache [13]. The length functional produces the same grids as their smoothness functional, the area functional produces grids that are close to, but not the same as, their area (volume) functional. It it thought that the differences are due to the truncation errors made in their approximation of the Euler–Lagrange equations by finite differences. An appropriate combination of the two functionals produces grids as suitable for numerical calculations as the Steinberg and Roache method does; however, the method presented here is considerably faster than theirs.

To check how fast the Discrete Method is for a model problem (see Figs. 2(a)–3(c)), the current version of the method was tested against an implementation of the homogeneous Thompson, Thames, and Mastin method (TTM)[14], or Winslow method [15], which solves a coupled system of elliptic equations using the Successive Overrelaxation Method (SOR).

The tolerance used for the TTM is 10-4 ; also, the use of one SOR iteration or two SOR iterations per outer (nonlinear) iteration was considered. In the Discrete Method, the sigma used for the combination of the two functionals was 0.1 for this test. The parameters for the minimization part were 10-5 tolerance for the gradient and the line search [12]. The number of nonlinear iterations (iter) as well as the time in seconds is presented for three differently sized grids for the model problem. This test was done in a Sun-3 workstation [6]. Although none of the methods have been highly tuned for speed, it seems clear that the present method is at least competitive with the most commonly used elliptic grid generation schemes.

9. Conclusions. The Discrete Variational Grid Generation Method presented has the capability of generating grids suitable for numerical calculations. The the-

ory presented here allows the development of efficient algorithms and, in addition, produces considerable insight into the nature of the minimization problems. Also, another functional can be added to obtain control over the orthogonality between the grid lines [7]. Moreover, it is clear that the method has a natural, but not straightforward, extension to three dimensions, and will have a structure analogous to the two-dimensional problem. Thus, it is reasonable to expect that the three-dimensional method will also be competitive with standard grid generation methods.

Work is proceeding on a Preconditioning Conjugate Gradient implementation which is expected to be faster, a truncated Newton implementation to take advantage of the structure of the Hessian, a parallel conjugate gradient implementation, and the solution adaptivity of the grid. This will clearly enhance the capability of the Discrete Grid Generation Method.

Acknowledgment. The author thanks Professor Pablo Barrera-Sanchez for his valuable suggestions.

REFERENCES

[1] P. BARRERA-SANCHEZ AND J. E. CASTILLO, *A large scale optimization problem arising from numerical grid generation.* Tech. Report 1, Institute for Computational Research, Department of Mathematics and Statistics, University of New Mexico, Albuquerque, NM, 1987.

[2] R. E. BLAHUT, *Fast Algorithms For Digital Signal Processing,* Addison-Wesley, New York, 1985.

[3] J. U. BRACKBILL AND J. S. SALTZMAN, *Adaptive zoning for singular problems in two dimensions,* J. Comput. Phys. , 46 (1982), pp. 342–368.

[4] J. E. CASTILLO, *Mathematical aspects of variational grid generation* I, in Numerical Grid Generation in Computational Fluid Dynamics, J. Hauser and C. Taylor, eds. , Pineridge Press Limited, Swansea, U. K. , 1986.

[5] ———, *On variational grid generation,* Ph. D. thesis, University of New Mexico, Albuquerque, NM, 1987.

[6] ———, *A direct variational grid generation method,* in Proc. Sixth IMACS International Symposium on Computer Methods for Partial Differential Equations, Lehigh University, Bethlehem, PA, June 1987.

[7] ———, *A direct variational grid generation method: Orthogonality control,* in Numerical Grid Generation in Computational Fluid Dynamics, S. Senguta, J. Hauser, P. R. Eiseman, and J. F. Thompson, eds. , Pineridge Press Limited, Swansea, U. K. , 1988.

[8] J. E. CASTILLO, S. STEINBERG, AND P. J. ROACHE, *Parameter estimation in variational grid generation,* Appl. Math. Comp. , 28 (1988), pp. 155–178.

[9] R. FLETCHER, *Unconstrained Optimization* 1, John Wiley, New York, 1980.

[10] E. ISAACSON AND H. B. KELLER, *Analysis of Numerical Methods,* John Wiley, New York, 1966.

[11] S. R. KENNON AND G. S. DULIKRAVICH, *A posteriori optimization of computational grids,* AIAA paper 85-0483, AIAA 23rd Aerospace Science Meeting, American Institute of Aeronautics and Astronautics, Reno, NV, January 1985.

[12] D. F. SHANNO, *Conjugate gradient methods with inexact searches,* Math. Oper. Res. , 3(1978), pp. 244–256.

[13] S. STEINBERG AND P. J. ROACHE, *Variational grid generation,* Numerical Methods for Partial Differential Equations, 2 (1986), pp. 71–96.

[14] J. F. THOMPSON, F. C. THAMES, AND C. E. MASTIN, *Automatic numerical generation of body-fitted coordinate systems for a field containing any number of arbitrary two-dimensional bodies.* J. Comput. Phys. , 15 (1974), pp. 299–319.

[15] A. M. WINSLOW, *Numerical solution of the quasilinear Poisson equation in a nonuniform triangle mesh,* J. Comput. Phys. , 2(1967), pp. 149–172.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 469-487, May 1991
© 1991 Society for Industrial and Applied Mathematics

PARALLEL HOMOTOPY ALGORITHM FOR THE SYMMETRIC TRIDIAGONAL EIGENVALUE PROBLEM*

TIEN-YIEN LI†, HONG ZHANG‡, AND XIAN-HE SUN§

Abstract. The parallel homotopy algorithm for finding few or all eigenvalues of a symmetric tridiagonal matrix is presented. The computations were executed on an NCUBE, a distributed memory multiprocessor. The numerical results show that the performance of our algorithm is strongly competitive with "divide and conquer" and bisection/multisection algorithms. The almost 100 percent efficiency seems to suggest that the natural parallelism of the homotopy method makes the algorithm an excellent candidate for a variety of architectures.

Key words. eigenvalues, multiprocessors, tridiagonal matrices, homotopy continuation method

AMS(MOS) subject classification. 65F15

1. Introduction. The homotopy continuation method can be used to solve eigenvalue problems [11]. Let **A** be a real symmetric and tridiagonal of the form

$$\mathbf{A}=\begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \ddots & \ddots & \\ & \ddots & & \beta_n \\ & & \beta_n & \alpha_n \end{bmatrix}. \tag{1.1}$$

If some $\beta_i = 0$, then $\mathbf{R}^n$ clearly decomposes into two complementary subspaces invariant under **A**. Thus the eigenproblem decomposes in an obvious way into two smaller subproblems. Therefore we will assume that each $\beta_i \neq 0$ and **A** is said to be *unreducible.* Let

$$\mathbf{H}: \mathbf{R}^n \times \mathbf{R} \times [0, 1] \to \mathbf{R}^n \times \mathbf{R}$$

be defined by

$$\begin{aligned} \mathbf{H}(\mathbf{x}, \lambda, t) &= (1-t)\begin{bmatrix} \lambda\mathbf{x}-\mathbf{D}\mathbf{x} \\ (\mathbf{x}^T\mathbf{x}-1)/2 \end{bmatrix} + t\begin{bmatrix} \lambda\mathbf{x}-\mathbf{A}\mathbf{x} \\ (\mathbf{x}^T\mathbf{x}-1)/2 \end{bmatrix} \\ &= \begin{bmatrix} \lambda\mathbf{x}-[(1-t)\mathbf{D}+t\mathbf{A}]\mathbf{x} \\ (\mathbf{x}^T\mathbf{x}-1)/2 \end{bmatrix}. \end{aligned} \tag{1.2}$$

Here **D** is the *initial matrix*, a specially chosen symmetric matrix such that all its eigenpairs are available. If **D** is chosen in such a way that $\mathbf{A}(t) \equiv (1-t)\mathbf{D} + t\mathbf{A}$ is an unreducible tridiagonal matrix for $t \neq 0$, then the solution set of $\mathbf{H}(\mathbf{x}, \lambda, t) = 0$ consists of $2n$ disjoint smooth curves, each of which can be parameterized by t and joins an eigenpair of **D** to one of **A** [4]. We call each of these curves $(\mathbf{x}(t), \lambda(t))$ a *homotopy curve* or *eigenpath.* Due to the symmetry of the solutions, only n of the curves need to be followed to find all eigenpairs of **A**. The algorithm has been successfully implemented in [4]. Preliminary computations indicate that the continuation approach to the symmetric eigenvalue problem may yield a reliable $O(n^2)$ algorithm for computing all eigenvalue-eigenvector pairs of a symmetric tridiagonal matrix.

* Received by the editors March 27, 1989; accepted for publication (in revised form) February 6, 1990. This research was supported in part by National Science Foundation grant DMS-8902663.

† Department of Mathematics, Michigan State University, East Lansing, Michigan 48824-1027.

‡ Department of Mathematical Sciences, Clemson University, Clemson, South Carolina 29634-1907.

§ Department of Computer Science, Michigan State University, East Lansing, Michigan 48824-1027. Present address, Ames Laboratory, Iowa State University, Ames, Iowa 50011-3020.

Modern scientific computing is marked by the advent of vector and parallel computers, and the search for algorithms that are to a large extent parallel in nature. The homotopy algorithm is to a large degree parallel, in the sense that each eigenpath is followed independently of the others. In this respect, it stands in contrast to the highly serial QR algorithm. Thus, given n processors, the continuation method has the potential to find all eigenpairs of a symmetric tridiagonal matrix in $O(n)$ time.

The purpose of this paper is to present a parallel homotopy algorithm for tracing the eigenpaths of $\mathbf{H}(\mathbf{x}, \lambda, t) = 0$. To begin, in § 2 we remedy the ill-conditioning problem of the serial homotopy algorithm suggested in [4]. The numerical scheme in [4] follows each curve in $O(n)$ operations when the separation of eigenvalues is sufficient. When $\mathbf{A}(t)$ has close eigenvalues for some $t \in [0, 1]$, the method slows down, and $O(n)$ path-following time can be lost. When such situations occur, we trace, in our algorithm, the eigenspace corresponding to the cluster of eigenvalues of $\mathbf{A}(t)$ instead. The eigenspace is well conditioned if the associated cluster of eigenvalues is well separated from all the other eigenvalues [6]. This leads to a substantial improvement of the algorithm in [4].

Second, we use the "divide-and-conquer" principle in our algorithm. To be more precise, we "divide" the matrix $\mathbf{A}$ in (1.1) by choosing the initial matrix

$$\mathbf{D} = \begin{bmatrix} \mathbf{A}_0 & & & \\ & \mathbf{A}_1 & & \\ & & \ddots & \\ & & & \mathbf{A}_{p-1} \end{bmatrix}, \tag{1.3}$$

where $\mathbf{A}_i$ $(i = 0, \cdots, p-1)$ are $n/p \times n/p$ diagonal blocks of $\mathbf{A}$ (for simplicity, we assume n/p is an integer). The eigenpairs of $\mathbf{A}_i$ are solved first. This is similar to the "divide" step of Cuppen's method [1]. In a method different from Cuppen's method, we "conquer" the matrix $\mathbf{A}$ by following the homotopy curves of $\mathbf{H}(\mathbf{x}, \lambda, t) = 0$ in (1.2) with $\mathbf{D}$ in (1.3). The major advantage of the homotopy method is that our "conquer" step requires almost no communication of the processors. The details of the parallel algorithm will be described in § 4.

In [4], $\mathbf{D}$ is always chosen as a diagonal matrix with the benefit that the derivatives of $\lambda(t)$ and $\mathbf{x}(t)$ are immediately available at $t = 0$. In § 3, we show that when $\mathbf{D}$ is a block diagonal matrix as in (1.3), the derivatives of $\lambda(t)$ and $\mathbf{x}(t)$ can also be obtained with minimal computations.

The algorithm has been implemented on an NCUBE, a distributed-memory multiprocessor machine. The preliminary results are very encouraging. Our numerical results presented in § 5 indicate that the *efficiency* of the parallel homotopy algorithm is almost 100 percent for large-dimensional matrices. This seems to suggest that the natural parallelism makes the homotopy algorithm an excellent candidate for multiprocessor machines. It is somewhat difficult to compare our algorithm with the Dongarra-Sorensen implementation of Cuppen's method [1] and the bisection/multisection algorithm [5] directly since most of their results were executed on shared-memory machines or different local-memory machines. However, the numerical results show that the *speedup* of our parallel algorithm over EISPACK is at least as good as those of the methods in [1] and [5] on the matrices presented in those papers.

A striking surprise is that the homotopy method does not seem to suffer from the effects of nearly equal eigenvalues as much as other algorithms do. It can solve such ill-conditioned problems as the Wilkinson matrices [10] much faster than BISECT+ TINVIT even in serial mode of execution (Tables 5.3 and 5.5).

2. Following the eigenspace. Let matrices $\mathbf{A}, \mathbf{D}$ be as in (1.1) and (1.3) and let $\mathbf{A}(t) = (1-t)\mathbf{D} + t\mathbf{A}$ be an unreducible tridiagonal matrix for $t \neq 0$. Differentiating $\mathbf{H}(\mathbf{x}(t), \lambda(t), t) = 0$ in (1.2) with respect to t yields

$$\begin{bmatrix} \lambda(t)\mathbf{I} - \mathbf{A}(t) & \mathbf{x}(t) \\ \mathbf{x}(t)^T & 0 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}}(t) \\ \dot{\lambda}(t) \end{bmatrix} = \begin{bmatrix} (\mathbf{A}-\mathbf{D})\mathbf{x}(t) \\ 0 \end{bmatrix}. \tag{2.1}$$

Hence, the eigenpaths $(\mathbf{x}(t), \lambda(t))$ of $\mathbf{H}(\mathbf{x}(t), \lambda(t), t) = 0$ can be considered as solutions of the ordinary differential equation in (2.1). Note that the matrix

$$\begin{bmatrix} \lambda(t)\mathbf{I} - \mathbf{A}(t) & \mathbf{x}(t) \\ \mathbf{x}(t)^T & 0 \end{bmatrix}$$

is nonsingular if $\mathbf{A}(t)$ is unreducible. The basic features of the curve-following scheme in [4] to trace an eigenpath $(\mathbf{x}(t), \lambda(t))$ from $t = 0$ to $t = 1$ are the following:

(i) Prediction.
(ii) Correction.
(iii) Checking.
(iv) Selection of stepsize.

We briefly summarize these features by giving an overview of the algorithm. Assume that after i steps the approximate value $(\mathbf{x}^*(t_i), \lambda^*(t_i))$ of the eigenpair at t_i is known. Multiplying on the left by $\mathbf{x}(t)^T$ on the top equation of (2.1), we obtain

$$\dot{\lambda}(t) = \mathbf{x}(t)^T(\mathbf{A}-\mathbf{D})\mathbf{x}(t). \tag{2.2}$$

The first attempt of the stepsize h is always chosen to be $1 - t_i$. This extremely liberal choice of stepsize can be justified because of the efficient checking algorithm. For a new stepsize h, a two-step ODE method is used on (2.2) to predict the eigenvalue $\lambda^0(t_{i+1})$ at t_{i+1}, then the inverse power method with the predicted eigenvalue shift is used to obtain a predicted eigenvector $\mathbf{x}^0(t_{i+1})$, at t_{i+1}. That is, we solve

$$(\mathbf{A}(t_{i+1}) - \lambda^0(t_{i+1})\mathbf{I})\mathbf{y} = \mathbf{x}^*(t_i),$$

and let $\mathbf{x}^0(t_{i+1}) = \mathbf{y}/\|\mathbf{y}\|$. Then correct $\mathbf{x}^0(t_{i+1})$ with Rayleigh quotient iteration to the approximate value $(\mathbf{x}^*(t_{i+1}), \lambda^*(t_{i+1}))$. The Sturm sequences are computed at $\lambda^0(t_{i+1})$ and $\lambda^*(t_{i+1})$ to check if the correct eigenpath is followed. If not, the stepsize is cut in half.

As mentioned in [4], the algorithm can be inefficient if $\mathbf{A}(t)$ has close eigenvalues for some $t \in [0, 1]$. In fact, while the eigenvalues of $\mathbf{A}(t)$ are distinct theoretically for $t \neq 0$, the separation of the eigenvalues can be as small as the machine accuracy. At these points, the eigenvectors are ill-conditioned, which may cause the homotopy algorithm to slow down severely. To deal with this problem, we propose, in the following, an improved algorithm based on the suggestion [6] that if a cluster of eigenvalues is well separated from all the other eigenvalues, the associated *eigenspace* is well conditioned. Thus, at such t where a cluster of eigenvalues of $\mathbf{A}(t)$ occurs, instead of following a single eigenpath, we trace the corresponding eigenspace.

A detailed explanation of this procedure is given as follows:

1. **Detection of a cluster.** When $(\mathbf{x}^*(t_i), \lambda^*(t_i))$ is taken as the approximate eigenpair at t_i, the Sturm sequences at $\lambda^*(t_i) \pm \varepsilon$ are computed to determine the number of eigenvalues of $\mathbf{A}(t_i)$, denoted by m, in the interval $(\lambda^*(t_i) - \varepsilon, \lambda^*(t_i) + \varepsilon)$. We choose ε to be the error tolerance of the eigenvalue curve. For what we discussed here, we assume, of course, $m > 1$.

2. **Construction of the corresponding eigenspace.** Once a cluster of m eigenvalues at t_i is detected, the perturbed inverse power method with eigenvalue $\lambda^*(t_i)$ as shift is used to construct an approximation of the corresponding eigenspace at t_i. Wilkinson [10] maintains that a separation of neighboring eigenvalues on the order of three times machine precision is sufficient to produce independent eigenvectors which span an approximation of the corresponding eigenspace. In order to construct an m-dimensional approximated eigenspace at t_i, $m-1$ equations

$$(\mathbf{A}(t_i)-(\lambda^*(t_i)+j\ \text{eps})\mathbf{I})\mathbf{x}_j=\mathbf{r}, \qquad j=1,\cdots,m-1$$

are solved, where $\text{eps}=\text{machep}*\|\mathbf{A}(t_i)\|_F$, machep is a machine-dependent parameter specifying the relative precision of floating point arithmetic, and $\mathbf{r}$ is a vector with all components equal to eps. Let $\mathbf{x}_m=\mathbf{x}^*(t_i)$. The subspace spanned by $\{\mathbf{x}_1,\cdots,\mathbf{x}_m\}$ is accepted as an approximated eigenspace of $\lambda(t_i)$. This version of perturbed inverse iteration has gained acceptance as implemented in the subroutine TINVIT of EISPACK [8].

3. **Following the eigenspace.**
   (1) Orthonormalize the $n\times m$ matrix $\mathbf{S}=[\mathbf{x}_1,\cdots,\mathbf{x}_m]$ to obtain an $n\times m$ orthonormal matrix $\mathbf{Q}$.
   (2) Find the projection of the matrix operator $\mathbf{A}(t_i+h)$ onto the column space of $\mathbf{Q}$:

$$\mathbf{P}=\mathbf{Q}^T\mathbf{A}(t_i+h)\mathbf{Q}. \tag{2.3}$$

   Here, $\mathbf{P}$ is an $m\times m$ matrix.
   (3) Solve for the eigensystem of the projected operator:

$$\mathbf{P}\mathbf{z}_k=\mu_k\mathbf{z}_k, \qquad k=1,\cdots,m.$$

   (4) Calculate

$$\mathbf{g}_k=\mathbf{Q}\mathbf{z}_k \qquad (k=1,\cdots,m).$$

   The subspace spanned by $\{\mathbf{g}_1,\cdots,\mathbf{g}_m\}$ serves as an approximation of the eigenspace of $\mathbf{A}(t_i+h)$.
   (5) Solve

$$(\mathbf{A}(t_i+h)-\mu_k\mathbf{I})\mathbf{x}_k=\mathbf{g}_k, \qquad k=1,\cdots,m.$$

   The subspace $\{\mathbf{x}_1,\cdots,\mathbf{x}_m\}$ is an improved approximate eigenspace of $\mathbf{A}(t_i+h)$.
   (6) Repeat (1)-(5) until the residual

$$r=\max_{1\le k\le m}\|\mathbf{A}(t_i+h)\mathbf{g}_k-\mu_k\mathbf{g}_k\|$$

   is smaller than the eigenvalue error tolerance. $\{(\mu_k,\mathbf{g}_k),\ k=1,\cdots,m\}$ is then taken as the eigenpair of $\mathbf{A}(t_i+h)$.

The procedure (1)-(5) is known as the Rayleigh-Ritz procedure [6] and μ_k $(k=1,\cdots,m)$ is the Ritz value. The iteration process used here is called the subspace iteration. Our experiments show that once the separation of the Ritz values is good

enough, instead of continuing the subspace iteration, we may use the Rayleigh quotient iteration, with $\{(\mathbf{g}_k, \mu_k),\ k=1, \cdots, m\}$ as input data, to approximate m eigenpairs $(\mathbf{x}_{k_1}(t_i+h),\ \lambda_{k_1}(t_i+h)), \cdots, (\mathbf{x}_{k_m}(t_i+h), \lambda_{k_m}(t_i+h))$ of $\mathbf{A}(t_i+h)$. It also gives satisfactory eigenpairs of $\mathbf{A}(t_i+h)$ and is obviously faster, since less computations are involved.

Remark 2.1. At $t=0$, $\{\mathbf{x}_1, \cdots, \mathbf{x}_m\}$ is orthonormal and each $\mathbf{x}_i$ has only n/p possible nonzero components. Step (1) can be skipped. In (2), forming matrix $\mathbf{P}$ takes only $1/p$th operation counts of the normal computation of (2.3).

The algorithm which follows the eigenspace described in this section has been incorporated into the Li-Rhee algorithm [4] for serial computations. It leads to a substantial improvement.

3. The initial matrix D. Again, multiplying on the left by $\mathbf{x}(t)^T$ and using (1.2) on the top equation of (2.1), we obtain

$$\dot{\lambda}(t) = \mathbf{x}(t)^T(\mathbf{A}-\mathbf{D})\mathbf{x}(t). \tag{3.1}$$

Hence,

$$|\dot{\lambda}(t)| \leqq \|\mathbf{A}-\mathbf{D}\|_2. \tag{3.2}$$

Note that $\dot{\lambda}(0)=0$.

Inequality (3.2) indicates that the *closer* the matrices $\mathbf{A}$ and $\mathbf{D}$, the better conditioned the eigenvalue curve will be. Hence, the selection of $\mathbf{D}$ is a critical aspect of our algorithm. This fact is evidenced by the following example.

Example 3.1.

$$\boldsymbol{A} = \begin{bmatrix} 10 & 1 & & & & & & & \\ 1 & 9 & 2 & & & & & & \\ & \cdot & \cdot & \cdot & & & & & \\ & & \cdot & 1 & \cdot & & & & \\ & & & \cdot & 0.5 & \cdot & & & \\ & & & & \cdot & 1.5 & \cdot & & \\ & & & & & \cdot & \cdot & 19 & \\ & & & & & & 19 & 9.5 & \end{bmatrix}.$$

Here, $\mathbf{A}$ is a 20×20 matrix and $\mathbf{D}$ is chosen as in (1.3), consisting of diagonal blocks of size $20/p$ of $\mathbf{A}$. For different values of p, we have different sizes of blocks. For instance, when $p=20$, $\mathbf{D}$ is a diagonal matrix. The eigenvalue curves of $\mathbf{H}(\mathbf{x}, \lambda, t)=0$ in (1.2) are shown in Fig. 3.1.

Remark 3.1. Of course, our algorithm has its optimal simplicity when $\mathbf{D}$ is chosen as diagonal part of $\mathbf{A}$. For such $\mathbf{D}$, from our experience on numerous random examples, the conditioning of the eigenvalue curves $\lambda(t)$ depends heavily on the standard deviation of subdiagonal elements of $\mathbf{A}$. Especially when the magnitudes of all off-diagonal elements $\{\beta_j,\ j=2, \cdots, n\}$ are approximately equal, the eigenvalue curves only change slightly in $[0, 1]$. While this phenomenon still is under investigation, picking $\mathbf{D}$ given $\mathbf{A}$ remains a critical open problem.

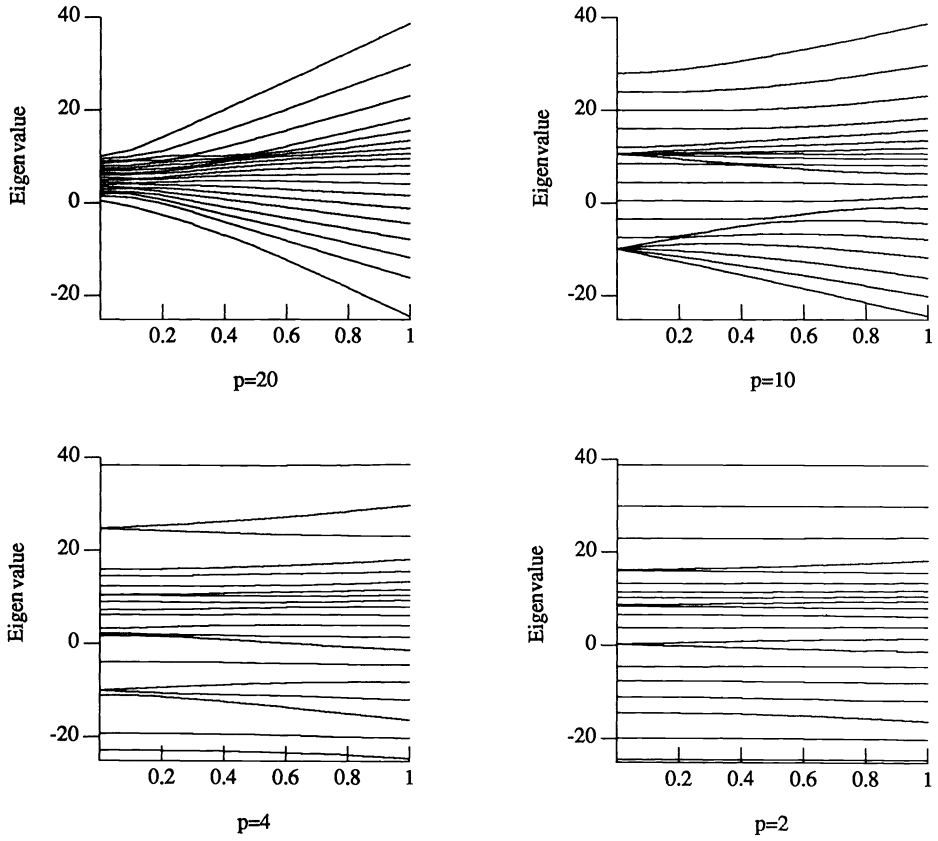


FIG. 3.1

In [4], the initial matrix **D** is always chosen as the diagonal part of **A** when the diagonal elements of **A** are different. In such cases, the derivatives of $\lambda(t)$ and $\mathbf{x}(t)$ at $t=0$ are immediately available without any computations. To make the initial matrix **D** even closer to **A**, we normally choose **D** as the diagonal blocks of **A** in our algorithm. It turns out that the derivatives of $\lambda(t)$ and $\mathbf{x}(t)$ at $t=0$ can also be obtained with minimal computations.

For positive integers $k_0, k_1, \cdots$, let $\bar{k}_{-1}=0$ and $\bar{k}_i = k_0+\cdots+k_i$, $i=0, 1, \cdots$. For the unreducible tridiagonal symmetric matrix

$$\mathbf{A}=\begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \ddots & \ddots & \\ & \ddots & & \beta_n \\ & & \beta_n & \alpha_n \end{bmatrix}$$

let

$$\mathbf{D}=\begin{bmatrix} \mathbf{A}_0 & & & \\ & \mathbf{A}_1 & & \\ & & \ddots & \\ & & & \mathbf{A}_{p-1} \end{bmatrix}$$

where $\mathbf{A}_i$ is the $k_i \times k_i$ matrix

$$\begin{bmatrix} \alpha_{\bar{k}_{i-1}+1} & \beta_{\bar{k}_{i-1}+2} & & & \\ \beta_{\bar{k}_{i-1}+2} & \alpha_{\bar{k}_{i-1}+2} & & & \\ & \ddots & \ddots & & \\ & & \alpha_{\bar{k}_{i-1}+k_i-1} & \beta_{\bar{k}_i} & \\ & & \beta_{\bar{k}_i} & \alpha_{\bar{k}_i} & \end{bmatrix}.$$

Then

$$\mathbf{A}-\mathbf{D}=\left[\begin{array}{c|c|c|c|c} \mathbf{0} & \beta_{\bar{k}_0+1} & & & \\ \hline \beta_{\bar{k}_0+1} & \mathbf{0} & \beta_{\bar{k}_1+1} & & \\ \hline & \beta_{\bar{k}_1+1} & \cdots & \beta_{\bar{k}_{p-2}+1} & \\ \hline & & \beta_{\bar{k}_{p-2}+1} & \mathbf{0} & \end{array}\right].$$

Let $\mathbf{A}_i = \mathbf{Q}_i \mathbf{\Lambda}_i \mathbf{Q}_i^T$, $i = 0, \cdots, p-1$, where $\mathbf{\Lambda}_i = (0, \lambda_l^{(i)}, 0)$ is a diagonal matrix consisting of eigenvalues of $\mathbf{A}_i$, and $\mathbf{Q}_i$ is a $k_i \times k_i$ orthonormal matrix consisting of eigenvectors of $\mathbf{A}_i$. In our algorithm, both $\mathbf{\Lambda}_i$ and $\mathbf{Q}_i$ are already known from previous computations. Let

$$\mathbf{\Lambda}=\begin{bmatrix} \mathbf{\Lambda}_0 & & & \\ & \mathbf{\Lambda}_1 & & \\ & & \ddots & \\ & & & \mathbf{\Lambda}_{p-1} \end{bmatrix}, \qquad \mathbf{Q}=\begin{bmatrix} \mathbf{Q}_0 & & & \\ & \mathbf{Q}_1 & & \\ & & \ddots & \\ & & & \mathbf{Q}_{p-1} \end{bmatrix}.$$

Then, $\mathbf{D} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ and

$$\dot{\lambda}(t) = \mathbf{x}(t)^T(\mathbf{A}-\mathbf{D})\mathbf{x}(t) = \mathbf{x}(t)^T(\mathbf{A}-\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T)\mathbf{x}(t) = \mathbf{x}(t)^T\mathbf{Q}(\mathbf{Q}^T\mathbf{A}\mathbf{Q}-\mathbf{\Lambda})\mathbf{Q}^T\mathbf{x}(t).$$

Hence,

$$\dot{\lambda}(t) = \tilde{\mathbf{x}}(t)^T\mathbf{B}\tilde{\mathbf{x}}(t)$$

where

$$\tilde{\mathbf{x}}(t) = \mathbf{Q}^T\mathbf{x}(t), \tag{3.3}$$

$$\mathbf{B} = \mathbf{Q}^T\mathbf{A}\mathbf{Q}-\mathbf{\Lambda} = \mathbf{Q}^T(\mathbf{A}-\mathbf{D})\mathbf{Q}.$$

For simplicity, we assume that the elements in matrix $\mathbf{\Lambda}$ are in increasing order.

It can easily be seen that if $\lambda_j(0)$ is the jth eigenvalue of $\mathbf{D}$, then

$$\tilde{\mathbf{x}}(0) = \mathbf{e}_j \equiv [0, 0, \cdots, \overset{j\text{th}}{1}, 0, \cdots, 0]^T.$$

And,

$$\mathbf{B}=\mathbf{Q}^T(\mathbf{A}-\mathbf{D})\mathbf{Q}=\left[\begin{array}{c|c|c|c|c} \mathbf{0} & \mathbf{B}_0 & & & \\ \hline \mathbf{B}_0 & \mathbf{0} & \mathbf{B}_1 & & \\ \hline & \mathbf{B}_1 & \cdots & \mathbf{B}_{p-2} & \\ \hline & & \mathbf{B}_{p-2} & \mathbf{0} & \end{array}\right],$$

where

$$\mathbf{B}_i = \beta_{\bar{k}_i+1}\mathbf{q}^i_{k_i}(\mathbf{q}_1^{i+1})^T, \qquad i=0,\cdots,p-2,$$

and $(\mathbf{q}^i_{k_i})^T$, $(\boldsymbol{q}_1^{i+1})^T$ are the last and first row vectors of $\mathbf{Q}_i$ and $\mathbf{Q}_{i+1}$, respectively.

From (2.1), we have

$$(\lambda(t)\mathbf{I}-\mathbf{A}(t))\dot{\mathbf{x}}(t) = (\mathbf{A}-\mathbf{D})\mathbf{x}(t) - \dot{\lambda}(t)\mathbf{x}(t),$$
$$\mathbf{x}^T(t)\cdot\dot{\mathbf{x}}(t) = 0.$$

Multiplying $\mathbf{Q}^T$ on the left and substituting (3.3) yields, when $t=0$,

$$(\lambda_j(0)\mathbf{I}-\boldsymbol{\Lambda})\dot{\tilde{\mathbf{x}}}(0) = \mathbf{B}\tilde{\mathbf{x}}(0) - \dot{\lambda}(0)\mathbf{e}_j = \mathbf{b}_j,$$
$$\mathbf{e}_j^T\cdot\dot{\tilde{\mathbf{x}}}(0) = 0,$$

where $\mathbf{b}_j$ is the jth column of $\mathbf{B}$ with the special structure

$$\mathbf{b}_j^T = [0,\cdots,d_{\bar{k}_{i-2}+1},\cdots,d_{\bar{k}_{i-1}},0,\cdots,0,d_{\bar{k}_{i-1}+1},\cdots,d_{\bar{k}_i},0,\cdots,0]^T \quad \text{if } \bar{k}_{i-1}<j\leqq\bar{k}_i.$$

Since $\lambda(0)\mathbf{I}-\boldsymbol{\Lambda}$ is a diagonal matrix, $\dot{\tilde{\mathbf{x}}}(0)$ can be easily solved and

$$\dot{\mathbf{x}}(0) = \mathbf{Q}\dot{\tilde{\mathbf{x}}}(0) = \mathbf{Q}_{i-1}(\lambda_j-\boldsymbol{\Lambda}_{i-1})^{-1}\begin{bmatrix} d_{\bar{k}_{i-2}+1}\\ \vdots\\ d_{\bar{k}_{i-1}}\end{bmatrix} + \mathbf{Q}_{i+1}(\lambda_j-\boldsymbol{\Lambda}_{i+1})^{-1}\begin{bmatrix} d_{\bar{k}_{i-1}+1}\\ \vdots\\ d_{\bar{k}_i}\end{bmatrix},$$

which requires only $k_{i-1}^2+k_{i-1}+k_i^2+k_i$ multiplications. On the other hand, differentiating (3.2), we have

$$\ddot{\lambda}(t) = 2\mathbf{x}^T(t)(\mathbf{A}-\mathbf{D})\dot{\mathbf{x}}(t) = 2\mathbf{x}^T(t)\mathbf{Q}\mathbf{B}\mathbf{Q}^T\dot{\mathbf{x}}(t) = 2\tilde{\mathbf{x}}^T(t)\mathbf{B}\dot{\tilde{\mathbf{x}}}(t).$$

At $t=0$, and if $\lambda_j(0)$ is the jth eigenvalue of $\mathbf{D}$,

$$\ddot{\lambda}(0) = 2\mathbf{b}_j^T\cdot\dot{\tilde{\mathbf{x}}}(0) = 2\left[\sum_{l=1}^{k_{i-1}}\frac{d^2_{k_{i-2}+l}}{\lambda_j-\lambda_l^{(i-1)}} + \sum_{l=1}^{k_i}\frac{d^2_{k_{i-1}+l}}{\lambda_j-\lambda_l^{(i+1)}}\right].$$

Along the same line of arguments, the higher derivatives of $\lambda(t)$, $\mathbf{x}(t)$ at $t=0$ can also be easily obtained with minimal computations.

4. Parallel algorithm. Part of what makes the homotopy algorithm attractive is its parallelism, in the sense that each eigenpath is followed independently of the others. After some sorting in the beginning, the communication between processing elements is unnecessary when the homotopy curves are followed simultaneously by different processors. The algorithm has been implemented on an NCUBE, a distributed-memory multiprocessor machine in which each memory module is physically associated with each processor and an interprocessor communication network provides a mechanism for communication between processors.

Using p processors, the parallel homotopy algorithm consists of the following steps:

1. Initiating. For

$$\mathbf{A} = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \ddots & \ddots & \\ & \ddots & & \beta_n \\ & & \beta_n & \alpha_n \end{bmatrix},$$

we choose, in our homotopy $\mathbf{H}(\mathbf{x},\lambda,t)$ in (1.2), the initial matrix

$$\mathbf{D} = \begin{bmatrix} \mathbf{A}_0 & & & \\ & \mathbf{A}_1 & & \\ & & \ddots & \\ & & & \mathbf{A}_{p-1} \end{bmatrix},$$

where $\mathbf{A}_i$ $(i=0, \cdots, p-1)$ are diagonal blocks of $\mathbf{A}$. We first compute all the eigenvalues of $\mathbf{A}_i$ by the best sequential algorithm on the ith processor concurrently for each i, $0 \leqq i \leqq p-1$.

2. **Broadcasting and sorting.** To identify the order of the eigenvalues of $\mathbf{A}_i$ $(i=0, \cdots, p-1)$, considered as eigenvalues of $\mathbf{D}$, we broadcast the eigenvalues from every processor to all the others first. Then, sort these eigenvalues on each processor concurrently. The Alternate Direction Exchange Algorithm (ADE) [2], [7] is used for the broadcasting. Namely, for broadcasting an array of length k (the number of eigenvalues being computed at the previous step) from one processor to all the others, and then sorting, the following steps are taken in a d-cube $(d=\log_2 p)$:
   For $l=1, 2, \cdots, d$
   (i) Processors are divided into two $(d-1)$-cubes, T_0 and T_1, according to the value of bit $l-1$ in their binary labels.
   (ii) Processors in corresponding positions of T_0 and T_1 exchange sorted array of length $2^{l-1}k$.
   (iii) Each processor merges its own sorted array with the one received to form a new array of length $2^l k$.

As can be seen, there are d communications. The amount of data doubles in size during each step. The total data transferred is

$$k+2k+\cdots+2^{d-1}k=k(p-1).$$

The total data comparison is at most

$$(2k-1)+(4k-1)+\cdots+(2^d k-1)=2k(p-1)-d.$$

Comparing them to the computation cost, they can almost be ignored when p is much smaller than n.

3. **Following the eigenpaths.** Divide the work as equally as possible for p processors. Each processor follows its own eigenpaths independently. The Li-Rhee algorithm [4], together with the procedure described in § 2, is used here.

Remark 4.1. Step 1 is the "divide" part of our algorithm and the rest of the steps comprise the "conquer" part. Depending on the availability of the processing elements and the size of the problem, "divide" and "conquer" may be performed inside the intial matrix $\mathbf{D}$ when we solve the eigenpairs of the subblocks $\mathbf{A}_i$ of $\mathbf{D}$.

Remark 4.2. As mentioned in [4], a further advantage of the homotopy algorithm is the *order preserving property*: Since $\mathbf{A}(t)$ is unreducible for each $t \neq 0$, the eigenvalue curves $\lambda_1(t), \cdots, \lambda_n(t)$ are disjoint in $(0, 1]$. It follows that each $\lambda(t)$ joins an eigenvalue of the initial matrix $\mathbf{D}$ to one of $\mathbf{A}$ of the same order. As a consequence, the homotopy method can find any specifically desired eigenvalue, say the fourteenth of $\mathbf{A}$, by following the eigenpath $(\mathbf{x}(t), \lambda(t))$, which starts from the eigenpair of $\mathbf{D}$ corresponding to the fourteenth eigenvalue. Note that this eigenvalue of $\mathbf{A}$ is obtained without calculating any other eigenvalues. This property is very important in practice, since, when large eigenvalue problems arise, it is rarely of any interest whatsoever to compute all the eigenvalues.

In addition to computational savings, the communication and sorting costs at step 2 can also be cut dramatically when our parallel algorithm is executed. For instance, if the first $k(\leqq n/p)$ eigenpairs of $\mathbf{A}$ are desired, we modify (ii) and (iii) of ADE algorithm as follows:

(ii′) Processors in corresponding positions of T_0 and T_1 exchange sorted arrays of length k.

(iii′) Each processor merges two corresponding subarrays and keeps the first k components only.

The amount of data is kept constant in each communication step. The data transferred and sorted is then reduced to kd and $(2k-1)d$, respectively.

Remark 4.3. On some occasions in practice, only the eigenvalues in a specified interval $[a, b]$ are in demand. The homotopy algorithm may achieve this purpose without finding all the eigenvalues. It is easy to verify that the eigenvalues of $\mathbf{A}-\mathbf{D}$ are $\{0, \pm|\beta_{\bar{k}_i+1}|\}$ if $\bar{k}_i \geqq 2$ $(i=0, \cdots, p-1)$. Since $\mathbf{A}=\mathbf{D}+(\mathbf{A}-\mathbf{D})$, by using the monotonicity theorem [6], we obtain, for each $i=1, \cdots, n$,

$$\lambda_i(1)-\lambda_n[\mathbf{A}-\mathbf{D}] \leqq \lambda_i(0) \leqq \lambda_i(1)-\lambda_1[\mathbf{A}-\mathbf{D}],$$

where subindex denotes the order of the eigenvalue. Let $\beta=\max_{0\leqq i\leqq p-1}|\beta_{\bar{k}_i+1}|$. Then,

$$a-2\beta \leqq \lambda_i(0) \leqq b+2\beta \quad \text{if some } k_j=1,$$

$$a-\beta \leqq \lambda_i(0) \leqq b+\beta \quad \text{if all } k_j>1, \quad j=0, \cdots, p-1,$$

if $\lambda_i(1) \in [a, b]$. Hence, in order to extract all the eigenvalues of $\mathbf{A}$ inside the interval $[a, b]$, only those eigenpaths with starting eigenvalues, eigenvalues of $\mathbf{D}$, inside the interval $[a-\beta, b+\beta]$ (or $[a-2\beta, b+2\beta]$ if some $k_j=1$) are necessary to follow. In our program, we use EISPACK's subroutine BISECT to compute the eigenvalues of subblock $\mathbf{A}_i$. The incorporation of this strategy is straightforward.

Remark 4.4. In addition to the natural parallelism mentioned before, the homotopy algorithm holds another potential parallelism: the algorithm for following a *single* eigenpath can also be parallelized. This is practically important when the required number of eigenpairs is less than the number of the processors or when the local memory capacity of the machine is not enough for matrices of large dimension. The main computations of our algorithm for following a single eigenpath only involve solving symmetric tridiagonal linear equations of form $\mathbf{My}=\mathbf{b}$. The parallelizing can be achieved by solving the linear systems in parallel. At present, several parallel tridiagonal solvers are available [3] and a parallel tridiagonal solver with pivoting is under investigation [9].

The program of the parallel homotopy algorithm is listed in the Appendix.

5. Experimental results. In this section we compare our implementation of the parallel homotopy algorithm with BISECT+TINVIT, IMTQL2, which are standard subroutines in EISPACK [8]. Also, we compare the performance of the homotopy method with both the divide and conquer algorithm (D&C) [1] and bisection/multisection algorithm (B/M) [5], which are currently considered as the best parallel algorithms for the symmetric eigenproblem. It is rather difficult to compare our results with D&C's and B/M's directly since most of their results were executed on a shared-memory machine or a different local-memory machine. We calculate the speedup of our algorithm over IMTQL2, which has somewhat different goals, for the purpose of indirectly comparing our results with those in [1] and [5]. As a reference, computing all the eigenpairs by IMTQL1+TINVIT is about 20 percent faster than BISECT+TINVIT on our examples. Our experiments are performed on NCUBE, which is a local-memory machine. The algorithm itself is not limited to this memory model. In fact, we believe that a better result can be expected on a shared-memory machine.

In using TINVIT, we modified it in two ways. First, we orthogonalize the eigenvectors that correspond to the close eigenvalues, since our algorithm does so. Second, we modified TINVIT such that it will do one more inverse iteration for each eigenvector than the normal TINVIT does if the residual $\|\mathbf{Ax}(1)-\lambda(1)\mathbf{x}(1)\|$ is larger than the error tolerance, since we use a similar strategy in our algorithm. These slight

modifications resulted in better accuracy in BISECT+TINVIT, as well as in our algorithm.

There are two commonly accepted measures for the *speedup*. First we compare the established serial algorithms with the parallel algorithm in concern and the speedup is defined as

$$S_p = \frac{\text{execution time using the established sequential algorithm}}{\text{execution time using parallel algorithm with } p \text{ processors}}.$$

The second one measures the parallel inherence of the algorithm which is defined as

$$S'_p = \frac{\text{execution time using one processor}}{\text{execution time using } p \text{ processors}}.$$

The *efficiency* is the ratio of the speedup and p. The speedup and efficiency of our parallel algorithm over BISECT+TINVIT, IMTQL2 and sequential homotopy algorithm on matrices $(1, 2, 1)$, and the Wilkinson matrices $\mathbf{W}^+_{2k+1} = (1, |k+1-i|, 1)$, $2k+1$ is the dimension [10, p. 308], are shown in Tables 5.1–5.7 and Figs. 5.1 and 5.2, where N denotes the dimension of the matrix and *nodes* represents the number of processors being used. For the Wilkinson matrix, the initial matrix $\mathbf{D}$ was chosen as the diagonal matrix of $\mathbf{A}$ based on the observations we described in Remark 3.1. To avoid huge multiplicities of the eigenvalues, we did not choose $\mathbf{D}$ as the diagonal matrix for matrix $(1, 2, 1)$. $\mathbf{D}$ was chosen as the diagonal block matrix taken from $(1, 2, 1)$ as in (1.3). The sizes of the blocks are the same $(=n/p)$ and the number of blocks equals the number of processors $(=p)$ being used. The maximal number of the processors being used is 16 because the sizes of our test examples are relatively small.

TABLE 5.1
BISECT+TINVIT *and* IMTQL2 *versus the parallel homotopy algorithm on* $(1, 2, 1)$ *matrix.*

N	Nodes	Execution time (sec)			BISECT+TINVIT / homotopy	IMTQL2 / homotopy
		Parallel homotopy**	BISECT+ TINVIT	IMTQL2		
64	1	9.65	7.08	28.90	0.7	3.0
	4	2.49			2.8	11.6
	8	1.72			4.1	16.8
128	1	37.78	27.38	219.13	0.7	5.8
	4	9.62			2.8	22.8
	8	6.28			4.4	34.9
256	1	147.87	105.97	*	0.7	
	4	37.27			2.8	
	8	24.37			4.3	
	16	18.36			5.8	
512	1	602.25	410.38	*	0.7	
	4	151.82			2.7	
	8	94.47			4.3	
	16	69.07			5.9	

* Exceed memory capacity.

** When homotopy algorithm was executed with one processor, the initial matrix $\mathbf{D}$ was chosen as diagonal block matrix consisting of four $N/4 \times N/4$ blocks.

TABLE 5.2
Sequential homotopy algorithm versus parallel homotopy algorithm on (1, 2, 1) *matrix.*

N	Nodes	Execution time (sec)		S'_p	Efficiency $(=S'_p/p)$
		Parallel homotopy	Sequential homotopy*		
64	4	2.49	9.65	3.9	0.970
	8	1.72	12.93	7.5	0.941
128	4	9.62	37.78	3.9	0.982
	8	6.28	49.07	7.8	0.977
256	4	37.27	147.87	4.0	0.992
	8	24.37	189.23	7.8	0.971
	16	18.36	279.80	15.2	0.953
512	4	151.82	602.25	4.0	0.992
	8	94.47	741.90	7.9	0.982
	16	69.07	1085.11	15.7	0.982

* The algorithm was executed with one processor. The initial matrix **D** was chosen as the diagonal blocks of (1, 2, 1) ($p(N/p)\times(N/p)$ blocks, where p is the number of nodes being used by the corresponding parallel algorithm). Hence the execution times of the sequential algorithm are different for the same N.

TABLE 5.3
BISECT+TINVIT *versus the parallel homotopy algorithm on the Wilkinson matrix.* $\mathbf{W}^+_{2k+1} = (1, |k+1-i|, 1)$.

N	Nodes	Execution time (sec)		BISECT+TINVIT / homotopy
		Parallel homotopy	BISECT+TINVIT	
31	1	1.12	1.30	1.2
	4	0.31		4.2
	8	0.18		7.1
63	1	3.30	4.63	1.4
	4	0.86		5.4
	8	0.48		9.7
127	1	9.07	16.52	1.8
	4	2.34		7.1
	8	1.21		13.6
255	1	28.35	62.63	2.2
	4	7.21		8.7
	8	3.67		17.1
	16	1.92		32.7
511	1	97.83	232.65	2.4
	4	24.63		9.4
	8	12.35		18.8
	16	6.26		37.2

TABLE 5.4

Sequential homotopy algorithm versus parallel homotopy algorithm on the Wilkinson matrix.

N	Nodes	Execution time (sec) Parallel homotopy	S'_p	Efficiency $(=S'_p/p)$
31	1	1.12		
	4	0.31	3.6	0.898
	8	0.18	6.1	0.763
63	1	3.30		
	4	0.86	3.9	0.964
	8	0.48	6.9	0.866
127	1	9.07		
	4	2.34	3.9	0.969
	8	1.21	7.5	0.934
255	1	28.35		
	4	7.21	3.9	0.983
	8	3.67	7.7	0.966
	16	1.92	14.8	0.925
511	1	97.83		
	4	24.63	4.0	0.993
	8	12.35	7.9	0.990
	16	6.26	15.6	0.977

TABLE 5.5

Speedup for the homotopy algorithm as a function of the number of eigenvalues and eigenvectors.

K	Execution time (sec) Homotopy (1 CE)	Homotopy (4 CEs)	BISECT+ TINVIT	Homotopy(1 CE) / Homotopy(4 CEs)	BISECT+TINVIT / homotopy(1 CE)
8	1.37	0.34	4.94	4.0	3.6
16	2.75	0.69	9.73	4.0	3.5
32	5.49	1.37	19.32	4.0	3.5
64	10.98	2.75	38.47	4.0	3.5
128	21.96	5.49	76.70	4.0	3.5
256	43.88	10.97	152.82	4.0	3.5
511	97.83	24.63	232.65	4.0	2.4

Test matrix $\mathbf{W}^+_{2k+1}$ is of order 511. K is the number of desired eigenvalues and eigenvectors. CE denotes the number of processors.

All computations were executed with single precision. The eigenvalue error tolerance was chosen as $\varepsilon = 1.E-8 \times \|\mathbf{A}\|_F$ (that is, the residual $\|\mathbf{Ax}-\lambda\mathbf{x}\| \leqq \varepsilon$). The timings for BISECT+TINVIT, IMTQL2, and sequential homotopy were obtained by executing them on a 0-cube.

The dotted line in Figs. 5.1 and 5.2 stands for the *perfect speedup*, the speedup which equals the number of the processors being used. As can be seen, the efficiency of our algorithm is almost 100 percent for large-dimensional matrices. This seems to indicate that the natural parallelism makes the homotopy algorithm an excellent

TABLE 5.6
Residual and orthonormality of computed eigenvalues and eigenvectors on (1, 2, 1) *matrix.*

		$\max_i \|\mathbf{A}\mathbf{x}_i - \lambda_i \mathbf{x}_i\|_2$			$\max_{i,j} \|\mathbf{X}^T\mathbf{X} - \mathbf{I}\|_{i,j}$		
N	Nodes	Parallel homotopy	BISECT+ TINVIT	IMTQL2	Parallel homotopy	BISECT+ TINVIT	IMTQL2
64	1		8.66×10^{-7}	2.43×10^{-6}		3.64×10^{-5}	2.37×10^{-6}
	4	9.98×10^{-7}			4.44×10^{-6}		
	8	9.09×10^{-7}			4.07×10^{-6}		
128	1		8.00×10^{-7}	3.65×10^{-6}		3.70×10^{-5}	4.40×10^{-6}
	4	9.07×10^{-7}			2.86×10^{-5}		
	8	9.53×10^{-7}			2.06×10^{-5}		
256	1		9.59×10^{-7}			9.87×10^{-5}	
	4	9.95×10^{-7}			4.71×10^{-5}		
	8	9.96×10^{-7}			4.69×10^{-5}		
	16	9.96×10^{-7}			5.71×10^{-5}		
512	1		9.36×10^{-7}			4.78×10^{-4}	
	4	9.93×10^{-7}			3.53×10^{-4}		
	8	9.81×10^{-7}			3.03×10^{-4}		
	16	9.80×10^{-7}			2.39×10^{-4}		

TABLE 5.7
Residual and orthonormality of computed eigenvalues and eigenvectors on the Wilkinson matrix.

	$\max_i \dfrac{\|\mathbf{A}\mathbf{x}_i - \lambda_i \mathbf{x}_i\|_2}{\|\mathbf{A}\|_F}$		$\max_{i,j} \|\mathbf{X}^T\mathbf{X} - \mathbf{I}\|_{i,j}$	
N	Homotopy	BISECT+TINVIT	Homotopy	BISECT+TINVIT
31	9.48×10^{-9}	2.00×10^{-8}	5.55×10^{-7}	1.12×10^{-4}
63	4.10×10^{-9}	1.37×10^{-8}	5.45×10^{-7}	3.95×10^{-3}
127	8.61×10^{-9}	1.41×10^{-8}	2.18×10^{-5}	6.27×10^{-3}
255	2.20×10^{-9}	1.37×10^{-8}	2.18×10^{-5}	4.46×10^{-2}
511	6.68×10^{-9}	1.37×10^{-8}	2.18×10^{-5}	4.29×10^{-1}

candidate for multiprocessor machines. A remarkable surprise is that the homotopy method does not seem to suffer from the effects of nearly equal eigenvalues as other algorithms do. It can solve such ill-conditioned problems as the Wilkinson matrices much faster than BISECT+TINVIT even in serial mode of execution (Tables 5.3 and 5.5).

As mentioned in § 4, the homotopy method can find any specifically desired eigenvalues. In Table 5.5, we measure the speedup of the homotopy algorithm as a function of the number of desired eigenpairs for a Wilkinson matrix of order 511. On four processors, the speedup over one processor does not decrease as the number of desired eigenpairs decreases. An interesting phenomenon is that, on one processor, the efficiency of the homotopy algorithm over BISECT+TINVIT for computing a small number of eigenpairs can sometimes be much better than the case when total eigenpairs are calculated. This is mainly due to the fact that the conditionings of different eigenvalue curves may be quite different.

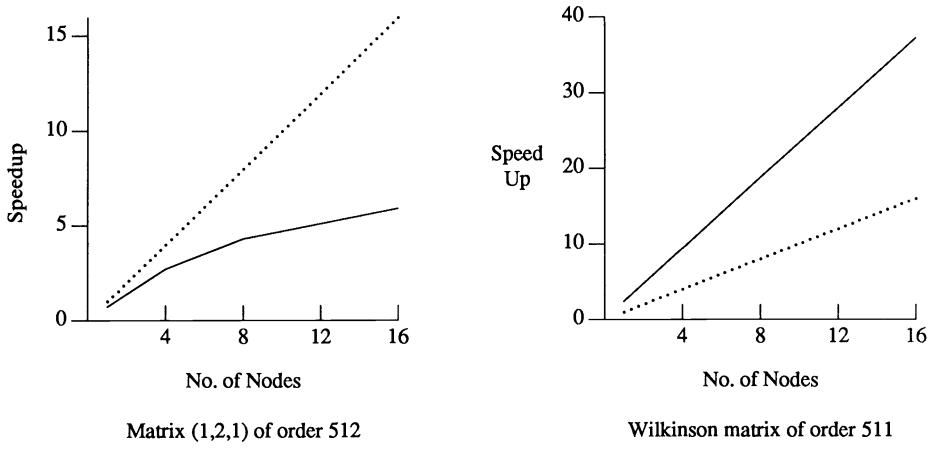


FIG. 5.1. *Speedup versus* BISECT+TINVIT.

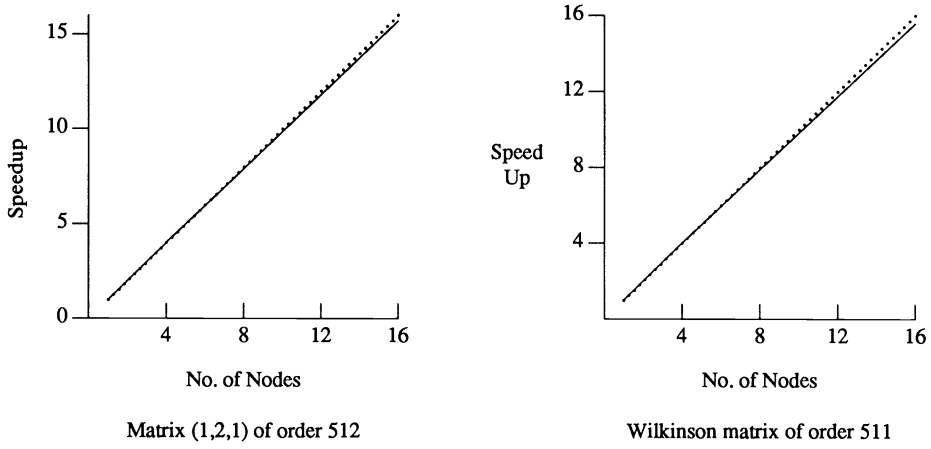


FIG. 5.2. *Speedup versus the sequential homotopy algorithm.*

TABLE 5.8

Comparison of our new homotopy algorithm with the Li-Rhee homotopy algorithm.

Test matrix = (1, 2, 1)		Test matrix = (1, 4, 1)	
N	New homotopy / BISECT+TINVIT	N	Li-Rhee homotopy / BISECT+TINVIT
128	1.38	100	4.18
256	1.40	200	7.12
512	1.47	400	8.13

TABLE 5.9
A comparison on the Alliant FX/8 *for* TQL2 *versus the parallel algorithm on* (1, 2, 1) *matrix.*

N	Ratio TQL2/SESUPD	TQL2 $\|\mathbf{Ax}-\lambda\mathbf{x}\|$	SESUPD $\|\mathbf{Ax}-\lambda\mathbf{x}\|$	TQL2 $\|\mathbf{X}^T\mathbf{X}-\mathbf{I}\|$	SESUPD $\|\mathbf{X}^T\mathbf{X}-\mathbf{I}\|$
100	9.4	9.6×10^{-15}	1.9×10^{-15}	7.8×10^{-15}	5.5×10^{-16}
200	15.4	1.7×10^{-14}	2.7×10^{-15}	1.1×10^{-14}	2.2×10^{-15}
300	17.7	1.9×10^{-14}	3.2×10^{-15}	1.7×10^{-14}	2.6×10^{-15}
400	20.0	2.9×10^{-14}	4.0×10^{-15}	2.0×10^{-14}	9.2×10^{-15}

* Reproduced by permission of J. J. Dongarra and D. C. Sorensen.

TABLE 5.10
Speedup over TQL2 *on the Alliant for computing all the eigenvalues and eigenvectors.*

i	Algorithm	Time (TQL2 on 1 CE) / Time (algorithm i on 8 CEs)	Time (TQL2 on 8 CEs) / Time (algorithm i on 8 CEs)
1	TREPS1	32.9	7
2	TREPS2	131.5	28
3	TQL2	4.7	1
4	BISECT+TINVIT	3.6	.8
5	SESUPD	27.1	5.8

* Reproduced by permission of S. S. Lo, B. Philippe, and A. Sameh.
Test matrix $(-1, 2, -1)$ is of order 500.

TABLE 5.11
Time and speedup for TREPS1 *on the Alliant* FX/8.

CEs	Time (sec)	Speedup
1	115.59	1.0
2	57.12	2.0
4	28.89	4.0
8	14.78	7.8

* Reproduced by permission of S. S. Lo, B. Philippe, and A. Sameh.
Test matrix $(-1, 2, -1)$ is of order 500.

TABLE 5.12.
Multisection on a 5-cube: fraction of total computation time spent in different steps.

Test matrix	Isolation	Extraction	Eigenvectors
[1, 2, 1]	0.52	0.41	0.07
[2, 2, 2]	0.42	0.50	0.08

* Reproduced by permission of I. C. F. Ipsen and E. R. Jessup.
Test matrices are of order 64.

Tables 5.6 and 5.7 show the residual and orthonormality of computed eigenvalues and eigenvectors on (1, 2, 1) and Wilkinson matrices. Note that our results for the residual and orthogonality properties of the Wilkinson matrices are better than that of BISECT+TINVIT even though we have modified TINVIT by orthogonalizing the eigenvectors corresponding to the close eigenvalues.

As we described in § 2, the Li-Rhee algorithm [4] may lose its efficiency when $\mathbf{A}(t)$ has close eigenvalues for some $t \in [0, 1]$. And when this problem arises, we use subspace iteration to "follow" the eigensubspace in our algorithm. Evidenced by the numerical results [4, Tables 5.1 and 5.2], this technique provides a considerable improvement of the Li-Rhee algorithm. To be more precise, we list in Table 5.8 the ratios of these two sequential homotopy algorithms over BISECT+TINVIT. Here BISECT+TINVIT is used as a medium comparing these two algorithms.

We list in Tables 5.9–5.12 some of the results of D&C [1] and B/M [2], [5]. Using EISPACK's subroutine IMTQL2, an indirect comparison between our algorithm and theirs may be obtained. In Table 5.9 [1, Table 8.3], when eight processors are being used, the speedup of D&C algorithm (SESUPD) over IMTQL2 is 9.4 for matrix (1, 2, 1) of order 100, whereas ours is 34.9 for the same matrix of order 128. Table 5.10 [5, Table 7b] shows the speedups of B/M (two versions: TREPS1 and TREPS2) and D&C over IMTQL2. Note that the last column of Table 5.1 shows that the speedup of the homotopy algorithm over IMTQL2 doubles as the dimension of the matrix doubles if eight processors are used. This result suggests that the speedup of the homotopy algorithm is approximately 140 for the matrix (1, 2, 1) of order 512, which is at least as good as TREPS2. Comparing our results on Table 5.2 with the speedup of B/M algorithm in Table 5.11 [5, Table 3], we see that the performance of the homotopy method on the matrix (1, 2, 1) is strongly competitive with B/M algorithm.

One big advantage of B/M algorithm is that it can compute the eigenvalues without computing eigenvectors. However, Table 5.12 [2, Table 1] shows that the computation of eigenvectors for the matrix (1, 2, 1) takes only seven percent of total execution time on a 5-cube Intel's iPSC. This seems to indicate that the eigenvalue computation time dominates the total parallel execution time for B/M algorithm, making our comparisons reliable.

Appendix.

Algorithm HOMOTOPY-EIGENVALUE
Input: **A**, **D** as in (1.1) and (1.3)
Output: eigenpairs of **A**
begin
    parallel do
        Compute the eigenvalues of submatrices $\mathbf{A}_i$ by BISECT
    od
    Sort the eigenvalues of **D**
    repeat
        parallel do
            Choose an eigenvalue, compute the corresponding eigenspace by TINVIT
            Initiate $t = 0$
            repeat
                $h = 2(1 - t)$
                repeat
                    $h \leftarrow h/2$
                    if the multiplicity of the eigenvalue = 1

```
                    Compute the eigenpair at t+h by LI-RHEE Algorithm
                else
                    Compute the eigenspace at t+h by SUBSPACE
                      ITERATION
                fi
                Check the order and multiplicity of the newly computed
                  eigenvalue
            until the order of the newly computed eigenvalue is right
            t ← t+h
            if the multiplicity of the newly computed eigenvalue > 1
                Construct corresponding eigenspace
            fi
        until t = 1
    od
until all eigenpairs are extracted
end
```

Subroutine SUBSPACE ITERATION.

```
Input: matrix A, subspace {x_1, ···, x_m}
Output: m eigenpairs of A
begin
    repeat
        Orthonormalize {x_1, ···, x_m} to obtain matrix Q
        Form P = Q*AQ
        Solve the eigenpairs: {(z_k, μ_k), k = 1, ···, m} of P
        Form g_k = Qz_k (k = 1, ···, m)
        Solve (A − μ_k)x_k = g_k (k = 1, ···, m)
    until r = max_{1≦k≦m} ‖Ag_k − μ_k g_k‖ < error tolerance
end
```

Algorithm LI-RHEE.

```
Input: eigenpair (x, λ) at t_{i−1} and t_i
Output: eigenpair (x, λ) at t_{i+1}
begin
    Compute λ̇(t_i)
    if t_i = 0
        Compute ẋ(t_i) and λ̈(t_i)
        Predict λ(t_{i+1}) by second-order Taylor expansions
        Predict x(t_{i+1}) by first-order Taylor expansions
    else
        Predict λ(t_{i+1}) by polynomial extrapolation at λ(t_{i−1}), λ̇(t_{i−1}), λ(t_i), λ̇(t_i)
        Predict x(t_{i+1}) by polynomial extrapolation at x(t_{i−1}), x(t_i)
    fi
    x ← x(t_{i+1})
    λ ← λ(t_{i+1})
    repeat
        Solve (A(t_{i+1}) − λ)y = x
        λ ← λ + x* · y/‖y‖
        x ← y/‖y‖
    until ‖(A(t_{i+1}) − λ)x‖ < error tolerance
end
```

REFERENCES

[1] J. J. DONGARRA AND D. C. SORENSEN, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 139–154.

[2] I. C. F. IPSEN AND E. R. JESSUP, *Two methods for solving the symmetric tridiagonal eigenvalue problem on the hypercube*, in Proc. 2nd Conference on Hypercube Multiprocessors, Knoxville, TN, September 29–October 1, 1986, pp. 627–647.

[3] S. L. JOHNSSON, *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 354–392.

[4] T. Y. LI AND N. H. RHEE, *Homotopy algorithm for symmetric eigenvalue problems*, Numer. Math., 55 (1989), pp. 265–280.

[5] S. S. LO, B. PHILIPPE, AND A. SAMEH, *A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 155–165.

[6] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[7] Y. SAAD AND M. H. SCHULZ, *Data communication in hypercubes*, Res. Report 428, Computer Science Department, Yale University, New Haven, CT, 1985.

[8] B. T. SMITH et al., *Matrix Eigensystem Routines*-EISPACK *Guide*, Second Edition, Springer-Verlag, New York, 1976.

[9] X. H. SUN, H. ZHANG, AND L. M. NI, *Parallel algorithms for solution of tridiagonal systems on multicomputers*, submitted.

[10] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, New York, 1965.

[11] M. T. CHU, *A simple application of the homotopy method to symmetric eigenvalue problems*, Linear Algebra Appl., 59 (1984), pp. 85–90.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 488-504, May 1991
© 1991 Society for Industrial and Applied Mathematics
002

POLAR DECOMPOSITION AND MATRIX SIGN FUNCTION CONDITION ESTIMATES*

CHARLES KENNEY† AND ALAN J. LAUB†

Abstract. This paper presents reliable condition estimation procedures, based on Fréchet derivatives, for polar decomposition and the matrix sign function. For polar decomposition, the condition number for complex matrices is equal to the reciprocal of the smallest singular value, and rather surprisingly, for real matrices it is equal to the reciprocal of the average of the two smallest singular values. By using inverse power methods, both of these condition numbers can be evaluated at a fraction of the cost of finding the polar decomposition.

Except for special cases, such as for normal matrices, the condition number of the matrix sign function does not have such a precise characterization. However, accurate condition estimates can be obtained by using explicit forms of the Fréchet derivative, or its finite-difference approximation, with a matricial inverse power method. These methods typically require two extra sign function evaluations, and it is an open problem whether accurate estimates can be obtained for a fraction of a function evaluation, as is the case for the polar decomposition. Related results for the stable Lyapunov equation and Newton's method for the matrix square root problem are discussed.

Key words. polar decomposition, matrix sign function, conditioning

AMS(MOS) subject classifications. 65F35, 15A12

1. Introduction.

Both the polar decomposition and the matrix sign function play important roles in many matrix algorithms [1]-[3], [8], [9], [14], [17]. The sensitivity of these matrix functions is determined by the norms of their Fréchet derivatives. More specifically, let $F = F(X)$ be a matrix function which is continuously differentiable at X in the sense that there exists a linear matrix operator $L(\cdot) = L(\cdot, X, F)$ such that for any matrix Z,

$$\lim_{\delta \to 0} \frac{F(X+\delta Z) - F(X)}{\delta} = L(Z). \tag{1.1}$$

Then L is the Fréchet derivative of F at X and we define the absolute and relative condition numbers of F at X, with respect to a matrix norm $\|\cdot\|$, by $K_a(F, X) = \|L\| = \max_{\|Z\| \neq 0} \|L(Z)\|/\|Z\|$, and $K_r(F, X) = \|L\| \|X\|/\|F(X)\|$ when $\|F(X)\| \neq 0$. This definition is consistent with the condition theory of Rice [16] (see [13]). For δ small and $\|Z\| = 1$, we see from (1.1) that

$$\|\Delta F\| \equiv \|F(X+\delta Z) - F(X)\| \cong \|L(Z)\|\delta \leqq \|L\|\delta,$$

which motivates the use of $\|L\|$ as a condition measure for the mapping $X \to F(X)$.

As a simple example, if $F(X) = X^2$ then

$$\frac{F(X+\delta Z) - F(X)}{\delta} = ZX + XZ + \delta Z^2,$$

so $L(Z) = ZX + XZ$ and $\|L\| \leqq 2\|X\|$.

* Received by the editors June 21, 1989; accepted for publication (in revised form) December 4, 1989. This research was supported by National Science Foundation (and Air Force Office of Scientific Research) grant ECS 87-18897, National Science Foundation grant DMS88-00817, and Air Force Office of Scientific Research contract AFOSR-89-0167.

† Department of Electrical and Computer Engineering, University of California, Santa Barbara, California 93106.

In the following, we give explicit representations for the derivatives of both the polar decomposition and matrix sign functions, and discuss numerical procedures for estimating the condition numbers of these functions.

2. Polar decomposition. Let $A \in \mathbb{C}^{m\times m}$ be nonsingular. Then (e.g., [9, Thm. 1.1]) A has a unique polar decomposition (U, H) such that

$$A = UH, \tag{2.1}$$

where $U \in \mathbb{C}^{m\times m}$ is unitary,

$$I = U^*U, \tag{2.2}$$

and $H \in \mathbb{C}^{m\times m}$ is Hermitian and positive definite,

$$H = H^* > 0. \tag{2.3}$$

In this paper, a superscript * will denote the complex conjugate transpose of a matrix. The polar decomposition can be obtained from the singular value decomposition of A: if

$$A = P\Sigma Q^*, \tag{2.4}$$

where P and Q are unitary and $\Sigma = \operatorname{diag}(\sigma_1, \cdots, \sigma_m)$ with $\sigma_1 \geqq \sigma_2 \geqq \cdots \geqq \sigma_m > 0$, then

$$U = PQ^*, \tag{2.5}$$

$$H = Q\Sigma Q^*. \tag{2.6}$$

Since $A^* = Q\Sigma P^*$, A^* has the polar decomposition (V, G) where

$$V = QP^* = U^*, \tag{2.7}$$

and

$$G = P\Sigma P^* = UHU^*. \tag{2.8}$$

A popular method for finding U is to let $X_0 = A$ and use the globally convergent Newton iteration [9]

$$X_{n+1} = \tfrac{1}{2}(X_n + X_n^{-*}) \tag{2.9}$$

where $X_n^{-*} = (X_n^*)^{-1}$. Then $\lim_{n\to+\infty} X_n = U$ and H can be recovered from (2.1) and (2.2):

$$H = U^*A.$$

Let $F_1(A) = U$, $F_2(A) = H$ and let L_U and L_H denote the respective Fréchet derivatives:

$$L_U(Z) \equiv L(Z, A, F_1), \qquad L_H(Z) \equiv L(Z, A, F_2).$$

Our goal is to find expressions for the polar condition numbers $K_U = \|L_U\|$ and $K_H = \|L_H\|$. For convenience we will work with the Frobenius norm $\|A\|^2 = \Sigma|a_{ij}|^2$. An important result along these lines is due to Higham [9].

THEOREM 2.1. *Let* $\Delta H = L_H(\Delta A)$ *and* $\Delta U = L_U(\Delta A)$. *Then* $H\Delta H + \Delta H H = A^*\Delta A + \Delta A^* A$, $\Delta U = (\Delta A - U\Delta H)H^{-1}$. *Moreover,* $K_H \leqq \sqrt{2}$ *and*

$$K_U \leqq (1+\sqrt{2})\sqrt{m}\,\|A^{-1}\|. \tag{2.10}$$

Proof. This is Theorem 2.5 of [9] and is based on perturbations of the identities $H^2 = A^*A$ and $U = AH^{-1}$. □

This result is quite nice because the important term $\|A^{-1}\|$ can be estimated during the first step of the recursion (2.9). Moreover, the next theorem shows that the upper bound (2.10) is close to the true value of K_U; in fact, we can exhibit the exact value of K_U.

THEOREM 2.2. *Let $G = UHU^*$ as in* (2.8) *and let $\Delta H = L_H(\Delta A)$, $\Delta U = L_U(\Delta A)$. Then*

$$G\Delta U + \Delta U H = \Delta A - U\Delta A^* U, \tag{2.11}$$

and the (absolute) polar condition number for U is given by

$$K_U = \frac{1}{\sigma_m} \tag{2.12}$$

where $\sigma_m = \sigma_m(A)$ is the smallest singular value of A.

Proof. Standard perturbation arguments, such as those used in [9] or [13], when applied to (2.1)-(2.3), give

$$\Delta A = \Delta U H + U\Delta H, \tag{2.13}$$

$$\begin{aligned} 0 &= \Delta U^* U + U^* \Delta U, \\ \Delta H &= \Delta H^*. \end{aligned} \tag{2.14}$$

Now take the conjugate transpose of (2.13) and multiply on the right and left by U. Then use (2.14) to get

$$U\Delta A^* U = UH\Delta U^* U + U\Delta H = -UHU^*\Delta U + U\Delta H. \tag{2.15}$$

Subtracting (2.15) from (2.13) gives

$$\Delta A - U\Delta A^* U = UHU^*\Delta U + \Delta U H = G\Delta U + \Delta U H. \tag{2.16}$$

This proves (2.11). To prove (2.12), note that by (2.5), (2.6), and (2.8), we may rewrite (2.16) as

$$P\Sigma P^*\Delta U + \Delta U Q\Sigma Q^* = \Delta A - PQ^*\Delta A^* PQ^*.$$

Multiply on the left by P^* and on the right by Q to get

$$\Sigma P^*\Delta U Q + P^*\Delta U Q\Sigma = P^*\Delta A Q - Q^*\Delta A^* P. \tag{2.17}$$

Let $\Delta u = P^*\Delta U Q$ and $\Delta a = P^*\Delta A Q$. Since P and Q are unitary $\|\Delta u\| = \|\Delta U\|$, $\|\Delta a\| = \|\Delta A\|$, and (2.17) can be written as

$$\Sigma\Delta u + \Delta u\Sigma = \Delta a - \Delta a^*. \tag{2.18}$$

From (2.18) and $\Sigma = \operatorname{diag}(\sigma_1, \cdots, \sigma_m)$ we have

$$\begin{aligned} \Delta u_{ij} &= \frac{1}{\sigma_i + \sigma_j}(\Delta a_{ij} - \overline{\Delta a_{ji}}), \\ |\Delta u_{ij}|^2 &\leqq \frac{1}{(\sigma_i + \sigma_j)^2}(|\Delta a_{ij}| + |\Delta a_{ji}|)^2 \\ &\leqq \frac{1}{2\sigma_m^2}(|\Delta a_{ij}|^2 + |\Delta a_{ji}|^2) \end{aligned} \tag{2.19}$$

where we have used σ_i, $\sigma_j \geqq \sigma_m$ and $(|a| + |b|)^2 = |a|^2 + 2|a||b| + |b|^2 \leqq 2(|a|^2 + |b|^2)$. Summing over i and j in (2.19) gives

$$\|\Delta u\|^2 \leqq \frac{1}{2\sigma_m^2}(\|\Delta a\|^2 + \|\Delta a^*\|^2) = \frac{\|\Delta a\|^2}{\sigma_m^2}.$$

That is,

$$\|\Delta U\| = \|L_U(\Delta A)\| \leqq \frac{\|\Delta A\|}{\sigma_m},$$

so

$$\|L_U\| = \max_{\|Z\| \neq 0} \frac{\|L_U(Z)\|}{\|Z\|} \leqq \frac{1}{\sigma_m}. \tag{2.20}$$

The upper bound in (2.20) is attained if we set $\Delta A = P \Delta a Q^*$, where $\Delta a_{mm} = \sqrt{-1}$ and $\Delta a_{ij} = 0$ otherwise. □

Remark. The smallest singular value of a matrix A is easily estimated by the inverse power iteration method of [7] and so K_U can be efficiently estimated during the first step of (2.9).

If A is a real nonsingular matrix, then the matrices P and Q in the singular value decomposition of A can also be taken to be real. In this case it makes sense to restrict the allowable perturbations ΔA to the set of real matrices. For example, in (2.9) roundoff errors associated with finite precision arithmetic will be real if $X_0 = A$ is real. However, the maximizing perturbation in the proof of Theorem 2.2 is complex, thus raising the possibility that the real condition number

$$K_U(\text{real}) \equiv \max_{Z \in \mathbb{R}^{m \times m}} \frac{\|L_U(Z)\|}{\|Z\|} \tag{2.21}$$

may be much smaller than the complex condition number: $K_U(\text{real}) \ll K_U(\text{complex})$ when A is real. In fact, this is the case whenever $\sigma_m(A)$ is small compared to $\sigma_{m-1}(A)$, as the next theorem shows.

THEOREM 2.3. $K_U(\textit{real}) = 2/(\sigma_m + \sigma_{m-1})$.

Proof. Let ΔA be real and let $\Delta U = L_U(\Delta A)$. As in the proof of Theorem 2.2,

$$\Sigma \Delta u + \Delta u \Sigma = \Delta a - \Delta a^T$$

where $\Delta u = P^T \Delta U Q$, $\Delta a = P^T \Delta A Q$ are both real. Thus

$$(\Delta u)_{ij} = \frac{\Delta a_{ij} - \Delta a_{ji}}{\sigma_i + \sigma_j}.$$

Now when $i = j$, $(\Delta u)_{ij} = 0$. For $i \neq j$,

$$|\Delta u|_{ij}^2 \leqq \frac{2(\Delta a_{ij}^2 + \Delta a_{ji}^2)}{(\sigma_i + \sigma_j)^2} \leqq \frac{2(\Delta a_{ij}^2 + \Delta a_{ji}^2)}{(\sigma_m + \sigma_{m-1})^2}.$$

Summing over i and j gives

$$\|\Delta u\|^2 \leqq \frac{2(\|\Delta a\|^2 + \|\Delta a^T\|^2)}{(\sigma_m + \sigma_{m-1})^2} = \frac{4\|\Delta a\|^2}{(\sigma_m + \sigma_{m-1})^2},$$

$$\|\Delta u\| = \|\Delta U\| = \|L_U(\Delta A)\| \leqq \frac{2\|\Delta a\|}{\sigma_m + \sigma_{m-1}} = \frac{2\|\Delta A\|}{\sigma_m + \sigma_{m-1}}.$$

Thus $\max_{Z \in \mathbb{R}^{m \times m}} \|L_U(Z)\| / \|Z\| \leqq 2/(\sigma_m + \sigma_{m-1})$. Moreover, the upper bound is attained if we set $\Delta a_{m,m-1} = -\Delta a_{m-1,m} = 1/\sqrt{2}$ with $\Delta a_{ij} = 0$ otherwise. □

In order to illustrate Theorems 2.2 and 2.3, suppose that a singular value decomposition of A is used to compute the polar factors U and H from (2.5), (2.6). As in the backward error analysis of Higham [9, § 4], the computed factors U_C and H_C satisfy

$$U_C = U_1 + \Delta U_1, \qquad \|\Delta U_1\|_2 \leqq c\varepsilon,$$

$$H_C = H_1 + \Delta H_1, \qquad H_C^* = H_C, \qquad \|\Delta H_1\|_2 \leqq c\varepsilon \|H_1\|_2,$$

$$U_1 H_1 = A + \Delta A, \qquad \|\Delta A\|_2 \leqq c\varepsilon \|A\|_2$$

where U_1 is unitary, H_1 is Hermitian and positive semidefinite, c is a small constant near one, and ε is the relative machine precision. Moreover, if $\|A\|_2\|A^{-1}\|_2 < 1/c\varepsilon$ then H_1 is positive definite and (U_1, H_1) are the polar factors of $A+\Delta A$. That is, $U = F_1(A)$, $U_1 = F_1(A+\Delta A)$, and

$$\begin{aligned} \|U_1 - U\| &= \|F_1(A+\Delta A) - F_1(A)\| \\ &\leqq \|L_U\|\,\|\Delta A\| + O(\|\Delta A\|^2) \\ &\leqq c\varepsilon\sqrt{m}\,\|A\|_2\|L_U\| + O(\varepsilon^2). \end{aligned}$$

Thus

$$\begin{aligned} \frac{\|U_C - U\|}{\|U\|} &\leqq \frac{\|U_C - U_1\|}{\|U\|} + \frac{\|U_1 - U\|}{\|U\|} \\ &\leqq c\varepsilon + c\varepsilon\|A\|_2\|L_U\| + O(\varepsilon^2). \end{aligned}$$

From this, we can expect that the relative error in the computed factor U_C is bounded by a small multiple of the term

$$R \equiv \varepsilon\|A\|_2\|L_U\|.$$

The following example clearly demonstrates the dichotomy between real and complex perturbations of the polar factor U.

Example 1. Let

$$A_0 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & \mu \\ -1/\mu & 1 \end{bmatrix}, \quad T = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad U_0 = \frac{1}{\Gamma}\begin{bmatrix} 2 & \bar{\gamma} \\ -\gamma & 2 \end{bmatrix}$$

where

$\gamma = \bar{\mu} + 1/\mu$, $\Gamma = (4+\gamma\bar{\gamma})^{-1/2}$. Set $A = TA_0T$, $U = TU_0T$. Then

$$\sigma_1(A) = \frac{1}{\sqrt{2}}\left[\left(|\mu| + \frac{1}{|\mu|}\right)^2 + \left[\left(|\mu| + \frac{1}{|\mu|}\right)^4 - 16\right]^{1/2}\right]^{1/2} \cong \frac{|\mu|}{\sqrt{2}},$$

$$\sigma_2(A) = \frac{1}{\sigma_1(A)} \cong \frac{\sqrt{2}}{|\mu|},$$

for $|\mu|$ large. Because we know U exactly we can measure how well U is computed by (2.9) and compare the accuracy attained with either the real or the complex condition numbers. The results are given in Tables 1 and 2, respectively, for $\mu = 10^k$ and $\mu = (1+i)10^k$ with $0 \leqq k \leqq 6$. In Tables 1 and 2, U_c denotes the solution computed via (2.9) coded in Fortran on a SUN 3/60 computer using double precision (machine epsilon $= \varepsilon \cong 1.4 \cdot 10^{-17}$) arithmetic. Table 1 shows that the complex condition number is not appropriate when A is real, and that the real condition number accurately reflects the error in the computed solution. Conversely, when A is complex, as in Table 2, the reverse is true.

TABLE 1
Condition estimates when μ is real in Example 1.

Parameter value	Relative error in the computed solution	Real relative condition number times the machine epsilon	Complex relative condition number times the machine epsilon
μ	$\dfrac{\|U_c - U\|}{\|U\|}$	$\dfrac{2\varepsilon}{(\sigma_m + \sigma_{m-1})}\dfrac{\|A\|}{\|U\|}$	$\dfrac{\varepsilon}{\sigma_m}\dfrac{\|A\|}{\|U\|}$
10^0	0	1.39×10^{-17}	1.39×10^{-17}
10^1	1.11×10^{-16}	1.93×10^{-17}	5.01×10^{-16}
10^2	2.29×10^{-16}	1.97×10^{-17}	4.92×10^{-14}
10^3	4.07×10^{-17}	1.97×10^{-17}	4.91×10^{-12}
10^4	1.15×10^{-16}	1.97×10^{-17}	4.91×10^{-10}
10^5	2.19×10^{-16}	1.97×10^{-17}	4.91×10^{-8}
10^6	1.14×10^{-16}	1.97×10^{-17}	4.91×10^{-6}

TABLE 2
Condition estimates when μ is complex in Example 1.

Parameter value	Relative error in the computed solution	Real relative condition number times the machine epsilon	Complex relative condition number times the machine epsilon
μ	$\dfrac{\|U_c - U\|}{\|U\|}$	$\dfrac{2\varepsilon\|A\|}{(\sigma_m + \sigma_{m-1})\|U\|}$	$\dfrac{\varepsilon\|A\|}{\sigma_m\|U\|}$
$(1+i)\times10^0$	2.23×10^{-16}	1.43×10^{-17}	1.89×10^{-17}
$(1+i)\times10^1$	1.11×10^{-15}	1.95×10^{-17}	9.93×10^{-16}
$(1+i)\times10^2$	2.41×10^{-13}	1.97×10^{-17}	9.82×10^{-14}
$(1+i)\times10^3$	1.07×10^{-12}	1.97×10^{-17}	9.82×10^{-12}
$(1+i)\times10^4$	3.16×10^{-9}	1.97×10^{-17}	9.82×10^{-10}
$(1+i)\times10^5$	2.05×10^{-7}	1.97×10^{-17}	9.83×10^{-8}
$(1+i)\times10^6$	2.83×10^{-5}	1.97×10^{-17}	9.83×10^{-6}

In practice, we need to estimate $\sigma_m(A)$ and $\sigma_{m-1}(A)$ in order to evaluate $K_U(\text{real}) = 2/(\sigma_m + \sigma_{m-1})$. On the first step of (2.9) we have available to us A^{-1}, so we can use a two vector version of the inverse power method to estimate σ_m and σ_{m-1}. Given v_1 and v_2 with $v_1^* v_2 = 0$, define $\tilde{z}_1$ and $\tilde{z}_2$ by $A\tilde{z}_i = v_i$. Now orthogonalize $\tilde{z}_1$ and $\tilde{z}_2$ to get z_1, z_2:

$$z_1 = \tilde{z}_1/\|\tilde{z}_1\|_2, \qquad z_2 = (\tilde{z}_2 - z_1^*\tilde{z}_2 z_1)/\|\tilde{z}_2 - z_1^*\tilde{z}_2 z_1\|_2,$$

where $\|\cdot\|_2$ denotes the vector 2-norm. The second part of the cycle gives $\tilde{v}_1$, $\tilde{v}_2$ from $A^*\tilde{v}_i = z_i$. Then $\|\tilde{v}_1\|_2 \cong 1/\sigma_m$ and $\|\tilde{v}_2\|_2 \cong 1/\sigma_{m-1}$. Better estimates can be obtained by reorthogonalizing $\tilde{v}_1$, $\tilde{v}_2$ to get v_1 and v_2 and then repeating the cycle. The orthogonalization after solving $A\tilde{z}_i = v_i$ is redundant but protects the accuracy of the estimates for very ill-conditioned problems ($\sigma_m \ll \sigma_{m-1}$).

In sum, for the polar decomposition of a nonsingular matrix, accurate condition estimates can be obtained with very little computational effort. In the next section, we show that accurate condition estimates for the matrix sign function are considerably more expensive, with the computational cost being equal to two or more sign function evaluations.

3. Matrix sign function. The matrix sign function is closely related to the polar decomposition since each essentially solves a square root problem. In this section, we assume that $X \in \mathbb{C}^{m\times m}$ has no eigenvalues on the imaginary axis. For a complex number ω, with $\operatorname{Re}\omega \neq 0$, let

$$\operatorname{sgn}(\omega) = \begin{cases} 1 & \text{if } \operatorname{Re}\omega > 0, \\ -1 & \text{if } \operatorname{Re}\omega < 0. \end{cases} \tag{3.1}$$

The matrix analogue, $\operatorname{sgn}(X)$, can be defined by using the Jordan form of X (see [2]). Let $X = T(D+N)T^{-1}$, where T is nonsingular, $D = \operatorname{diag}(d_1, \cdots, d_m)$, and N is nilpotent and commutes with D. Define

$$\operatorname{sgn}(X) = T \operatorname{Diag}(\operatorname{sgn}(d_1), \cdots, \operatorname{sgn}(d_m))T^{-1}. \tag{3.2}$$

From this it follows that $S \equiv \operatorname{sgn}(X)$ is a diagonalizable matrix, commutes with X,

$$SX = XS, \tag{3.3}$$

and is a square root of the identity

$$I = S^2. \tag{3.4}$$

A Cauchy integral representation for $\operatorname{sgn}(X)$ can also be given [17], which shows that the mapping $X \to \operatorname{sgn}(X)$ is analytic. We could also define $\operatorname{sgn}(X)$ to be the limit of the sequence $\{X_n\}$ given by the Newton recursion (see [1])

$$X_{n+1} = \tfrac{1}{2}(X_n + X_n^{-1}), \qquad X_0 = X. \tag{3.5}$$

Equations (3.3) and (3.4) yield one representation of the Fréchet derivative of $\operatorname{sgn}(X)$, while (3.5) gives another via the chain rule for composite matrix functions. In the following, for $F(X) = \operatorname{sgn}(X)$, let $L(Z) = L(Z, F, X)$ denote the derivative of F at X in the (matrix) direction Z.

THEOREM 3.1. *Let $S = \operatorname{sgn}(X)$, $\Delta S = L(\Delta X)$ and $B = SX$. Then ΔS satisfies the (uniquely solvable) Sylvester equation*

$$B\Delta S + \Delta S B = \Delta X - S\Delta X S. \tag{3.6}$$

Proof. Relations (3.3) and (3.4) give

$$S\Delta X + \Delta S X = \Delta X S + X \Delta S, \tag{3.7}$$

$$0 = S\Delta S + \Delta S S. \tag{3.8}$$

Rewrite (3.7) as

$$X\Delta S - \Delta S X = S\Delta X - \Delta X S. \tag{3.9}$$

In general, neither (3.8) nor (3.9) has a unique solution, but together they determine ΔS. Multiply (3.9) on the left by S to get $SX\Delta S - S\Delta S X = \Delta X - S\Delta X S$. Now use (3.8) as $-S\Delta S = \Delta S S$ to write

$$SX\Delta S + \Delta S S X = \Delta X - S\Delta X S. \tag{3.10}$$

This is the same as (3.6). Since the eigenvalues of SX lie in the open right-half complex plane, (3.10) is uniquely solvable for ΔS, given ΔX. (See [15].) □

Remark. In the special case where X has all of its eigenvalues in the open right-half plane, $\operatorname{sgn}(X) = I = \operatorname{sgn}(X + \delta\Delta X)$ for δ sufficiently small. That is, $\operatorname{sgn}(X)$ is then locally constant and completely insensitive to small perturbations. This can also be seen from (3.10) because $\Delta X - S\Delta X S = 0$ when $S = I$, thus giving $\Delta S = 0$ independent of ΔX. This means that $L \equiv 0$ and the condition number $\|L\|$ is also zero, for this special case.

Superficially (3.6) is very close in form to (2.11), which raises the hope that we could use (3.6) to characterize the norm of L for sgn (X), just as (2.11) led to expressions for the conditioning of the polar decomposition. Unfortunately, we are unable to capitalize on the similarity between (3.6) and (2.11), principally because G and H in (2.11) are Hermitian and related via (2.6) and (2.8), whereas $B = SX$ is not Hermitian and generally is far from normal. We can, however, obtain a result in the special case when X is normal.

THEOREM 3.2. *Let X be normal with $X = P\Lambda P^*$ where P is unitary and $\Lambda = \operatorname{diag}(\lambda_1, \cdots, \lambda_m)$. If all of the eigenvalues of X lie in the open right-half plane or all lie in the open left-half plane, then*

$$\|L\| = 0. \tag{3.11}$$

Otherwise,

$$\|L\| = \frac{2}{\min |\lambda_i + \lambda_j|} \equiv \alpha, \tag{3.12}$$

where the minimum is taken over all eigenvalues of opposite sign: $\operatorname{sgn}(\lambda_i)\operatorname{sgn}(\lambda_j) < 0$.

Proof. Since $X = P\Lambda P^*$, $S = \operatorname{sgn}(X) = P \operatorname{sgn}(\Lambda) P^*$ and $SX = P\Lambda_+ P^*$, where $\Lambda_+ = \operatorname{diag}(\operatorname{sgn}(\lambda_1)\lambda_1, \cdots, \operatorname{sgn}(\lambda_m)\lambda_m)$. Multiply (3.6) on the left by P^* and on the right by P to get $\Lambda_+ \Delta s + \Delta s \Lambda_+ = \Delta x - s \Delta x s$ where $\Delta s = P^* \Delta S P$, $\Delta x = P^* \Delta X P$, and $s = \operatorname{sgn}(\Lambda)$. Then $\|\Delta s\| = \|\Delta S\|$, $\|\Delta x\| = \|\Delta X\|$ and

$$\Delta s_{ij} = \frac{(1 - \operatorname{sgn}(\lambda_i)\operatorname{sgn}(\lambda_j))}{(\operatorname{sgn}(\lambda_i)\lambda_i + \operatorname{sgn}(\lambda_j)\lambda_j)} \Delta x_{ij}.$$

If λ_i and λ_j are in the same half plane, then $\operatorname{sgn}(\lambda_i)\operatorname{sgn}(\lambda_j) = 1$ and $\Delta s_{ij} = 0$. If all of the eigenvalues are in the same half plane, then (3.11) follows immediately, since $L(\Delta X) = 0$ for any ΔX. When λ_i and λ_j are in opposite half planes $\operatorname{sgn}(\lambda_i)\operatorname{sgn}(\lambda_j) = -1$ and

$$\Delta s_{ij} = \left(\frac{2 \Delta x_{ij}}{\lambda_i - \lambda_j} \right) \operatorname{sgn}(\lambda_i).$$

Thus, for α as in (3.12), $|\Delta s_{ij}| \leqq \alpha |\Delta x_{ij}|$, and $\|L(\Delta X)\| = \|\Delta S\| \leqq \alpha \|\Delta X\|$. This gives $\|L\| \leqq \alpha$. Now let i and j be such that $2/|\lambda_i + \lambda_j| = \alpha$ and select $\Delta x_{ij} = 1$, $\Delta x_{kl} = 0$ otherwise. Then $\|\Delta S\| = \alpha \|\Delta X\|$ which gives $\|L\| = \alpha$. □

From this result we see that the sensitivity of the sign function for normal matrices is completely determined by the smallest distance between eigenvalues across the imaginary axis. (This was also proved by Byers for symmetric matrices in [6].) The smaller this distance, the greater the ill-conditioning. Similar results can be obtained by using the Jordan form when X is diagonalizable but not necessarily normal. Using essentially the same analysis we obtain

$$\|L\| \leqq \alpha \|T\|^2 \|T^{-1}\|^2 \tag{3.13}$$

where T is the similarity matrix which diagonalizes X. However, this bound tends to be very conservative when T is ill-conditioned and, moreover, algorithms to compute the Jordan canonical form in a fixed precision of arithmetic are not generally reliable.

Byers [6] has given some general conditioning bounds that involve the Schur decomposition of X and the separation between the positive and negative invariant subspaces of X. However, the computational cost of evaluating these bounds may be high, especially for large-scale problems in which the sign function has been selected as an alternative to Schur decomposition. Instead we use the condition estimation

procedure in [13]. For this it is helpful to have a second characterization of the Fréchet derivative of the matrix sign function.

THEOREM 3.3. *For X_n defined by* (3.5), *let $\Delta X_0 = \Delta X$ and set*

$$\Delta X_{n+1} = \tfrac{1}{2}(\Delta X_n - X_n^{-1}\Delta X_n X_n^{-1}). \tag{3.14}$$

Then $\lim_{n\to+\infty} \Delta X_n = L(\Delta X) = \Delta S$.

Proof. Let $F_1(X) \equiv \frac{1}{2}(X + X^{-1})$, $F_2(X) = F_1(F_1(X)), \cdots, F_n(X) = F_1(F_{n-1}(X))$. Then $F_n(X) = X_n$ and $\lim_{n\to+\infty} F_n(X) = \operatorname{sgn}(X)$. Let $L_1(\cdot, X)$ and $L_n(\cdot, X)$ denote the Fréchet derivatives of F_1 and F_n at X, respectively. Then

$$L_1(\Delta X, X) = \tfrac{1}{2}(\Delta X - X^{-1}\Delta X X^{-1}) = \Delta X_1 \tag{3.15}$$

as in (3.14). By Lemma 6.2 in [13], the Fréchet derivative of the composition $F_2 = F_1 \circ F_1$ is simply the composition of the corresponding Fréchet derivatives at $F_1(X)$ and X, respectively:

$$L_2(\Delta X, X) = L_1(L_1(\Delta X, X), F_1(X)).$$

This can also be written, by (3.15), as

$$\Delta X_1 = \tfrac{1}{2}(\Delta X_0 - X_0^{-1}\Delta X X_0^{-1}),$$
$$\Delta X_2 = \tfrac{1}{2}(\Delta X_1 - X_1^{-1}\Delta X X_1^{-1})$$

where $L_2(\Delta X, X) = \Delta X_2$. Continuing in this way we see that $L_n(\Delta X, X) = \Delta X_n$ as in (3.14). Now an analysis similar to that given in [13] shows that

$$\lim_{n\to+\infty} L_n(\Delta X, X) = L(\Delta X, X) = \Delta S. \qquad \square$$

Given ΔX, Theorem 3.1 or 3.3 allows us to find $L(\Delta X)$, which is crucial in terms of estimating the norm of L. A third method of approximating $L(\Delta X)$ is simply to use the relation (1.1) for sufficiently small δ:

$$L(\Delta X) \cong (\operatorname{sgn}(X + \delta\Delta X) - \operatorname{sgn}(X))/\delta. \tag{3.16}$$

Since the Frobenius norm of $L = L(\cdot, X)$ is equal to the 2-norm of the associated Kronecker matrix, the power method estimation technique of [13] can be used. Starting with Z_0 of unit Frobenius norm, let W and Z_1 satisfy

$$\tilde{W} = L(Z_0, X), \tag{3.17}$$

$$W = \tilde{W}/\|\tilde{W}\|, \tag{3.18}$$

$$Z_1 = L^*(W, X) \tag{3.19}$$

where $L^*(\cdot, X)$ denotes the conjugate transpose of the linear operator $L(\cdot, X)$ and is given by $L^*(W, X) = L(W, X^*)$. For suitably chosen Z_0, $\|Z_1\| \cong \|L\|$ and more accurate estimates can be obtained by repeating the cycle with $Z_0 = Z_1/\|Z_1\|$. (A similar estimation procedure for the 1-norm of L can be given [10].) For the method of evaluating $L(\Delta X)$ in Theorem 3.1, (3.17)-(3.19) take the form of two Sylvester equations, with $B = SX$:

$$B\tilde{W} + \tilde{W}B = Z_0 - SZ_0S, \tag{3.20}$$

$$W = \tilde{W}/\|\tilde{W}\|, \tag{3.21}$$

$$B^*Z_1 + Z_1B^* = W - S^*WS^*. \tag{3.22}$$

Note that this last equation may be written as

$$BZ + ZB = W^* - SW^*S, \qquad Z_1 = Z^*. \tag{3.23}$$

How expensive is it to solve (3.20) and (3.22) (or (3.23))? If we use a matrix sign approach to solve (3.20), i.e., if we exploit the fact that (see [17])

$$\operatorname{sgn}\begin{bmatrix} B & \frac{1}{2}(Z_0 - SZ_0S) \\ 0 & -B \end{bmatrix} = \begin{bmatrix} I & \tilde{W} \\ 0 & -I \end{bmatrix}, \tag{3.24}$$

then $\tilde{W} = \lim_{n\to+\infty} Y_n$ where

$$X_{n+1} = \tfrac{1}{2}(X_n + X_n^{-1}), \qquad X_0 = B, \tag{3.25}$$

$$Y_{n+1} = \tfrac{1}{2}(Y_n + X_n^{-1} Y_n X_n^{-1}), \qquad Y_0 = \tfrac{1}{2}(Z_0 - SZ_0S). \tag{3.26}$$

This takes one matrix inversion and two multiplies per step and converges at exactly the same rate as (3.5), which requires one inversion per step. How does this compare with the method of evaluating $L(\Delta X)$ in Theorem 3.3? Using (3.14), we may write (3.17)-(3.19) as

$$W_{n+1} = \tfrac{1}{2}(W_n - X_n^{-1} W_n X_n^{-1}), \qquad W_0 = Z_0, \tag{3.27}$$

$$Y_{n+1} = \tfrac{1}{2}(Y_n - X_n^{-*} Y_n X_n^{-*}), \tag{3.28}$$

$$Z_1 = \lim_{n\to+\infty} Y_n, \tag{3.29}$$

where $X_{n+1} = \frac{1}{2}(X_n + X_n^{-1})$; $X_0 = X$ as in (3.5) and $Y_0 = \lim_{n\to+\infty} W_n / \| W_n \|$. From (3.27)-(3.29), this method of evaluating $L(\Delta X)$ is just as expensive as the method (3.20)-(3.22) based on (3.25)-(3.26). Note, however, that some savings is possible since on the first step (3.27) can be evaluated concurrently with the determination of $\operatorname{sgn}(X)$ via (3.5). This is not true for (3.25)-(3.26) since S is unknown until $\operatorname{sgn}(X) = S$ has been found.

The last method, based on the finite-difference approximation (3.16), takes the form

$$\tilde{W} = (\operatorname{sgn}(X + \delta Z_0) - \operatorname{sgn}(X))/\delta, \tag{3.30}$$

$$W = \tilde{W} / \| \tilde{W} \|,$$

$$Z_1 = (\operatorname{sgn}(X^* + \delta W) - \operatorname{sgn}(X^*))/\delta, \tag{3.31}$$

with $\| L \| \cong \| Z_1 \|$. This method requires two sign function evaluations per cycle, when we use the fact that $\operatorname{sgn}(X^*) = (\operatorname{sgn}(X))^*$.

Independently of which method is used to evaluate $L(Z_0)$, if Z_0 is not properly chosen then several cycles of (3.17)-(3.19) may be necessary to get a good estimate of $\| L \|$. In general, because each of the three methods for finding $L(Z_0)$ is iterative in nature, it is not feasible to use "look-ahead" methods to select Z_0, such as those in [7] and [5].

In the absence of a heuristic procedure, Z_0 can be generated randomly since it will almost always then have a significant component in the matrix direction which maximizes $\| L(Z_0) \|$. (See [7] for a discussion of this point.)

The following lemma is very useful for generating examples to test condition estimates.

LEMMA 3.4. *Let* $X = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ *be real. Then*

$$\operatorname{sgn}(X) = \begin{cases} \dfrac{1}{\sqrt{(a-d)^2 + 4bc}} \begin{bmatrix} a-d & 2b \\ 2c & d-a \end{bmatrix} & \text{if } ad < bc, \\ \operatorname{sgn}(a+d) I \quad \text{if } ad > bc \text{ and } a + d \neq 0. \end{cases} \tag{3.32}$$

Otherwise, $\operatorname{sgn}(X)$ *is undefined.*

Proof. If $ad < bc$, then X has one real positive eigenvalue and one real negative eigenvalue. Manipulating the Jordan form (3.2) gives the first expression in (3.32). If

$ad > bc$ and $a+d \neq 0$, then X has both eigenvalues in either the right- or left-half plane depending on whether $\operatorname{sgn}(a+d)$ is positive or negative. In all other cases, X has eigenvalues on the imaginary axis and $\operatorname{sgn}(X)$ is undefined. □

Example 2. Let $X = TDT$ where

$$T = T^{-1} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

is the 4×4 Walsh matrix and

$$D = \operatorname{diag}(D_1, D_2), \quad D_1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad D_2 = \begin{bmatrix} 1 & \mu \\ -1/\mu & 1 \end{bmatrix} \quad \text{for } \mu \in \mathbb{R}.$$

Then $\operatorname{sgn}(X) = T \operatorname{sgn} DT$ where $\operatorname{sgn} D = \operatorname{diag}(\operatorname{sgn}(D_1), \operatorname{sgn}(D_2))$ with $\operatorname{sgn}(D_1) = (1/\sqrt{33})\left[\begin{smallmatrix} -3 & 4 \\ 6 & 3 \end{smallmatrix}\right]$ and $\operatorname{sgn}(D_2) = I$ by Lemma 3.4. Table 3 gives the results of computing $\operatorname{sgn}(X)$ by (3.5), which we denote by S_c, and compares the relative error in the computed solution with the condition estimates obtained by using (3.30)-(3.31) with $\delta = 1000\varepsilon\|X\|$ as per the recommendations in [13]. Here ε is the relative machine precision and $\mu = 10^k$ for $0 \leqq k \leqq 6$.

TABLE 3
Condition estimates versus error for Example 2.

Parameter value	Relative error in the computed solution	Relative condition number times the relative machine precision
μ	$\dfrac{\|S_c - S\|}{\|S\|}$	$\dfrac{\|L\|_{\text{est}}\|X\|\varepsilon}{\|S\|}$
10^0	1.13×10^{-16}	4.77×10^{-17}
10^1	2.67×10^{-16}	5.62×10^{-16}
10^2	3.25×10^{-14}	4.81×10^{-14}
10^3	5.22×10^{-12}	4.81×10^{-12}
10^4	2.61×10^{-10}	4.82×10^{-10}
10^5	1.98×10^{-8}	4.83×10^{-8}
10^6	2.59×10^{-6}	3.6×10^{-6}

Remark. In order to relate the condition number to the error in the computed solution, we need to know that the algorithm used is backwards stable (i.e., solves a nearby problem). Although Byers [6] has shown that Newton's method is backwards stable for well-conditioned matrix sign problems, this may not be true for ill-conditioned problems. Because of this, a large condition number for a particular problem should simply be viewed as an indication that the computed solution may not be accurate.

4. Condition estimates for Lyapunov equations. In this section we show that reliable condition estimates for Lyapunov equations can be obtained at a cost of about 25 percent of the effort needed to solve the Lyapunov equation by the matrix sign function method. Let $A \in \mathbb{R}^{m\times m}$ be stable in the sense that all of its eigenvalues lie in the open left-half complex plane. The solution, Z, to the Lyapunov equation

$$A^T Z + ZA = -Q, \tag{4.1}$$

can be found [17] from the sign function identity

$$\operatorname{sgn}\begin{bmatrix} -A^T & \frac{1}{2}Q \\ 0 & A \end{bmatrix} = \begin{bmatrix} I & Z \\ 0 & -I \end{bmatrix}. \tag{4.2}$$

The associated Newton iteration is

$$A_{n+1} = \tfrac{1}{2}(A_n + A_n^{-1}), \qquad A_0 = A, \tag{4.3}$$

$$Z_{n+1} = \tfrac{1}{2}(Z_n + A_n^{-T} Z_n A_n^{-1}), \qquad Z_0 = \tfrac{1}{2}Q. \tag{4.4}$$

The Fréchet derivative of the Lyapunov mapping $(A, Q) \to Z$ is given by

$$\Delta Z = L(\Delta A, \Delta Q),$$

where

$$A^T \Delta Z + \Delta Z A = -(\Delta Q + \Delta A^T Z + Z \Delta A). \tag{4.5}$$

Let $\Omega(M) \equiv A^T M + MA$. Then (4.5) can be written as

$$\begin{aligned} &\Delta Z = -\Omega^{-1}(\Delta Q + \Delta A^T Z + Z \Delta A), \\ &\|\Delta Z\| \leqq \|\Omega^{-1}\|(\|\Delta Q\| + 2\|Z\|\,\|\Delta A^T\|). \end{aligned} \tag{4.6}$$

From (4.6), we see that $\|\Omega^{-1}\|$ determines the conditioning of (4.1). For a detailed discussion, see [12], in which it is proved that for the 2-norm

$$\|\Omega^{-1}\|_2 = \|H\|_2, \tag{4.7}$$

where H satisfies

$$A^T H + HA = -I. \tag{4.8}$$

In order to estimate $\|H\|_2$, we can supplement (4.3), (4.4) with

$$H_{n+1} = \tfrac{1}{2}(H_n + A_n^{-T} H_n A_n^{-1}), \qquad H_0 = \tfrac{1}{2}I. \tag{4.9}$$

Moreover, we really do not need to take very many steps in (4.9) in order to get a good estimate of $\|H\|_2$. The following lemma provides a means of determining when to stop in (4.9).

LEMMA 4.1. *Let H_n be defined by* (4.9). *Then*

$$A^T H_n + H_n A = \frac{A_n^T + A_n}{2}. \tag{4.10}$$

Proof. For $n = 0$, (4.10) is an identity since $H_0 = \frac{1}{2}I$. As an induction hypothesis, assume that (4.10) is true for n. Then, by (4.9)

$$A^T H_{n+1} + H_{n+1} A = \tfrac{1}{2}(A^T H_n + H_n A) + \tfrac{1}{2}(A^T A_n^{-T} H_n A_n^{-1} + A_n^{-T} H_n A_n^{-1} A).$$

But by (4.3), $A_n^{-1} A = A A_n^{-1}$ and $A^T A_n^{-T} = A_n^{-T} A^T$. So,

$$\begin{aligned} A^T H_{n+1} + H_{n+1} A &= \frac{1}{2}(A^T H_n + H_n A) + \frac{1}{2} A_n^{-T}(A^T H_n + H_n A) A_n^{-1} \\ &= \frac{1}{2}\frac{(A_n^T + A_n)}{2} + \frac{1}{2} A_n^{-T} \frac{(A_n^T + A_n)}{2} A_n^{-1} \\ &= \frac{1}{2}\left(\frac{A_n^T + A_n^{-T}}{2} + \frac{A_n + A_n^{-1}}{2}\right) \\ &= \frac{1}{2}(A_{n+1}^T + A_{n+1}). \end{aligned}$$

□

LEMMA 4.2. *Let* $\|(A_n^T + A_n)/2 + I\|_2 \equiv \xi < 1$. *Then*

$$\frac{\|H_n\|_2}{1+\xi} \leqq \|H\|_2 \leqq \frac{\|H_n\|_2}{1-\xi}. \tag{4.11}$$

Proof. Since $A^T H + HA = -I$, $A^T(H_n - H) + (H_n - H)A = (A_n^T + A_n)/2 + I$, by the last lemma. Thus $H_n - H = \Omega^{-1}((A_n^T + A_n)/2 + I)$, $\|H - H_n\|_2 \leqq \|\Omega^{-1}\|_2 \xi = \|H\|_2 \xi$, because of (4.7). From this we have $\|H\|_2 \leqq \|H_n\|_2 + \|H - H_n\|_2 \leqq \|H_n\|_2 + \xi\|H\|_2$. Consequently, $\|H\|_2 \leqq \|H_n\|_2/(1-\xi)$. Similarly, $\|H_n\|_2/(1+\xi) \leqq \|H\|_2$. □

To illustrate this result, if $\xi \leqq \frac{1}{2}$, then $\|H_n\|_2$ determines $\|H\|_2$ to within a factor of $(1+\xi)/(1-\xi) \leqq 3$, which is certainly adequate for condition estimation purposes.

How expensive is this estimate? Extensive numerical testing shows that the condition $\xi \leqq \frac{1}{2}$ is satisfied in about half the steps needed for convergence of Z_n to Z in (4.4). Each step of (4.3), (4.4) takes one inversion and two matrix multiplies, or about three matrix multiplies total if we count an inversion as approximately equivalent to a multiply. Since each H_n is symmetric, it takes about $\frac{3}{2}$ of a matrix multiply to evaluate (4.9). If $\xi \leqq \frac{1}{2}$, then assuming half the steps needed for convergence of Z_n, the total cost of estimating $\|H\|_2$ is less than 25 percent of the cost of solving (4.1) via (4.3), (4.4). If Q is symmetric then this changes to about 27 percent, and for parallel implementations the ratios are even lower.

5. Asymptotic conditioning. For the square root problem $S^2 = C$, the sensitivity of S with respect to perturbations in C is governed by the Fréchet relation $S\Delta S + \Delta S S = \Delta C$. However, it has also been recognized that the ratio $\gamma = \|S\|^2/\|S^2\|$ has some bearing on sensitivity [4], [5], [11]. For the matrix sign function ($C = I$), this ratio determines what we will call the asymptotic condition number of S. In this section, we restrict ourselves to the special case $C = I$.

As motivation, note that $\operatorname{sgn}(X) = \operatorname{sgn}(X_n)$ for each X_n in the Newton sequence (3.5), but the conditioning of the mappings $X \to \operatorname{sgn}(X)$ and $X_n \to \operatorname{sgn}(X_n)$ can be very different. This is principally because the first mapping depends on the eigenvalues as well as the eigenstructure of X whereas the second depends only on the eigenstructure of X (when n is large enough so that the eigenvalues of X_n are near ± 1). Thus we are led to the idea of studying the conditioning of $Y \to \operatorname{sgn}(Y) = S$ for Y near S, and we will let $L(\cdot, Y)$ denote the Fréchet derivative of this map. The next theorem shows that the norm of L is essentially equal to $\|S\|^2$.

THEOREM 5.1. *Let* Y *be near enough to* $S = \operatorname{sgn}(Y)$ *so that* $\|Y - S\|\,\|S\| < 1$. *Then*

$$\frac{\|S\|^2 - 1}{2(1 + \|Y - S\|\,\|S\|)} \leqq \|L(\cdot, Y)\| \leqq \frac{\|S\|^2 + 1}{2(1 - \|Y - S\|\,\|S\|)}, \tag{5.1}$$

and

$$\frac{\|S\|^2 - 1}{2} \leqq \|L(\cdot, S)\| \leqq \frac{\|S\|^2 + 1}{2}. \tag{5.2}$$

Proof. Let $\Delta S = L(\Delta Y, Y)$. Then by Theorem 3.1,

$$(SY)\Delta S + \Delta S(SY) = \Delta Y - S\Delta YS. \tag{5.3}$$

Writing $Y = S + Y - S$ and rearranging terms,

$$2\Delta S = \Delta Y - S\Delta YS + S(S - Y)\Delta S + \Delta S S(S - Y),$$

$$2\|\Delta S\| \leqq (\|S\|^2 + 1)\|\Delta Y\| + 2\|Y - S\|\,\|S\|\,\|\Delta S\|,$$

$$\|\Delta S\| \leqq \frac{(\|S\|^2 + 1)}{2(1 - \|Y - S\|\,\|S\|)}\|\Delta Y\|.$$

That is,

$$\|L(\Delta Y, Y)\| \leqq \left[\frac{\|S\|^2+1}{2(1-\|Y-S\|\,\|S\|)}\right]\|\Delta Y\|,$$

which proves (5.1). Letting $Y \to S$ gives $\|L(\cdot, S)\| \leqq (\|S\|^2+1)/2$. Let $\sigma = \|S\|$ and let $Sv = \sigma u$, $S^*u = \sigma v$ where u and v are the corresponding left and right singular vectors, respectively. For $\Delta Y = vu^*$,

$$2\Delta S + S(Y-S)\Delta S + \Delta S S(Y-S) = vu^* - \sigma^2 uv^*,$$

$$2(\|\Delta S\| + \|Y-S\|\,\|S\|\,\|\Delta S\|) \geqq (\sigma^2-1)\|\Delta Y\| = (\|S\|^2-1)\|\Delta Y\|.$$

That is, $\|\Delta S\| \geqq (\|S\|^2-1)\|\Delta Y\|/2(1+\|Y-S\|\,\|S\|)$, which gives (5.1) and (5.2) in the limit as $Y \to S$. □

As an example, suppose that $X = \operatorname{diag}(1/\alpha, -1/\alpha, 1)$ for some scalar $\alpha > 1$. Then $\operatorname{sgn}(X) = \operatorname{diag}(1, -1, 1)$. Since X is normal we can use Theorem 3.2 to get $\|L(\cdot, X)\| = \alpha$. However, this problem is asymptotically well-conditioned. By Theorem 5.1, $0 \leqq \|L(\cdot, S)\| \leqq 1$, and in fact since S is normal, $\|L(\cdot, S)\| = 1$ by Theorem 3.2.

From Theorem 3.3 and the above, Newton's method at the solution propagates errors according to the linear map (cf. [11])

$$\Delta X_{n+1} = \frac{\Delta X_n - S\Delta X_n S}{2}.$$

The eigenvalues of this mapping are either zero or one but this is not sufficient to guarantee that $\|\Delta X_n\|$ remains bounded. For example, the matrix $M = \left[\begin{smallmatrix} 1 & u \\ 0 & 1 \end{smallmatrix}\right]$ has eigenvalues equal to one but $M^k = \left[\begin{smallmatrix} 1 & uk \\ 0 & 1 \end{smallmatrix}\right]$, so for almost all initial vectors x_0, the sequence defined by $x_{n+1} = Mx_n$ diverges.

Instead we rely on the rather remarkable property that $\Delta X_1 = \Delta X_2 = \cdots$. To see this write

$$\Delta X_1 = \frac{\Delta X_0 - S\Delta X_0 S}{2},$$

$$\begin{aligned}\Delta X_2 &= \frac{\Delta X_1 - S\Delta X_1 S}{2} \\ &= \frac{\Delta X_0 - S\Delta X_0 S}{4} - \frac{S(\Delta X_0 - S\Delta X_0 S)S}{4} \\ &= \frac{\Delta X_0 - S\Delta X_0 S}{2} = \Delta X_1.\end{aligned}$$

A similar type of analysis can be applied to the polar decomposition, and shows that this problem is always asymptotically well-conditioned.

THEOREM 5.2. *Let Y be a nonsingular matrix with polar decomposition (U, H), and let $L(\cdot, Y)$ denote the Fréchet derivative of the mapping $Y \to U$. If $\|Y-U\| < 1$, then*

$$\|L(\cdot, Y)\| \leqq \frac{1}{1-\|Y-U\|}, \tag{5.4}$$

$$\|L(\cdot, U)\| = 1. \tag{5.5}$$

Proof. By Theorem 2.2, $\|L(\cdot, Y)\| = 1/\sigma_m$ where $\sigma_m = \sigma_{\min}(Y)$. But

$$Y = U + (Y-U)$$

so

$$\frac{1}{\sigma_m} = \|Y^{-1}\| = \|U^*(I+(YU^*-I))^{-1}\|$$

$$= \|(I+(YU^*-I))^{-1}\|$$

$$\leqq \frac{1}{1-\|YU^*-I\|} = \frac{1}{1-\|Y-U\|}.$$

This proves (5.4). As $Y \to U$, $\sigma_{\min}(Y) \to 1$ so $\|L(\cdot, Y)\| \to 1$. □

6. Refinement of approximate square roots. Each element of the sequence $\{X_n\}$ defined by (3.5) will always commute with X in theory, but due to finite precision arithmetic the computed matrices $(X_n)_c$ may not satisfy $X(X_n)_c - (X_n)_c X = 0$. In this case the limiting value of the sequence will not be $\operatorname{sgn}(X)$, and we are faced with the problem of refining an approximate solution $Y = S + \Delta S$ where Y does not commute with X and $\operatorname{sgn}(Y) \neq \operatorname{sgn}(X) \equiv S$. Since (3.5) will not restore commutativity with X once it is lost, we must seek a different approach if we wish to improve the approximate solution Y. First consider the more general square root problem [11]

$$S^2 = B, \tag{6.1}$$

and the associated Newton method

$$X_{n+1} = \tfrac{1}{2}(X_n + X_n^{-1}B). \tag{6.2}$$

The next lemma shows the effect that commutativity error plays in the convergence of $\{X_n\}$ to S.

LEMMA 6.1. *Let $E_n = X_n - S$, and $C_n = X_n X - X X_n \equiv [X_n, X]$. Then*

$$E_{n+1} = \tfrac{1}{2}X_n^{-1}E_n^2 + \tfrac{1}{2}X_n^{-1}[S, X_n], \tag{6.3}$$

$$C_{n+1} = \tfrac{1}{2}(C_n - X_n^{-1}C_n X_n^{-1}).$$

Proof. By (6.2)

$$E_{n+1} = \frac{1}{2}(X_n + X_n^{-1}S^2) - S$$

$$= \frac{X_n^{-1}}{2}(X_n^2 - 2X_nS + S^2)$$

$$= \frac{X_n^{-1}}{2}(X_n^2 - X_nS - SX_n + S^2) + \frac{1}{2}X_n^{-1}(SX_n - X_nS)$$

$$= \frac{X_n^{-1}}{2}E_n^2 + \frac{X_n^{-1}}{2}[S, X_n].$$

The commutator relation is proved in a similar manner. □

In order to suppress the buildup of commutator error it is much better to use the closely related method:

$$X_nX_{n+1} + X_{n+1}X_n = X_n^2 + B. \tag{6.4}$$

Because this method is obtained by applying quasi linearization to the relation $S^2 = B$ it can be regarded as the "true" version of Newton's method for (6.1), and is equivalent to (6.2) if X_{n+1} and X_n commute.

LEMMA 6.2. *Let* $\{X_n\}$ *be given by* (6.4). *Then* $X_nE_{n+1}+E_{n+1}X_n=E_n^2$, *and*

$$X_nC_{n+1}+C_{n+1}X_n=(X_n-X_{n+1})C_n+C_n(X_n-X_{n+1}).$$

Proof. From (6.4)

$$\begin{aligned}X_nE_{n+1}+E_{n+1}X_n &= X_n(X_{n+1}-S)+(X_{n+1}-S)X_n\\ &= S^2+X_n^2-X_nS-SX_n\\ &= (X_n-S)^2\\ &= E_n^2.\end{aligned}$$

To prove the commutator relation, expand terms and use (6.4). □

If the eigenvalues of X_n all lie in the right-half plane (or all lie in the left-half plane) then (6.4) is uniquely solvable for X_{n+1} and the commutator error goes to zero by Lemma 6.2 as $X_n\to S$. However, for the sign function ($S^2=I$), this eigenvalue condition is generally not satisfied. Moreover, since (6.4) requires the solution of a Sylvester equation for each n, it is generally too expensive to use.

If loss of commutivity becomes a problem, it is natural to seek a remedy in terms of the original matrices $X=X_0$ and B. The following theorem shows that this can be done by using the methods in the proof of Theorem 3.1.

THEOREM 6.3. *Let* $Y=S+\Delta S$, *where* $S^2=B$ *and* S *commutes with* X. *Then*

$$YX\Delta S+\Delta SYX=YXY-BX+\Delta S^2X. \tag{6.5}$$

Proof. Using $Y=S+\Delta S$, we have

$$Y^2=S^2+\Delta SS+S\Delta S+\Delta S^2=S^2+\Delta SY+S\Delta S. \tag{6.6}$$

Also, since $XS=SX$,

$$YX-XY=(S+\Delta S)X-X(S+\Delta S)=\Delta SX-X\Delta S. \tag{6.7}$$

Multiply (6.7) on the left by Y and use (6.6):

$$\begin{aligned}Y\Delta SX-YX\Delta S &= Y^2X-YXY\\ &= (S^2+\Delta SY+S\Delta S)X-YXY\\ &= S^2X+\Delta SYX+S\Delta SX-YXY.\end{aligned}$$

Rearranging terms gives (with $B=S^2$)

$$YX\Delta S+\Delta SYX=Y\Delta SX-S\Delta SX+YXY-BX=\Delta S^2X+YXY-BX. \qquad \square$$

As an example, if we let $X=T\begin{bmatrix}1 & \mu\\ & 1\end{bmatrix}T$, where $T=(1/\sqrt{2})\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$, then

$$S=T\begin{bmatrix}0 & \sqrt{\mu}\\ 1/\sqrt{\mu} & 0\end{bmatrix}T=\operatorname{sgn}(X).$$

For $\mu=2\times10^8$, after 20 steps of (3.5) the relative error in X_n is about 1.9×10^{-8}. If we neglect the term ΔS^2X in (6.5) and solve for ΔS in

$$YX\Delta S+\Delta SYX=YXY-X$$

as in (6.5) with $Y=X_n$, $B=I$, we find that $\hat{S}\equiv X_n-\Delta S$ has a relative error of about 4.1×10^{-13}.

Remark. Because (6.5) has the same form as (3.6), we can make a condition estimate at the same time that refinement is done, at a considerable savings in cost.

7. Conclusion. Reliable condition estimates for the polar decomposition and matrix sign functions are numerically feasible, although there is (currently) a wide difference in the cost of such estimates. For the polar decomposition, exact characterization of both the complex and real condition numbers (Theorems 2.2 and 2.3) in terms of smallest and second smallest singular values of A leads to a very cheap estimation scheme. Simple examples, for both the real and complex case, show that these condition numbers accurately measure the expected error in the computed solution. For the matrix sign function, good condition estimates can also be obtained, and for normal matrices the conditioning is inversely proportional to the smallest distance between eigenvalues across the imaginary axis. In the nonnormal case, no computationally simple characterization of the conditioning of the matrix sign is known and accurate condition estimators, based on the power method, use the equivalent of two extra sign function evaluations. What is really needed is an accurate condition estimate for the sign function that costs only a fraction of the effort needed to evaluate sgn (X).

Acknowledgment. We would like to thank Jianqin Mao of the Department of Automatic Control, Beijing Institute of Aeronautics and Astronautics, Beijing, China for fruitful discussions on the conditioning of polar decompositions.

Note added in proof. We have recently become aware of some interesting extensions of [9] for the case of polar factors. See [6a].

REFERENCES

[1] L. A. BALZER, *Accelerated convergence of the matrix sign function method of solving Lyapunov, Riccati and other matrix equations*, Internat. J. Control, 32 (1980), pp. 1057–1078.

[2] G. BIERMAN, *Computational aspects of the matrix sign function solution to the* ARE, in Proc. 23rd Conference on Decision and Control, Las Vegas, NV, 1984, pp. 514–519.

[3] Å. BJÖRCK AND C. BOWIE, *An iterative algorithm for computing the best estimate of an orthogonal matrix*, SIAM J. Numer. Anal., 8 (1971), pp. 358–364.

[4] Å. BJÖRCK AND S. HAMMARLING, *A Schur method for the square root of a matrix*, Linear Algebra Appl., 52/53 (1983), pp. 127–140.

[5] R. BYERS, *A* LINPACK*-style condition estimator for the equation* $AX - XB^T = C$, IEEE Trans. Automat. Control, AC-29 (1984), pp. 926–927.

[6] ———, *Numerical stability and instability in matrix sign function based algorithms*, in Computational and Combinatorial Methods in Systems Theory, C. I. Byrnes and A. Lindquist, eds., North-Holland, New York, 1986, pp. 185–200.

[6a] C. CHEN AND J. SUN, *Perturbation bounds for the polar factors*, J. Comput. Math., 7 (1989), pp. 397–401.

[7] A. CLINE, C. B. MOLER, G. W. STEWART, AND J. H. WILKINSON, *An estimate for the condition number of a matrix*, SIAM J. Numer. Anal., 16 (1979), pp. 368–375.

[8] E. DENMAN AND A. BEAVERS, *The matrix sign function and computations in systems*, Appl. Math. Comput., 2 (1976), pp. 63–94.

[9] N. J. HIGHAM, *Computing the polar decomposition—with applications*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1160–1174.

[10] ———, FORTRAN *codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation*, ACM Trans. Math. Software, 14 (1988), pp. 381–396.

[11] ———, *Newton's method for the matrix square root*, Math. Comp., 46 (1986), pp. 537–549.

[12] G. HEWER AND C. KENNEY, *The sensitivity of the stable Lyapunov equation*, SIAM J. Control Optim., 26 (1988), pp. 321–344.

[13] C. KENNEY AND A. J. LAUB, *Condition estimates for matrix functions*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 191–209.

[14] Z. KOVARIK, *Some iterative methods for improving orthonormality*, SIAM J. Numer. Anal., 7 (1970), pp. 386–389.

[15] P. LANCASTER, *Explicit solutions of linear matrix equations*, SIAM Rev., 12 (1970), pp. 554–566.

[16] J. R. RICE, *A theory of condition*, SIAM J. Numer. Anal., 3 (1966), pp. 287–310.

[17] J. D. ROBERTS, *Linear model reduction and solution of the algebraic Riccati equation by use of the sign function*, Internat. J. Control, 32 (1980), pp. 677–687.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 505-520, May 1991
© 1991 Society for Industrial and Applied Mathematics
003

SOLUTION OF THE NONLINEAR RADIATIVE TRANSFER EQUATIONS BY A FULLY IMPLICIT MATRIX MONTE CARLO METHOD COUPLED WITH THE ROSSELAND DIFFUSION EQUATION VIA DOMAIN DECOMPOSITION*

T. N'KAOUA†

Abstract. In this article, a new fully implicit Monte Carlo method for solving the nonlinear radiative transfer equations is presented. This scheme is both temperature and frequency implicit, and the particle tracking is much faster than in the standard Fleck method. This tracking is used to build an energy balance matrix. This leads to an original matrix formulation of the energy balance equation, where the entries of the matrix are computed by a Monte Carlo method.

This matrix formulation of the energy balance equation allows this method to be easily coupled with the Rosseland diffusion equation, which is generally needed in optically thick media, via domain decomposition. This coupling scheme between the two methods is also fully implicit.

Numerical tests are shown to prove the efficiency of the matrix Monte Carlo method; the results are better than those obtained by Fleck's method in that there are smaller computation times and there is a better robustness of the time discretization.

Key words. radiative transfer, Monte Carlo method, domain decomposition

AMS(MOS) subject classifications. 85A25, 65C05, 65C20

Introduction. When matter is hot enough (as in a plasma), it spontaneously emits radiation. Then this radiation heats the media that it is propagated through, and can eventually be scattered by the electrons (Thomson scattering). We can consider "light" as particules (photons) which travel across matter and interact with it.

The radiative transfer equations consist of two equations:

—The transfer equation, which describes the propagation, emission, and absorption of light.

—The energy balance equation, which describes the effect of the energy exchanges on the temperature.

If the medium is transparent (transparent means that the photons have a great mean free path), the propagation has a transport behaviour. In that case, we must take into account all the characteristics of the phenomena (direction, frequency, etc.), and it is quite usual and natural to use a Monte Carlo method to solve the radiative transfer equations.

On the other hand, if the medium is very opaque (small mean free path), the propagation can be considered as a diffusion process which can be described more easily. This implies that the Monte Carlo method, which takes into account all the details of the propagation, becomes uselessly detailed, but also too expensive, inefficient, and then unusable. So, in such media, it is natural to use the Rosseland diffusion equation, which describes the radiation propagation through opaque media (this equation is proved to be the asymptotic limit of the radiative transfer equations as the opacity, inverse of the mean free path, goes to infinity; see [1], [4]).

Since we want to use a Monte Carlo method in transparent zones, one of the main problems is to use the "best" temperature in the transfer equation in order to represent

* Received by the editors May 24, 1989; accepted for publication (in revised form) December 29, 1989.

† Département de Mathématiques Appliquées, Commissariat à l'Energie Atomique Limeil-Valenton, 94195 Villeneuve Saint-Georges Cedex, France.

correctly the emission of light. This means that the temperature which is used in the emission of energy should be as close as possible to the temperature computed at the end of the timestep by the energy balance equation (we naturally use a time discretization). As this energy balance equation takes into account both emission and absorption (which naturally depend on each other), it is easy to understand the advantages and the difficulties of using a "good" temperature. If the scheme is not implicit (using the previous timestep temperature, for example), it gives very oscillatory and innacurate results, as the cells need to wait during an entire timestep to emit back the absorbed energy. This implies that nonimplicit schemes need very small timesteps (see [5]).

In this paper, we propose using a fully implicit scheme. By fully implicit, we mean that the emission temperature is exactly the timestep end one: emission and absorption are then rigorously linked. Moreover, the tracking of the particles that we get by applying a Monte Carlo method to our scheme, is very simplified in comparison with the classical Fleck scheme (which is not really fully implicit; see [2], [5], or §§ 2.1 and 3, below). Then, as we shall see, we get a matrix formulation of the energy balance equation, where the entries of the matrix are computed by Monte Carlo.

On the other hand, as the Rosseland diffusion equation is used in optically thick media, we propose an implicit coupling scheme of the two methods via domain decomposition. In fact, it seems very difficult to couple Fleck's method with the Rosseland diffusion equation, for the formulations of the two methods are very different. But here, with the new method, we can build a natural and implicit coupling scheme, as both methods are expressed with matrices.

After the presentation of the basic equations, we shall expose (on a model problem) the two possible time discretizations: Fleck's, based on the "extrapolation" of a good temperature, and our, which is the real implicit time discretization of the equations. In the following sections, we shall present how the Monte Carlo method can be applied to both discretizations, and give some numerical results.

1. Notation and equations.

1.1. Notation.

c: speed of light,
h: Planck constant,
a: radiation constant,
K: Boltzmann constant,
G: number of frequency groups,
ν: frequency,
x: position,
t: time,
Ω: direction of propagation,
$T(x, t)$: temperature,
$\phi(x, t)$: acT^4,
$\sigma_\nu(T, x)$: opacity,
$E(\phi)$: energy density,
$I(x, t, \Omega, \nu)$: radiative intensity,
V: volume,
Q: Thomson scattering operator,
Δt: timestep,
$B(\nu, T)$ is the Planck function:

$$B(\nu, T) = \frac{8h\pi\nu^3}{c^2}(e^{h\nu/KT} - 1)^{-1}.$$

We have

$$\int_0^{+\infty} B(\nu\, T)\, d\nu = acT^4.$$

We denote

$$b(\nu, T) = B(\nu, T)/acT^4$$

so that

$$\int_0^{+\infty} b(\nu, T)\, d\nu = 1.$$

A frequency discretization in G groups $[\nu_k, \nu_{k+1}]$ of $[0, \nu_{\max}]$ is given.

Let b_k be the reduced Planck function in group k:

$$b_k(T) = \frac{\int_{\nu_k}^{\nu_{k+1}} B(\nu, T)\, d\nu}{acT^4}.$$

σ_p is the total Planck opacity and σ_k is the Planck opacity in group k:

$$\sigma_p(T) = \frac{\int_0^{+\infty} \sigma_\nu(T) B(\nu, T)\, d\nu}{acT^4} = \sum_k \sigma_k b_k,$$

$$\sigma_k(T) = \frac{\int_{\nu_k}^{\nu_{k+1}} \sigma_\nu(T) b(\nu, T)\, d\nu}{b_k(T)}.$$

Let us now give some explanations about the underlying physics and the meaning of all these terms.

I and T are the basic unknown functions. $T(x, t)$ is the usual electronic temperature. $I(x, t, \nu, \Omega)$, the radiative intensity, is the amount of energy at position x, time t, in frequency ν, travelling in direction Ω, per unit of time, frequency, solid angle, and area orthogonal to Ω.

σ_ν is the opacity; it represents both the "resistance" of matter to the light, and the emission coefficient at frequency ν. σ_p is an analogue coefficient integrated over frequency.

Assuming local thermodynamical equilibrium, the total emission of matter at temperature T is (due to Stephan's law) $\sigma_p acT^4$ (per unit of time and volume). The Planck function gives the distribution of this energy among the frequencies.

E is the classical energy density of matter (energy per unit of volume).

The functions E, σ_ν, and σ_p are supposed to be given analytically or by reference tables (characteristics of matter).

We shall also take into account the scattering of the photons on the electrons (Thomson scattering). This scattering only affects the direction of the propagation. The Thomson scattering operator can be written as

$$Q(I) = k_s I(\Omega) - k_s \int_{s^2} I(\Omega') \frac{3}{16\pi} (1 + (\Omega.\Omega')^2)\, d\Omega',$$

where k_s is the Thomson scattering coefficient and S^2 is the unit sphere of R^3.

As the emission depends on $\phi = acT^4$, we shall take ϕ as a basic unknown function in place of T.

1.2. Equations. The radiative transfer equations consist of the transfer equation coupled with the energy balance equation. They can be written together with initial conditions:

$$
\begin{aligned}
&\frac{1}{c}\frac{\partial I}{\partial t}+\Omega\cdot\nabla I+\sigma_\nu I+Q(I)=\sigma_\nu b(\nu,\phi)\frac{\phi}{4\pi};\\
&I(x,0,\Omega,\nu)=I_0(x,\Omega,\nu),\\
&\frac{\partial E(\phi)}{\partial t}=\int_0^{+\infty}\int_{S^2}\sigma_\nu\left(I-b(\nu,\phi)\frac{\phi}{4\pi}\right)d\Omega\,d\nu,\\
&\phi(x,0)=\phi_0(x).
\end{aligned}
\tag{1}
$$

The term

$$\frac{1}{c}\frac{\partial I}{\partial t}+\Omega\cdot\nabla I$$

is the transport term and just means that light travels at speed c; $\sigma_\nu I$ is the absorbed energy; $Q(I)$ is Thomson scattering; $\sigma_\nu b(\nu)\phi$ is the emitted energy at frequency ν.

The energy balance equation means that the total variation of matter energy comes from the difference between emission and absorption of radiation. Emission makes the temperature decrease and absorption makes it increase. Naturally it is a continuous phenomenon. But, if after time discretization the emission temperature is not the right one, the continuous phenomena is transformed into a discrete one; it means that we force matter to wait during an entire timestep to emit back the absorbed energy. This explains why we want an implicit scheme.

The numerical difficulties in solving these equations come both from the non-linearity and the large number of independent variables (t, x, ν, Ω). It is classical to use a Monte Carlo method to solve these kinds of equations.

We shall not deal here with the difficult problem of frequency discretizations, and we shall use the Planck opacity in each frequency group. It means that we assume we can replace, without problems, the continuous variable ν by the discrete one k and write the multigroup equations in the following way:

$$
\begin{aligned}
&\frac{1}{c}\frac{\partial I_k}{\partial t}+\Omega\cdot\nabla I_k+\sigma_k I_k+Q(I_k)=\sigma_k b_k\frac{\phi}{4\pi},\\
&I_k(0)=I_{0,k},\\
&\frac{\partial E(\phi)}{\partial t}=\int_{S^2}\sum_k\sigma_k I_k\,d\Omega-\sigma_p\phi,\\
&\phi(0)=\phi_0
\end{aligned}
\tag{1'}
$$

where I_k is the integral of I over group k: $[\nu_k, \nu_{k+1}]$.

It is classical to use the previous timestep values of the opacities and the reduced Planck functions in the methods we are dealing with. So, we shall consider that they are constant on the timesteps $\sigma_k=\sigma_k(\phi_0)$ and $b_k=b_k(\phi_0)$.

In order to make this paper as clear as possible, we will now present the different ways of discretizing these equations on an easier-to-handle model problem.

2. Time discretizations on a model problem. We consider the following simpler *linear* system of equations

$$\frac{\partial I}{\partial t}+\Omega\cdot\nabla I+\sigma I=\sigma\phi,$$

$$I(0)=I_0,$$

$$\gamma\frac{\partial\phi}{\partial t}+\sigma\phi-\sigma\int_{S^2}I\,d\Omega=0,$$

$$\phi(0)=\phi_0.$$

These equations can be obtained from the initial system (1′) after several simplifying assumptions:

—One frequency group,

—$E(x, t)$ linear in ϕ (with $\gamma(x)>0$),

—No Thomson scattering,

—$\sigma=\sigma(x)$,

—"$c=1$" and "$4\pi=1$."

In this section, we are interested in the solution of these equations on the timestep $[0, \Delta t]$.

We can consider this system as an "ordinary" system of two equations with two unknowns ϕ and I. Then there exists two ways of solving it:

—Compute ϕ as a function of I and get an equation for I.

—Compute I as a function of ϕ and get an equation for ϕ.

2.1. Fleck's discretization. Fleck's discretization uses the first "idea," while ours uses the second one.

Fleck's strategy consists of computing ϕ as a function of I from the discretized energy balance equation

$$\gamma\frac{\phi-\phi_0}{\Delta t}+\sigma\phi-\sigma\int_0^{\Delta t}\int_{S^2}I\,d\Omega\,dt=0,$$

which gives

$$\phi=f\phi_0+(1-f)\frac{1}{\Delta t}\int_0^{\Delta t}\int_{S^2}I\,d\Omega\,dt$$

with

$$f=\frac{\gamma}{\gamma+\sigma\Delta t}.$$

But if we use this value of ϕ in the transfer equation, the integral term cannot be treated by a Monte Carlo method (which is the final goal of the time discretization). So Fleck's method uses, in fact, the following expression of ϕ, approximating the integral over t:

$$\phi^*=f\phi_0+(1-f)\int_{S^2}I\,d\Omega.$$

Now the integral term represents an isotropic scattering which can easily (but expensively) be handled by a Monte Carlo method. Using ϕ^* in place of ϕ in the transfer equation, we get the so-called Fleck transfer equation.

When solving this equation by a standard Monte Carlo method (by tracking particles), we compute the energy absorptions

$$\frac{1}{\Delta t}\int_0^{\Delta t}\int_{S^2} I\,d\Omega$$

in each cell, in order to finally solve the energy balance equation

$$\gamma\frac{\hat{\phi}-\phi_0}{\Delta t}+\sigma\hat{\phi}-\sigma\int_0^{\Delta t}\int_{S^2} I\,d\Omega\,dt=0$$

and get the timestep and temperature $\hat{\phi}$.

We can easily see that in general $\phi^*\neq\hat{\phi}$. This means that this scheme is not fully implicit.

2.2. The fully implicit time discretization. Our strategy consists of noting that we can explicitly write I as a function of ϕ, when I and ϕ satisfy

$$\frac{\partial I}{\partial t}+\Omega\cdot\nabla I+\sigma I=\sigma\phi,$$

$$I(0)=I_0.$$

This system is equivalent to

$$I=T_tI_0+\int_0^t T_{t-s}(\sigma\phi)\,ds,$$

where T_t is the semigroup generated by $-(\Omega\cdot\nabla+\sigma)$:

$$T_tv=e^{-\Sigma_\Omega(x,\,t)}v(x-\Omega t,\,\Omega)$$

and

$$\Sigma_\Omega(x,\,t)=\int_0^t\sigma(x-\Omega z)\,dz.$$

(For the sake of simplicity, we have written the semigroup in the case when the domain is the entire space R^N; the general case gives no difficulty.)

This expression of I is *exact* and we can use it in the energy balance equation

$$\gamma(x)\frac{\partial\phi}{\partial t}+\sigma\phi-\sigma\int_{S^2}\int_0^t T_{t-s}(\sigma\phi)\,ds\,d\Omega-\sigma\int_{S^2}T_tI_0\,d\Omega=0.$$

We get a rigorous equation for ϕ.

The key point of our method is that we are going to compute the space discretized version of the operator

$$\int_{S^2}\int_0^t T_{t-s}(\sigma\bullet)\,ds\,d\Omega$$

and not the value of this operator on a particular ϕ.

After space and time discretization, the discrete version of the operator written above is a *matrix M*, so that we can write the energy balance equation in the following convenient way:

$$\gamma\frac{\hat{\phi}-\phi_0}{\Delta t}+M\hat{\phi}=S,$$

which can be solved in a fully implicit way. Therefore, the temperature used in the transfer equation is exactly $\hat{\phi}$.

We can immediately note that this method can be extended to the nonlinear case ($E(\phi)$ being nonlinear in ϕ). The previous equation is still valid, replacing

$$\gamma \frac{\hat{\phi} - \phi_0}{\Delta t}$$

with

$$\frac{E(\hat{\phi}) - E(\phi_0)}{\Delta t}.$$

Taking into account Thomson scattering is easy in both time discretizations (when using a Monte Carlo method): we treat this scattering during the tracking of particles only as it affects the direction of propagation.

3. Monte Carlo for Fleck's discretization. On the complete equations, we need to first linearize the energy balance equation. The computations are quite straightforward and give

$$\phi^* = f\phi_0 + (1-f) \int_{S^2} \sum_k \frac{\sigma_k}{\sigma_p} I_k \, d\Omega,$$

where

$$f = \left(1 + \Delta t \sigma_p \frac{\partial E(\phi_0)^{-1}}{\partial \phi}\right)^{-1}.$$

Putting ϕ^* in place of ϕ in the transfer equation, we get

$$\frac{1}{c}\frac{\partial I_k}{\partial t} + \Omega \cdot \nabla I_k + (\sigma'_k + \sigma''_k) I_k + Q(I_k) = \sigma'_k b_k \frac{\phi_0}{4\pi} + \frac{\sigma_k}{\sigma_p} b_k \int_{S^2} \sum_m \sigma''_m I_m \, d\Omega$$

with $\sigma'_k = f\sigma_k$ and $\sigma''_k = (1-f)\sigma_k$.

The energy balance equation becomes

$$E(\hat{\phi}) - E(\phi_0) = \int_{S^2} \int_0^{\Delta t} \sum_k \sigma'_k I_k \, d\Omega \, dt - f\sigma_p \phi_0 \Delta t.$$

We can note that, as in the linear problem, the scheme is not implicit, and now, on the nonlinear model, the discretized equations are not rigorous because of the linearization.

A mesh is given. These equations are solved using a standard Monte Carlo method (assuming that the temperature is constant on each cell). It means that we simulate the transfer equation with particles in order to compute the energy absorptions.

Particles (photons) are emitted (uniformly in space and time in their birth cell). They are characterized by their position, direction, energy, and frequency group. They represent an energy $\Delta t V f \sigma_p \phi_0$, and they are spectrally distributed according to the local Planck function.

During tracking, the particles may undergo different events:

—Thomson collisions (scattering without loss of weight).

—"Fleck's collisions" (absorption and instantaneous isotropic Planck emission) due to the integral term in transfer Fleck's equation.

—Boundary crossings.

The weight of the particles decreases exponentially with respect to time (to take into account the absorption term) as in an ordinary particle method. The corresponding

energy is transferred to the cells. So, after tracking the particles, we get the energy emissions and absorptions in each cell. Then, the energy balance equation is solved by a standard Newton method, separately in each cell.

Fleck's method is better than the "natural" (explicit) Monte Carlo method which uses the previous timestep temperature in the transfer equation.

However, Fleck's method is not completely implicit and still needs quite small timesteps (see [5]). It remains expensive: for each Fleck's collision, we must compute the intersections of the particles' trajectories with the cell's boundaries. These computations are the most expensive ones in the Monte Carlo method.

These remarks justify the development of a better method which must be both fully implicit and less expensive.

4. The MIMC (matrix implicit Monte Carlo) method.

4.1. Space discretization. From now on, ϕ denotes the value at the end of the timestep ($\hat{\phi}$ in the previous sections). As for the model problem, we consider the following system, which is the implicit time discretization of (1'):

$$
\begin{aligned}
&\frac{1}{c}\frac{\partial I_k}{\partial t}+\Omega\cdot\nabla I_k+\sigma_k I_k+Q(I_k)=\sigma_k b_k\frac{\phi}{4\pi},\\
&I_k(0)=I_{0,k},\\
&E(\phi)-E(\phi_0)=\int_{S^2}\int_0^{\Delta t}\sum_k \sigma_k I_k\,dt\,d\Omega-\sigma_p\phi\Delta t,\\
&\phi(0)=\phi_0.
\end{aligned}
\tag{2}
$$

This time discretization can be shown to be of order one (see [6], [7]).

We split the transfer equation into two equations:

—One describing the effect of the initial condition ($\int_{S^2} T_t I_0\,d\Omega$ in the model problem).

—The other one describing implicitly the emission of the cells ($\int_{S^2}\int_0^t T_{t-s}(\sigma\bullet)\,ds\,d\Omega$ in the model problem).

So we define the functions u_k and $v_{i,k}$ solutions of the equations:

$$
\begin{aligned}
&\frac{1}{c}\frac{\partial u_k}{\partial t}+\Omega\cdot\nabla u_k+\sigma_k u_k+Q(u_k)=0,\\
&u_k(0)=u_{0,k}=I_{0,k},
\end{aligned}
\tag{3}
$$

$$
\begin{aligned}
&\frac{1}{c}\frac{\partial v_{i,k}}{\partial t}+\Omega\cdot\nabla v_{i,k}+\sigma_k v_{i,k}+Q(v_{i,k})=\sigma_k b_k\frac{\chi_i}{4\pi},\\
&v_{i,k}(0)=0,
\end{aligned}
\tag{4}
$$

where χ_i is the characteristic function of the cell i. We assume that ϕ is constant on each cell. $v_{i,k}$ is the radiative intensity in group k, assuming that only cell i emits radiation with a ϕ equal to one.

Instead of (2), we solve (3), (4) coupled with the energy balance equation.

So, since $\phi=\sum_i \chi_i\phi_i$ (where ϕ_i is the value of ϕ in cell i), we get

$$
I_k=u_k+\sum_i v_{i,k}\phi_i
$$

as the transfer equation in (2) is linear in ϕ.

Using this formula in the energy balance equation and integrating the expression over the cell i, we get a *ϕ-linear* expression in the right-hand side of the energy balance equation

$$V_i(E(\phi_i)-E(\phi_{0,i}))=\int_i\int_{S^2}\int_0^{\Delta t}\sum_k \sigma_k u_k\,dt\,d\Omega\,dx+\sum_{j,k}\left(\int_i\int_{S^2}\int_0^{\Delta t}\sigma_k v_{j,k}\,dt\,d\Omega\,dx\right)\phi_j$$
$$-\sigma_{p,i}\Delta t V_i\phi_i,$$

which can be written as

$$V_i(E(\phi_i)-E(\phi_{0,i}))=S_i+\sum_j M_{i,j}\phi_j,$$

with

$$S_i=\int_i\int_{S^2}\int_0^{\Delta t}\sum_k \sigma_k u_k\,dt\,d\Omega\,dx$$

and

$$M_{i,j}=\sum_k\int_i\int_{S^2}\int_0^{\Delta t}\sigma_k v_{j,k}\,dt\,d\Omega\,dx-\delta_{i,j}\sigma_{p,i}\Delta t V_i,$$

where

$$\delta_{i,j}=1 \quad \text{if } i=j, \qquad 0 \quad \text{if } i\neq j.$$

The integrals in S_i and $M_{i,j}$ are computed by a standard Monte Carlo method (in the same way as in Fleck's method. But here "Fleck's collisions" no longer exist, as the integral term due to Fleck's discretization is no longer present in the transfer equation). These terms are the discrete versions of the operators that we mentioned in the model problem.

Instead of approximating ϕ from the energy balance equation, we compute the energy transfers as implicit functions of ϕ.

We solve (3), (4) by a Monte Carlo method:

(3) by tracking the particles coming from the previous timesteps or from the boundary conditions,

(4) by tracking "implicit" photons (without "real" weights) and computing implicit energy transfers.

These computations allow us to construct the energy transfer matrix M, which will then be used in the energy balance equation.

Besides its implicitness, this method should be cheaper than Fleck's method, as the only events in the tracking are Thomson collisions and boundary crossings.

4.2. Details of computations. Let us recall that a frequency discretization and a mesh are given.

The subscripts i and j will always denote cells, while k will denote frequency groups.

On $[0, \Delta t]$, the energy emitted by cell j in group k is

$$\Delta t V_j \sigma_{j,k} b_k \phi_j.$$

When particles emitted in cell j, in group k, cross the cell i, they give to i the following energy:

$$A_{i,j}^k(1-e^{-\sigma_{i,k}l_i})\Delta t V_j\sigma_{j,k}b_k\phi_j,$$

where l_i is the distance travelled in cell i and $A^k_{i,j}$ is the attenuation due to the crossing of the other cells before arriving in cell i.

So the energy absorbed in cell i coming from all the frequency groups and from all the cells can be written as

$$\sum_j \Delta t V_j \sum_k A^k_{i,j}(1-e^{-\sigma_{i,k}l_i})\sigma_{j,k}b_k\phi_j.$$

We set

$$D_{i,j} = \Delta t V_j \sum_k A^k_{i,j}(1-e^{-\sigma_{i,k}l_i})\sigma_{j,k}b_k.$$

So this absorbed energy is

$$\sum_j D_{i,j}\phi_j.$$

The terms $D_{i,j}$ are easily computed during the tracking. In fact, they are also computed in Fleck's method. But in Fleck's method, we effectively compute $D_{i,j}\phi_j$ and the implicitness is lost.

The global emission of cell i is

$$\Delta t V_i \sigma_i \phi_i.$$

Then we define the energy balance matrix M in the following way:

$$M_{i,j} = D_{i,j} - \delta_{i,j} V_i \Delta t \sigma_i.$$

The energy balance equation can then be written as

$$E(\phi) - E(\phi_0) = M\phi + S,$$

where S represents the energy absorptions of the previous timesteps particles (and due to sources or boundary conditions, if necessary).

So we now have an implicit matrix formulation of the energy balance equation. The energy at the end of the timestep is evaluated according to the temperature at the end of the timestep.

It seems to be quite new and interesting to use a Monte Carlo method to build a matrix. Here Monte Carlo is used to compute the integral of an operator; usually, it is used to compute ordinary numerical integrals.

Let us now give some practical details for the implementation of the Monte Carlo method. We emit N photons in each cell, uniformly on the timestep, representing an energy "1" (unity). The implicitness of the scheme can be seen in this "1" instead of a preset value of ϕ.

From the frequency point of view, every particle will contribute to the terms $A^k_{i,j}$ for *all the frequency groups k*. It means that the particles used to construct the matrix M carry all the frequency groups. This is possible, as Thomson scattering is frequency independent. It is not really more expensive (than attributing only one frequency group to each particle), and as we shall see further, it must give better results. So we attribute to every particle G weights (one in each frequency group). This is not possible in Fleck's method, as Fleck's collisions depend upon frequency.

4.3. Description of the algorithm. Let e_n be the initial total weight of particle n ($e_n = 1/N$ if the generation is uniform in space).

We initialize an array $A(n, k)$ (implicit weights for each frequency group k and each particle n) as

$$A(n, k) = \Delta t V_{i(n)} \sigma_{i(n),k} b_k,$$

where $i(n)$ is the birth cell of the particle n. This array will allow us to build the matrix M and to compute the successive attenuations during the tracking.

We initialize M as

$$M_{i,j} = -\delta_{i,j}\Delta t V_i \sigma_i.$$

When a particle, born in cell j, crosses cell i along a distance l_i, we compute an attenuation and an energy absorption for all the frequency groups:

$$M_{i,j} + e_n \sum_k A(n,k)(1 - e^{-\sigma_{i,k} l_i}) \to M_{i,j} \quad \text{(absorption)},$$

$$A(n,k)\, e^{-\sigma_{i,k} l_i} \to A(n,k) \quad \text{(attenuation)}.$$

So at the end of the tracking, we get the matrix M and all the implicit attenuations of the particles. After the energy balance equation solution (see below), we get ϕ in all the cells.

We have for each photon a weight in every frequency group:

$$e_{n,k} = e_n A(n,k)\phi_{i(n)}.$$

We then need to represent this multiweights particle by only one photon, in order to represent the initial radiative intensity of the next timestep.

To conserve the global energy, the final photon must have the weight

$$\sum_k e_{n,k}$$

and its frequency group is sampled according to $A(n,k)$.

We can now see the advantage of computing weights for all the frequency groups. The frequency group here is chosen as a function of the *final* spectral distribution. In Fleck's method, as we cannot know what this final radiative intensity will be, the frequency group is chosen at the beginning of the timestep, in function of the birth spectrum. That is why we said that the scheme is also frequency implicit.

In order to get a really efficient practical method, we now have to study the solution of the energy balance equation.

4.4. Energy balance equation solution. This equation is

$$E(\phi) - E(\phi_0) = M\phi + S.$$

To deal with the nonlinearity of E, we use a Newton's method:

$$\left(\frac{\partial E}{\partial \phi}(\phi^q) - M\right)\Delta\phi^{q+1} = M\phi^q + S + E(\phi^0) - E(\phi^q),$$

where q is the Newton iterations number.

We have to solve this linear system for every Newton iteration.

THEOREM 1. *M is strictly diagonally dominant.*

Proof. $|M_{j,j}|$ is the energy born in j, leaving j and $M_{j,j} < 0$.

On column j, for $i \neq j$, the entries $M_{i,j}$ are the energy absorptions in the cells i, coming from cell j. The sum of these terms is necessarily less than the energy leaving cell j, so

$$\sum_{i \neq j} M_{i,j} < |M_{j,j}|.$$

COROLLARY. *The Jacobi method for*

$$\left(\frac{\partial E}{\partial \phi}(\phi^q) - M\right)\Delta\phi^{q+1} = M\phi^q + S + E(\phi^0) - E(\phi^q)$$

is convergent. It means that, D being the diagonal of M, to solve

$$(E'(\phi^q) - M)X = B$$

we use the Jacobi iterative method

$$\left(\frac{\partial E}{\partial \phi}(\phi^q) - D\right)X^{n+1} = (M - D)X^n + B.$$

5. A coupling scheme with the Rosseland diffusion equation. The Monte Carlo method cannot be used in optically thick media (see [6]), so in such media we use the Rosseland diffusion equation. The direction and frequency variables no longer appear in this equation: the propagation in opaque media is modelled as a diffusion process. The whole phenomenon can be described by only one variable: the temperature (or ϕ in order to use the same variable as in Monte Carlo). This equation can be obtained by several methods: asymptotic developments of the radiative transfer equations as the opacity goes to infinity (see [1], [4]), or assuming several physical properties (thermodynamical equilibrium and radiative intensity linear in Ω).

The Rosseland diffusion equation is

$$\frac{\partial}{\partial t}\left(E(\phi) + \frac{\phi}{c}\right) - \nabla \cdot \left(\frac{1}{3\sigma_R}\nabla\phi\right) = 0$$

with some initial and boundary conditions, and where σ_R is the Rosseland opacity

$$\sigma_R(T) = \left(\int_0^{+\infty} \frac{\partial B(\nu, T)}{\partial T}\, d\nu\right)\left(\int_0^{+\infty} \frac{1}{\sigma_\nu(T)} \frac{\partial B(\nu, T)}{\partial T}\, d\nu\right)^{-1}.$$

We solve this equation using a standard implicit finite difference method. So that, after time and space discretization, the equation can be written in the following *pleasant* way:

$$E(\phi) - E(\phi_0) = M\phi + S.$$

We divide the domain into two zones: the first one treated with the Monte Carlo method, the second one (supposed to be opaque) with the Rosseland diffusion equation.

Let d be a diffusion cell with one face coupled with the Monte Carlo zone. The classical Marshak relation links the energy E^+ leaving the Monte Carlo zone, to the energy E^- leaving the diffusion zone

$$E^- = (1 - \xi)E^+ + \Delta t s_d \xi \frac{\phi_d}{4}$$

where

$$\xi = \frac{4}{3\sigma_R L + 2}.$$

L is the half depth of d, s_d is the area of the face coupled with the Monte Carlo zone, and σ_R is the Rosseland opacity of cell d.

This relation can easily be obtained after some straightforward calculations, assuming that the radiative intensity is linear in Ω at the interface.

We denote by j the Monte Carlo cells which "lighten" the cell d during the timestep. With the same notation as before,

$$E^+ = \sum_j \Delta t V_j \sum_k A^k_{d,j}\sigma_{j,k}\phi_j,$$

which means that all the energy arriving in d is absorbed (which is true, as d is supposed to be opaque).

We can now write the energy balance at the interface:

$$E = E^+ - E^- = \xi \sum_j \Delta t V_j \sum_k A^k_{d,j} \sigma_{j,k} \phi_j - \Delta t s_d \xi \frac{\phi_d}{4}.$$

This relation will be used as a "boundary condition" in the diffusion zone. In fact, this formula is an implicit evaluation of the flux at the interface between the Monte Carlo zone and the diffusion one. It is a very interesting relation, as it links the diffusion cell temperature to the temperatures of *all* the Monte Carlo cells which participate to the boundary flux. We do not evaluate this flux by only "counting" the energy that crosses the interface as in a standard Monte Carlo method. Here we get a *really implicit* relation between the temperatures.

We also need to represent the diffusion zone emission:

$$E^- = (1-\xi)E^+ + \Delta t s_d \xi \frac{\phi_d}{4}.$$

The term $(1-\xi)E^+$ is classically a "return" or feedback operator with coefficient $(1-\xi)$: we return the particles arriving in d with an energy multiplied by this coefficient. The spectral treatment is simple: the implicit particles have their weights distributed according to the local spectrum; the explicit ones have their unique frequency group sampled, also according to the local spectrum.

To simulate the term $\Delta t s_d \xi \phi_d / 4$, we emit implicit photons, as this emission is also linear in ϕ_d. We track these particles, compute the attenuations, and build the matrix as before.

In the diffusion zone, we build the standard diffusion matrix.

We get one global matrix to solve the energy balance equation, both in the Monte Carlo zone and in the diffusion zone. This matrix has the following structure:

$$\begin{pmatrix} D & C1 \\ C2 & M \end{pmatrix}.$$

D is the diffusion matrix; M is the Monte Carlo matrix; $C1$ and $C2$ are the coupling matrices arising from the Marshak relation.

This coupling scheme is completely implicit, as it couples effectively the timestep end temperatures of the diffusion cells to the timestep end temperatures of the "lightening" and "lightened" Monte Carlo cells.

We can see that the computation of that global matrix is possible, thanks to our time discretization of the full radiative transfer equations. It is very interesting to get such a simple coupling scheme for such different methods (statistical and deterministic).

Remark. Why is the Monte Carlo method not suited to opaque media?

We said that the Monte Carlo method could not give good results in optically thick media (see [6]). This is a well-known result. To get a "good" matrix M, we need a correct evaluation of the energy transfers. But if the cells are very opaque (mean free path very small with respect to the dimension of the cells), the particles will all have a zero weight when they get out of their birth cell, and the evaluation of the flux will be wrong, unless we generate a huge number of particles.

In fact, it is clear that the entries of the matrix M go to zero when the opacity increases, using a constant number of particles.

Another way to see this is to consider the asymptotic analysis of the radiative transfer equations as the opacity goes to infinity: *the first-order term* goes to zero, and the *second-order* term alone gives the Rosseland diffusion equation (see [1], [4]). So it is natural that the matrix M becomes singular when we pass from the transport behaviour to the diffusion behaviour.

For this reason, we developed a coupling scheme between the Monte Carlo method and the Rosseland diffusion equation.

6. Numerical experiment. We present here a numerical test of the Monte Carlo method (without coupling with diffusion). This test was inspired by Fleck and Cummings's test [2]. It is a plane geometry case. The domain consists of three successive media:

Medium 1: 2.0 cm long, 5 cells, opacity denoted Σ (Kramers model without Thomson scattering).

Medium 2: 0.4 cm long, 8 cells, opacity $1000 \times \Sigma$.

Medium 3: 1.6 cm long, 4 cells, opacity Σ.

The initial temperature is 0.05 kev (1 kev = 11.6 million Kelvin degrees). The boundary condition on medium 1 is a given surface temperature of 1 kev. The boundary condition on medium 3 is a zero incoming flux.

We denote the Fleck method by FMC and the matrix implicit Monte Carlo method by MIMC.

We used a CRAY/XMP/416 to achieve the computations.

Figure 1 shows "temperatures versus time" in the first cell of medium 2 and in the last cell of medium 3. The timestep is 2.10^{-11} seconds.

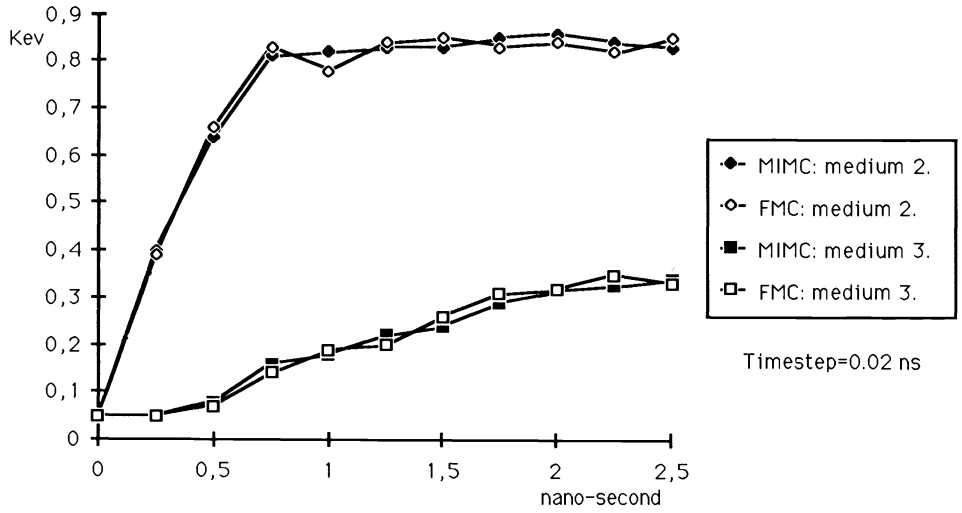


FIG. 1. *Temperature versus time. Media* 2 *and* 3. *Timestep* = 0.02 ns.

The ratio of the computational times between FMC and MIMC is 80. If we accelerate FMC by the random walk algorithm described in [3], this ratio decreases to 5, still favorable to MIMC.

Moreover, Figs. 2 and 3 show the great robustness of MIMC with respect to the size of the timestep ("Reference" designs the results of the computations with the small timestep 2.10^{-11} seconds). Using a timestep of 5.10^{-10} seconds, MIMC still gives correct results, which is no longer the case of FMC.

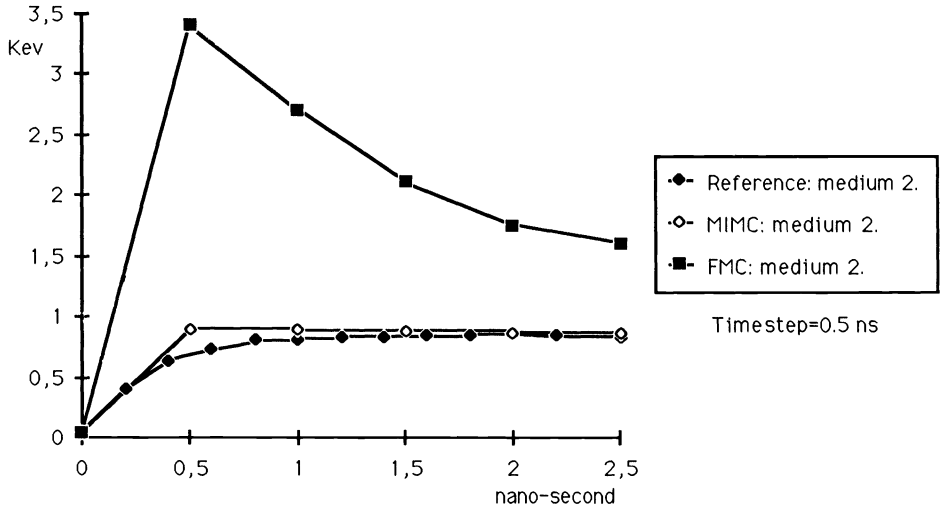


FIG. 2. *Temperature versus time. Medium* 2. *Timestep* = 0.5 ns.

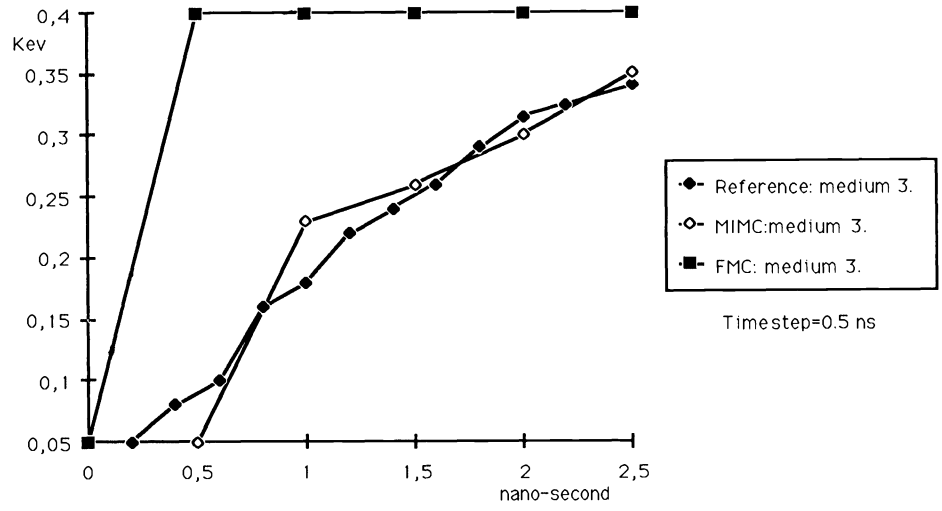


FIG. 3. *Temperature versus time. Medium* 3. *Timestep* = 0.5 ns.

We did not encounter any difficulties concerning the Jacobi method for solving the linear systems. The algorithm always converged in very few iterations, even in more complicated computations with several hundreds of cells. It seems to be quite interesting to note that, even with a poor solving algorithm (Jacobi), the method gives good results with important time savings.

7. Conclusion. The MIMC method seems to be very interesting for the following reasons:

—The time discretization is really implicit.

—In the Monte Carlo algorithm, the tracking is simpler and cheaper than the classical one.

—The computation of the Monte Carlo matrix M is easy.

—The numerical example proves its feasibility and its superiority over Fleck's method by the time savings and the robustness to larger timesteps.

—The coupling scheme with the Rosseland diffusion equation (which remains to be tested) is fully implicit and "natural."

REFERENCES

[1] C. BARDOS, F. GOLSE, B. PERTHAME, AND R. SENTIS, *The nonaccretive radiative transfer equations: Global existence and Rosseland approximation*, J. Funct. Anal., 77 (1988), pp. 434–460.

[2] J. A. FLECK AND J. D. CUMMINGS, *An implicit Monte Carlo scheme for calculating time and frequency dependent nonlinear radiation transport*, J. Comput. Phys., 8 (1971), pp. 313–342.

[3] J. GIORLA AND R. SENTIS, *A random walk method for solving radiative transfer equations*, J. Comput. Phys., 70 (1987), pp. 145–165.

[4] E. W. LARSEN, G. C. POMRANING, AND V. C. BADHAM, *Asymptotic analysis of radiative transfer problems*, J. Quant. Radiat. Transfer, 29 (1983), pp. 285–309.

[5] E. W. LARSEN AND B. MERCIER, *Analysis of a Monte Carlo method for the nonlinear radiative transfer equations*, J. Comput. Phys., 71 (1987), pp. 50–64.

[6] T. N'KAOUA AND R. SENTIS, *A new time discretization for the radiative transfer equations. Analysis and comparison with the standard discretization*, Note C.E.A. N-2617, Commissariat à l'Energie Atomique, Paris, France, 1989; SIAM J. Numer. Anal., submitted.

[7] T. N'KAOUA, *A new time discretization for the radiative transfer equations and development of an implicit matrix particle method*, C. R. Acad. Sci. Paris Sér. I. Math. Paris, 309 (1989), pp. 707–711.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 521-546, May 1991
© 1991 Society for Industrial and Applied Mathematics
004

A PRACTICAL APPROACH TO NONLINEAR FUZZY REGRESSION*

AIVARS CELMIŅŠ†

Abstract. This paper presents a new method of mathematical modeling in an uncertain environment. The uncertainties of data and model are treated using concepts of fuzzy set theory. The model fitting principle is the minimization of a least squares objective function. A practical modeling procedure is obtained by restricting the type of data and parameter fuzziness to conical membership functions. Under this restriction, the model fitting problem can be solved numerically with the aid of any least squares software for regression with implicit constraint equations. The paper contains a short discussion of the geometry of fuzzy point and function spaces with conical membership functions, and illustrates the application of fuzzy regression with an example from terminal ballistics.

Key words. fuzzy nonlinear models, fuzzy data, fuzzy regression, least squares, conical membership function

AMS(MOS) subject classifications. 62J02, 65C20, 62F99

1. Introduction.

We consider regression problems where the mathematical model is a set of equations between observables. The solution of such a problem strongly depends on the method by which the uncertainties of the data and/or model are treated. Among the different methods available for the treatment of uncertainty, the fuzzy regression presented in this paper offers some advantages. The advantages are, first, that the method allows us to formalize in a natural way linguistic data, including their intrinsic vagueness. Second, identical formalisms can be used for the treatment of linguistic and numerical data, so that a common data base can be established for both types of data. Finally, the analysis does not depend on the existence of a known or postulated error distribution in a hypothetical universe. Of course, probabilistic concepts, if appropriate, also can be included in the analysis.

Special types of fuzzy regression problems have been treated by several authors. Tanaka, Uejima, and Asai [12], [13], developed linear programming algorithms for regression with fuzzy linear models and special types of fuzzy data. The algorithm of [12] was applied by Heshmaty and Kandel [9] to economic forecasts. Diamond suggested in [6] and [7] a least squares approach and in [8] a maximum likelihood approach to fuzzy linear regression problems with special fuzzy data. Celmiņš has treated regression of nonlinear models with a slightly more general type of data in the special cases where either only the data [3] or only the model [4] is fuzzy. In the present paper, the results of [3] and [4] are extended to a general regression problem, where the model equations are nonlinear and the model as well as the data are fuzzy. Some restrictions are imposed on the generality in order to obtain a method which is algorithmically simple. The restrictions involve a parametrization of the membership functions which are assumed to be conical functions. Under this restriction, we can solve the model fitting problem with the aid of regression software for implicitly formulated constraints, supplementing such software only with normalization procedures for some of the input.

In § 2 we develop some simple concepts and properties of a special fuzzy point space. These results are used in § 3 to derive a formalism for a fuzzy regression. Section 4 presents an example of fuzzy regression in a terminal ballistics problem.

* Received by the editors November 17, 1986; accepted for publication (in revised form) February 12, 1990.

† U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, Maryland 21005-5066.

2. Fuzzy geometry. In this section, we provide basic definitions of fuzzy set theory and develop simple concepts of analytic geometry in a fuzzy environment. The purpose of the section is to establish practical tools for model fitting, and to provide a geometrical interpretation of the fitting and its results. Therefore, the exposition is restricted to a special type of fuzzy space for which we obtain algorithms that are easily applicable to real life problems.

2.1. Basic definitions. In presenting the basic definitions we follow, in essence, Zimmermann [16]. A classical set is normally defined as a collection of elements of $e \in E$. Each element can either belong or not belong to a set $H \subseteq E$. The set can be described by a characteristic function which equals unity if e is an element of H and is zero if e does not belong to H. We call such a set a *crisp* set.

DEFINITION. A *fuzzy set* $\tilde{H}$ in E is a set of ordered pairs

$$\tilde{H} = \{e, \mu_H(e) | e \in E\}. \tag{2.1}$$

$\mu_H(e)$ is called the membership function or grade of membership of e. The range of $\mu_H(e)$ is a subset of nonnegative numbers whose supremum is finite.

In this paper, we assume that the range of the membership function is the interval $[0, 1]$ and that elements with a zero degree of membership are not included in $\tilde{H}$. If the membership function assumes only the values zero and one, then the set $\tilde{H}$ is nonfuzzy (*crisp*) and μ_H is its characteristic function.

The concept of a fuzzy set allows us to express uncertainty in a manner which is different from probabilistic concepts and interval analysis. Thus, for instance, the concept "about 5" might be represented by a fuzzy number $\tilde{5}$ whose membership function $\mu_5(x)$ has a positive maximum value at $x = 5$ and decreases to zero as $|x-5|$ increases. Zimmermann [16] discusses many examples of such concepts and their representation as fuzzy sets.

2.2. Fuzzy point space. A fuzzy point in R_n is a fuzzy set of points $X \in R_n$. Let the membership value of any element of the set be given by a membership function $\mu(X)$. In this paper, we only consider a particular type of fuzzy point which is characterized by fuzzy position vectors with *conical membership functions.*

DEFINITION. A *regular* (*not regular*) *fuzzy vector* $\tilde{A}$ in R_n is the pair

$$\tilde{A} = \{A, P_A\}, \tag{2.2}$$

where $A \in R_n$, and P_A is a positive definite (semidefinite) $(n \times n)$-matrix. A is called the *apex* of $\tilde{A}$ and P_A is called the *panderance matrix* of $\tilde{A}$.

With this definition, a fuzzy vector in an n-dimensional space is specified by $n \cdot (n+3)/2$ real numbers. Let Q_A be the Moore–Penrose generalized inverse of P_A and let $X_A \subseteq R_n$ be the linear space spanned by the columns of P_A. We denote by $\|\cdot\|_A$ the following elliptic norm of the distance between an arbitrary $X \in R_n$ and A:

$$\|X - A\|_A = \begin{cases} [(X-A)^T \cdot Q_A \cdot (X-A)]^{1/2} & \text{if } X - A \in X_A, \\ +\infty & \text{otherwise.} \end{cases} \tag{2.3}$$

DEFINITION. The *conical membership function* of $\tilde{A}$ is

$$\mu_A(X) = 1 - \min\{1, \|X - A\|_A\}. \tag{2.4}$$

The *fuzzy point* $\tilde{A}$ is the set of all vectors X with the property $\mu_A(X) > 0$. If the panderance matrix P_A is positive definite, then the boundary of the support of μ_A and the level surfaces $\mu_A(X) = \text{constant}$ are hyperellipsoids in R_n and (2.4) defines a cone. The conical membership function μ_A is normalized to $0 \leqq \mu_A \leqq 1$, and a fuzzy point

with a conical membership function is a convex fuzzy set [14]. If the panderance matrix P_A is semidefinite, then the cone is degenerate in R_n and is a regular cone in the subspace X_A.

In R_1, the conical membership function is the triangular function

$$\mu_a(x) = 1 - \min\{1, |x-a|/s_a\}, \tag{2.5}$$

where $s_a = \sqrt{P_a}$ is the spread of the function. In R_n, the membership function of each component of $\tilde{A}$ is a triangular function with a spread that equals the square root of the corresponding diagonal element of P_A. The triangular function is, of course, the projection of the cone (2.4) on the corresponding axis. Figure 1 gives a schematic illustration of the conical membership function of a fuzzy point (vector) in R_2.

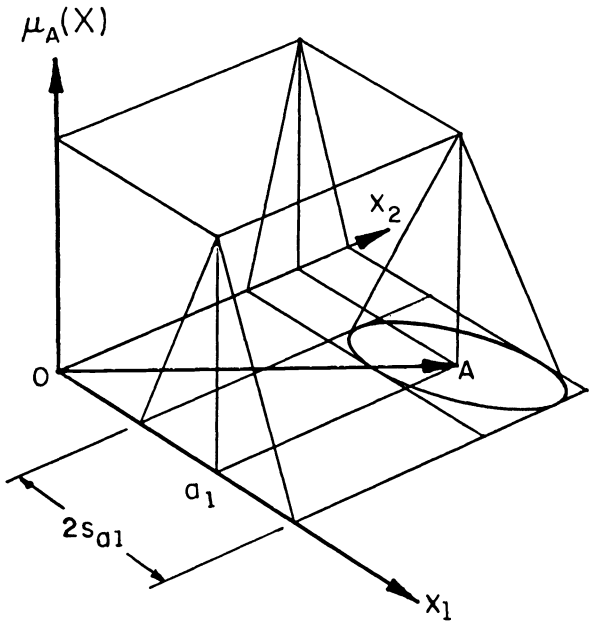


FIG. 1. *Conical membership function in* R_2.

The off-diagonal elements of P_A indicate possible interactions [15] between the components of $\tilde{A}$. These elements can be conveniently represented by dimensionless *concordances* which we define for a regular fuzzy point as follows:

$$c_{ki} = p_{ki}/\sqrt{p_{kk}p_{ii}}, \qquad k, i = 1, \cdots, n, \tag{2.6}$$

where c_{ki} and p_{ki} are elements of a concordance matrix and the panderance matrix P_A, respectively.

We introduce a structure in the space of fuzzy points by the following definition of *discord* and *grade of collocation* between fuzzy points.

DEFINITION. The *discord* between two fuzzy points $\tilde{A}$ and $\tilde{B}$ is

$$D(\tilde{A}, \tilde{B}) = \min_{X \in R_n} \max\{\|X-A\|_A, \|X-B\|_B\}. \tag{2.7}$$

We note that the discord is not a distance in the sense of functional analysis, because it does not satisfy the triangle inequality.

If the panderance matrix of one of the vectors in (2.7), say P_B, approaches a zero-matrix, then $\mu_B(X)$ approaches the characteristic function of the apex B. That is, the corresponding $\tilde{B}$ approaches the crisp point B. The corresponding limit of the discord is

$$\lim_{\tilde{B}\to B} D(\tilde{A}, \tilde{B}) = \|B - A\|_A. \tag{2.8}$$

DEFINITION. The *grade of collocation* between two fuzzy points $\tilde{A}$ and $\tilde{B}$ is

$$\gamma(\tilde{A}, \tilde{B}) = \max_{X\in R_n} \min\{\mu_A(X), \mu_B(X)\}. \tag{2.9}$$

The grade of collocation and the discord are related by

$$\gamma(\tilde{A}, \tilde{B}) = 1 - \min\{1, D(\tilde{A}, \tilde{B})\}. \tag{2.10}$$

2.3. Fuzzy functions. We interpret functions as mappings of a domain in R_n to a range in R_r. Usually, we shall assume that the mapping function is twice continuously differentiable with respect to all arguments of interest, i.e., with respect to $X \in R_n$ and with respect to any free parameters, if present. If some of the arguments are fuzzy numbers or fuzzy vectors, then the image of a crisp or fuzzy point in R_n is a fuzzy point in R_r. For consistency with crisp mappings, we shall always assume that the image of the apex of the argument is obtained by replacing in the function all fuzzy arguments by their apexes. That is, the apex of a fuzzy point is mapped onto the apex of the image. The panderance matrix of the image can be computed in terms of the panderance matrices of the argument and parameters by formulas which are derived in this section.

2.3.1. Linear functions. We first consider a mapping of R_n onto itself by an affine coordinate transformation, and derive a panderance propagation formula for such mappings from the requirement that the membership value of any $X \in R_n$ should be the same in both coordinate systems.

PROPOSITION 1. *Let Z, Z_0, X, and $A \in R_n$, let D be a nonsingular $(n \times n)$-matrix, and let*

$$Z = L(X) = D \cdot X + Z_0. \tag{2.11}$$

The image of the fuzzy point which is represented by the fuzzy vector

$$\tilde{A} = \{A, P_A\} \tag{2.12}$$

is the fuzzy point (vector)

$$\tilde{B} = \{B, P_B\}, \tag{2.13}$$

with the apex

$$B = D \cdot A + Z_0 \tag{2.14}$$

and the panderance matrix

$$P_B = D \cdot P_A \cdot D^T. \tag{2.15}$$

Proof. Equating the membership values of an $X \in R_n$ in the new and old coordinate systems, respectively, we obtain

$$\mu_B(L(X)) = \mu_A(X),$$

or, from the definition (2.4),

$$\|L(X) - B\|_B^2 = \|X - A\|_A^2 \tag{2.16}$$

in a neighborhood of A. If $X = A$, then the right-hand side of (2.16) vanishes and the left-hand side produces (2.14). Using this result, we express the left-hand side of (2.16) by

$$(D \cdot X + Z_0 - B)^T \cdot Q_B \cdot (D \cdot X + Z_0 - B) = (D \cdot (X - A))^T \cdot Q_B \cdot (D \cdot (X - A)) = (X - A)^T \cdot D^T \cdot Q_B \cdot D \cdot (X - A). \tag{2.17}$$

Hence, from (2.16)

$$D^T \cdot Q_B \cdot D = Q_A, \tag{2.18}$$

or, because D is not singular,

$$P_B = D \cdot P_A \cdot D^T. \qquad \square$$

The *extension principle for fuzzy functions* [15] states that, if Z is a function $L(X)$ and $\tilde{B} = L(\tilde{A})$, then we should have the relation

$$\mu_B(Z) = \max_{X : L(X) - Z = 0} \mu_A(X). \tag{2.19}$$

Formulas (2.14) and (2.15) are also componentwise consistent with this principle in cases where the affine mapping (2.11) is not a coordinate transformation. The consistency is a consequence of the following proposition.

PROPOSITION 2. *Let Z and $Z_0 \in R_r$, X and $A \in R_n$, let D be a nonzero $(r \times n)$-matrix of rank r, and let*

$$Z = L(X) = D \cdot X + Z_0. \tag{2.20}$$

Then the image $\tilde{B}$ of a regular fuzzy vector $\tilde{A}$ as defined by (2.14) *and* (2.15) *satisfies the extension principle condition* (2.19).

Proof. We formulate the right-hand side of (2.19) as the following constrained minimization problem:

$$\begin{aligned} &\text{Minimize} \quad W = (X - A)^T \cdot P_A^{-1} \cdot (X - A), \\ &\text{subject to} \quad D \cdot (X - A) = Z - B. \end{aligned} \tag{2.21}$$

We solve (2.21) using a Lagrange multiplier vector k and defining a modified objective function

$$\check{W} = \tfrac{1}{2}(X - A)^T \cdot P_A^{-1} \cdot (X - A) - k^T \cdot (D \cdot (X - A) - Z + B). \tag{2.22}$$

The normal equations for the minimization of $\check{W}$ are

$$P_A^{-1} \cdot (X - A) - D^T \cdot k = 0, \qquad D \cdot (X - A) - Z + B = 0. \tag{2.23}$$

Solving the first equation in (2.23) for k, we obtain

$$k = (D \cdot P_A \cdot D^T)^{-1} \cdot D \cdot (X - A). \tag{2.24}$$

We substitute this expression into the first equation (2.23) and multiply the result from left by $(X - A)^T$, obtaining

$$(X - A)^T \cdot P_A^{-1} \cdot (X - A) = (X - A)^T \cdot D^T \cdot (D \cdot P_A \cdot D^T)^{-1} \cdot D \cdot (X - A). \tag{2.25}$$

This equation holds at the solution of the minimization problem. The left-hand side of the equation is the minimum value of $\check{W}$ that we are seeking. The right-hand side can be simplified by using the second (2.23) and (2.15) to yield $(Z - B)^T \cdot P_B^{-1} \cdot (Z - B)$. Hence,

$$[(X - A)^T \cdot P_A^{-1} \cdot (X - A)]_{\min} = (Z - B)^T \cdot P_B^{-1} \cdot (Z - B). \tag{2.26}$$

$\square$

COROLLARY 1. *If the mapping*

$$Z = L(X) = D \cdot X + Z_0 \tag{2.27}$$

maps X into a subspace of R_n and $\mu_B(Z)$ is defined by (2.14) *and* (2.15), *then* (2.19) *holds in that subspace.*

Proof. Let the dimension of the subspace be s. Then, by a proper coordinate transformation, (2.27) can be rearranged in the form

$$Z_1 = D_1 \cdot X + Z_{10}, \qquad Z_2 = 0, \tag{2.28}$$

where D_1 is an $(s \times n)$-matrix with the rank s. Proposition 2 applies to the first s equations (2.28), proving the corollary.

If $\tilde{A}$ is not a regular fuzzy point in R_n, then only the subspace X_A (defined in the paragraph following (2.2)), where X has a positive membership value, is relevant for the mapping. This subspace is mapped by (2.20) into another linear subspace $Z_A \subseteq R_n$. Proposition 2 and Corollary 1 directly apply to the mapping of X_A into Z_A.

So far we have considered a crisp linear function, (2.20), with a possibly fuzzy argument X. Now we let the function be fuzzy and assume for simplicity that the argument X is crisp. That is, we consider the function

$$Z = \tilde{L}(X) = \tilde{D} \cdot X + \tilde{Z}_0, \tag{2.29}$$

where some of the elements of $\tilde{D}$ and $\tilde{Z}_0$ are fuzzy numbers or vectors. By definition, a fuzzy linear function is a subset in a function space S. That space consists of all linear functions which map R_n into R_r. Its dimension is $r(n+1)$, i.e., the number of elements of D and Z_0. Each element of S is defined by a particular set of the elements of D and Z_0, which in turn has a well-defined nonnegative membership value, given by the apexes and panderances of $\tilde{D}$ and $\tilde{Z}_0$. We assign to the corresponding function $L(X)$ the same membership value thus defining the fuzzy function set $\tilde{L}(X)$. The image of a crisp X by (2.29) is fuzzy due to the fuzziness of the function: a given X produces several images in R_r, each with a corresponding membership value. Also, a given $Z \in R_r$ possibly can be the image of several points $X \in R_n$ whereby each of the corresponding mappings has a specific membership value. According to the extension principle for fuzzy functions we assign to the image $Z = \tilde{L}(X)$ the supremum of these values as its membership value.

To calculate the apex B and the panderance matrix of the image $\tilde{B} = \tilde{L}(A)$ we can invoke the Proposition 2, because (2.29) can be considered as a crisp linear function (defined by the crisp elements of X) of the fuzzy arguments $\tilde{D}$ and $\tilde{Z}_0$. Let T_1 and $\tilde{T}_2 = \{T_2, P_{T2}\}$ be, respectively, the vector of all crisp and all fuzzy elements in $\tilde{D}$ and $\tilde{Z}_0$, and let (2.29) be expressed by

$$Z = F(X, T_1, \tilde{T}_2). \tag{2.30}$$

Then,

$$\tilde{B} = F(A, T_1, \tilde{T}_2) \tag{2.31}$$

has a panderance matrix which formally is given by

$$P_B = \frac{\partial F}{\partial T_2} \cdot P_{T2} \cdot \left(\frac{\partial F}{\partial T_2}\right)^T. \tag{2.32}$$

We present two examples. First, let z be a scalar function

$$z = f(X; T) = g(X; C, z_0) = X^T \cdot C + z_0 \tag{2.33}$$

and let T be the fuzzy vector

$$\tilde{T} = \left\{ \begin{pmatrix} C \\ z_0 \end{pmatrix}, P_T \right\}. \tag{2.34}$$

Then, for $X = A$, the apex of the function value is

$$z = b = A^T \cdot C + z_0 \tag{2.35}$$

and its panderance matrix is

$$P_b = (A^T, 1) \cdot P_T \cdot \begin{pmatrix} A \\ 1 \end{pmatrix}. \tag{2.36}$$

As a second example, assume that in (2.29) the panderance matrix P_D of the fuzzy elements of $\tilde{D}$ is diagonal, and that all concordances between elements of $\tilde{D}$ and $\tilde{Z}_0$ are zero. The formula (2.32), evaluated at $X = A$, then is

$$P_B = \begin{pmatrix} \sum_{i=1}^{n} P_{D1i} A_i^2 & 0 & \cdots & 0 \\ 0 & \sum_{i=1}^{n} P_{D2i} A_i^2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sum_{i=1}^{n} P_{Dri} A_i^2 \end{pmatrix}. \tag{2.37}$$

2.3.2. Nonlinear functions. We consider twice-differentiable functions

$$Z = F(W), \tag{2.38}$$

where $Z \in R_r$ and $W \in R_n$. Let the argument W be a fuzzy vector $\tilde{C}$. Then the value of Z is a fuzzy vector $\tilde{B}$ with the apex

$$B = F(C) \tag{2.39}$$

and a panderance matrix which may be approximately computed by expanding the function $F(W)$ at $W = C$. The linear part of the expansion is

$$L(W) = F(C) + \frac{\partial F}{\partial W} \cdot (W - C), \tag{2.40}$$

where $\partial F / \partial W$ is evaluated at $W = C$. Applying the panderance propagation formula (2.15) to the linearized function (2.40), we obtain the approximation

$$P_B \approx \frac{\partial F}{\partial W} \cdot P_C \cdot \left(\frac{\partial F}{\partial W} \right)^T. \tag{2.41}$$

For the intended application (mathematical model fitting) it is convenient to split the argument W of a function into two sets: observables X and parameters T, that is, to express model functions $F(W)$ in the form

$$Z = F(X, T). \tag{2.42}$$

Either X or T, or both, can be fuzzy and the approximate formula (2.41) may be applied accordingly. In particular, if $X = \tilde{A}$ and $T = \tilde{C}$, and the panderance matrix of the vectors $\tilde{A}$ and $\tilde{C}$ is

$$P_W = \begin{pmatrix} P_A & P_{AC} \\ P_{CA} & P_C \end{pmatrix}, \tag{2.43}$$

then the linearized form (2.41) of the panderance propagation formula is

$$P_B \approx \frac{\partial F}{\partial X} \cdot P_A \cdot \left(\frac{\partial F}{\partial X}\right)^T + \frac{\partial F}{\partial T} \cdot P_C \cdot \left(\frac{\partial F}{\partial T}\right)^T + \frac{\partial F}{\partial X} \cdot P_{AC} \cdot \left(\frac{\partial F}{\partial T}\right)^T + \frac{\partial F}{\partial T} \cdot P_{CA} \cdot \left(\frac{\partial F}{\partial X}\right)^T. \tag{2.44}$$

We consider as an example the scalar function

$$z = f(X, T, z_0) = X^T \cdot T + z_0 \tag{2.45}$$

with the fuzzy arguments $X = \tilde{A}$, $T = \tilde{C}$, and $z_0 = \tilde{\nu}$. The function can be expanded at the apex (a, C, ν) of its argument to yield

$$f(X, T, z_0) = A^T \cdot C + \nu + C^T \cdot (X - A) + A^T \cdot (T - C) + (z_0 - \nu) + (T - C)^T \cdot (X - A). \tag{2.46}$$

Using the approximate formula (2.44) we obtain

$$P_B \approx C^T \cdot P_A \cdot C + A^T P_C \cdot A + P_\nu + 2A^T \cdot P_{CA} \cdot C + 2P_{\nu A} \cdot C + 2A^T \cdot P_{C\nu}. \tag{2.47}$$

The approximation comes about by the neglect of the last summand in (2.46), i.e., the nonlinear part of the function. Its contribution generally is small if T is close to C and X is close to A.

The advantages of the approximate calculation of the panderance of a function by (2.41) are twofold. First, the formula is simple and can be easily included in algorithms using fuzzy arithmetic. Second, it produces function values which have conical membership functions. That is, with this approximation, one stays in conical fuzzy vector spaces (including degenerate vectors with semidefinite panderance matrices). The disadvantage is, as in any approximation, that in certain cases the approximation may not be sufficiently accurate.

2.3.3. Fuzzy function space. In analogy to the linear functions discussed above we may consider the nonlinear function

$$Z = F(X, \tilde{T}) \tag{2.48}$$

with a fuzzy parameter vector $\tilde{T}$ as a fuzzy function of a crisp argument:

$$Z = \tilde{F}(X). \tag{2.49}$$

To define the fuzzy set of functions $\tilde{F}$, we specify a corresponding membership function. In the case $\tilde{F}(X) = F(X, \tilde{T})$, each member of the set $\tilde{F}$ is given by a particular parameter value $t \in \tilde{T}$. We assign to the function $F(X, t)$ the membership value of t:

$$\mu_{FT}(F(X, t)) = \mu_T(t). \tag{2.50}$$

In this paper, we only consider fuzzy functions of the described type, that is, we assume that a fuzzy function always is defined in terms of a fuzzy function parameter.

To measure the separation between elements of a fuzzy function space, we define the following norm $\|\cdot\|_{FT}$ for the distance of the crisp function $F(X, t)$ from the crisp "apex" function $F(X, T)$ with the membership value one:

$$\|F(X, t) - F(X, T)\|_{FT} = \|t - T\|_T = [(t - T)^T \cdot P_T^{-1} \cdot (t - T)]^{1/2}. \tag{2.51}$$

(If $\tilde{T}$ is not a regular point, then P_T^{-1} is replaced by Q_T and the norm defined in accordance with (2.3).) With this definition, we have, in analogy to (2.4),

$$\mu_{FT}(F(X, t)) = 1 - \min\{1, \|F(X, t) - F(X, T)\|_{FT}\}. \tag{2.52}$$

The discord and the grade of collocation between two fuzzy elements of the fuzzy function space are defined in analogy to (2.7) and (2.9), respectively.

2.4. Fuzzy equations. A fuzzy equation is a fuzzy set of equations. We obtain a fuzzy equation, for instance, by setting a fuzzy function equal to a crisp or fuzzy zero. Let $X \in R_n$, $T \in R_p$, and the function $F(X, T) \in R_r$. We further assume that F is at least twice continuously differentiable in a proper domain,

$$r \leqq n \quad \text{and} \quad r \leqq p. \tag{2.53}$$

We define a fuzzy function as in § 2.3.3, and obtain a fuzzy equation by setting the function equal to a crisp vector with r vanishing components:

$$\tilde{F}(X) = F(X, \tilde{T}) = 0. \tag{2.54}$$

We shall assume that the right-hand side of a fuzzy equation always is a crisp vector. By permitting it to be a fuzzy vector, we do not gain additional generality, but merely increase the number of fuzzy parameters by r. That is, the fuzzy equations

$$\tilde{F}(X) = \tilde{0} \tag{2.55}$$

and

$$\tilde{G}(X) = \tilde{F}(X) - \tilde{0} = 0 \tag{2.56}$$

are equivalent sets of equations in a sense which we shall describe shortly. Any crisp $t \in \tilde{T}$ defines a particular crisp equation of the fuzzy set (2.54). That crisp equation either has no solution or defines a crisp solution $X_F(t)$. We assign to the equation and to its solution the membership value of the function $F(X, t)$, that is, the value $\mu_T(t)$.

The solution $X_F(t)$ of the crisp equation $F(X, t) = 0$ is in R_n an $(n-r)$-dimensional hypersurface which is the set (locus) of all points X that satisfy the equation. The fuzzy equation (2.54) defines in R_n a fuzzy surface $\tilde{X}_F$, that is, a fuzzy set of $(n-r)$-dimensional surfaces. The surface corresponding to the parameter value t is assigned the membership value $\mu_T(t)$. The fuzzy surface $\tilde{X}_F$ contains all points X which satisfy (2.54) for some $t \in \tilde{T}$. To an $X \in \tilde{X}_F$ we assign the largest membership value of all solution surfaces that pass through X, that is,

$$\mu_{XF}(X) = \sup_{t:\, F(X,\, t)=0} \mu_{FT}(F(X, t)) = \sup_{t:\, F(X,\, t)=0} \mu_T(t). \tag{2.57}$$

If $X \notin \tilde{X}_F$, i.e., if X does not solve the (2.54) for any $t \in \tilde{T}$, then we assign to X the membership value zero.

Now we consider (2.55) and (2.56). Their equivalency is related to the following proposition.

PROPOSITION 3. *Let* $X \in R_n$, $C \in R_r$, $\tilde{C} = \{C, P_C\}$, *and the function* $G(X) \in R_r$. *Then the solutions of*

$$G(X) = \tilde{C} \tag{2.58}$$

and

$$F(X, \tilde{C}) = G(X) - \tilde{C} = 0 \tag{2.59}$$

are identical fuzzy sets.

Proof. Equation (2.58) is a fuzzy set of equations. Let its solution be $\tilde{X}_G$. Because one side of the equation is crisp, we may use (2.57) to construct the solution in terms of its membership value. The result is

$$\mu_{XG}(X) = \sup_{t:\, G(X)=t} \mu_C(t). \tag{2.60}$$

On the other hand, according to (2.57) the solution of (2.59) is a fuzzy set with the membership function

$$\mu_{XF}(X) = \sup_{t: G(X)-t=0} \mu_C(t). \tag{2.61}$$

Because the delimiters for the maximization of $\mu_C(t)$ in (2.60) and (2.61) are equivalent crisp expressions, $\mu_{XG}(X) = \mu_{XF}(X)$. □

COROLLARY 2. *The solutions of* (2.55) *and* (2.56) *are identical.*

Proof. This is the special case $\tilde{C} = \tilde{0}$ of Proposition 3.

Proposition 3 and Corollary 2 show that certain algebraic manipulations of fuzzy equations are permitted in the sense that they do not affect the solution of the equations. We note, however, that not all manipulations which are permitted for crisp equations can be applied to fuzzy equations. For the present paper, the important consequence of Proposition 3 is that we can assume on the right-hand side of a fuzzy equation a crisp vector with zero-valued components.

The definition (2.57) of the membership function of $\tilde{X}_F$ is not practical for numerical algorithms. We provide, therefore, another equivalent definition in terms of a norm for the distance between X and the solution of $F(X, T) = 0$, i.e., between any $X \in R_n$ and the crisp solution surface $X_F(T)$. We define such a norm by

$$\|X - X_F\|_{XF} = \inf_{t: F(X,t)=0} \|t - T\|_T \tag{2.62}$$

and assign to $\|X - X_F\|$ an infinite value if $F(X, t) \neq 0$ for all $t \in R_p$. With this definition, (2.57) is equivalent to

$$\mu_{XF}(X) = 1 - \min\{1, \|X - X_F\|_{XF}\}. \tag{2.63}$$

This formulation is better suited for numerical calculations than (2.57), because the norm (2.62) can be approximately computed as shown by the following proposition.

PROPOSITION 4. *Let the r components of* $F(X, t)$ *be linearly independent functions of t. Then the following approximation of* (2.62) *is exact for functions* $F(X, t)$ *that are linear with respect to t*:

$$\|X - X_F\|_{XF} \approx \left[F(X,t)^T \cdot \left(\frac{\partial F}{\partial t} \cdot P_T \cdot \left(\frac{\partial F}{\partial t} \right)^T \right)^{-1} \cdot F(X,t) \right]^{1/2}. \tag{2.64}$$

Proof. We use the definition (2.62) and formulate the left-hand side of (2.64) as the following constrained minimization problem:

$$\begin{aligned} &\text{Minimize} \quad W = (t-T)^T \cdot P_T^{-1} \cdot (t-T), \\ &\text{subject to} \quad F(X,t) = 0. \end{aligned} \tag{2.65}$$

Using a Lagrange multiplier vector k, we obtain for the problem (2.65) a modified objective function

$$\check{W} = \tfrac{1}{2}(t-T)^T \cdot P_T^{-1} \cdot (t-T) - k^T \cdot F(X,t). \tag{2.66}$$

The corresponding normal equations are

$$P_T^{-1} \cdot (t-T) - F_t^T \cdot k = 0, \qquad F(X,t) = 0, \tag{2.67}$$

where $F_t = \partial F / \partial t$. After elimination of k from the first equation (2.67) and multiplication of the result from the left by $(t-T)^T$, we obtain the following formula for $\|t-T\|^2$ at the solution:

$$(t-T)^T \cdot P_T^{-1} \cdot (t-T) = (t-T)^T \cdot F_t^T \cdot (F_t \cdot P_T \cdot F_t^T)^{-1} \cdot F_t \cdot (t-T). \tag{2.68}$$

On the right-hand side of (2.68) we approximate

$$\frac{\partial F(X, t)}{\partial t} \cdot (t - T) \approx -F(X, T) \tag{2.69}$$

(because $F(X, t) = 0$), and also use the argument (X, T) instead of (X, t) in the product $F_t \cdot P_T \cdot F_t^T$. The result is the right-hand side of (2.64).

If F is linear with respect to t, then (2.69) holds exactly, and F_t is independent of t. Therefore, (2.64) is exact in the linear case. □

Now we investigate the structure of the membership function $\mu_{XF}(X)$ in the vicinity of the apex of the function. The apex itself is the crisp hypersurface $X_F(T)$, defined by $F(X, T) = 0$. The level surfaces $\mu_{XF} = \gamma$, or $\|X - X_F\|_{XF} = 1 - \gamma$, are given approximately by the equation (see (2.64))

$$F^T \cdot (F_t \cdot P_T \cdot F_t^T)^{-1} \cdot F = (1 - \gamma)^2. \tag{2.70}$$

Let $Y \in X_F(T)$, i.e., Y be a point of the apex of $X_F(T)$. Then, $F(Y, T) = 0$ and, expanding F in (2.70) in terms of $X - Y$ and only keeping the first nonzero term, we obtain for the level surface the equation

$$(X - Y)^T \cdot F_X^T \cdot (F_t \cdot P^T \cdot F_t^T)^{-1} \cdot F_X \cdot (X - Y) = (1 - \gamma)^2, \tag{2.71}$$

where $F_X = \partial F / \partial X$. Within the accuracy provided by the one-term approximation, the structure of the level surface in the vicinity of Y is characterized by the matrix

$$Q_Y = F_X^T \cdot (F_t \cdot P_T \cdot F_t^T)^{-1} \cdot F_X. \tag{2.72}$$

We recall that $F \in R_r$ and $r \leqq n$. The rank s of the matrix Q_Y is, therefore, at most equal to $r \leqq n$. (It equals r if F_X has a full row rank.) If $s = n$, then the solution $\tilde{X}_F$ is a fuzzy point in R_n, and the surfaces $\|X - X_F\|_{XF} = 1 - \gamma$ are hyperellipsoids in R_n. If $s < n$, then the matrix Q_Y is only semidefinite and the surfaces are hypercylinders. If γ is close to one, then the boundaries of the support of $\mu_{XF}(X) - \gamma$ are in case $s < n$ cylindrical surfaces that are approximately parallel to the apex surface $X_F(T)$.

If the apex $X_F(T)$ is a point in R_n then its distance norm $\|\cdot\|_A$ from a fuzzy point $\tilde{A}$ is defined by (2.2)-(2.3). If the apex $X_F(T)$ is a surface in R_n then we define the distance by

$$\begin{aligned} \|X_F - A\|_A &= \min_{X \in X_F} [(X - A)^T \cdot P_A^{-1} \cdot (X - A)]^{1/2} \\ &= \min_{X : F(X, T) = 0} [(X - A)^T \cdot P_A^{-1} \cdot (X - A)]^{1/2}. \end{aligned} \tag{2.73}$$

An approximate formula for $\|X_F - A\|_A$ is provided by the following proposition.

PROPOSITION 5. *Let the r components of F be linearly independent functions of X. Then the following approximate formula for* (2.73) *is exact for functions* $F(X, t)$ *that are linear with respect to* X:

$$\|X_F - A\|_A \approx [F^T \cdot (F_X \cdot P_A \cdot F_X^T)^{-1} \cdot F]^{1/2}, \tag{2.74}$$

where F and $\partial F / \partial X$ *are evaluated at* (A, T).

Proof. The proof follows the same steps as the proof of Proposition 4, with appropriate changes of the arguments of $F(X, T)$.

The separation between a fuzzy point $\tilde{A}$ and the fuzzy surface $\tilde{X}_F$ may be measured in analogy to (2.7) by the discord

$$D(\tilde{A}, \tilde{X}_F) = \min_{X \in R_n} \max \{\|X - A\|_A, \|X - X_F\|_{XF}\}. \tag{2.75}$$

The grade of collocation between $\tilde{A}$ and $\tilde{X}_F$ is (see (2.10))

$$\gamma(\tilde{A}, \tilde{X}_F) = 1 - \min\{1, D(\tilde{A}, \tilde{X}_F)\}. \tag{2.76}$$

In some important special cases the discord $D(\tilde{A}, \tilde{X}_F)$ can be explicitly computed. These cases are covered by the following Propositions 6 and 7.

PROPOSITION 6. *Let F_X and F_t be full rank constant coefficient matrices; let*

$$F(X, t) = F_0 + F_X \cdot X + F_t \cdot t, \tag{2.77}$$

$A \notin X_F(T)$; let P_A be nonsingular; let

$$\|X - A\|_A = [(X - A)^T \cdot P_A^{-1} \cdot (X - A)]^{1/2}, \tag{2.78}$$

$$\|X - X_F\|_{XF} = [F(X, T)^T \cdot (F_t \cdot P_T \cdot F_t^T)^{-1} \cdot F(X, T)]^{1/2}; \tag{2.79}$$

and let $L(X)$ be the locus of points where $\|X - A\|_A = \text{const.}$ is tangent to $\|X - X_F\|_{XF} = \text{const.}$ Necessary and sufficient for the locus to be a straight line is that

$$\beta \cdot F_t \cdot P_T \cdot F_t^T = F_X \cdot P_A \cdot F_X^T \tag{2.80}$$

for some positive β.

Proof. We first note that $L(X)$ by definition contains the point A, and that along any straight line through A the planes tangent to $\|X - A\|_A = \text{const.}$ are parallel to each other. The tangent plane which passes through a point $C \neq A$ is spanned by vectors $X - C$ that satisfy the equation

$$(C - A)^T \cdot P_A^{-1} \cdot (X - C) = 0. \tag{2.81}$$

The plane tangent to $\|X - X_F\|_{XF} = \text{const.}$ which passes through a point $C \notin X_F(T)$ is spanned by vectors $X - C$ that satisfy the equation

$$F(C, T)^T \cdot (F_t \cdot P_T \cdot F_t^T)^{-1} \cdot F_X \cdot (X - C) = 0. \tag{2.82}$$

Let $B \in X_F(T)$, i.e., $F(B, T) = 0$. Then, $F(C, T) = F_X \cdot (C - B)$ and (2.82) can be expressed in the form

$$(C - B)^T \cdot F_X^T \cdot (F_t \cdot P_T \cdot F_t^T)^{-1} \cdot F_X \cdot (X - C) = 0. \tag{2.83}$$

Equation (2.83) shows that planes through C and tangent to $\|X - X_F\|_{XF} = \text{const.}$ are parallel to each other for all such C which are located on any straight line that passes through a $B \in X_F$, i.e., through the apex of $\tilde{X}_F$.

We find a point $B \in X_F(T)$ which also is a point of the locus $L(X)$ by solving the following minimization problem:

Minimize $W = (B - A)^T \cdot P_A^{-1} \cdot (B - A)$,

subject to $F(B, T) = 0$.

Using a Lagrange multiplier vector k, we obtain a modified objective function

$$\hat{W} = \tfrac{1}{2}(B - A)^T \cdot P_A^{-1} \cdot (B - A) - k^T \cdot F(B, T)$$

and the corresponding normal equations

$$P_A^{-1} \cdot (B - A) - F_X^T \cdot k = 0, \qquad F(B, T) = 0. \tag{2.84}$$

Eliminating k from the first equation (2.84), we obtain

$$B - A = P_A \cdot F_X^T \cdot (F_X \cdot P_A \cdot F_X^T)^{-1} \cdot F_X \cdot (B - A),$$

or, because $F(B, T) = F(A, T) + F_X \cdot (B - A) = 0$,

$$B - A = -P_A \cdot F_X^T \cdot (F_X \cdot P_A \cdot F_X^T)^{-1} \cdot F(A, T). \tag{2.85}$$

The plane tangent to $\|X-A\|_A = \text{const.}$ through the point B is spanned by vectors $X-B$ which satisfy the equation (see (2.81))

$$(B-A)^T \cdot P_A^{-1} \cdot (X-B) = 0,$$

or, using (2.85),

$$F(A, T)^T \cdot (F_X \cdot P_A \cdot F_X^T)^{-1} \cdot F_X \cdot (X-B) = 0. \tag{2.86}$$

On the other hand, the surface $\|X-X_F\|_{XF} = \text{const.}$, which passes through the point A, has a tangent plane that is spanned by vectors $X-A$ satisfying the equation (see (2.82))

$$F(A, T)^T \cdot (F_t \cdot P_T \cdot F_t^T)^{-1} \cdot F_X \cdot (X-A) = 0. \tag{2.87}$$

Necessary and sufficient for $L(X)$ to be a straight line is that the planes defined by (2.86) and (2.87), respectively, are parallel to each other. Inspecting the equations, we see that necessary and sufficient for the planes to be parallel is that (2.80) holds. □

PROPOSITION 7. *Let the premises of Proposition* 6 *hold, and let* (2.80) *be satisfied. Then,*

$$D(\tilde{A}, \tilde{X}_F) = \frac{\|X_F - A\|_A \cdot \|A - X_F\|_{XF}}{\|X_F - A\|_A + \|A - X_F\|_{XF}}, \tag{2.88}$$

where $\|X_F - A\|_A$ *is given by* (2.74) *and* $\|A - X_F\|_{XF}$ *is given by* (2.79).

Proof. The discord $D(\tilde{A}, \tilde{X}_F)$ is defined by (2.75). Because $\|X-A\|_A$ as well as $\|X-X_F\|_{XF}$ are unbounded continuous functions, differentiable everywhere except at their zeros, the minimum in (2.75) is assumed at a point C where $\|C-A\|_A = \|C-X_F\|_{XF}$, and the planes tangent to $\|X-A\|_A = \text{const.}$ and tangent to $\|X-X_F\|_{XF} = \text{const.}$ coincide. C necessarily is a point of the locus $L(X)$, or, according to Proposition 6, a point of the straight line through A and B, the latter point being given by (2.85). Therefore, we can compute D by first finding that point C between A and B where both norms are equal, and then computing the value of the norms at $X = C$.

Let

$$C = A + \alpha \cdot (B-A) \tag{2.89}$$

be a point of $L(X)$. Then, by definition,

$$\|C-A\|_A = |\alpha| \cdot \|B-A\|_A = |\alpha| \cdot \|X_F - A\|_A$$

and

$$\|C-X_F\|_{XF} = |1-\alpha| \cdot \|A-X_F\|_{XF}.$$

Both expressions are equal for

$$\alpha = \frac{\|A-X_F\|_{XF}}{\|X_F-A\|_A + \|A-X_F\|_{XF}}.$$

Hence,

$$D(\tilde{A}, \tilde{X}_F) = \|C-A\|_A = \alpha \cdot \|X_F - A\|_A$$

is given by (2.88). □

COROLLARY 3. *Let* $F(X, t) = F_0 + F_X \cdot X + F_t \cdot t$ *be a scalar linear function of* X *and* t. *Then the discord* $D(\tilde{A}, \tilde{X}_F)$ *is given by* (2.88).

Proof. If F is a scalar linear function, then (2.80) is satisfied and Proposition 7 holds.

COROLLARY 4. *Let $\tilde{A}$ and $\tilde{B}$ be fuzzy points in R_n, and $P_A = \beta^2 \cdot P_B$. Then the discord between $\tilde{A}$ and $\tilde{B}$ is given by*

$$D(\tilde{A}, \tilde{B}) = \frac{\beta}{1+\beta} \cdot \|B - A\|_A. \tag{2.90}$$

Proof. We define $\tilde{B}$ as the solution of the fuzzy equation

$$F(X, \tilde{B}) = X - \tilde{B} = 0.$$

Propositions 6 and 7 hold for this function and, therefore

$$D(\tilde{A}, \tilde{B}) = \frac{\beta \cdot \|B - A\|_A^2}{(1+\beta) \cdot \|B - A\|_A} = \frac{\beta}{1+\beta} \cdot \|B - A\|_A. \qquad \square$$

If $F(X, \tilde{T})$ is not linear in X and T, then the discord formula (2.88) may be applied to a linearized approximation of the equation $F(X, \tilde{T}) = 0$, e.g., to a fuzzy set of equations with the elements

$$F(X, t) = F_X \cdot (X - Y) + F_t \cdot (t - T) = 0,$$

where $Y \in X_F(T)$. For scalar F this yields a reasonable value for the discord if t is sufficiently close to T and X is sufficiently close to $X_F(T)$. If F is not scalar, then also the condition (2.80) needs to be satisfied in order to have a reasonable approximation of the discord. In particular, in the case $r = n$ the condition (2.80) is important.

We make a final remark about the fuzziness of the surface $F(X, \tilde{T}) = 0$. Its spread is defined by (2.62) and (2.63). It is easy to see that the spread depends on the formulation of the equation $F(X, \tilde{T}) = 0$. Therefore, two forms of the equation which produce the same crisp apex surface $X_F(T)$ (defined by $F(X, T) = 0$) generally have different spreads, i.e., they define different fuzzy surfaces $\tilde{X}_F$. This dependence of $\tilde{X}_F$ on the equation formulation is an intrinsic property of fuzzy equations.

3. Fuzzy model fitting.

3.1. Fuzzy models. We consider mathematical models of observable events that are formulated as twice differentiable fuzzy equations among components of a fuzzy observable vector $\tilde{X}$. That is, the models are fuzzy equations of the type

$$F(\tilde{X}, \tilde{T}) = 0, \tag{3.1}$$

where F is a twice differentiable vector function. It is convenient to assume that the model parameter vector $\tilde{T}$ consists of two parts, $\tilde{T}_f$ and $\tilde{T}_p$. The first part, $\tilde{T}_f$, represents free model parameters. The second part, $\tilde{T}_p$, consists of prescribed parameters. (Either of these can be an empty set, of course.) The apex T_f of $\tilde{T}_f$ is determined by fitting the model to observations while the apex T_p is predetermined. The corresponding panderance matrices P_{Tf} and P_{Tp} of these vectors may be prescribed or determined concurrently with other unknowns of the regression problem.

We assume throughout the rest of this section that the dimensions of F, X, and T are r, n, and p, respectively. We will, however, permit the functions F_i and the corresponding dimensions r_i and n_i to be different for different observation vectors $\tilde{X}_i$. For simplicity, we assume that the model parameter dimension p is the same for all observations. This is not a restriction, because not all model functions F_i need to depend explicitly on all parameters.

3.2. Model fitting principles. We determine the unknown model parameter vector T by maximizing the membership values of the corrected observations as well as the membership values of the model surface at the corrected observations. Because the membership functions have a normalized maximum of one, the maximization is equivalent to a minimization of the deviations of the membership values from one. In particular, we minimize the sum of the squares of such deviations.

The constraints of the model fitting problem are the equations

$$F_i(X_i + c_i,\ T + e_i) = 0, \qquad i = 1, \cdots, s. \tag{3.2}$$

That is, local adjustments are applied to the apexes X_i of the observations $\tilde{X}_i = \{X_i, P_{Xi}\}$ as well as to the apex $T^T = (T_f^T, T_p^T)$ of the model parameter vector. The local corrections are only applied to the fuzzy components of $\tilde{X}_i$ and $\tilde{T}$. That is, if a component of these vectors is crisp, then the corresponding local adjustment is zero. The nonzero adjustments are obtained by minimizing the objective function

$$W = \sum_{i=1}^{S} [(1 - \mu_{Xi}(X_i + c_i))^2 + (1 - \mu_T(T + e_i))^2] \tag{3.3}$$

subject to the constraints (3.2).

The solution of the model fitting problem consists of the model parameter vector $\tilde{T}$ and the set of the local adjustments (residuals) c_i and e_i. In the n_i-dimensional space of the ith observation set, the fitted surface (the model) is the fuzzy solution of the equation $F_i(X, \tilde{T}) = 0$. It is reasonable to require that the observations $\tilde{X}_i$ are close to the fitted surface. In our fuzzy environment we express such a requirement in terms of the grade of collocation between $\tilde{X}_i$ and $\tilde{X}_{Fi}$, that is, by

$$\gamma(\tilde{X}_i, \tilde{X}_{Fi}) > \overset{*}{\gamma}_i, \qquad i = 1, \cdots, s. \tag{3.4}$$

The choice of $\overset{*}{\gamma}_i$ depends on the particular application and expresses the modeller's confidence in the adequacy of the model and in the reliability of the data panderance estimates. Typically we specify different desired values of $\overset{*}{\gamma}_i$ in the course of an investigation or set all desired values equal to zero and calculate a posteriori the attained grades of collocation. The attained grades then might be used as outlier indicators.

Using the relation (2.76) between $\gamma(\tilde{X}_i, \tilde{X}_{Fi})$ and the discord $D(\tilde{X}_i, \tilde{X}_{Fi})$, and the discord formula (2.88), this condition can be approximated by

$$\frac{\|X_{Fi} - X_i\|_{Xi} \cdot \|X_i - X_{Fi}\|_{XFi}}{\|X_{Fi} - X_i\|_{Xi} + \|X_i - X_{Fi}\|_{XFi}} < 1 - \overset{*}{\gamma}_i, \tag{3.5}$$

or in case of a scalar linearizable model, by

$$\frac{\|c_i\|_{Xi} \cdot \|e_i\|_T}{\delta_i \cdot \|c_i\|_{Xi} + \varepsilon_i \cdot \|e_i\|_T} < 1 - \overset{*}{\gamma}_i, \tag{3.6}$$

where the norms are given by (2.3) in terms of the panderances P_{Xi} and P_T,

$$\delta_i = 1 - \frac{F_i(X_i,\ T + e_i)}{F_i(X_i,\ T)} \tag{3.7}$$

and

$$\varepsilon_i = 1 - \frac{F_i(X_i + c_i,\ T)}{F_i(X_i,\ T)}. \tag{3.8}$$

The corrector ε_i appears in (3.6) because $\|c_i\|_{Xi}$ is not the norm of the whole distance of the observed point X_i from the apex X_{Fi}, but only of a part of it. A linear approximation yields $\|c_i\|_{Xi} = \varepsilon_i \cdot \|X_{Fi} - X_i\|_{Xi}$ and, correspondingly, $\|e_i\|_T = \delta_i \cdot \|X_i - X_{Fi}\|_{XFi}$.

If at the solution all the μ_{Xi} and μ_T in the expression (3.3) are positive then we can use the definition (2.4) and express the objective function (3.3) in terms of the norms $\|c_i\|_{Xi}$ and $\|e_i\|_T$ as follows:

$$W_N = \sum_{i=1}^{S} [\|c_i\|_{Xi}^2 + \|e_i\|_T^2] = \sum_{i=1}^{S} [c_i^T \cdot P_{Xi}^{-1} \cdot c_i + e_i^T \cdot P_T^{-1} \cdot e_i]. \tag{3.9}$$

Because W_N is a differentiable function of c_i and e_i for all nonzero values of the residuals, the numerical minimization of W_N subject to (3.2) is much simpler than the minimization of W. After the solution is obtained we may check the membership values μ_{Xi} and μ_T as well as the inequality constraints (3.4) and take appropriate action if the membership values are not positive or (3.4) are not satisfied. Usually they are satisfied if model and data are reasonable. The testing of the conditions is most conveniently done in terms of the discords $D(\tilde{X}_i, \tilde{X}_{Fi})$, i.e., by (3.5), because the values of D also provide a measure for the identification of outliers. The appropriate action again depends on the particular application and little can be said in general. It may involve a removal of outliers, reexamination of panderance estimates, change of model, etc.

If the panderance matrices P_{Xi} and P_T are given, then (3.2), (3.4), and (3.9) define a constrained minimization problem where the unknowns are T, c_i, and e_i. More common are situations where the parameter panderance matrix P_{Tf} of the free parameters $\tilde{T}_f$ is not known. Then it can be iteratively determined making use of the panderance propagation formula (2.41). For such an iteration, we assume an initial estimate $P_{Tf,0}$, numerically solve the minimization problem, and compute the ensuing panderance matrix $P_{Tf,1}$ using the linearized panderance propagation formula (2.41). Then this matrix is normalized (see below) and taken as an initial estimate, etc.

The linearized panderance propagation formula produces a panderance matrix with a norm proportional to the norms of the input panderances. Therefore, an iteration without a normalization of the iterated panderance matrix between iterations generally does not converge to a nonzero finite panderance matrix of the model parameters. The iteration with normalization determines on the other hand only the relative fuzziness and concordances of the parameters and thereby allows us to prescribe the relative importance of model and data fuzziness, respectively. Let this relative importance be expressed by a scalar parameter, say α. Let another parameter β express the relative importance of the fuzziness of the free and prescribed parameters, respectively. We express the objective function W_N as a function of the two parameters as follows:

$$W_N(\alpha, \beta) = \sum_{i=1}^{S} q_i^2, \tag{3.10}$$

$$q_i^2 = (c_i^T, e_i^T) \cdot P_{qi}^{-1} \cdot \begin{pmatrix} c_i \\ e_i \end{pmatrix}, \tag{3.11}$$

and

$$P_{qi}(\alpha, \beta) = \begin{pmatrix} (1-\alpha)^2 \cdot P_{Xi} & 0 & 0 \\ 0 & \alpha^2 \cdot \beta^2 \cdot P_{Tf} & 0 \\ 0 & 0 & \alpha^2 \cdot (1-\beta)^2 \cdot P_{Tp} \end{pmatrix}. \tag{3.12}$$

The two parameters α and β may have values between zero and one. If $\alpha = 0$ then we have an adjustment problem with fuzzy data and crisp model. If $\alpha = 1$ then the data are crisp and the model is fuzzy. If $\alpha > 0$ and $\beta = 0$ then the data are fuzzy, the free model parameters are crisp and the prescribed model parameters are fuzzy. If $\alpha > 0$ and $\beta = 1$ then the data are fuzzy and the model has only free and fuzzy parameters. By choosing appropriate values for α and β one can obtain a number of different combinations of conditions. We give three examples of such combinations. First, if all the panderance matrices P_{Xi}, P_{Tf}, and P_{Tp} are prescribed, then we use (3.12) with $\alpha = \frac{2}{3}$ and $\beta = \frac{1}{2}$. Second, if P_{Xi} and P_{Tp} are given, and P_{Tf} is iteratively determined, then we specify α and β by two conditions. First, we require that the relative weights of the contributions $\sum \|c_i\|^2_{Xi}$ and $\sum \|e_{pi}\|^2_{Tp}$ to W_N are the same. Second, we require that the contribution of $\sum \|e_{fi}\|^2_{Tf}$ (after the normalization of the iterated parameter panderance matrix P_{Tf}) to W_N have the same weight as the sum of the contributions of $\sum \|c_i\|^2_{Xi}$ and $\sum \|e_{pi}\|^2_{Tp}$. The first requirement is satisfied if

$$\alpha = \frac{1}{2-\beta}. \tag{3.13}$$

The second requirement we express by the equation

$$\sum_{i=1}^{S} \|c_i\|^2_{Xi} + \sum_{i=1}^{S} \|e_{pi}\|^2_{Tp} - \sum_{i=1}^{S} \|e_{fi}\|^2_{Tf} = 0. \tag{3.14}$$

Equations (3.13) and (3.14) are a system of equations for α and β, because the residuals depend on α and β. We can obtain the numerical solution of this system, for instance, using a regula falsi algorithm for β.

As a third example for the choice of α and β we assume that the P_{Xi} are known and P_{Tf} is iteratively determined, as before, and that P_{Tp} is known only up to scaling factor. Now, P_{Tf}, as well as P_{Tp}, contains arbitrary scaling factors. In this case, we postulate the following two conditions for α and β. First, we require that the contribution of $\sum \|e_{fi}\|^2_{Tf}$ to W_N receives the same weight as the contribution of $\sum \|e_{pi}\|^2_{Tp}$. Second, we require that the sum of all contributions of the parameter residuals has the same weight as the contribution $\sum \|c_i\|^2_{Xi}$ of the residuals of the observations. The two conditions can be expressed in the form of the following two equations:

$$G(\alpha, \beta) = \sum_{i=1}^{S} \|e_{fi}\|^2_{Tf} - \sum_{i=1}^{S} \|e_{pi}\|^2_{Tp} = 0, \tag{3.15}$$

$$H(\alpha, \beta) = -\sum_{i=1}^{S} \|c_i\|^2_{Xi} + \sum_{i=1}^{S} \|e_{fi}\|^2_{Tf} + \sum_{i=1}^{S} \|e_{pi}\|^2_{Tp} = 0. \tag{3.16}$$

This system of equations for α and β we solve numerically by a regula falsi algorithm in the α, β-plane. (Note that $G(\alpha, 0) \leqq 0$, $G(\alpha, 1) \geqq 0$, $H(0, \beta) \leqq 0$, and $H(1, \beta) \geqq 0$, which assures the existence of a solution within the unit square.)

3.3. Numerical treatment. The central part of the numerical treatment of the model fitting problem is the solution of the following constrained minimization problem:

$$\text{Minimize} \quad W_N(\alpha, \beta) = \sum_{i=1}^{S} (c_i^T, e_i^T) \cdot P_{qi}^{-1} \cdot \begin{pmatrix} c_i \\ e_i \end{pmatrix}, \tag{3.17a}$$

$$\text{subject to} \quad F_i(X_i + c_i, T + e_i) = 0, \quad i = 1, \cdots, s, \tag{3.17b}$$

where the P_{qi} are defined by (3.12). This problem formally is equal to a usual least squares model fitting problem with vector data and implicit model equations [2].

Software for the solution of such problems is available, for instance from [1], and can be used to solve (3.17). The solution of the example presented in § 4 was computed using the utility routine COLSAC [1]. The result of a COLSAC calculation consists of the optimal values of c_i, e_i, and T, and a posterior estimate of the panderance matrix P_{Tf} computed with the panderance propagation formula (2.41) from the input panderance matrices. If no iterations are involved, i.e., if P_T is prescribed, then a single call of COLSAC concludes the calculations.

The inequality constraints (3.4) usually are inactive, except for outliers. If it is determined that data with large discords cannot be discarded or treated in the same manner as other data (i.e., the inequality constraints ignored), then we have a more complicated programming problem at hand. There is no guarantee that the problem has a solution and, if a solution exists, it generally is not very useful, because it particularly accommodates outliers. The treatment of problems with outliers and too stringent conditions (3.4) very much depends on the application and typically may involve a change of the model functions F_i. We shall not further pursue this type of problem, and we assume that the inequality constraints are inactive at the solution.

For certain ranges of α and β the convergence of the procedure for the determination of P_T can be quite slow. In such cases, one may first determine the matrix P_T by iteration for extreme values of α and β and use an interpolated matrix P_T for subsequent calculations with other α and β. Then, iteration to find the final P_T might be resumed in the vicinity of the final point in the α, β-plane.

4. Example.

4.1. Data and model. The vulnerability of armored vehicles depends among other things on the deformation of armor plates by the blast of a nearby explosion. References [10] and [11] describe a series of experiments in which such deformations were measured. We shall use data from these experiments in an example of fuzzy model fitting.

The relevant data consist of the charge energy, the distance of the charge from the armor plate, the thickness of the plate, and the maximum deflection of the plate. (The size of the unsupported span of the plate was the same in all experiments and, therefore, its influence on the deformation cannot be obtained from the data. However, in most of the observed cases, the size of the deformation was such that the influence of the plate's support reasonably could be assumed to be negligible.) Some of these data were measured quite precisely (e.g., the plate thickness), while other data were inaccurate and were obvious candidates for fuzzy numbers. Such a datum is, e.g., the strength of the plate's material which we chose to characterize by its ultimate yield stress. The yield stress is ill-defined and difficult to measure, and it was found to vary considerably between plate specimens [11]. In summary, it was assumed that all data were fuzzy numbers with appropriate spreads.

The regression model of the plate deformation is not derived from a mechanical theory of the event, because such a general theory does not exist. Instead, the model was determined by exploratory data analysis and dimensional arguments. This uncertainty about the form of the model was represented in the model specification by using fuzzy equations as model constraints. Thus, in this example, the model as well as the data are fuzzy.

The following two dimensionless quantities were used as variables:

$$A = \frac{z}{d}, \qquad B = \frac{E}{d^2 \cdot T \cdot \sigma_y}, \tag{4.1}$$

where z (m) is the maximum deflection, d (m) is the distance of the charge from the

TABLE 1
Input data.

No.	B	B-spread	A	A-spread	Concordance
1	105.26	21.17	0.76056	0.01369	0.053516
2	102.93	20.70	0.76508	0.01158	0.063669
3	118.21	23.81	0.98405	0.01692	0.063657
4	51.00	10.21	0.43281	0.00458	0.018055
5	64.24	12.87	0.55264	0.00386	0.034592
6	51.34	10.28	0.37383	0.00344	0.020756
7	64.99	13.02	0.41531	0.00336	0.029891
8	65.18	13.06	0.43272	0.00582	0.017976
9	85.16	17.07	0.51171	0.00408	0.039626
10	115.30	23.12	0.86140	0.00971	0.038086
11	123.73	24.82	1.02472	0.02159	0.022186
12	97.93	19.63	0.77744	0.00821	0.034652
13	107.91	21.64	0.87198	0.00772	0.044735
14	81.80	16.39	0.67031	0.00425	0.030151
15	87.36	17.50	0.77164	0.00593	0.026309
16	89.61	17.95	0.79166	0.00471	0.036016
17	101.66	20.37	0.92772	0.00577	0.038845
18	112.61	22.56	0.91424	0.01386	0.016990
19	119.06	23.85	0.92703	0.00606	0.039406
20	69.83	14.07	0.40709	0.00767	0.066639
21	135.94	27.54	0.96792	0.02033	0.116586
22	95.37	19.25	0.62306	0.01206	0.088135
23	116.19	23.51	0.77957	0.01784	0.088517
24	96.04	19.39	0.69616	0.01057	0.112396
25	114.02	23.05	0.78626	0.01431	0.111430
26	114.04	23.06	0.78291	0.01279	0.124088
27	131.31	26.70	1.28464	0.03112	0.101073
28	129.67	26.07	0.95972	0.01227	0.075380
29	122.87	24.70	1.02897	0.01251	0.079290
30	130.02	26.09	1.19877	0.01199	0.061773
31	130.85	26.26	1.13072	0.01138	0.061376
32	152.65	30.64	1.16320	0.01258	0.063302
33	123.06	24.68	1.05742	0.01652	0.032715
34	86.41	17.55	0.20707	0.00873	0.071527
35	110.83	22.60	0.63734	0.01885	0.129906
36	66.25	13.32	0.28792	0.00438	0.063388
37	86.75	17.46	0.63775	0.00852	0.094093
38	100.05	20.16	0.54149	0.00845	0.093388
39	115.95	23.30	0.65883	0.01016	0.062502
40	90.62	18.19	0.45640	0.00668	0.052095
41	102.40	20.57	0.59983	0.00682	0.075114
42	91.35	18.34	0.48248	0.00518	0.071063
43	90.17	18.15	0.40348	0.00625	0.081118
44	122.35	24.69	0.60291	0.00953	0.107990

plate, E (J) is the energy released by the explosion, T (m) is the thickness of the plate, and σ_y (Pa) is the ultimate yield stress of the plate's material. The maximum deflection was obtained from a detailed analysis of various observed deflections, as described in reference [5]; d, E, and T were obtained by simple measurements; and for σ_y a nominal handbook value was used.

Table 1 lists the data with their estimated spreads and concordances. The spreads of d, E, T, and σ_y were assumed to be constant and estimated by reasonable assumptions

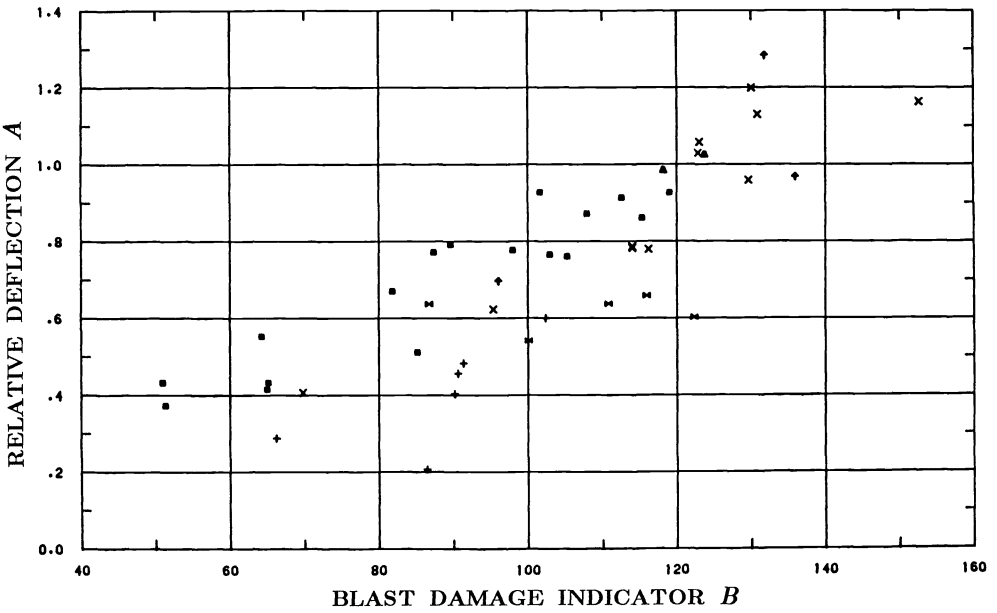


FIG. 2. *Observed plate deflection versus load.*

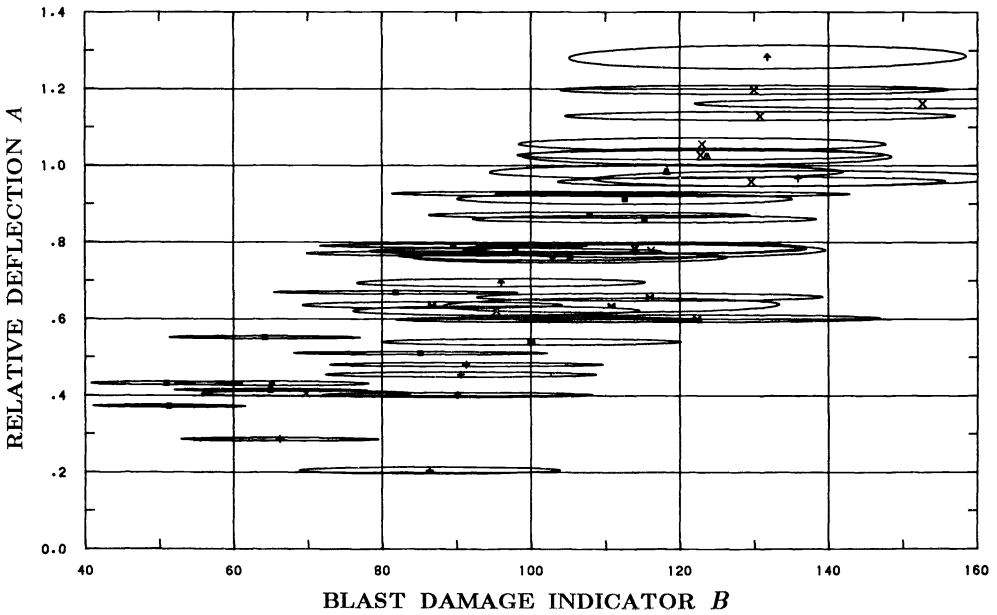


FIG. 3. *Observations with support ellipses.*

about measurement accuracy. The spread of the deflection z was obtained by the data analysis in reference [5]. The final spreads of A and B were computed from these estimates using (2.41). The concordances between the observed $\tilde{A}$ and the corresponding $\tilde{B}$ are not zero, because both depend on the distance $\tilde{d}$. Figures 2 and 3 display the data points without and with the corresponding supports of their membership functions, respectively. The different symbols of data points in the figures distinguish different subsets of the experiments, but the distinction is not important for the present example.

The model function was chosen to be

$$F(\tilde{A}, \tilde{B}; \tilde{a}, \tilde{b}, \tilde{c}) = \tilde{a} + \tilde{b} \cdot \tilde{B}^{\tilde{c}} - \tilde{A}. \tag{4.2}$$

The parameters $\tilde{a}$ and $\tilde{b}$ were assumed to be free and their spreads were not prescribed. The parameter $\tilde{c}$ was assumed to have the apex 1.50 and either a spread of 0.2 or an undetermined spread. Thus, in this example, the panderance matrix P_{Tf} of the free parameters is a (2×2)-matrix and the panderance matrix P_{Tp} of the prescribed parameter $\tilde{c}$ is a scalar. The data fitting was carried out as described in § 3 with the aid of the utility routine COLSAC [1].

4.2. Data and model spreads. The data panderance matrices P_{Xi} are given in terms of the data in Table 1 by

$$P_{Xi} = \begin{pmatrix} s_{Ai}^2 & s_{Ai} \cdot s_{Bi} \cdot s_{ABi} \\ s_{Ai} \cdot s_{Bi} \cdot c_{ABi} & s_{Bi}^2 \end{pmatrix}, \tag{4.3}$$

where s_{Ai} and s_{Bi} are the spreads of A_i and B_i, respectively, and c_{ABi} is the concordance between A_i and B_i. The parameter part of the regression panderance matrix (3.12) is

$$P_T = \begin{pmatrix} \alpha^2 \cdot \beta^2 \cdot P_{Tf} & 0 \\ 0 & \alpha^2 \cdot (1-\beta)^2 \cdot 0.2^2 \end{pmatrix}, \tag{4.4}$$

where the (2×2)-matrix P_{Tf} was determined by iteration. We calculated two cases. In one case, P_T was assumed as given by (4.4). In the other case, we assumed that the spread of the exponent $\tilde{c}$ (0.2 in (4.4)) is not known, that is, P_{Tp} was assumed to contain an unknown scaling factor.

In the former case, the normalization factor of P_{Tf} was chosen such that the maximum discord between any of the apexes X_i of the observations $\tilde{X}_i$, $i = 1, \cdots, s$, and the fuzzy surface $F(X, \tilde{T}_f, T_p) = 0$ was equal to five. In the latter case, we first normalized P_{Tf} and P_{Tp} separately such that the maximum discords between the observation apexes and $F(X, \tilde{T}_f, T_p) = 0$ and $F(X, T_f, \tilde{T}_p) = 0$, respectively, were equal to five. Then the complete matrix P_T (consisting of the two separately normalized submatrices) was normalized such that the maximum discord between the observation apexes and the surface $F(X, \tilde{T}_f, \tilde{T}_p) = 0$ was equal to five. We remind that the choice of a normalization procedure (in the present case, also the maximum discord values chosen for the normalization) is arbitrary in the sense that it affects only the numerical values of α and β, which regulate the relative weights of data and parameter residuals (see § 3.2).

The posterior fuzziness of the fitted model can be estimated using the panderance propagation formula (2.41) if the panderances of data and model parameters are all prescribed. If a part of the model parameter panderance matrix P_T is iteratively determined, then the result contains arbitrary parameters and a direct application of (2.41) cannot be justified. Therefore, we should in such cases determine the spread of the model by other means, also taking into account the actually observed deviations between data and fitted model.

In the present example, the fitted function was used in the form $\tilde{A} = A(B, \tilde{T}) = \tilde{a} + \tilde{b} \cdot B^{\tilde{c}}$, and its spread was adapted to observations by using two quadratic modular functions $M(B)$, one each for the positive and negative spreads of $\tilde{A}$. First, P_T was normalized using a factor Ψ such that the largest discord between the fitted curve $A(B, \tilde{T})$ and the X_i was five. This gave a reasonable spread increase pattern for the model outside the observed range of B but a too optimistic spread within the observed range. Therefore, modulator functions $M(B)$ were introduced to define modulated panderance matrices by

$$P_{T\,\mathrm{mod}} = M(B)^2 \cdot \Psi^2 \cdot P_T. \tag{4.5}$$

The modulator functions were determined such that the largest discord between the fitted $A(B, \tilde{T})$ and the observations $\tilde{X}_i$ was less than two. The form of the modulator functions was

$$M(B) = \max\{1, m_1 + m_2 \cdot (B - B_0)^2\} \tag{4.6}$$

with proper constants m_1, m_2, and B_0, and a different function was determined for positive and negative differences, respectively, between the fitted function and observations. The maximal discord value of two was established by numerical experiments and chosen because it produced a spread pattern in reasonable agreement with the scatter of observations. The final formula for the spread of $\tilde{A} = A(B, \tilde{T})$ is

$$s_A = \left[\frac{\partial A}{\partial T} \cdot P_{T\,\mathrm{mod}} \cdot \left(\frac{\partial A}{\partial T}\right)^T\right]^{1/2}. \tag{4.7}$$

4.3. Numerical results. Table 2 contains the numerical results in the two presented examples. Table 2 lists the optimal values of the model parameters, their spreads, and the concordance c_{ab} between a and b. The panderance matrix of the parameter vector is given in terms of the spreads and c_{ab} by (2.6). The quoted spreads correspond to a matrix that is normalized such that in the parameter space the largest discord between the locally corrected parameters and the apex of the parameter vector equals one and, therefore, the listed spread s_c of $\tilde{c}$ in Case 1 is not equal to the prescribed nominal value of 0.2. A comparison of the two cases illustrates the effect of assigning a fixed value to the spread of a model parameter. Thus, in the present example, the spread of $\tilde{c}$ is larger in Case 1 than in Case 2, as are the uncertainties of the other two model parameters $\tilde{a}$ and $\tilde{b}$. The increase of the latter two is a consequence of the condition (3.16). Plots of the final model curve $\tilde{A}(B)$ are shown in Figs. 4 and 5. The figures display for each data point its corresponding support ellipse (modified by the factor $1 - \alpha$, see (3.12)), the corrected data point $X_i + c_i$, and a short segment of the fitted function that passes through the corrected point and has the highest membership value μ_{FT}. (It is a segment of the curve $F(X, T + e_i) = 0$.) The spreads of the model curves shown in Figs. 4 and 5 are computed using the factor Ψ and the modulator functions $M(B)$ from Table 2 to modulate the parameter panderance matrix given at the top of the table (see § 4.2). The differences between the shapes of the spreads of the fitted curves in both figures illustrate how a prescribed model fuzziness may affect the result of the fitting.

An overview of the local parameter adjustments is provided by Figs. 6 and 7. They show in the parameter space three views of the locally adjusted parameter sets, i.e., the end points of the vectors $T + e_i = (a + e_{ai}, b + e_{bi}, c + e_{ci})^T$. Also shown are the contours of the support ellipsoid of the fuzzy parameter vector $\tilde{T}$. The ellipsoid corresponds to the spreads s_a, s_b, and s_c listed in Table 2.

TABLE 2
Model parameters.

Case 1. Prescribed apex and spread of $\tilde{c}$.	
$\tilde{a} =: \{-5.92 \cdot 10^{-2}, 31.60 \cdot 10^{-2}\}$,	$s_a \cdot \Psi = 5.83 \cdot 10^{-2}$
$\tilde{b} =: \{7.52 \cdot 10^{-4}, 3.68 \cdot 10^{-4}\}$,	$s_b \cdot \Psi = 0.68 \cdot 10^{-4}$
$\tilde{c} =: \{1.500, 0.134\}$,	$s_c \cdot \Psi = 0.025$
$c_{ab} = -0.952\,52$	
$\Psi = 0.184\,39$	
$M(B)_+ = 2.588 - 1.19 \cdot 10^{-4} \cdot (B - 48.4)^2$	
$M(B)_- = -2.617 + 3.75 \cdot 10^{-4} \cdot (B - 90.8)^2$	
$\alpha = 0.75$, $\beta = 0.67$	
Case 2. Prescribed apex and undetermined spread of $\tilde{c}$.	
$\tilde{a} =: \{-3.01 \cdot 10^{-2}, 20.88 \cdot 10^{-2}\}$,	$s_a \cdot \Psi = 12.28 \cdot 10^{-2}$
$\tilde{b} =: \{7.50 \cdot 10^{-4}, 2.28 \cdot 10^{-4}\}$,	$s_b \cdot \Psi = 1.52 \cdot 10^{-4}$
$\tilde{c} =: \{1.500, 0.040\}$,	$s_c \cdot \Psi = 0.024$
$c_{ab} = -0.955\,21$	
$\Psi = 0.587\,88$	
$M(B)_+ = 1.952 - 4.18 \cdot 10^{-4} \cdot (B - 80.4)^2$	
$M(B)_- = -2.641 + 3.80 \cdot 10^{-4} \cdot (B - 93.4)^2$	
$\alpha = 0.60$, $\beta = 0.43$	

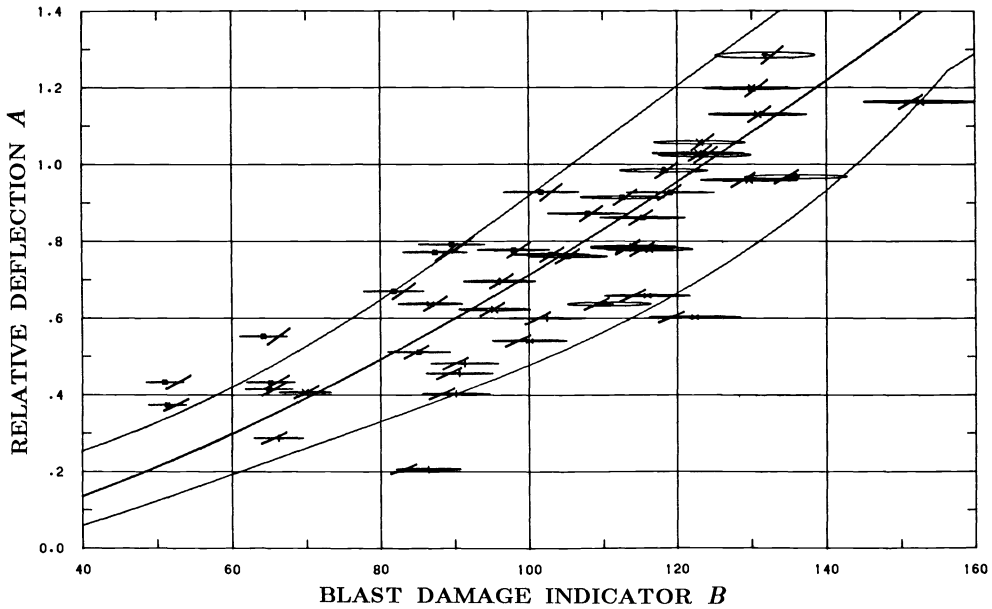


FIG. 4. *Model curve in Case* 1 *with prescribed spread of* $\tilde{c}$.

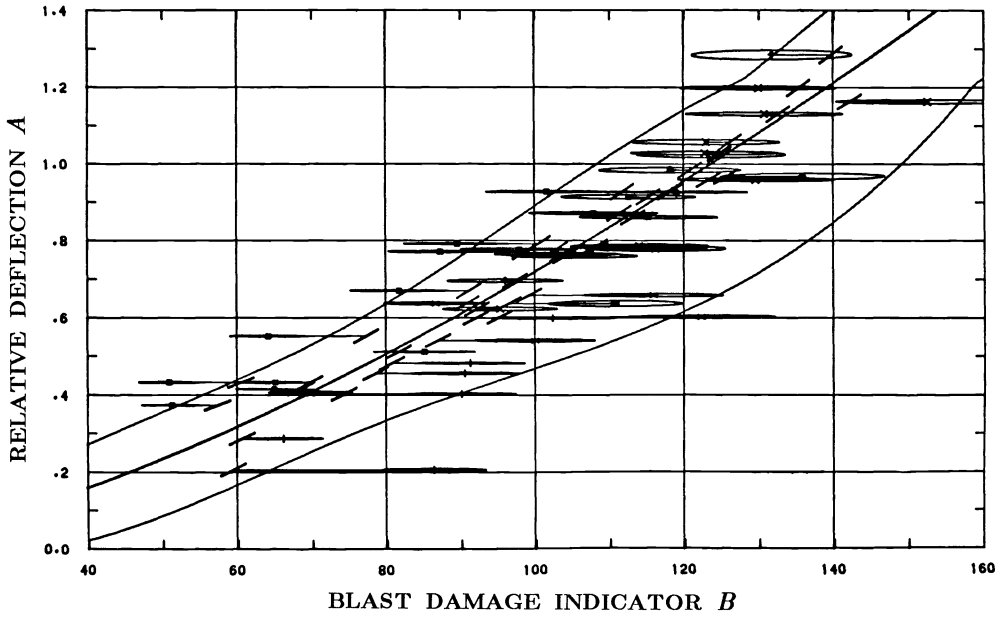


FIG. 5. *Model curve in Case 2 with undetermined parameter spreads.*

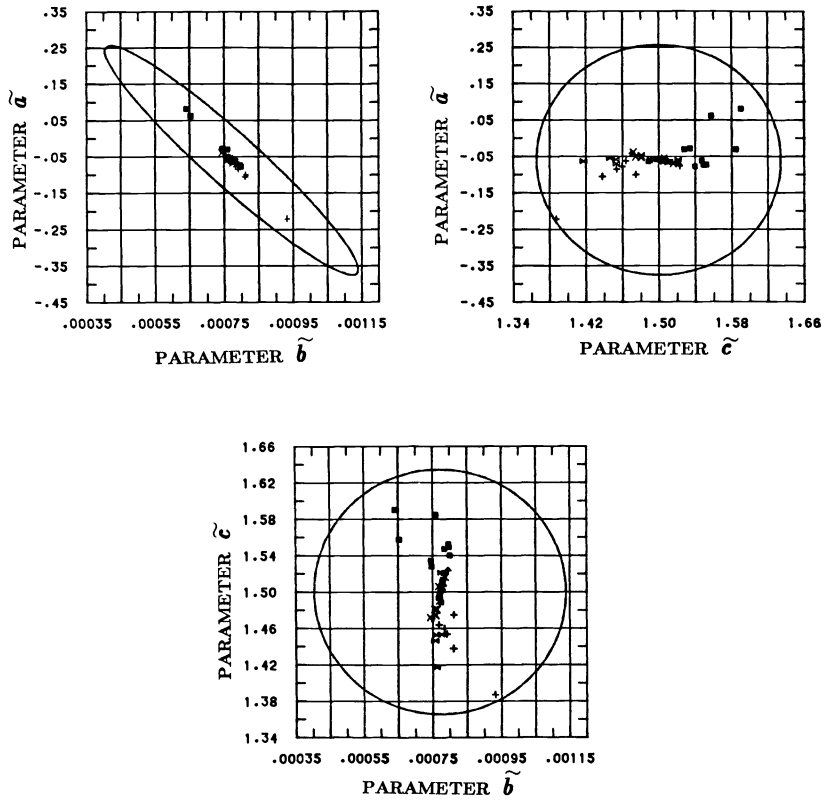


FIG. 6. *Local parameter adjustments in Case 1.*

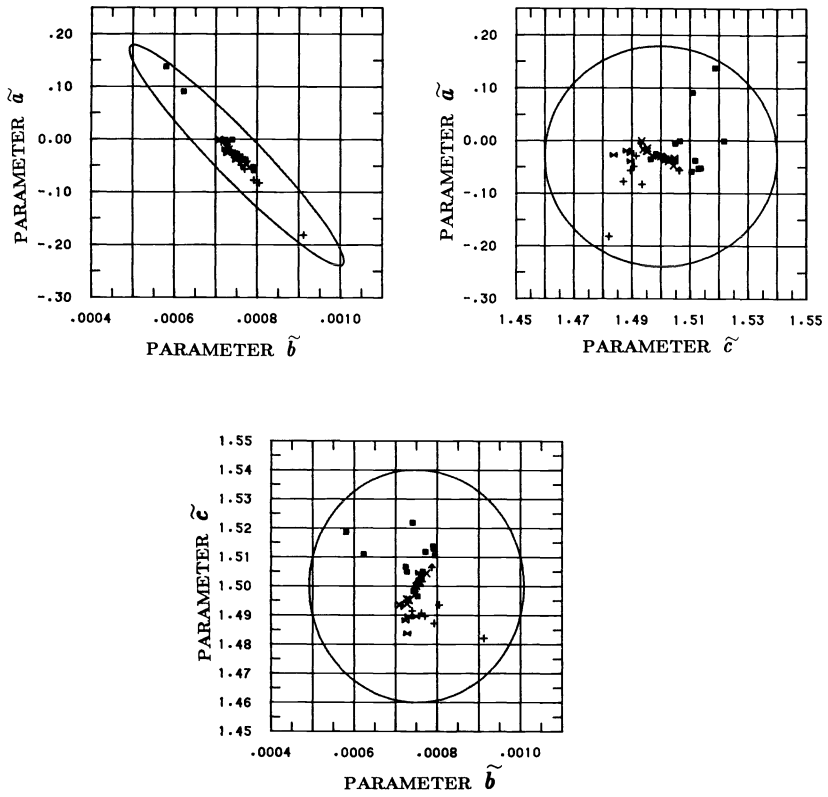


FIG. 7. *Local parameter adjustments in Case 2.*

5. Summary. We have developed algorithms for a special type of fuzzy regression. The algorithms are based on concepts of analytic geometry and restricted to data and model parameter vectors which are characterized by a particular type of conical membership function. The regression principle is to maximize in a least squares sense membership values of fuzzy observations and of fuzzy fitting functions. The restriction to conical membership functions and the employment of a least squares principle have the advantage that the calculations can be done with available software for least squares regression problems with implicit constraint functions. The model functions can be implicit and nonlinear with respect to data as well as with respect to model parameters, and the algorithms accept fuzzy data as well as fuzzy parameters, whereby the latter may be either free or prescribed. The method is more flexible than previously published methods, which are restricted either to linear models, [6]-[9], [12], and [13], or to crisp models [3], or crisp data [4]. (The membership functions are restricted to particular types in all of the quoted publications about fuzzy regression methods.) We feel that the possibility to freely choose the model functions is a considerable asset of the present method. The method has been applied to several real life problems, one of which is described in § 4.

REFERENCES

[1] A. CELMIŅŠ, *A manual for general least squares model fitting*, U.S. Army Ballistic Research Laboratory Report ARBRL-TR-02167, AD#B040229L, Aberdeen Proving Ground, MD, June 1979.

[2] A. CELMIŅŠ, *Least squares optimization with implicit model equations*, in Mathematical Programming with Data Perturbations II, A. V. Fiacco, ed., Marcel Dekker, New York, 1982, pp. 131-152.
[3] ———, *Least squares model fitting to fuzzy data*, Fuzzy Sets and Systems, 22 (1987), pp. 245-269.
[4] ———, *Multidimensional least-squares fitting of fuzzy models*, Math. Modelling, 9 (1987), pp. 669-690.
[5] ———, *A ballistic limit indicator for steel plates subject to blast*, U.S. Army Ballistic Research Laboratory Report BRL-TR-2928, AD#B125489, Aberdeen Proving Ground, MD, July 1988.
[6] P. DIAMOND, *Least squares fitting of several fuzzy variables*, in Proc. 2nd International Fuzzy Systems Association Congress, Tokyo, Japan, 1987, pp. 329-332.
[7] ———, *Fuzzy least squares*, Inform. Sci., 46 (1988), pp. 141-157.
[8] ———, *Some maximum likelihood estimates for the fuzzy linear model*, in Proc. Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, 1988, Urbino, Italy, 1988.
[9] B. HESHMATY AND A. KANDEL, *Fuzzy linear regression and its applications to forecasting in uncertain environment*, Fuzzy Sets and Systems, 15 (1985), pp. 159-191.
[10] A. R. KIWAN AND H. J. GOODMAN, *The response of rolled homogeneous steel armor plates to explosive blast loading*, U.S. Army Ballistic Research Laboratory Report ARBRL-TR-02393, AD#B063618, Aberdeen Proving Ground, MD, February 1982.
[11] A. R. KIWAN, *Experimental and analytical study of the response of clamped rolled homogeneous steel armor to close-in blast loading*, U.S. Army Ballistic Research Laboratory Report ARBRL-TR-02509, AD#B076399, Aberdeen Proving Ground, MD, July 1983.
[12] H. TANAKA, S. UEJIMA, AND K. ASAI, *Linear regression analysis with fuzzy model*, IEEE Trans. Systems Man Cybernet., 12 (1982), pp. 903-907.
[13] H. TANAKA AND H. ISHIBUCHI, *Identification of possibilistic linear systems by quadratic membership functions of fuzzy parameters*, in Proc. 3rd International Fuzzy Systems Association Congress, Seattle, WA, 1989, pp. 516-519.
[14] L. A. ZADEH, *Fuzzy sets*, Inform. and Control, 8 (1965), pp. 338-353.
[15] ———, *The concept of a linguistic variable and its application to approximate reasoning, Part* I, Inform. Sci., 8 (1975), pp. 199-249.
[16] H. J. ZIMMERMANN, *Fuzzy Set Theory—and Its Applications*, Kluwer-Nijhoff, Boston, MA, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 547-572, May 1991
© 1991 Society for Industrial and Applied Mathematics
005

MOVING POINT, PARTICLE, AND FREE-LAGRANGE METHODS FOR CONVECTION-DIFFUSION EQUATIONS*

M. D. REES† AND K. W. MORTON‡

Abstract. Recently proposed moving point and particle numerical methods for time-dependent convection-diffusion equations are briefly reviewed. The methods are applied to a one-dimensional model problem and an alternative and superior method is presented. This is generalised to higher spatial dimensions by reformulating it as a finite-volume or finite-element method on a moving triangulation. It is shown that when solving the heat equation on triangular meshes with Dirichlet data, it is important to use a certain type of triangulation for the methods to possess maximum principles. This triangulation (known as the locally Delaunay triangulation) is shown to be unique. Moreover, it possesses a "minimum-energy" property: hence it may be generated from an arbitrary given triangulation by a sequential process of mesh reconnection. This is important for the convection-diffusion algorithm, as using the transported triangulation from the last time level as an initial guess results in a very efficient mesh generation procedure. The two-dimensional algorithm is thus a free-Lagrange method and is applied to a model problem in two dimensions. Some ideas concerning the generalisation to three dimensions are briefly discussed.

Key words. moving point, particle, free-Lagrange, convection-diffusion, triangulation, finite volume, finite element, mass-lumped

AMS(MOS) subject classifications. 65M25, 65M50, 65M60

1. Introduction.

Recently, several authors [8], [30] have described so-called *moving point* and *particle* numerical methods for time-dependent convection-diffusion equations of the form

$$c_t + \nabla \cdot (\mathbf{a}(\mathbf{x}, t)c) = \nabla \cdot (b(\mathbf{x}, t)\nabla c), \tag{1}$$

$$c(\mathbf{x}, 0) = c_0(\mathbf{x}), \tag{2}$$

where $\mathbf{x} \in \mathbf{R}^d$, $t \in [0, T_F]$, and $\mathbf{a}$, b are known functions of their arguments.

The common feature of the proposed methods is the presence of a set of nodes (the *points* and *particles*, respectively) which are transported under the action of the convective field; their positions satisfy the equation

$$\frac{d\mathbf{x}}{dt} = \mathbf{a}(\mathbf{x}, t). \tag{3}$$

When written in coordinates moving with the convective field, (1) becomes

$$\frac{dc}{dt} = \nabla \cdot (b(\mathbf{x}, t)\nabla c) - (\nabla \cdot \mathbf{a})\, c. \tag{4}$$

We shall concentrate on the incompressible case $\nabla \cdot \mathbf{a} = 0$, so that (4) takes the form of the heat equation in the moving coordinate system. This representation of the convective term avoids many of the difficulties associated with its discretisation on an Eulerian mesh, such as excess numerical diffusion and under/overshoot. However, the diffusive term must now be approximated using information at the scattered nodal positions and the form of this approximation essentially differentiates the methods. We also note that we have the convection-dominated case particularly in mind (recent work in related areas includes [6], [7]).

* Received by the editors April 24, 1989; accepted for publication (in revised form) April 18, 1990.

† Braxton Associates, 90 Long Acre, London WC2E 9RA, United Kingdom. This author's research was supported in part by the Science and Engineering Research Council.

‡ Oxford University Computing Laboratory, 11 Keble Road, Oxford OX1 3QD, United Kingdom.

In § 2, we describe the methods in detail and then, in § 2.4, apply them to a one-dimensional model problem. For this model problem we present an alternative and superior method. The desire to generalise it to higher spatial dimensions leads to the idea of free-Lagrange methods (see [4], [5], [9], [10]). The two-dimensional method is thus formulated as a finite-element/volume method on a moving triangulation. In § 3 we show the importance of using a Delaunay-type triangulation in order that the method be stable and also that such a triangulation may be rapidly generated from an arbitrary given triangulation by a sequential process of reconnection. In § 4 the method is applied to a model problem in two dimensions. Finally, § 5 summarises and concludes the work and briefly describes some areas which we hope to develop in the future. It is hoped that the paper reviews, clarifies and develops ideas in several areas of computational mathematics.

2. Description of the methods. Here, we describe briefly the moving point and particle methods; the interested reader is referred to the cited references for more details.

2.1. The moving point method. We describe a moving point method based closely on that presented in [8]. A generalised treatment may be found in [34]; we present only the principles of the method here.

The scheme involves placing a set of points in the computational box initially (either uniformly or in a manner more appropriate to represent the initial data) and giving each point, labelled with a multi-index $\alpha, \beta \cdots$, an associated function value

$$C_\alpha^0 := c_0(\mathbf{x}_\alpha^0), \tag{5}$$

with the superscript denoting the time level. The basic stages in the algorithm are then the positional update and the concentration update due to the diffusion, which we assume to be constant for simplicity of presentation. The latter stage is the principal point of interest; it is performed by placing a difference stencil centred on the point (see Fig. 1) and differencing using any standard finite-difference formula; here we describe the method with respect to the usual five-point stencil in two dimensions. (The alignment of the stencil may be chosen arbitrarily but will, in the absence of reasons for choosing otherwise, generally be along the coordinate axes.)

However, in contrast to fixed grid finite-difference schemes, the function values at the stencil positions are unknown. Thus, they need to be obtained from some approximation scheme. This approximation scheme, together with the original choice of finite-difference stencil, may be used to classify these "finite-difference moving

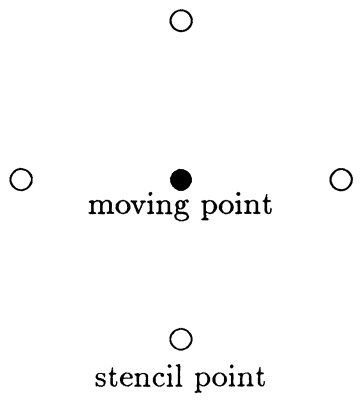


FIG. 1. *A moving point difference stencil.*

point" schemes. Explicit updates in time are used since the "inverse" problem of interrelating moving point values via the approximation scheme is extremely cumbersome. This does not severely restrict the timestep, since it is determined by the diffusion coefficient (which is small for problems for which the method is intended) and the stencil width (which, as we shall see later, if too small, will actually reduce the accuracy). We shall discuss the stability and accuracy of the method.

The explicit Euler update then reads

$$C_\alpha^{n+1} = (1-4\kappa)C_\alpha^n + \kappa \sum_{\beta \in \text{stencil}} (AC)_\beta^n. \tag{6}$$

The approximation scheme is a map $A: W_\infty^0(\mathbf{R}^d) \to W_\infty^0(\mathbf{R}^d)$ possessing the following properties:

(P1) $A(\mu f + \lambda g) = \mu Af + \lambda Ag$ for scalars μ, λ (linearity);

(P2) $\|Af\|_\infty \leqq \|f\|_\infty$ (stability);

so that for

$$0 \leqq 4\kappa \leqq 1 \tag{7}$$

we have

$$\sup_\alpha |C_\alpha^{n+1}| \leqq \sup_\alpha |C_\alpha^n|, \tag{8}$$

with a similar inequality holding for the error at the point positions but with an additional term $\bar{T}\Delta t$ on the right-hand side, where $\bar{T}$ bounds the truncation error.

Concerning the accuracy of the method, suppose the approximation scheme satisfies

(P3) $\|Af - f\|_\infty \leqq \Gamma h^s |f|_\infty^s$, $f \in W_\infty^s(\mathbf{R}^d)$ (accuracy),

where Γ is a constant, h a parameter representing the moving point spacing, and $|f|_\infty^r$ the Sobolev seminorm. Then, ignoring the errors due to time truncation, the error in the method behaves as $O(H^2 + h^s/H^2)$, where H is the width of the difference stencil. More detailed analysis, including the effects of time truncation are found in [8]. The treatment of more general difference stencils can be found in [34].

As examples of possible approximation schemes consider those based on a triangulation. The value at a stencil point P inside a triangle may be taken as

$$C_P = \sum_{i=1}^3 \lambda_i C_i \Big/ \sum_{i=1}^3 \lambda_i, \tag{9}$$

where $\lambda_i \geqq 0$ (see Fig. 2). Such a scheme has first-order error except in the case that λ_i is the area of the opposite subtriangle to vertex i, in which case C_P is the linear interpolant of the vertex values and hence the error is second order: a simple first-order scheme results from taking each $\lambda_i = \frac{1}{3}$.

As an alternative, we may interpolate linearly from the triangulation to the nodes of a fixed uniform mesh and then perform bilinear interpolation from the grid values to the stencil points. This considerably reduces the computation when compared to direct interpolation to the stencil points.

2.2. The particle method. Here we recall the important features of the particle method of Raviart in a rather informal manner. The reader is referred to [23], [31], and [30] for more detail and analysis. Earlier work on particle methods in general includes [16] and [17].

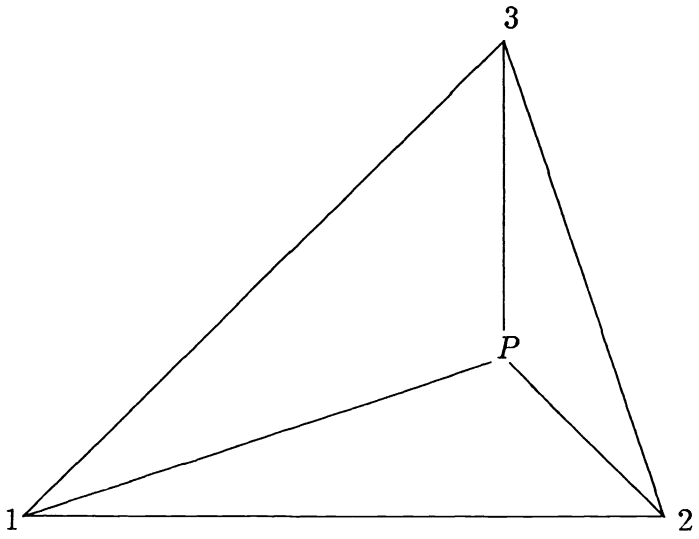


FIG. 2. *Linear interpolation from a triangulation.*

In this method the function $c(x, t)$ is approximated as a linear combination of Dirac measures:

$$c^h := \Pi^h c = \sum_{\alpha \in \mathbf{Z}^d} w_\alpha(t) c(\mathbf{x}_\alpha(t), t) \delta(\mathbf{x} - \mathbf{x}_\alpha(t)), \tag{10}$$

where $w_\alpha c_\alpha(t)$ is termed the *mass* of particle α. The scalar $w_\alpha(t)$ is the *volume* associated with particle α and satisfies

$$\frac{dw_\alpha}{dt}(t) = w_\alpha \nabla \cdot \mathbf{a}(\mathbf{x}_\alpha(t), t), \tag{11}$$

since the volume change is proportional to the Jacobian of $\mathbf{x}$, which itself satisfies (11).

A smooth representation of the function c is obtained by replacing each point particle by a smooth "particle" which has the same mass and has small support $O(\varepsilon)$. For this purpose, we introduce a smooth "cutoff" function $\tau(\mathbf{x})$ with the properties

$$\int_{\mathbf{R}^d} \tau(\mathbf{x})\, d\mathbf{x} = 1, \tag{12}$$

$$\int_{\mathbf{R}^d} \mathbf{x}^j \tau(\mathbf{x})\, d\mathbf{x} = 0 \quad \forall j \in \mathbf{N}^d, \qquad 1 \leqq |j| \leqq k-1, \tag{13}$$

$$\int_{\mathbf{R}^d} |\mathbf{x}|^k |\tau(\mathbf{x})|\, d\mathbf{x} < \infty \tag{14}$$

for some $k \in \mathbf{Z}_+$. For $\varepsilon > 0$, set

$$\tau_\varepsilon(\mathbf{x}) = \varepsilon^{-d} \tau\left(\frac{\mathbf{x}}{\varepsilon}\right). \tag{15}$$

Then $\tau_\varepsilon(\mathbf{x})$ also satisfies (12), (13), (14). The smooth representation of c is then given by

$$c_\varepsilon^h := (\Pi^h c) * \tau_\varepsilon = \sum_{\alpha \in \mathbf{Z}^d} w_\alpha(t) c(\mathbf{x}_\alpha(t), t) \tau_\varepsilon(\mathbf{x} - \mathbf{x}_\alpha(t)). \tag{16}$$

The cutoff function will generally be radially symmetric; a family of possible functions are given by the *super-Gaussian* functions:

$$G_{2n}(r) \propto \exp(-r^2)\frac{H_{2n+1}(r)}{r}, \tag{17}$$

for which $k = 2n+2$. The cases $n = 0, 1$ have been considered in [25] and [26]; when $n = 0$ we get the Gaussian.

Integrating (1) over the volume associated with particle α and using the approximation

$$\int_{box} f(x)\,dx \approx w_\alpha f(x_\alpha) \tag{18}$$

together with (11) gives

$$\frac{d(w_\alpha c_\alpha)}{dt} = w_\alpha \nabla \cdot (b\nabla c)_\alpha. \tag{19}$$

Raviart describes an approximation for the diffusive term as follows:

$$(\nabla \cdot (b\nabla c))_\alpha \approx \sum_\beta w_\beta (b_\alpha + b_\beta)(c_\beta - c_\alpha)\frac{(\mathbf{x}_\beta - \mathbf{x}_\alpha)\cdot \nabla \tau_\varepsilon(\mathbf{x}_\alpha - \mathbf{x}_\beta)}{|\mathbf{x}_\beta - \mathbf{x}_\alpha|^2}, \tag{20}$$

which represents a pairwise interaction of particles. Again, we refer the reader to [23] and [30] for more details and results regarding convergence.

We contrast the diffusive approximation (20) with that in the moving point method in which it is taken from local moving points only. We can explain the necessity for this "global" nature of the particle method as being due to the required smoothing of the mass distribution.

In summary, the particle method is defined by the equations

$$\frac{d\mathbf{x}_\alpha}{dt} = \mathbf{a}(\mathbf{x}_\alpha(t), t), \tag{21}$$

$$\mathbf{x}_\alpha(0) = \alpha h, \tag{22}$$

$$\frac{dw_\alpha}{dt} = w_\alpha(t)\nabla \cdot \mathbf{a}(\mathbf{x}_\alpha(t), t), \tag{23}$$

$$w_\alpha(0) = h^d, \tag{24}$$

$$\frac{d}{dt}(w_\alpha c_\alpha) = w_\alpha(t)\overline{(\nabla \cdot (b\nabla c))_\alpha}, \tag{25}$$

$$(w_\alpha c_\alpha)(0) = w_\alpha(0) c_0(\mathbf{x}_\alpha(0)), \tag{26}$$

where $\overline{(\nabla \cdot (b\nabla c))_\alpha}$ is the diffusion approximation obtained from (20).

Remarks. (1) If we require the desirable "positivity property" $g \geqq 0 \Rightarrow g_\varepsilon^h \geqq 0$, then $k \leqq 2$. Such a property may be important for the stability of the numerical method.

(2) The radial symmetry property on τ guarantees conservation of mass:

$$\sum_\alpha (w_\alpha c_\alpha)(t) = \sum_\alpha (w_\alpha c_\alpha)(0) \quad \forall t \geqq 0; \tag{27}$$

this is clearly a desirable property.

(3) The particle method can be applied to the vorticity-stream function formulation of the Navier–Stokes equations by using Green's function to solve the Poisson

equation. The resulting method is then known as a *vortex* method. Alternatively, the Poisson equation may be solved using finite-difference (or finite-element) techniques, leading to a *vortex-in-cell* method. Further material in this direction is found in [1], [12], [21], and [39].

2.3. Free-Lagrange methods. Methods of this type, as described in [9], can be regarded as developments of finite-difference methods in which the moving of the gridpoints has the effect of combining the convective terms in a system of equations with the time derivatives to give Lagrangian derivatives: and then the remaining terms are approximated on a mesh which changes with time. We describe below a particular gridfree method of this type, first for a one-dimensional problem and then, in § 3, for two-dimensional problems.

2.4. Application to a model problem. Here we compare results of numerical experiments with the moving point and particle methods above when applied to the one-dimensional incompressible convection-diffusion problem

$$c_t + c_x = bc_{xx}, \tag{28}$$

$$c_0(x) = \begin{cases} \sin^2 \pi(4x-1), & x \in (\tfrac{1}{4}, \tfrac{1}{2}), \\ 0 \quad \text{elsewhere on } [0,1] \end{cases} \tag{29}$$

for $x \in \mathbf{R}$, $t \in [0, T_F]$, with one-periodic boundary conditions.

Note that the solution possesses the conservation property

$$\int_0^1 c(x,t)\,dx = \frac{1}{8} \quad \forall t > 0, \tag{30}$$

and may also be expressed as a Poisson integral:

$$c(x,t) = (4\pi bt)^{-1/2} \int_{-\infty}^{\infty} c_0(s) \exp\left(-\frac{(x-s-t)^2}{4bt}\right) ds, \tag{31}$$

which, for given x, t, may be evaluated efficiently by means of the Romberg algorithm.

2.4.1. A gridfree method. In addition to the methods above, we propose a gridfree method in which the diffusion update for a point is taken by using the values at the neighbouring points directly. The method is shown to be superior to all the above methods over a wide range of parameter values and is conservative for the model problem. A slight modification allows us to keep the timestep bounded from zero at the expense of the conservation property; this may be useful in the case of large or variable diffusion coefficient. To avoid confusion, we call the moving point method of § 2.1 the "Farmer" moving point method; the gridfree method is also a moving point method in the sense that the points carry pointwise information. We now describe this gridfree method.

The solution of (28) reduces to the solution of the heat equation on the set of moving points. Thus, we could estimate a difference approximation to the diffusive term by using

$$c_{xx} = \frac{2}{h_{\alpha+1/2} + h_{\alpha-1/2}} \left(\frac{c_{\alpha+1} - c_\alpha}{h_{\alpha+1/2}} - \frac{c_\alpha - c_{\alpha-1}}{h_{\alpha-1/2}} \right) + O(h_{\alpha+1/2} - h_{\alpha-1/2}), \tag{32}$$

where $\alpha+1$, $\alpha-1$ are the points immediately to the right and left of point α (see Fig. 3).

The update procedure for point α is then

$$C_\alpha^{n+1} = C_\alpha^n + \frac{2b\Delta t}{h_{\alpha+1/2}+h_{\alpha-1/2}}\left(\frac{C_{\alpha+1}^n - C_\alpha^n}{h_{\alpha+1/2}} - \frac{C_\alpha^n - C_{\alpha-1}^n}{h_{\alpha-1/2}}\right). \tag{33}$$

The method is stable for

$$0 \leqq \frac{2b\Delta t}{h_{\alpha+1/2}h_{\alpha-1/2}} \leqq 1 \tag{34}$$

and defining

$$w_\alpha = \frac{h_{\alpha+1/2}+h_{\alpha-1/2}}{2}, \tag{35}$$

it possesses the conservation property

$$\sum_\alpha w_\alpha (C_\alpha^{n+1} - C_\alpha^n) = 0. \tag{36}$$

Note that, for this model problem, each w_α is constant.

In fact, such a scheme can easily be derived using a finite-volume framework; integrate $c_t = bc_{xx}$ over the cell associated with point α (i.e., the region $((x_{\alpha-1}+x_\alpha)/2, (x_\alpha+x_{\alpha+1})/2)$) and use the approximations

$$\int_{V_\alpha} c_t(x)\,dx \approx w_\alpha c_t(x_\alpha), \tag{37}$$

$$\int_{V_\alpha} c_{xx}(x)\,dx = [c_x]_{\partial V_\alpha} \approx \left(\frac{c_{\alpha+1}-c_\alpha}{h_{\alpha+1/2}} - \frac{c_\alpha - c_{\alpha-1}}{h_{\alpha-1/2}}\right), \tag{38}$$

where the t-argument has been suppressed for convenience.

If two points are very close or b is large, the timestep restriction given above may be too severe for practical computation. A modified scheme allows us to keep the timestep bounded above some threshold τ. To do this, instead of using the neighbouring points $\alpha-1$, $\alpha+1$ to perform the diffusion update, we choose points $\alpha+$, $\alpha-$ as close as possible to α subject to the constraints

$$h_{\alpha+} \geqq (2b\tau)^{1/2}, \tag{39}$$

$$h_{\alpha-} \geqq (2b\tau)^{1/2}. \tag{40}$$

Thus points within distance $(2b\tau)^{1/2}$ of point α are not used in the diffusion update for point α. The conservation law is not valid for this scheme, though it is more nearly satisfied as $b\tau$ is decreased; see the numerical experiments in Table 1.

2.4.2. Numerical results. The model problem (28), (29) with $b=0$, is commonly used to compare methods for hyperbolic conservation laws (see e.g. [28]); for ease of

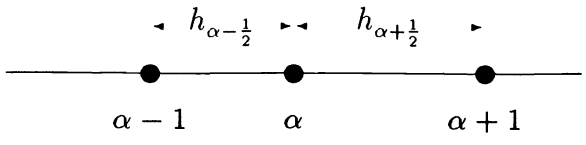


FIG. 3. *The gridfree scheme in one dimension.*

TABLE 1
Maximum absolute errors in density (units of 10^{-3}).

b	0	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Farmer	0	11.4	64.5	44.0	1.22
Gridfree	0	3.55	4.7	2.2	0.58
Particle	0	13.6	92.4	133	4.97

comparison, the space and time steps are taken to be the same as there, namely, $h = 0.02$, $\Delta t = 0.01$, $T_F = 6.12$. In the case of the three Lagrangian methods above, the initial positions are chosen to be of the form

$$x_i(0) = h * (i - 1 + \text{rand}\,(i)), \tag{41}$$

where each rand (i) is a random number chosen from a uniform distribution on [0, 1]. For this problem, different initial distributions of the above form did not qualitatively affect the results, so our results are for a sample distribution. The large mesh parameter H and the cutoff radius ε for the moving point and particle method were both chosen as 0.1. In general, such parameters will be related to (and optimally chosen from) the point/particle and grid spacings. The flexibility allowed in choosing these parameters may be considered to be an advantage for a model problem where the answer is known. For a realistic problem, it is a disadvantage; therefore, no attempt has been made to choose these parameters optimally—rather they are chosen to be "reasonable." The gridfree method we describe does not suffer from the presence of these extra parameters. We use the Gaussian as the cutoff function for the particle method.

The results for the methods for various b are shown in Fig. 4. We plot the solutions *at* the point/particle positions—this avoids the introduction of interpolation errors onto a regular grid which would render the comparison harder. Note that for $b = 0$, all methods are exact at the point positions; these results are not included. In addition, two tables are included; the first gives the maximum absolute error in the density at the point positions (also shown in braces on the graphs) and the second the percentage increase in mass for each method.

The Farmer method used is linear interpolation to a fixed mesh followed by linear interpolation to the stencil points. The gridfree method used here is actually the modified method described above, used to keep the timestep at 0.01.

We note the following points:

(1) The error for each method, regarded as a function of b, starts at zero, passes through a maximum and then decreases. This is because the truncation error depends on both the smoothness of the solution and the diffusion coefficient b. As b increases, the solution becomes smoother, which tends to decrease the error (see Table 1).

(2) The results clearly show the supremacy of the gridfree method over the entire range of parameter values. The modification used to keep the timestep limit away from zero does not seem to affect the conservation property significantly. As expected, the loss in conservation is greater for larger values of $b\tau$, since the points used in the update become further and further apart. The Farmer method is not conservative. Perhaps more sophisticated approximation methods to and from the fixed grid could alleviate this problem; for the vortex computations in [12], an effort is made to ensure conservation of the vorticity during the interpolation process (see Table 2).

(3) It may be argued that the use of a fixed grid is important in order to use Fast Fourier Transforms (FFTs) when solving (for instance) the Poisson equation for the

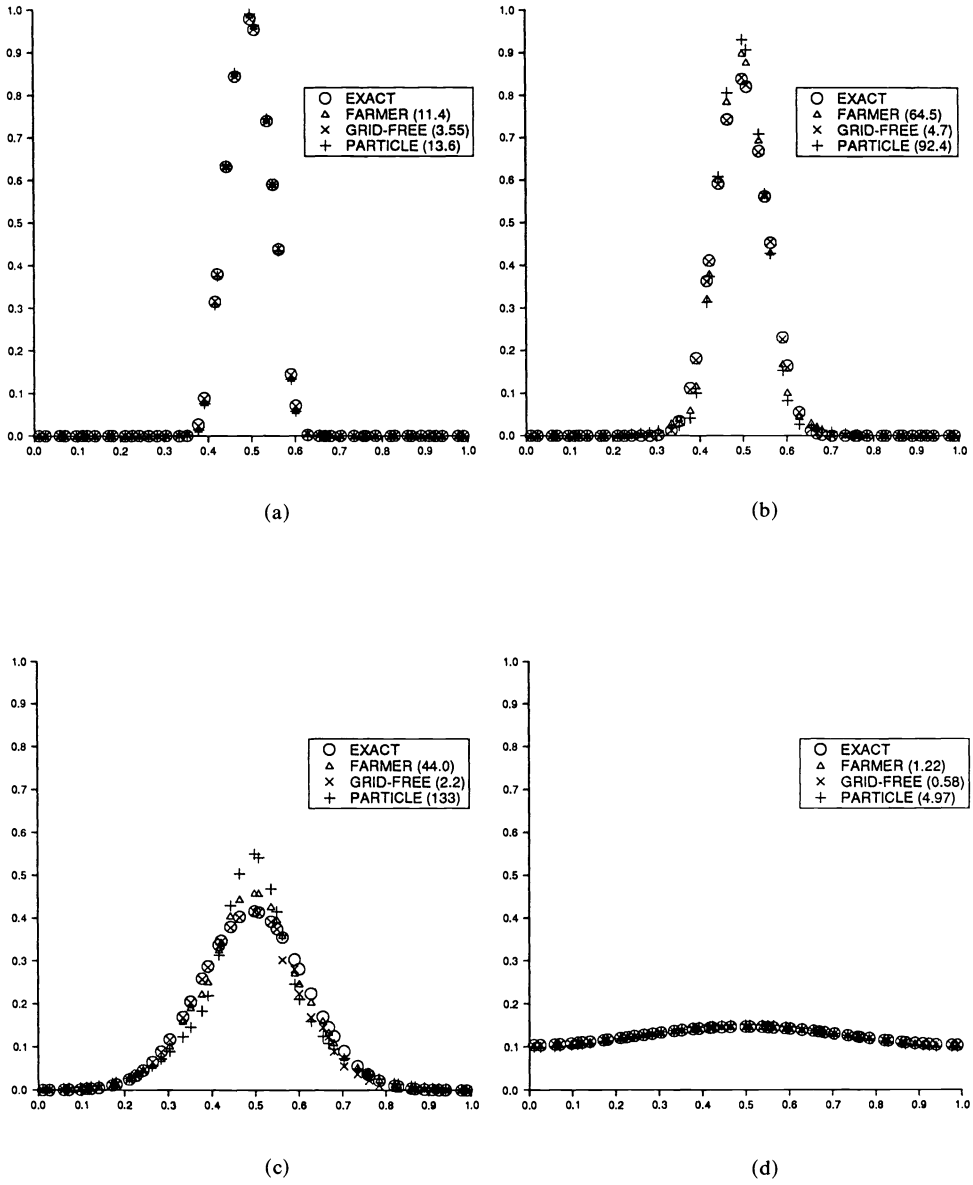


FIG. 4. *Farmer, gridfree, and particle methods.* (a) $b = 10^{-5}$; (b) $b = 10^{-4}$; (c) $b = 10^{-3}$; (d) $b = 10^{-2}$.

TABLE 2
Percentage increase in mass (units of 10^{-5}).

b	0	10^{-5}	10^{-4}	10^{-3}	10^{-2}
Farmer	0	375	3,510	20,000	33,000
Gridfree	0	−0.6	1.2	−7.2	−485
Particle	0	0	0	0	0

stream function in the vorticity-stream function formulation of the Navier–Stokes equations. Our gridfree method would not preclude this; the convection-diffusion equation could still be solved by the gridfree method and the Poisson equation by using FFTs on a fixed mesh.

(4) The particle method is generally far more computationally intensive than either the Farmer or gridfree method, as each particle is required to interact with every other. Our calculations were performed on a MicroVAX and a Convex, but further comparisons of the methods for different computer architectures would be valuable.

3. The two-dimensional gridfree method. The key to the generalisation of the one-dimensional method lies in the use of a finite-volume method on a triangulation. Once a triangulation is available, it is also natural to consider finite-element techniques. We recall that following the positional update of the points, we are effectively required to solve the heat equation. Thus, in § 3.1 we present a brief description of relevant properties of triangulations. In § 3.2 we discuss the numerical solution of the heat equation on triangular grids by finite-volume and finite-element techniques with Dirichlet data. We show that the stability of the method is dependent on the correct choice of triangulation. In § 3.3 we describe how to generate this required triangulation from an arbitrary triangulation; this leads to an efficient numerical method for the two-dimensional convection-diffusion problem.

3.1. A catalogue of triangulation properties. We may need to be able to triangulate a region which may be nonconvex or even multiply connected. Let $\Omega, \Omega_1, \cdots, \Omega_I$ be closed, bounded, simply connected polygonal regions of $\mathbf{R}^2$ with the properties:

(1) $\Omega_i \subset \Omega$ for $i = 1, \cdots, I$.

(2) Each Ω_i intersects Ω and the $\{\Omega_j : j = 1, \cdots, I\}$ at common vertices, along complete common edges, or not at all.

Remark. Without loss of generality it is actually possible to demand intersections only at vertices or not at all; simply combine the regions wherever common edges occur; we shall assume that this has been done.

Set

$$\Omega^* = \Omega \setminus \bigcup_{i=1}^{I} \Omega_i^0, \tag{42}$$

where Ω_i^0 denotes the interior of Ω_i. Suppose a finite set of nodes $\mathbf{P}$ is prescribed interior to Ω^*.

DEFINITION 3.1. A set of triangles $\triangle$ is said to be a *triangulation of* $(\mathbf{P}, \Omega^*)$ if and only if

(1) $\bigcup_{\tau \in \triangle} \tau = \Omega^*$.

(2) Any pair of triangles intersects only along a complete edge, at a vertex, or not at all.

Let N denote the number of distinct nodes (including boundary nodes) and N_B the number of boundary nodes counted according to multiplicity of the number of boundaries to which they belong.

LEMMA 3.1. *Every triangulation,* $\triangle$, *of* $(\mathbf{P}, \Omega^*)$ *has*:

(1) $3(N + I - 1) - N_B$ *(distinct) edges,*

(2) $2(N + I - 1) - N_B$ *triangles.*

COROLLARY 3.1. *The average number of neighbours for a vertex is bounded by six. The average tends to six as* $N \to \infty$ *with* $N_B / N \to 0$.

Figure 5 illustrates the above.

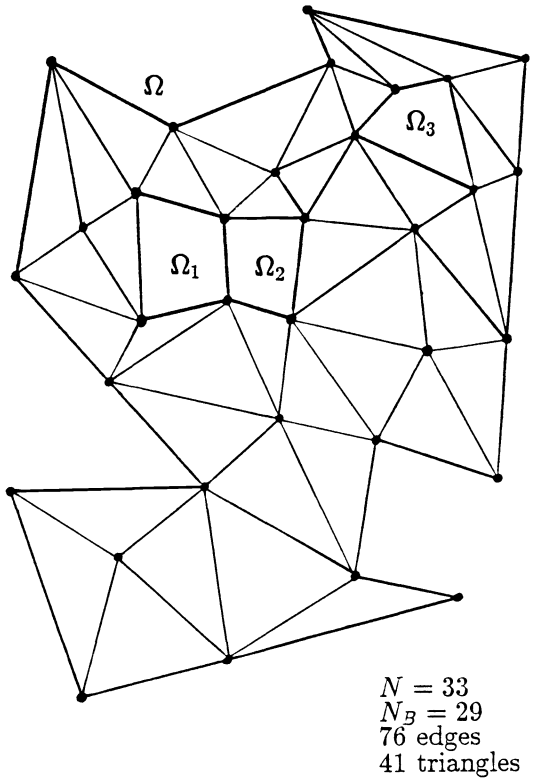


FIG. 5. *Triangulation with specified boundaries and interior nodes.* $N = 33$, $N_B = 29$, 76 *edges*, 41 *triangles.*

Suppose p_i $(i = 1, \cdots, 4)$ are the vertices of a convex quadrilateral, which are ordered counterclockwise; and also for $i = 1, \cdots, 4$ define s_i, t_i as the angles shown in Fig. 6. We shall let $\tau_{i,j,k}$ denote the triangle whose vertices are p_i, p_j, p_k, and $\odot_{i,j,k}$

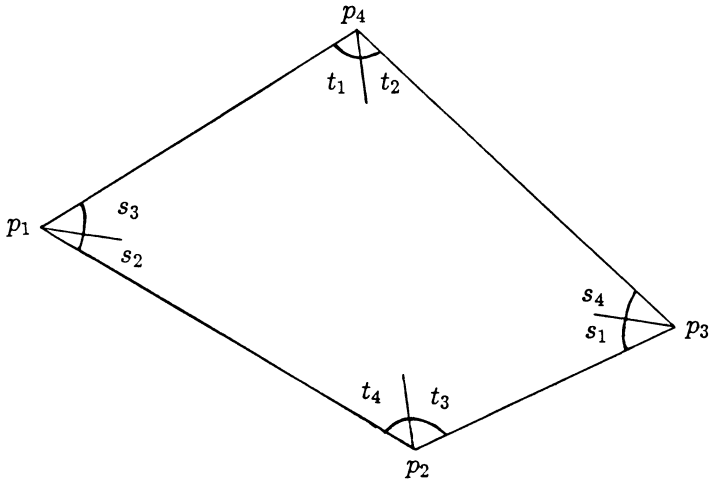


FIG. 6. *Notation for convex quadrilateral.*

the circumscribing circle to this triangle: $r(\tau)$ will denote the circumradius of triangle τ. Then we have Lemma 3.2.

LEMMA 3.2. *The following conditions are equivalent*:

(1) p_4 *lies strictly inside* $\odot_{1,2,3}$ *and* p_2 *strictly inside* $\odot_{1,3,4}$.

(2) $\sum_i t_i > \pi$.

(3) $t_i > s_i$, $i = 1, \cdots, 4$.

(4) *At least one of the following is true*:

(a) $r(\tau_{1,2,4}) < r(\tau_{1,2,3})$ *and* $r(\tau_{2,3,4}) < r(\tau_{1,3,4})$,

(b) $r(\tau_{1,2,4}) < r(\tau_{1,3,4})$ *and* $r(\tau_{2,3,4}) < r(\tau_{1,2,3})$.

(*For a cyclic quadrilateral,* p_2, p_4 *lie on* $\odot_{1,2,3} (= \odot_{1,3,4})$ *and the above inequalities become equalities with* (4)(a), (4)(b) *both true.*)

For a proof of this, see [34]. We deduce Lemma 3.3.

LEMMA 3.3. *The following* <u>local</u> *statements are equivalent*:

(L1) Angle-sum property. *For every interior edge, the sum of the subtended angles is less than or equal to* π.

(L2) Equi-angularity property. *For every interior edge whose associated quadrilateral is convex, replacement of the edge by the other diagonal does not increase the minimum of the six angles in the two triangles which form the quadrilateral.*

(L3) Circumradii property. *For every interior edge whose associated quadrilateral is convex, replacement of the edge by the other diagonal does not decrease any symmetric strictly isotone*[1] *function of the circumradii.*

The replacement of an edge in a convex quadrilateral by the other diagonal will be termed a *flip* of this edge.

DEFINITION 3.2. A triangulation $\triangle$ of $(\mathbf{P}, \Omega^*)$ is said to be locally Delaunay (i.e., is a LDT) if and only if one (and hence all) of (L1), (L2), (L3) is satisfied.

The characterisations (L2), (L3) are presented in [38]. Note that those edges which have angle sum equal to π are interior to some cyclic p-gon and that the interior edges to this p-gon may be connected in several different ways, and the angle sum for each interior edge is equal to π for each connection. Edges in a LDT which have angle sum strictly less than π are said to be "proper"; two LDTs are termed "equivalent" if and only if each proper edge of one occurs as a proper edge of the other and vice versa. The following result may be established by generalisation of an argument in [38] from convex to nonconvex regions.[2]

THEOREM 3.1. *All* LDT*s of* $(\boldsymbol{P}, \Omega^*)$ *are equivalent.*

As a result, we shall below refer to any LDT as *the* LDT. For convex regions, the LDT is known simply as the "Delaunay triangulation" ("DT") and possesses the following properties:

(1) The boundary of the triangulation defines the convex hull of the nodes.

(2) No node lies strictly inside the circumcircle of any triangle.

(3) Nearest neighbours are connected.

There has recently been much interest in the properties and generation of the Delaunay triangulation (see [3], [13], [18], [20], [22], [24], [29], [35]–[38], [41], [44]). Application areas include grid generation, interpolation, and surface drawing. In the context of the relationships between triangulations and numerical methods, see [10], [34], [45].

[1] Monotonic increasing.

[2] This generalisation should actually have been made in that text in order for the result to be valid, because, starting with a convex region, the first inductive step destroys the convexity.

3.2. Solution of the heat equation. We discuss the numerical solution of the heat equation with Dirichlet boundary conditions by finite-volume and finite-element techniques on triangular grids. We shall show that, for mass-lumped linear elements, the methods are equivalent apart from a weighting term, representing an area, associated with each node. The methods may thus be described and analysed in a common framework. In addition, we demonstrate that the stability of the method(s) may be guaranteed (via a maximum principle) if and only if each interior edge of the triangulation satisfies the "angle-sum" property (i.e., the triangulation is a LDT). The application of the results to elliptic equations will be discussed briefly. Much of the work here has been presented in [33].

Thus, we consider the solution of the equation

$$c_t(\mathbf{x}, t) = b\nabla^2 c(\mathbf{x}, t), \tag{43}$$

$$c(\mathbf{x}, 0) = c_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega^*, \tag{44}$$

$$c(\mathbf{x}, t) = g(\mathbf{x}, t), \qquad \mathbf{x} \in \partial\Omega^*, \tag{45}$$

with $t \in [0, T_F]$, b a positive constant, on a triangular grid. We let $\{p_i : i = 1, \cdots, n\}$ denote the nodes of the grid (vertices of the triangulation) labelled in such a way that $\{p_i : i = 1, \cdots, m\}$ are the interior nodes ($m < n$).

3.2.1. The finite-volume method. Given a triangulation, we may construct "cells" to associate with interior points in several ways: drawing the midside perpendiculars to the edges of the triangulation results in the dual or *Voronoi* graph[3]; alternatively the "median-dual" mesh may be constructed recalling that the medians of a triangle intersect its centre of mass (barycentre) (see Fig. 7). The latter will relate to the finite-element solution.

The finite-volume technique is applied on a triangulation by integrating the appropriate equations around the faces of the dual *Voronoi* graph. In our case we obtain

$$V_i \frac{\partial c_i}{\partial t} \approx \int_{V_i} \frac{\partial c}{\partial t} \tag{46}$$

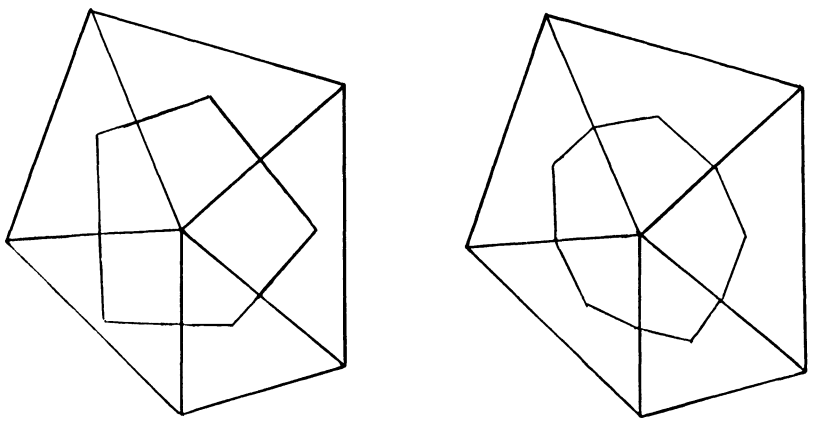


FIG. 7. *The Voronoi and "median-dual" meshes.*

[3] In fact, using this procedure, it is clear how to define an area for each vertex only when the dual graphs associated with adjacent triangles in the triangulation do not cross. This condition is equivalent to the "angle-sum" property. Usually, the Voronoi regions are used to define a triangulation, not vice versa.

$$= b \int_{V_i} \nabla^2 c \tag{47}$$

$$= b \int_{\partial V_i} \frac{\partial c}{\partial n}\, ds \tag{48}$$

$$\approx b \sum_{j \in N(i)} \frac{c_j - c_i}{|\mathbf{x}_j - \mathbf{x}_i|}\, l_{ij}, \tag{49}$$

where l_{ij} is the length of edge i-j (Fig. 8) and $\mathbf{x}_i$, $N(i)$ denote the position and the set of neighbours (i.e., those nodes connected to it in the triangulation) of node i.

Defining

$$K_{ij} = -\frac{bl_{ij}}{|\mathbf{x}_j - \mathbf{x}_i|}, \qquad i \neq j, \tag{50}$$

$$K_{ii} = -\sum_{j \neq i} K_{ij}, \tag{51}$$

we have Lemma 3.4.

LEMMA 3.4.

$$K_{ij} = -\frac{b}{2}\left[\frac{\sin(\theta_1 + \theta_2)}{\sin\theta_1 \sin\theta_2}\right], \qquad i \neq j, \quad i \text{ interior}, \tag{52}$$

where θ_1, θ_2 are the angles subtended by the interior edge (see Fig. 8).

Proof. Let $\bar{\theta}$ denote the angle subtended at node i by the line drawn between the vertices of the dual graph associated with the two triangles on either side of edge i-j (see Fig. 8). Then

$$\bar{\theta} + \theta_1 + \theta_2 = \pi. \tag{53}$$

Elementary geometry yields

$$\frac{l_{ij}}{|\mathbf{x}_i - \mathbf{x}_j|} = \frac{\sin\bar{\theta}}{2\sin\theta_1 \sin\theta_2}, \tag{54}$$

where l_{ij} is the length of this line; the result follows. □

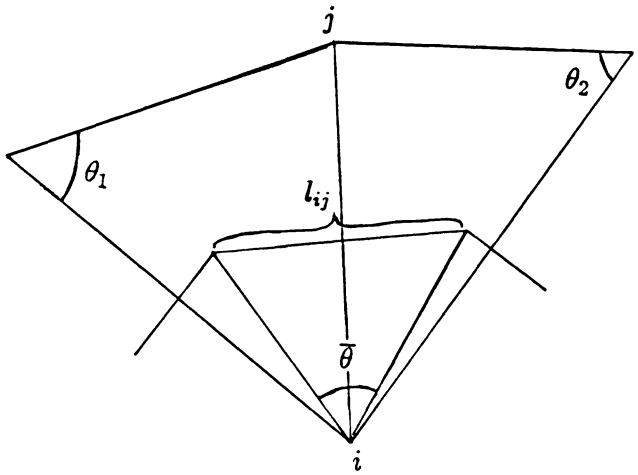


FIG. 8. *The i-j interface and notation for Lemma* 3.4.

We shall call K the *transfer matrix*. Recalling the angle-sum and uniqueness properties of the LDT (§ 3.1), we have Corollary 3.2.

COROLLARY 3.2. *The off-diagonal elements of the finite-volume transfer matrix are nonpositive if and only if the triangulation is a* LDT.

3.2.2. The finite-element method. Define a numerical approximation of (43), (44), (45) by

$$C(\mathbf{x}, t) = \sum_i C_i(t)\phi_i, \tag{55}$$

where ϕ_i is that piecewise linear function on the triangulation satisfying

$$\phi_i(\mathbf{x}_j) = \delta_{ij}. \tag{56}$$

We obtain

$$MC_t + KC = 0, \tag{57}$$

where M is the mass matrix and K the stiffness matrix:

$$M_{ij} = (\phi_i, \phi_j), \tag{58}$$

$$K_{ij} = b(\nabla\phi_i, \nabla\phi_j). \tag{59}$$

THEOREM 3.2. *The finite-element stiffness matrix is identical to the finite-volume transfer matrix.*

(Note. A more general result identifying the finite-volume transfer matrix with the finite-element stiffness matrix has recently been given by Hackbusch [14].)

Proof. It is well known that the element stiffness matrices have entries of the form

$$K_{ij}^e = -\frac{b}{2}\cot\theta, \tag{60}$$

where θ is the angle subtended by side *i-j* in the element. Summing the two components and using Lemma 3.4 gives the desired result. □

A technique often used to stabilise the finite-element equations and facilitate the solution procedure is mass lumping, in which the mass matrix M is replaced by the diagonal matrix W whose entries are the row sums of the consistent finite-element method. It is possible to show (see [42]) that no order of accuracy is lost due to this; the mass-lumped finite-element method is still $O(h^2)$ in appropriate norms. Thus we obtain

$$WC_t + KC = 0, \tag{61}$$

where

$$W_i = \sum_{j=1}^{n} M_{ij} \tag{62}$$

$$= \tfrac{1}{3} \text{ (area of surrounding triangles).} \tag{63}$$

We remark that a scheme equivalent to that obtained by mass-lumping is obtained by following either of the following (see [42]):

(1) Evaluate the mass matrix inner products by quadrature, with the integral over a triangle being approximated by the area multiplied by the mean of the vertex values.

(2) Using piecewise constant trial/test functions (which need not possess a bounded first derivative since they are not differentiated) for the mass-matrix inner products, the support of these functions being the "median-dual" mesh (Fig. 7).

In summary, both the mass-lumped linear finite-element method and the finite volume may be written as

$$DC_t + KC = 0, \tag{64}$$

where K is given by (51), (52) and, in the respective cases,

$$D_i = \begin{cases} \frac{1}{3} \text{ (area of triangles surrounding node } i) \\ \text{area associated with node } i \text{ in dual graph} \end{cases} \tag{65}$$

with the notation as used above. Note that these methods possess the conservation property

$$\sum_{i=1}^{m} (D_i C_i)_t = \text{boundary terms.} \tag{66}$$

Incorporating the boundary conditions, the equations may be written

$$C_t + K^1 C + K^2 G = 0, \tag{67}$$

where K^1 is the square symmetric $m \times m$ matrix corresponding to the interior nodes and K^2 is $m \times (n-m)$ matrix corresponding to the boundary nodes; G is the vector of boundary values. The augmented $m \times n$ matrix $K = (K^1 | K^2)$ has the properties:

$$\sum_{j=1}^{n} K_{ij} = 0, \qquad i = 1, \cdots, m, \tag{68}$$

$$K_{ij} \leqq 0, \qquad i = 1, \cdots, m, \qquad j = 1, \cdots, n, \qquad i \neq j, \tag{69}$$

where (69) is satisfied if and only if the LDT is used (by Corollary 3.2). We now state Theorem 3.3 (see [19], [34]).

THEOREM 3.3. *Properties* (68) *and* (69) *are necessary and sufficient for the system* (67) *to possess a maximum principle.*[4]

Proof. We prove only sufficiency; for the necessity of the condition, see [19]. We note first that the angle-sum property of an LDT ensures that there is always a path from an interior node to a boundary node along edges i-j for which $K_{ij} < 0$. This follows from elementary geometric arguments: it is clear that there must be at least three edges leaving node i for which $K_{ij} < 0$: and by similar arguments we can also show that it is impossible to construct a ring of triangles forming a barrier between an interior node and the boundary by having $K_{ij} = 0$ for all its interior edges.

Now suppose that the maximum M is attained at an interior node i at time t_0 with $0 < t_0 \leqq T_F$. Then at that point $C_t \geqq 0$ so that, from (68),

$$KC \equiv \sum_{j=1}^{n} K_{ij} C_j = \sum_{j=1}^{n} K_{ij} (C_j - M) \leqq 0.$$

By the usual arguments this implies that the maximum is attained at all neighbours j of i for which $K_{ij} < 0$; and because of the existence of a path to the boundary, at some boundary point. □

Remarks. (1) All equivalent LDTs produce identical finite-volume methods but this is not the case for the finite-element method in which the weights change whenever a reconnection of a nonproper edge occurs. This remark favours the finite-volume method, as do our numerical experiments (see [34]) which indicate that it is pointwise more accurate than the finite-element method.

[4] That is, for all boundary and initial data, the maximum value occurs at one of the points on the boundary $\partial\Omega^* \times \{0, T_F\}$.

(2) For variable diffusion coefficients, the angle-sum property is not sufficient to give (69). In this case, each interior edge of the triangulation must have the property that the subtended angles are *individually* less than $\pi/2$ in order that the contribution from each element has the correct sign. This property on the individual angles is too strict to hold in general.

3.3. The FLIP algorithm. We have seen that the LDT is unique (up to the equivalence defined earlier) and uniquely generates finite-volume/element methods with maximum principles. The results quoted above were on a fixed mesh; we wish to use a mesh whose nodal positions vary with time. Since the complete recomputation of the LDT at each time level would be highly computationally intense, we need an efficient algorithm to update it. It is clear that the transported LDT from the last time level (no longer the LDT of the nodes) will be an approximation to the LDT of the new node positions. Thus we seek an algorithm which creates the LDT from an arbitrary given triangulation. From the characterisations (L1), (L2), (L3) of § 3.1, a possible procedure is to iterate over the mesh, performing *flips* of diagonals whenever the local-Delaunay condition is not satisfied. Such an algorithm (called the "**FLIP**" algorithm) is discussed in [38] using the criterion (L2) to determine whether a *flip* is needed. Here we base the algorithm on (L1). This automatically avoids the technicality of checking for nonconvex quadrilaterals and yields a far more efficient method. In fact, it is hard to see how any other criterion could be more efficient, since checking the convexity of a quadrilateral would involve a similar calculation.

The angle-sum criterion has been considered previously in the context of mesh-reconnection (see [9]). However, we believe that our presentation both relates and develops these ideas more fully than has been done previously.

DEFINITION 3.3. For strictly isotone functions, $f: \mathbf{R} \to \mathbf{R}$, define the energy of the triangulation (with respect to f) by

$$E_f(\triangle) := \sum_{\tau \in \triangle} f(r(\tau)). \tag{70}$$

Then, after every *flip* we have $E_f(\triangle') < E_f(\triangle)$, where $\triangle'$ is the triangulation after the *flip*.[5] By the finiteness of the number of possible triangulations, we have Theorem 3.4.

THEOREM 3.4. *Given a triangulation* $\triangle_0$ *the* **FLIP** *algorithm converges to the locally Delaunay triangulation* $\triangle_1$.

COROLLARY 3.3. *The energy functional achieves its minimum at the* LDT.

The problem of finding the LDT may thus be reformulated as that of finding the "minimum-energy" triangulation subject to the constraints that the boundary edges (and vertices) and interior vertex positions are given.

Remarks. (1) Many free-Lagrange methods use "mixed" elements, for instance quadrilaterals and triangles. In this case, the results may be applied by treating all nontriangular elements as internal boundaries.

(2) The explicit calculation of angles is avoided by using the cosine rule and the following results (valid for $0 < A, B < \pi$) and stated in [2]: $\cos A|\cos A| + \cos B|\cos B| < 0 \Leftrightarrow A + B > \pi$.

(3) The order in which edges are considered in the **FLIP** algorithm (together with $\triangle_0$) determines the member of the equivalence class of the LDT that $\triangle_1$ is.

(4) Another criterion which has been used in the literature for the determination of when to perform a *flip* is to choose the shorter of the two diagonals (in convex

[5] The above argument is used in [38]; an alternative proof is possible from a linear ordering of triangulations based on (L2) (see [20]).

quadrilaterals). This criterion (rather than the angle-sum criterion, which they recognise to be numerically superior) is used in [10] since it is stated that the angle-sum criterion does not produce a "nearest neighbour" mesh. Whilst it is certainly true that the diagonal chosen using the angle-sum criterion is not necessarily the shorter, upon iterating over the mesh until convergence, each node *is* connected to its nearest neighbour, as we have seen in § 3.1.

(5) Any triangulation is reasonably close to the LDT in the following sense (established by a simple counting argument): *Given any interior node P of a triangulation, at least three of the edges which have one end-node as P satisfy the angle-sum condition.*

(6) Any subset of a LDT is the LDT of the (appropriately modified) region.

3.3.1. Experimentation. As an example of the application of the algorithm, consider an initial distribution of nodes as shown in Fig. 9, with the interior nodes subject to motion in the incompressible velocity field $\mathbf{a} = (u, v)$, where:

$$u = -y(1-x^2), \tag{71}$$

$$v = +x(1-y^2). \tag{72}$$

The resulting configuration at time 0.5 (using 50 Euler steps of 0.01) is shown in Fig. 10. The **FLIP** algorithm is then applied, resulting in the triangulation shown in Fig. 11 (this is the *DT*, since the region is convex). Only four iterations over the mesh were required. A timestep restriction is generally necessary to ensure that the triangulation property (Definition 3.1) is not destroyed by the transport operator. The allowed timestep is calculated by ensuring that the signed area of each triangle does not change sign during the course of the timestep.

We now consider the CPU time (on a microVAX) required to produce the DT as the number of nodes in the model problem varies (achieved by refining the same basic initial structure). Each configuration was transported using a timestep of 0.2 (chosen to preserve the triangulation property for all cases). In Fig. 12, results are shown for

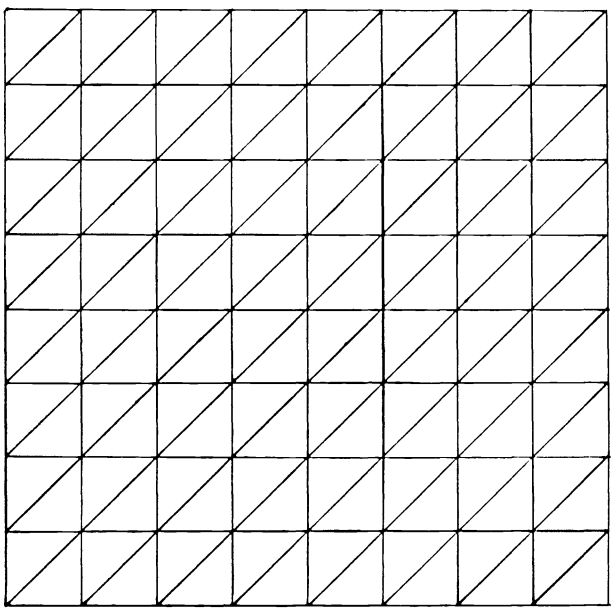


FIG. 9. *Initial triangulation.*

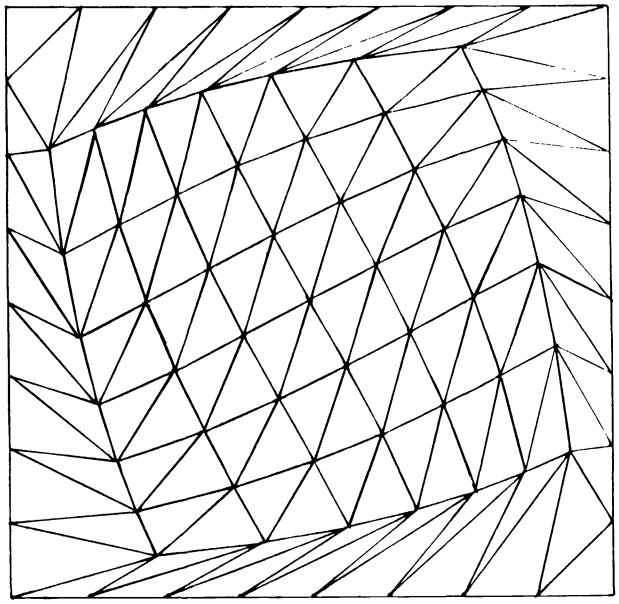


FIG. 10. *Transported triangulation.*

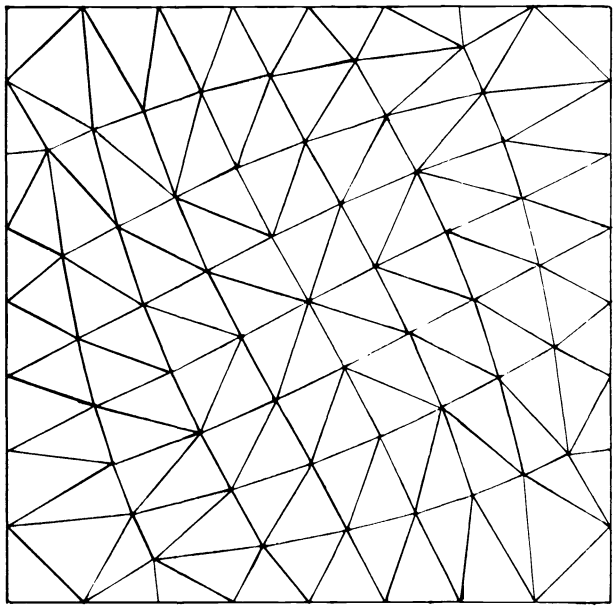


FIG. 11. *Delaunay triangulation generated by* **FLIP** *algorithm.*

the **FLIP** algorithm and for the "circle-check" algorithm.[6] As expected, the circle-check algorithm increases approximately quadratically in the number of nodes (but is known

[6] This is perhaps the simplest algorithm for producing the DT: nodes are added one-by-one and the configuration is modified appropriately if the new node lies in the circumcircle of any existing triangle. Thanks are due to P.K. Sweby (Department of Mathematics, Reading University, U.K.) for donation of a code for this algorithm.

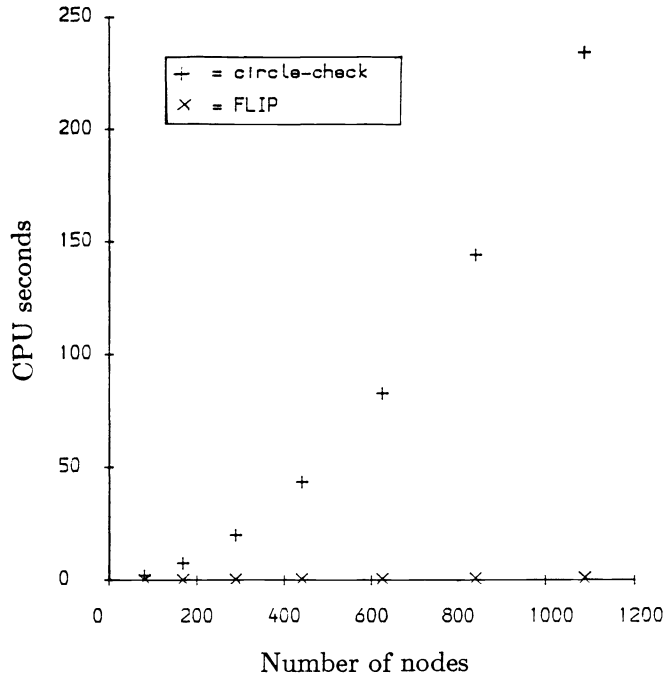


FIG. 12. CPU *times for "circle-check" and* **FLIP** *algorithms.* + *circle check;* × **FLIP.**

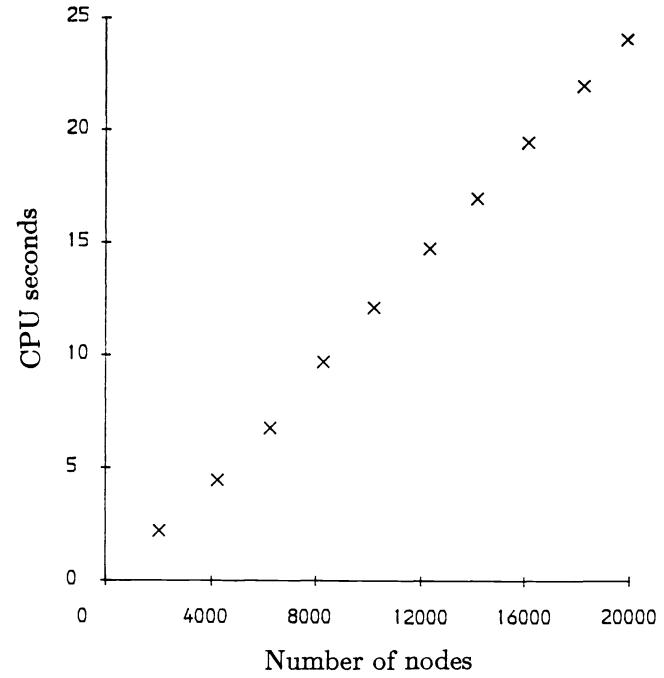


FIG. 13. CPU *times for* **FLIP** *algorithm for many nodes.*

to be fairly efficient for small numbers of nodes). The **FLIP** algorithm is far more efficient than the circle-check over the whole range and the time linear. Figure 13 shows CPU times for the **FLIP** algorithm as the number of nodes increases to $(141)^2$; this also seems to confirm the linearity of the algorithm.

More efficient algorithms than the "circle-check" for the complete computation of the LDT are available [3], [13], [20]. Sibson [38] states that the **FLIP** algorithm is unlikely to be more efficient for producing the DT than these. However, with our context in mind, we believe that the **FLIP** algorithm *is* more efficient; an analogous argument is that between the use of direct or iterative methods for the solution of systems of equations. (The optimal time for the complete triangulation of N nodes is asymptotically $O(N \log N)$ (see [37]) but such algorithms are usually *less* efficient for small numbers of nodes than the "circle-check.")

4. A two-dimensional model problem. We consider here the first of the test problems prepared at the CEGB/IAHR workshop held in 1981 (see Smith and Hutton [40] for a report). This is to solve the convection-diffusion equation

$$c_t + \mathbf{a} \cdot \nabla c = b \nabla^2 c \tag{73}$$

in the region $[-1, 1] \times [0, 1]$, where

$$\mathbf{a} = (2y(1-x^2), -2x(1-y^2)), \tag{74}$$

with boundary conditions

$$c_0(\mathbf{x}) = \begin{cases} 1+\tanh((2x+1)\alpha), & y=0, \quad x \in (-1, 0), \\ 1-\tanh\alpha \begin{cases} x=-1, & y \in [-1, 0], \\ y=+1, & x \in [-1, 1], \\ x=+1, & y \in [0, 1] \end{cases} \end{cases} \tag{75}$$

and $\alpha = 10$. At the outflow $\{y = 0, x \in (0, 1)\}$, Neumann conditions are imposed. More details are found in [27].

The problem is actually set as a steady state one, with a standard 20×10 square mesh specified; we solve it as an unsteady problem but with time-independent boundary conditions implying that steady state is approached as $t \to \infty$.

The solution to this problem was originally obtained before the development of the **FLIP** algorithm and we used the circle-check triangulation routine referred to in § 3.3.1. This routine required that the region to be triangulated be convex. Thus points were deleted as they left the outflow boundary; this meant that the solution profile at the outflow did not give a true representation of the accuracy of the method. This restriction would no longer apply since the development of the **FLIP** algorithm. Although the main test is to produce the outflow profile, an accurate profile does not imply that a method is performing well; some (e.g., finite-difference) methods produce large oscillations in the interior. In addition, a symmetry condition is enforced for some methods in the case where $b = 0$, which leads to unrepresentative accuracy. For these reasons, we present the results along the midline profile.

Moving points (about 200) are placed uniformly in the region initially and inserted uniformly along the inflow at a time interval appropriate to produce a corresponding distribution of points in the coordinate directions. A typical triangulation of the points at a later time is shown in Fig. 14: they are obtained by interpolation onto the standard 20×10 grid. The midline profiles of both the finite-volume approximation on the moving triangulation and the "exact" solution, calculated by using finite differences on a fine mesh, are shown in Fig. 16.

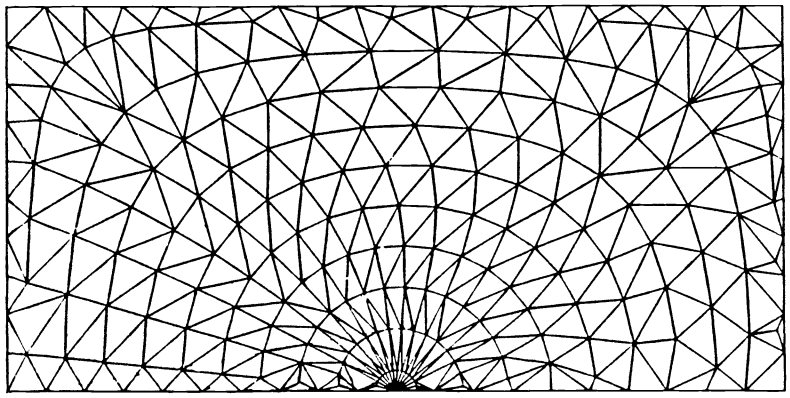


FIG. 14. CEGB *problem: typical triangulation.*

Steady-state surface profiles for $b = 0.002$, 0.01 and 0.1, viewed from the (1, 1) coordinate of the domain, are shown in Fig. 15: they are obtained by interpolation onto the standard 20×10 grid. The midline profiles of both the finite-volume approximation on the moving triangulation and the "exact" solution, calculated by using finite differences on a fine mesh, are shown in Fig. 16.

The finite-volume approximation was obtained using an implicit method for the diffusion update to reduce the number of complete retriangulations required. This has

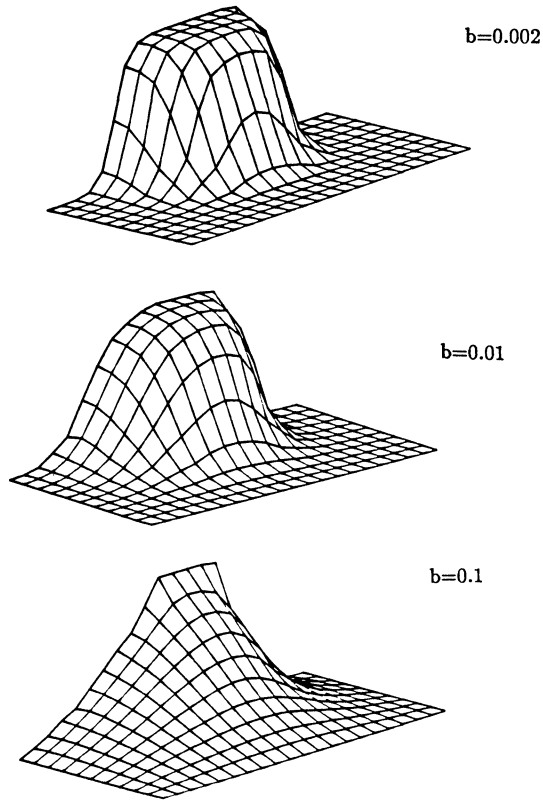


FIG. 15. *Steady state surface profiles for* $b = 0.002$, 0.01, *and* 0.1.

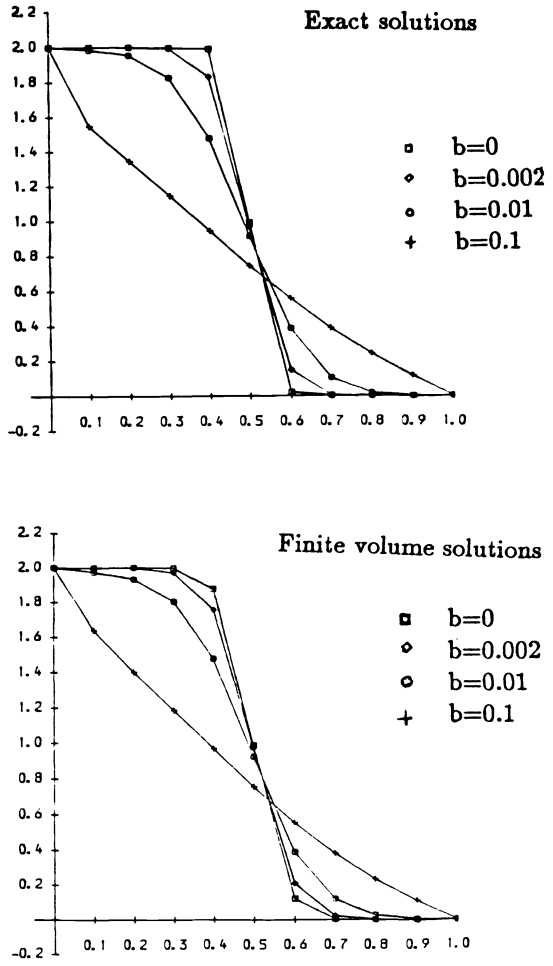


FIG. 16. *Exact and moving point finite-volume solutions.* □ $b=0$; ◇ $b=0.002$; ○ $b=0.01$; + $b=0.1$.

the added advantage that a constant timestep may be used, and hence different methods (which, when explicit, would have different timestep restrictions) may be compared, since they will have the same time accuracy. In addition, there is no problem with the size of the timestep when points become very close, so a modification similar to the one used in § 2.4.1 is not necessary. An interesting point did arise when we chose to use an explicit method; the finite-volume method admitted a diffusive timestep approximately 10 times that of the finite-element method. The diffusive timestep used for the results of Fig. 16 was 0.1 and the point positions were calculated by the classical fourth-order Runge–Kutta method with a timestep of 0.01.

The solutions are very accurate throughout all parameter ranges; the Péclet number (relative to a mesh of comparable spacing to that of the moving points) ranges from 2 to ∞. Indeed, it is hard to quantify what component of the error is due to the interpolation to the grid. Note that even when $b=0$ there are actually three sources of error:

(1) The projection of the inflow data (pointwise values).
(2) The error in the point positions.
(3) The final interpolation error.

5. Summary and concluding remarks. We have presented recently proposed moving point and particle methods for the solution of multidimensional convection-diffusion problems. The basic distinction between the methods is the type of information associated with each Lagrangian point; pointwise (density) information in the moving point method and integral information (mass) in the particle method. The result is that the particle method is conservative, whereas the moving point method is not. Both methods contain parameters, the stencil width and the cutoff radius, which need to be large compared to the point/particle spacing for satisfactory approximation results; their presence is not generally desirable. The methods have been compared for a one-dimensional model problem and we have described an alternative method which uses the values at neighbouring points to perform the diffusive update. This method has superior accuracy and can be reformulated as a finite-volume method on a moving triangulation when generalised to two dimensions.

For the two-dimensional case, we have discussed issues such as the formation of the triangulation and the properties of the resulting numerical methods on it. The equivalence of the mass-lumped linear finite-element and the finite-volume methods, apart from a weighting term associated with each point, has been established. Analysis of the methods also reveals that they possess maximum principles if and only if the triangulation satisfies a local "angle-sum" property. The latter guarantees the (essential) uniqueness of the triangulation, which we term a "locally Delaunay" triangulation. The angle-sum criterion is shown to be preferable to several equivalent local properties and, using it, we have developed and tested an iterative algorithm which generates the locally Delaunay triangulation from an arbitrary given triangulation, relying on its "minimum-energy" property for convergence time increasing approximately linearly with the number of points.

Some issues that have emerged when the method has been applied to a number of model problems in two dimensions include[7]: the importance of the accurate calculation of the positions of the points; the superiority of the finite-volume method; the data structures needed to use the algorithm to generate the triangulation can entail complex coding, particularly at flow boundaries between regions of differing material properties.

We summarise that the principal advantage to be gained by using this method is high accuracy throughout all parameter ranges: the absence of both excess numerical diffusion and over/undershoot is very important for the convection-dominated case; the finite-element/volume accuracy and flexibility is important for the diffusion-dominated case.

It is hoped to apply the method to vortex and other fluid dynamics models in the near future and to explore generalisations to three dimensions. For the latter, we foresee two barriers, namely the optimality of the Delaunay tetrahedralisation and the generalisation of the **FLIP** algorithm.

REFERENCES

[1] C. ANDERSON AND C. GREENGARD, *On vortex methods*, SIAM J. Numer. Anal., 22 (1985), pp. 413–440.

[2] A. H. ARMSTRONG, *A conservative free-Lagrange code*, in Lecture Notes in Physics 238, The Free-Lagrange Method, M. J. Fritts, W. P. Crowley, and H. E. Trease, eds., Springer-Verlag, Berlin, New York, 1985, pp. 255–280.

[7] See [34] for more details.

[3] A. BOWYER, *Computing Dirichlet tessellations*, Comput. J., 24 (1981), pp. 162–166.
[4] W. P. CROWLEY, *Free-Lagrange methods for compressible hydrodynamics in two space dimensions*, in Lecture Notes in Physics 238, The Free-Lagrange Method, M. J. Fritts, W. P. Crowley, and H. E. Trease, eds., Springer-Verlag, Berlin, New York, 1985, pp. 1–21.
[5] ———, *A free-Lagrange method for* 2-D *compressible flows*, Report UCRL-97323, Lawrence Livermore National Laboratory, Livermore, CA, 1987.
[6] J. DONEA, *A Taylor–Galerkin method for convective transport problems*, Internat. J. Numer. Methods Engrg., 20 (1984), pp. 101–119.
[7] J. DOUGLAS AND T. F. RUSSELL, *Numerical methods for convection-dominated diffusion problems based on combining the method of characteristics with finite element or finite difference procedures*, SIAM J. Numer. Anal., 19 (1982), pp. 871–885.
[8] C. L. FARMER, *A moving point method for arbitrary Péclet number multi-dimensional convection-diffusion equations*, IMA J. Numer. Anal., 5 (1985), pp. 465–480.
[9] M. J. FRITTS, W. P. CROWLEY, AND H. E. TREASE, EDS., *The free-Lagrange method*, in Lecture Notes in Physics 238, Springer-Verlag, Berlin, New York, 1985.
[10] M. J. FRITTS AND J. P. BORIS, *The Lagrangian solution of transient problems in hydrodynamics using a triangular mesh*, J. Comput. Phys., 31 (1979), pp. 173–215.
[11] M. J. FRITTS, *Three-dimensional algorithms for grid restructuring in free-Lagrange calculations*, in Lecture Notes in Physics 238, The Free-Lagrange Method, M. J. Fritts, W. P. Crowley, and H. E. Trease, eds., Springer-Verlag, Berlin, New York, 1985, pp. 122–144.
[12] J. M. R. GRAHAM, *Computation of viscous separated flow using a particle method*, in Numerical Methods for Fluid Dynamics III, K. W. Morton and M. J. Baines, eds., Oxford University Press, Oxford, 1988, pp. 310–317.
[13] P. J. GREEN AND R. SIBSON, *Computing Dirichlet tessellations in the plane*, Comput. J., 21 (1978), pp. 168–173.
[14] W. HACKBUSCH, *On first and second order box schemes*, Computing, 41 (1989), pp. 277–296.
[15] D. C. HANDSCOMB, *The Delaunay tessellation in three dimensions*, private communication, 1988.
[16] F. H. HARLOW, *The particle-in-cell computing method for fluid dynamics*, Methods in Computational Physics 3, B. Alder et al., eds., Academic Press, New York, 1964, pp. 319–343.
[17] R. W. HOCKNEY AND J. W. EASTWOOD, *Computer Simulation Using Particles*, McGraw-Hill, New York, 1981.
[18] D. A. H. JACOBS AND P. RUTTER, *Triangulating a set of nodes in two dimensions*, Report RD/L/2179N81, Central Electricity Research Laboratory, Surrey, UK, 1982.
[19] J. F. B. M. KRAAIJEVANGER, *Necessary and sufficient conditions for maximum principles*, private communication, 1988.
[20] D. T. LEE AND J. SCHACHTER, *Two algorithms for constructing a Delaunay triangulation*, Internat. J. Comput. Inform. Sci., 9 (1980), pp. 219–231.
[21] A. LEONARD, *Vortex methods for flow simulation*, J. Comput. Phys., 37 (1980), pp. 289–335.
[22] B. A. LEWIS AND J. S. ROBINSON, *Triangulation of planar regions with applications*, Comput. J., 21 (1978), pp. 324–332.
[23] S. MAS-GALLIC AND P.-A. RAVIART, *Particle approximation of convection-diffusion problems*, Numerical Analysis Report 86013, Université Pierre et Marie Curie, Paris, France, 1986.
[24] D. H. MCLAIN, *Two-dimensional interpolation from random data*, Comput. J., 19 (1976), pp. 178–181. Errata, Comput. J., 19 (1976), p. 384.
[25] J. J. MONAGHAN, *Particle methods for hydrodynamics*, Comput. Phys. Reports, 3 (1985), pp. 71–124.
[26] J. J. MONAGHAN AND R. A. GINGOLD, *Shock simulation by the particle method*, J. Comput. Phys., 52 (1983), pp. 374–389.
[27] K. W. MORTON AND B. W. SCOTNEY, *Petrov–Galerkin methods and diffusion-convection problems in* 2-D, in The Mathematics of Finite Elements and Applications V, J. R. Whiteman, ed., Academic Press, London, 1985, pp. 343–366.
[28] K. W. MORTON AND P. K. SWEBY, *A comparison of flux limited difference methods and characteristic Galerkin methods for shock modelling*, J. Comput. Phys., 73 (1987), pp. 203–230.
[29] J. M. NELSON, *A triangulation algorithm for arbitrary planar domains*, Appl. Math. Modelling, 2 (1978), pp. 151–159.
[30] P.-A. RAVIART, *Particle numerical models in fluid dynamics*, in Numerical Methods for Fluid Dynamics II, K. W. Morton and M. J. Baines, eds., Oxford University Press, Oxford, 1985, pp. 231–254.
[31] ———, *Particle approximation of linear hyperbolic equations of the first order*, Lecture Notes in Mathematics 1066, Springer-Verlag, Berlin, New York, 1984, pp. 142–158.
[32] ———, *An analysis of particle methods*, Lecture Notes in Mathematics 1127, Springer-Verlag, Berlin, New York, 1985.

[33] M. D. REES, *Numerical solution of the heat equation on triangular grids*, Report 88/2, Oxford University Computing Laboratory, Oxford, U.K., 1988.

[34] ———, *Moving point, particle and free-Lagrange methods*, Ph.D. thesis, Oxford University, Oxford, U.K., 1988.

[35] R. RIEDINGER, M. HABAR, P. OELHAFEN, AND H. J. GÜNTHERODT, *About the Delaunay-Voronoi tessellation*, J. Comput. Phys., 74 (1988), pp. 61–72.

[36] C. A. ROGERS, *Packing and Covering*, Cambridge Tracts in Mathematical Physics, Vol. 54, Cambridge University Press, Cambridge, U.K., 1964.

[37] M. I. SHAMOS, *Computational Geometry*, Springer-Verlag, Berlin, New York, 1977.

[38] R. SIBSON, *Locally equiangular triangulations*, Comput. J., 21 (1978), pp. 243–245.

[39] P. A. SMITH AND P. K. STANSBY, *Wave-induced bed flows by a Lagrangian vortex scheme*, J. Comput. Phys., 60 (1985), pp. 489–516.

[40] R. M. SMITH AND A. G. HUTTON, *The numerical treatment of convection—a performance comparison of current methods*, J. Numer. Methods Heat Transfer, 5 (1982), pp. 439–461.

[41] M. TANEMURA, T. OGAWA, AND N. OGITA, *A new algorithm for three-dimensional Voronoi tessellation*, J. Comput. Phys., 51 (1983), pp. 191–207.

[42] V. THOMÉE, *Galerkin finite element methods for parabolic problems*, in Lecture Notes in Mathematics 1054, Springer-Verlag, Berlin, New York, 1984.

[43] H. E. TREASE, *Three dimensional free-Lagrangian hydrodynamics*, in Lecture Notes in Physics 238, The Free-Lagrange Method, M. J. Fritts, W. P. Crowley, and H. E. Trease, eds., Springer-Verlag, Berlin, New York, 1985, pp. 145–157.

[44] D. F. WATSON, *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*, Comput. J., 24 (1981), pp. 167–172.

[45] A. M. WINSLOW, *Numerical solution of quasilinear Poisson equation on a nonuniform triangle mesh*, J. Comput. Phys., 2 (1967), pp. 149–172.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 573-594, May 1991
© 1991 Society for Industrial and Applied Mathematics
006

BALANCING SPACE AND TIME ERRORS IN THE METHOD OF LINES FOR PARABOLIC EQUATIONS*

J. LAWSON†, M. BERZINS†, AND P. M. DEW†

Abstract. A new error control strategy for the time integration in the solution of parabolic equations using the method of lines is presented. The strategy aims to approximately balance the spatial discretisation and time integration errors so that they are of the same order of magnitude, but so that the time integration error is less than the spatial discretisation error. This is achieved by making use of the individual contributions of the spatial discretisation error and the time integration error to an existing estimate of the global error in the numerical solution. The new strategy is presented in the light of this global error indicator and a comparison between the new error control strategy and a similar existing strategy is made. Numerical results are used to illustrate the performance of this strategy.

Key words. parabolic equations, method of lines, spatial and temporal errors

AMS(MOS) subject classifications. 65M20, 65M15

1. Introduction.

The method of lines is widely used in general-purpose software for the integration of time-dependent parabolic and elliptic-parabolic partial differential equations (pde's). Two of the factors influencing the performance of the method of lines are the choice of a spatial discretisation method and the positioning of the spatial discretisation points. The points should be chosen so that the computed solution accurately models the exact solution to the pde, or, in other words, so that the spatial discretisation error is controlled as far as possible. Once the spatial mesh has been chosen, it is desirable to integrate the ordinary differential equation (ode) system in time with just sufficient accuracy so that the temporal error does not significantly corrupt the spatial accuracy. However, in most existing software based on the method of lines, the standard procedure is to control the local time error per step [9] with respect to a supplied accuracy tolerance. It is difficult for the user to select a tolerance which is related to the spatial discretisation error. In addition, controlling the local time error per step does not always guarantee equivalent control of the global time error, although, for a given problem, decreasing the local time error will generally lead to a smaller global time error.

The purpose of this paper is to present a new error control strategy for the time integration which uses the global error estimating algorithm of Berzins [3], as a means of approximately balancing the contributions of the spatial and temporal errors to the global error in the computed solution. This leads to a strategy in which the local time error is controlled in such a way that it is a fraction of the change in the spatial error over each timestep. In this way the error tolerance used in the time integration is varied according to the size of the spatial discretisation error. The analysis of the new strategy shows that the time integration error remains below the spatial discretisation error and that the strategy is a form of local error per unit time step control, in contrast to the more usual local error per timestep control.

This paper is structured in the following way. Section 2 outlines the background theory and notation, given in Berzins [3], needed to describe the global error indicator. This allows the main contribution of the paper to be given in § 3 where the new error

* Received by the editors December 5, 1988; accepted for publication (in revised form) March 12, 1990.
† School of Computer Studies, The University, Leeds LS2 9JT, United Kingdom. The work of the first author was supported by Shell Research Limited and the Science and Engineering Research Council of the United Kingdom through a CASE studentship.

control strategy is fully described. In § 4, this new strategy is compared to that presented by Schönauer, Schnepf, and Raith [16], which is one of the few existing algorithms which estimate and combine the spatial and temporal error estimates. Finally, in §§ 5 and 6, a summary and discussion of the numerical experiments is presented. It is shown that the error control strategy appears to offer an effective method of balancing the spatial and temporal errors in the method of lines.

2. Problem class and method of solution.

2.1. Problem class. The problem class considered here is sufficiently general to illustrate the algorithm for error estimation. The algorithm extends naturally to systems of pde's and to equations in more than one space dimension, providing that the same method of lines approach is employed.

For notational convenience, the class of parabolic pde's to be considered will be written as

$$\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}\, r\left(x, t, u, \frac{\partial u}{\partial x}\right)+f\left(x, t, u, \frac{\partial u}{\partial x}\right), \tag{2.1}$$

defined on $(x, t)\in\Omega=[a, b]\times(0, t_e]$. The boundary conditions are of the form

$$\beta(a, t)r\left(a, t, u(a, t), \frac{\partial u}{\partial x}\right)=\alpha(a, t)u(a, t)-g_a(t) \tag{2.2}$$

and

$$\beta(b, t)r\left(b, t, u(b, t), \frac{\partial u}{\partial x}\right)=\alpha(b, t)u(b, t)-g_b(t) \tag{2.3}$$

for $t\in(0, t_e]$. The initial condition has the form

$$u(x, 0)=u_0(x), \qquad x\in[a, b]. \tag{2.4}$$

It is assumed that the pde defined above is well posed and has a unique continuous solution $u(x, t)$ for all $(x, t)\in\Omega$. The spatial mesh, the points at which the solution is to be computed, is defined by

$$\delta: a=x_1<x_2<\cdots<x_N=b. \tag{2.5}$$

This mesh partitions the interval $[a, b]$ into $N-1$ subintervals of length h_j, where

$$h_j=x_{j+1}-x_j, \qquad h=\max h_j, \qquad j=1, 2, \cdots, N-1. \tag{2.6}$$

The method of lines is used to solve the pde numerically. That is, the pde is discretised in space and the resulting system of time-dependent ode's is solved using existing techniques for solving ode's. Second-order finite-difference methods are commonly used to spatially discretise many of the parabolic equations that arise in practice. Although the error control strategy proposed in this paper can be used in conjunction with any of these schemes, the modified version of the box scheme [11] proposed by Skeel and Berzins [19] is used here. This scheme is particularly convenient as it spatially discretises the pde above into an ode system in normal form. The discretisation method can be written as

$$\frac{\partial U}{\partial t}(x_j, t)=\frac{1}{h_{j-1}+h_j}\left[2(R_{j+1/2}-R_{j-1/2})+(h_{j-1}f_{j-1/2}+h_jf_{j+1/2})\right],$$

where $R_{j+1/2}$ and $R_{j-1/2}$ are defined by

$$R_{j+1/2} = r\left(\frac{x_j + x_{j+1}}{2}, t, \frac{U(x_j, t) + U(x_{j+1}, t)}{2}, \frac{U(x_{j+1}, t) - U(x_j, t)}{h_j}\right),$$

$$R_{j-1/2} = r\left(\frac{x_j + x_{j-1}}{2}, t, \frac{U(x_j, t) + U(x_{j-1}, t)}{2}, \frac{U(x_j, t) - U(x_{j-1}, t)}{h_{j-1}}\right)$$

and $j = 2, \cdots, N-1$. The quantities $f_{j+1/2}$ and $f_{j-1/2}$ are defined similarly and $U(x_j, t)$ is the approximate solution defined by the spatial discretisation method at the point x_j. The boundary condition at $x = a$ is implemented as

$$\beta(a, t)(c_{1i/2})\frac{\partial U}{\partial t}(x_1, t) = 2(\beta(a, t), R_{1i/2} - (\alpha(a, t)u(a, t) - g_a(t)))/h_1 + \beta(a, t)f_{1i/2}$$

and the condition at $x = b$ is treated similarly. When $\beta = 0$, substitution for the solution, u, will lead to a system of time-dependent ode's, although in practice it is easier to leave the boundary conditions as coupled algebraic equations. The initial condition is defined by evaluating the function $u_0(x)$ at the spatial meshpoints

$$U(x_i, 0) = u_0(x_i), \qquad i = 2, \cdots, N-1.$$

For ease of exposition in the rest of the paper, we shall assume that this system of time-dependent ode's can be written as the normal form ode system

$$\dot{\underline{U}} = \underline{F}_N(t, \underline{U}(t)), \tag{2.7}$$

where the N-dimensional vector is defined by

$$\underline{U}(t) = \begin{bmatrix} U(x_1, t) \\ U(x_2, t) \\ \vdots \\ U(x_N, t) \end{bmatrix}.$$

In the case of at least one Dirichlet boundary condition, the system must be modified accordingly. In practice, (2.7) is a stiff or mildly stiff system of time-dependent ode's that can be solved using software based on the backward differentiation formulae, e.g., [5], [9].

2.2. Error control for stiff ode's. In most of the codes available for solving time-dependent ode's, the routines attempt to control the local time integration error in the computed solution with regard to an accuracy tolerance supplied by the user, TOL. The initial value problem to be solved is given by equation (2.7) with the true solution $\{\underline{U}(t_n)\}_{n=0}^{p}$ approximated by $\{\underline{Y}(t_n)\}_{n=0}^{p}$ at a set of discrete times $0 = t_0 < t_1 < \cdots < t_p = t_e$ by a time integration method with requested local error accuracy TOL. After taking a stepsize of size k_n, from $[t_n, t_{n+1}]$, the global time error in the ode solution at time t_{n+1} and for a tolerance TOL, is given by

$$\underline{ge}(t_{n+1}, \text{TOL}) = \underline{U}(t_{n+1}) - \underline{Y}(t_{n+1}). \tag{2.8}$$

The local solution on $[t_n, t_{n+1}]$, $\underline{y}_{n+1}(t, \text{TOL})$, is defined as the solution of the ivp

$$\dot{\underline{y}}_{n+1}(t, \text{TOL}) = \underline{F}_N(t, \underline{y}_{n+1}(t, \text{TOL})), \qquad \underline{y}_{n+1}(t_n, \text{TOL}) = \underline{Y}(t_n), \tag{2.9}$$

and so the local error per step (LEPS) at t_{n+1} can be given by

$$\underline{le}_{n+1}(\text{TOL}) = \underline{Y}(t_{n+1}) - \underline{y}_{n+1}(t_{n+1}, \text{TOL}) \tag{2.10}$$

and the local error per unit per step (LEPUS) is given by $\underline{le}_{n+1}(\text{TOL})/k_n$.

Control over either the LEPS or the LEPUS, with respect to an accuracy tolerance TOL can be achieved by varying the time stepsize and also by using formulae of different orders. In order to balance the spatial and temporal errors, it is desirable that the error control strategy should yield a solution that satisfies tolerance proportionality; that is, there exists a linear relationship between the global time error and the requested accuracy. An error control strategy is said to satisfy tolerance proportionality (Stetter [20]) if the numerical solution is such that if $\underline{ge}(t, \mathrm{TOL})$ is the global time error at time t for an accuracy requirement TOL, then

$$\underline{ge}(t, \mathrm{TOL}) = \underline{\nu}(t)\mathrm{TOL} + o(\mathrm{TOL}), \tag{2.11}$$

where $\underline{\nu}(t)$ is independent of TOL and $\underline{\nu}(t)$ and $\underline{\nu}'(t)$ are bounded on $[0, t_e]$. Here, $o(\mathrm{TOL})$ denotes a term that is numerically negligible compared with terms of order TOL in the same equation. The following theorem shows how tolerance proportionality can be achieved by employing suitable local error control strategies.

THEOREM 1 (Stetter [20]). *Equation* (2.11) *is satisfied for all* t *if and only if the local error* $\underline{le}_{n+1}(\mathrm{TOL})$ *obtained for a fixed accuracy tolerance* TOL *satisfies*

$$\underline{le}_{n+1}(\mathrm{TOL}) = \bar{\underline{\gamma}}(t_n, t_{n+1})k_n\mathrm{TOL} + o(\mathrm{TOL}), \quad t \in [t_n, t_{n+1}], \quad n = 0, \cdots, p, \tag{2.12}$$

where $\bar{\underline{\gamma}}(t, \tau)$ *behaves like an integral mean over* $[t, \tau]$ *of a function that is independent of* TOL *and bounded on* $[0, t_e]$ *and* $k_n = t_{n+1} - t_n$.

From this theorem, it follows that in order to obtain tolerance proportionality, the LEPUS must be controlled rather than the LEPS.

2.3. The global error indicator of Berzins [3]. An efficient strategy which controls the local time error in such a way that the spatial discretisation error dominates can be developed through considering the global error indicator of Berzins [3].

The vector of the values of the global error at the spatial meshpoints, at any time t, is defined by

$$\underline{E}(t) = \underline{u}(t) - \underline{V}(t), \tag{2.13}$$

where $\underline{u}(t)$ is the restriction of the exact pde solution to the mesh δ, i.e.,

$$[\underline{u}(t)]_i = u(x_i, t), \qquad i = 1, \cdots, N.$$

The vector $\underline{E}(t)$ may also be written as a combination of the restriction of the pde spatial discretisation error $\underline{es}(t)$, as defined by

$$\underline{es}(t) = \underline{u}(t) - \underline{U}(t),$$

and the ode global error $\underline{ge}(t, \mathrm{TOL})$, that is,

$$\underline{E}(t) = \underline{es}(t) + \underline{ge}(t, \mathrm{TOL}). \tag{2.14}$$

The solution of the variational equation (see Shampine [17])

$$\dot{\underline{W}} = J\underline{W}, \qquad \underline{W}(t_n) = \underline{ge}(t_n, \mathrm{TOL}) \tag{2.15}$$

(where $J = \partial \underline{F}_N / \partial \underline{U}$) is related to the global time error at the end of the step $\underline{ge}(t_{n+1}, \mathrm{TOL})$ by

$$\underline{ge}(t_{n+1}, \mathrm{TOL}) \approx \underline{W}(t_{n+1}) + \underline{le}_{n+1}(\mathrm{TOL}). \tag{2.16}$$

Experiments, e.g., [7], [10], have shown that the reliability of the global time error estimate can be improved by using a modified version of the local time error, that is, by using

$$\underline{le}_{n+1}(\mathrm{TOL}) = M^{-1}\widehat{\underline{le}}_{n+1}(\mathrm{TOL}), \tag{2.17}$$

where $\underline{\hat{le}}_{n+1}$ is the usual local error estimate when the ode system is in normal form and $M = I - k_n \gamma J$. This is so, even, as in this case, when the ode system is in normal form. The equation used by Berzins [3], for the evolution of $\underline{es}(t)$ is

$$\dot{\underline{es}}(t) = \frac{\partial \underline{F}_N}{\partial \underline{U}} \underline{es}(t) + \underline{TE}(t, \underline{u}(t)), \qquad \underline{es}(0) = \underline{0}, \tag{2.18}$$

where

$$\underline{TE}(t) = \dot{\underline{u}} - \underline{F}_N(t, \underline{u}(t))$$

is the spatial truncation error. Since, in general, the exact solution is unknown the spatial truncation error must be estimated. Berzins [3] shows that if the spatial discretisation error dominates the time integration error, then the estimate of the spatial truncation error found through Richardson extrapolation techniques is reliable. Equations (2.15) and (2.18) can be solved using the theta method and so the global error at t_{n+1} can be given by

$$\underline{E}(t_{n+1}) = \underline{es}(t_{n+1}) + \underline{W}(t_{n+1}) + \underline{le}_{n+1}(\mathrm{TOL}). \tag{2.19}$$

2.4. Computational considerations. When solving the system of ode's (2.7) using backward differentiation formulae, the use of $\theta = 1$ in calculating $\underline{W}(t)$ and $\underline{es}(t)$ leads to the efficient procedure (Berzins [3]) defined by

$$\underline{E}(t_{n+1}) = M^{-1}(\underline{E}(t_n) + k_n \underline{TE}(t_{n+1})) + \underline{le}_{n+1}(\mathrm{TOL}). \tag{2.20}$$

If the more usual form of the local error estimate $\underline{\hat{le}}_{n+1}(\mathrm{TOL})$ is used by the time integrator, then (2.17) can be used to rewrite (2.20) as

$$\underline{E}(t_{n+1}) = M^{-1}[\underline{E}(t_n) + \underline{\hat{le}}_{n+1}(\mathrm{TOL}) + k_n \underline{TE}(t_{n+1})]. \tag{2.21}$$

This means that the global error indicator still incorporates the modified form of the local error estimate given by equation (2.17). Although this procedure is of zero order, it has been found to work in practice. Alternative estimates for $\underline{E}(t_{n+1})$ are given by Berzins [3]; these could be used in place of that given by (2.21).

3. Balancing the space and time errors in the method of lines. The aim in this section is to develop a time integration error control strategy, in which the accuracy tolerance is automatically selected and subsequently adjusted. In order for the method of lines to be used efficiently, the time integration error should not dominate the error due to the spatial discretisation of the pde. However, it is difficult to select a priori an ode tolerance that will ensure that this is so. This is particularly difficult if the LEPS is controlled by the integrator, since the relationship between the ode global error and the chosen accuracy tolerance is not clear. In addition, the spatial accuracy may vary with time, so any fixed tolerance used in the ode integrator is unlikely to be related to the size of the changing spatial error. Thus, an ode tolerance which is related to the spatial discretisation error in some way and which can be modified accordingly as the spatial discretisation error varies is required. Furthermore, the estimation of the spatial truncation error relies on the time integration error being dominated by the spatial discretisation error and Theorem 1 of Stetter [20] suggests that the error control strategy should be based on a form of LEPUS control. The precise form of this error control will be established by considering the relative contributions of the local time error and the spatial discretisation error to the global error in the numerical solution.

3.1. A LEPUS control strategy. The following analysis is based on the assumption that the spatial discretisation error has the form

$$\underline{es}(t) = h^p \bar{\underline{\omega}}(t) + O(h^{p+1}) \tag{3.1a}$$

and that the truncation error of the spatial discretisation method has the form

$$\underline{TE}(t) = h^p \underline{\omega}(t) + O(h^{p+1}), \tag{3.1b}$$

where $\bar{\omega}(t)$ and $\underline{\omega}(t)$ are bounded functions that depend on the derivatives of the true solution to the pde. The modified box scheme [19] used in this paper is second order, that is $p = 2$, but the extension to other values of p is straightforward. From (2.21), it can be seen that the estimate of the global error at t_{n+1} is governed by the global error at the previous time level t_n, the spatial truncation error and the ode local error over the current timestep. Thus, it follows that for the contribution of the spatial error to dominate the contribution of the temporal error to the global error, the time local error must be controlled, in some way, with respect to $k_n \underline{TE}(t_{n+1})$. This immediately suggests that the following local error control strategy be employed in the ode integrator:

$$\|\underline{\hat{le}}_{n+1}(\mathrm{TOL})\| < \varepsilon k_n \|\underline{TE}(t_{n+1})\|, \tag{3.2a}$$

that is,

$$\|\underline{\hat{le}}_{n+1}(TOL)\| < \varepsilon k_n h^2 \|\underline{\omega}(t_{n+1})\|, \tag{3.2b}$$

for h sufficiently small and where $\underline{\omega}(t)$ is a bounded function. In the strategies above, $\|\cdot\|$ is a weighted maximum or averaged L_2 vector norm. In this case the fixed global LEPUS tolerance TOL is effectively given by $\varepsilon \max_t \|\underline{TE}(t)\|$ as from (3.2b) it follows that

$$\|\underline{\hat{le}}_{n+1}(\mathrm{TOL})\| < \varepsilon k_n h^2 \max_t \|\underline{\omega}(t)\|.$$

This expression shows that the LEPUS is controlled and that the local time error is restricted to be less than a fraction ε of the spatial truncation error. It must now be shown that if the local time error is controlled in this way then the spatial discretisation error will dominate the global time error and also that the estimated truncation error $\underline{TE}_{\mathrm{est}}(t)$ will be a suitable approximation to the true truncation error $\underline{TE}_{\mathrm{true}}(t)$.

It is assumed, first of all, that the LEPUS is actually controlled with respect to the true spatial truncation error $\underline{TE}_{\mathrm{true}}(t_{n+1})$, that is,

$$\|\underline{\hat{le}}_{n+1}(\mathrm{TOL})\| < \varepsilon k_n \|\underline{TE}_{\mathrm{true}}(t_{n+1})\|, \tag{3.3}$$

where, from (2.18)

$$\dot{\underline{es}}(t) = J\underline{es}(t) + \underline{TE}_{\mathrm{true}}(t).$$

Since (3.3) is a LEPUS control strategy, we know from Theorem 1 of Stetter [20] that the global time error is proportional to $\varepsilon \max_t \|\underline{TE}_{\mathrm{true}}(t)\|$, at time t, or $\varepsilon h^2 \max_t \|\underline{\omega}(t)\|$, using (3.1b). Therefore, it follows that for a suitably small value of ε, the $O(h^2)$ spatial discretisation error will dominate the $O(\varepsilon h^2)$ global time error. In practice, the effect of using the estimated spatial truncation error, rather than the exact value, when estimating the global error must be considered. Thus the error control strategy at each step is given by

$$\|\underline{\hat{le}}_{n+1}(\mathrm{TOL})\| < \varepsilon k_n \|\underline{TE}_{\mathrm{est}}(t_{n+1})\| \tag{3.4a}$$

and in this case, the LEPUS tolerance is given by $\mathrm{TOL} = \varepsilon \max_t \|\underline{TE}_{\mathrm{est}}(t)\|$.

The restriction of the spatial discretisation error to the coarse mesh

$$\delta^c : a = z_1 < z_2 < \cdots < z_M = b,$$

where $z_i = x_{2i-1}$, $i = 1, \cdots, M$, $M = (N+1)/2$, and N is odd, $\underline{es}^c(t)$, can be given by (Berzins [3])

$$\begin{aligned}\dot{\underline{es}}^c(t) &= J_M \underline{es}^c(t) + \underline{TE}^c_{\text{true}}(t) \\ &= J_M \underline{es}^c(t) + \underline{TE}^c_{\text{est}}(t) + \tfrac{4}{3}(\dot{\underline{ge}}^c(t, \text{TOL}) - J_M \underline{ge}^c(t, \text{TOL})),\end{aligned}$$

which may be written as

$$\dot{\underline{es}}^c(t) - \tfrac{4}{3}\dot{\underline{ge}}^c(t, \text{TOL}) = J_M(\underline{es}^c(t) - \tfrac{4}{3}\underline{ge}^c(t, \text{TOL})) + \underline{TE}^c_{\text{est}}(t). \tag{3.4b}$$

Now, from the local error control strategy (3.4a) and from Theorem 1 of Stetter [20], the global time error is proportional to $\text{TOL} = \varepsilon \max_t \|\underline{TE}_{\text{est}}(t)\|$, that is,

$$\|\underline{ge}(t, \text{TOL})\| = O(\text{TOL}).$$

Therefore, the restriction of $\underline{ge}(t, \text{TOL})$ to the coarse mesh δ^c, $\underline{ge}^c(t, \text{TOL})$, also satisfies

$$\|\underline{ge}^c(t, \text{TOL})\| = O(\text{TOL}).$$

Defining $\text{TOL}^* = \max_t \|\underline{TE}^c_{\text{est}}(t)\|$, where $\underline{TE}^c_{\text{est}}(t)$ is the spatial truncation error defined on the coarse mesh, which has the form, for a sufficiently small value of h,

$$\underline{TE}^c_{\text{est}}(t) = 4h^2 \underline{\omega}^*(t).$$

Hence,

$$\|\underline{ge}^c(t, \text{TOL})\| = O\left(\text{TOL}^* \varepsilon \frac{\omega_r}{4}\right),$$

where

$$\omega_r = \frac{\max_t \|\underline{\omega}(t)\|}{\max_t \|\underline{\omega}^*(t)\|}$$

and so

$$\left\| \frac{4}{3} \underline{ge}^c(t, \text{TOL}) \right\| = O\left(\text{TOL}^* \varepsilon \frac{\omega_r}{3}\right). \tag{3.4c}$$

The solution to equation (3.4b), with initial conditions $\underline{es}^c(t_0) = \underline{ge}^c(t_0) = 0$ is given by (Coppel [8])

$$\underline{es}^c(t) - \frac{4}{3}\underline{ge}^c(t, \text{TOL}) = Y(t) \int_{t_0}^{t} Y^{-1}(s) \underline{TE}^c_{\text{est}}(s)\, ds,$$

where $Y(t)$ is the fundamental matrix (Coppel [8]) for

$$\dot{\underline{es}}^c(t) - \frac{4}{3}\dot{\underline{ge}}^c(t, \text{TOL}) = J_M\left(\underline{es}^c(t) - \frac{4}{3}\underline{ge}^c(t, \text{TOL})\right).$$

This shows that

$$\left\| \underline{es}^c - \frac{4}{3}\underline{ge}^c(t, \text{TOL}) \right\| = O(\text{TOL}^*).$$

A comparison of this and (3.4c) shows that, because of the $\varepsilon/3$ term in (3.4c), the spatial error dominates on the coarse mesh for a suitably small value of ε. Interpolating onto the fine mesh will lead to the same result—that is, the spatial discretisation error will dominate when using the error control strategy (3.4a).

3.2. A new LEPUS control strategy. Instead of comparing the local time error simply with the spatial truncation error, the local time error can be controlled with respect to the contribution of the spatial truncation error and the existing error from previous timesteps to the global error at the end of the next timestep. The following theorem shows that by controlling the local time error so that it is a fraction ε of the growth in the global error (without the local time error) over the interval $k_n = t_{n+1} - t_n$, the global time error will be dominated by the spatial discretisation error.

THEOREM 2. *The local time error control strategy given by*

$$\|\underline{le}_{n+1}(\mathrm{TOL})\| < \varepsilon \|\underline{E}(t_{n+1}) - \underline{E}(t_n) - \underline{le}_{n+1}(\mathrm{TOL})\| \tag{3.5}$$

will, for a suitable constant ε, yield a time integration error which is dominated by the spatial discretisation error. The assumption made is that the spatial discretisation error and the spatial trunction error have the same order of accuracy.

Proof. This theorem uses that given by Stetter [20] to show that the strategy given by (3.5) controls the LEPUS with respect to a tolerance that ensures that the spatial discretisation error dominates.

From (2.18), the growth of the spatial discretisation error over the timestep $[t_n, t_{n+1}]$ is governed, approximately, by

$$\underline{es}(t_{n+1}) - \underline{es}(t_n) = \int_{t_n}^{t_{n+1}} J\underline{v}(t)\, dt + \int_{t_n}^{t_{n+1}} \underline{TE}(t)\, dt, \qquad \underline{v}(t_n) = \underline{es}(t_n)$$

and the growth of the global time error over the same interval is governed by a similar equation (see Stetter [20], Shampine [18])

$$\underline{ge}(t_{n+1}, \mathrm{TOL}) - \underline{ge}(t_n, \mathrm{TOL}) = \underline{le}_{n+1}(\mathrm{TOL}) + \int_{t_n}^{t_{n+1}} J\underline{W}(t)\, dt,$$

where $\underline{W}(t_n) = \underline{ge}(t_n, \mathrm{TOL})$. Thus, the growth of the global error in the method of lines is governed, approximately, by

$$\underline{E}(t_{n+1}) - \underline{E}(t_n) = \underline{le}_{n+1}(\mathrm{TOL}) + \int_{t_n}^{t_{n+1}} J(\underline{v}(t) + \underline{W}(t))\, dt + \int_{t_n}^{t_{n+1}} \underline{TE}(t)\, dt,$$

where $\underline{E}(t_n) = \underline{v}(t_n) + \underline{W}(t_n)$ is the global error at time t_n. Therefore, the local error control strategy (3.5) may be written as

$$\|\underline{le}_{n+1}(\mathrm{TOL})\| \leqq \varepsilon \left\| \int_{t_n}^{t_{n+1}} (J(\underline{v}(t) + \underline{W}(t)) + \underline{TE}(t))\, dt \right\|. \tag{3.6}$$

It shall now be shown, by an inductive proof, that this is a LEPUS control strategy with respect to $\mathrm{TOL} = \varepsilon h^2$. Consider the strategy (3.6) over the first timestep $(t_0, t_1]$, that is,

$$\|\underline{le}_1(\mathrm{TOL})\| \leqq \varepsilon \left\| \int_{t_0}^{t_1} (J(\underline{v}(t) + \underline{W}(t)) + \underline{TE}(t))\, dt \right\|,$$

where $\underline{TE}(t)$ is proportional to h^2 since the spatial discretisation method is second order (assuming that the spatial truncation error and spatial discretisation error have the same rate of convergence). Now $\underline{v}(t)$ is the solution of $\dot{\underline{v}}(t) = J\underline{v}(t) + \underline{TE}(t)$ and so $\underline{v}(t)$ can be given by

$$\begin{aligned} \underline{v}(t) &= Y(t)\, Y^{-1}(t_0)\underline{v}(t_0) + Y(t) \int_{t_0}^{t} Y^{-1}(s)\underline{TE}(s)\, ds \\ &= Y(t) \int_{t_0}^{t} Y^{-1}(s)\underline{TE}(s)\, ds, \qquad t \in (t_0, t_1], \end{aligned}$$

where $Y(t)$ is the fundamental matrix (Coppel [8]) for the equation $\dot{\underline{v}}(t) = J\underline{v}(t)$. Since $\underline{TE}(t)$ is proportional to h^2 it follows that $\underline{v}(t)$, $t \in (t_0, t_1]$, is also proportional to h^2. On the other hand, $\underline{W}(t)$ can be defined in terms of the fundamental matrix $Y(t)$ for the homogeneous linear equation (2.15), $\dot{\underline{W}}(t) = J\underline{W}(t)$ (Coppel [8])

$$\begin{aligned}\underline{W}(t) &= Y(t)\,Y^{-1}(t_0)\,\underline{W}(t_0)\\ &= Y(t)\,Y^{-1}(t_0)\underline{ge}(t_0, \mathrm{TOL})\\ &= 0, \qquad t \in (t_0, t_1].\end{aligned}$$

Thus, the strategy given by (3.6) over the first timestep $(t_0, t_1]$ is equivalent to

$$\begin{aligned}\|\underline{le}_1(\mathrm{TOL})\| &\leqq \varepsilon \left\| \int_{t_0}^{t_1} \left\{ JY(t) \int_{t_0}^{t} Y^{-1}(s)\underline{TE}(s)\,ds + \underline{TE}(t) \right\} dt \right\|\\ &= \mathrm{TOL} \left\| \int_{t_0}^{t_1} \left\{ JY(t) \int_{t_0}^{t} Y^{-1}(s)\underline{\omega}(s)\,ds + \underline{\omega}(t) \right\} dt \right\|,\end{aligned}$$

where $\mathrm{TOL} = \varepsilon h^2$ and $\underline{\omega}(t)$ is given by (3.1b). This can also be expressed as

$$\underline{le}_1(\mathrm{TOL}) = \underline{v}(t_1)k_1\,\mathrm{TOL} + O(\mathrm{TOL}), \tag{3.7}$$

where

$$\underline{v}(t_1) = \frac{1}{k_1} \int_{t_0}^{t_1} \left\{ JY(t) \int_{t_0}^{t} Y^{-1}(s)\underline{\omega}(s)\,ds + \underline{\omega}(t) \right\} dt,$$

and $\underline{v}(t)$ is independent of TOL and is bounded on $(t_0, t_1]$. Thus the LEPUS is controlled with respect to $\mathrm{TOL} = \varepsilon h^2$ on the first timestep, according to Stetter's theorem [20]. This means that, from (3.7), condition (2.11) is satisfied; that is,

$$\underline{ge}(t_1, \mathrm{TOL}) = \bar{\underline{v}}(t_1)\mathrm{TOL} + O(\mathrm{TOL}),$$

where, in this case, $\bar{\underline{v}}(t_1) = k_n\underline{v}(t_1)$.

Now, assume that strategy (3.5) controls the LEPUS with respect to $\mathrm{TOL} = \varepsilon h^2$ for time $t = t_2, \cdots, t_n$. Therefore, at $t = t_n$,

$$\underline{ge}(t_n, \mathrm{TOL}) \propto \varepsilon h^2.$$

The control strategy over the timestep $[t_n, t_{n+1}]$ must now be considered, that is,

$$\|\underline{le}_{n+1}(\mathrm{TOL})\| \leqq \varepsilon \left\| \int_{t_m}^{t_{n+1}} J(\underline{v}(t) + \underline{W}(t)) + \underline{TE}(t)\,dt \right\|.$$

Again, $\underline{v}(t)$ is the solution of $\dot{\underline{v}}(t) = J\underline{v}(t) + \underline{TE}(t)$, $\underline{v}(t_n) = \underline{es}(t_n)$ and so $\underline{v}(t)$ can be given as

$$\underline{v}(t) = Y(t)\,Y^{-1}(t_n)\underline{es}(t_n) + Y(t) \int_{t_n}^{t} Y^{-1}(s)\underline{TE}(s)\,ds, \qquad t \in [t_n, t_{n+1}], \tag{3.8}$$

where $Y(t)$ is the fundamental matrix for the equation $\dot{\underline{v}}(t) = J\underline{v}(t)$ (Coppel [8]). On the other hand, $\underline{W}(t)$ can be defined in terms of the fundamental matrix $Y(t)$ for the homogeneous linear equation (2.15) (Coppel [8]), that is,

$$\underline{W}(t) = Y(t)\,Y^{-1}(t_n)\underline{ge}(t_n, \mathrm{TOL}), \qquad t \in [t_n, t_{n+1}].$$

On setting $C(t) = Y(t)Y^{-1}(t_n)$ and $\underline{ge}(t_n, \text{TOL}) = \underline{\gamma}_n(t_n)\text{TOL}$ where $\text{TOL} = \varepsilon h^2$, strategy (3.6) over the timestep $[t_n, t_{n+1}]$ is equivalent to

$$\|\underline{le}_{n+1}(t, \text{TOL})\|$$

$$\leqq \text{TOL} \left\| \int_{t_n}^{t_{n+1}} \left\{ JC(t)(\bar{\omega}(t_n) + \varepsilon \underline{\gamma}_n(t_n)) + JY(t) \int_{t_n}^{t} Y^{-1}(s)\underline{\omega}(s)\, ds + \underline{\omega}(t) \right\} dt \right\|, \tag{3.9}$$

where $\underline{\omega}(t)$ and $\bar{\omega}(t)$ are defined by equations (3.1b) and (3.1a), respectively. From the inductive hypothesis $\underline{ge}(t_n, \text{TOL}) = O(\varepsilon h^2)$, $\underline{\gamma}_n(t_n)$ must also be bounded above. Also $\underline{ge}(t_n, \text{TOL})$ must be bounded above by $\underline{es}(t_n)$ for a sufficiently small value of ε. Therefore,

$$\underline{le}_{n+1}(\text{TOL}) = \underline{\upsilon}(t_{n+1})k_n\text{TOL} + O(\text{TOL}),$$

where

$$\underline{\upsilon}(t_{n+1}) = \frac{1}{k_n}\left(\int_{t_n}^{t_{n+1}} \left\{ JC(t)(\bar{\omega}(t_n) + \varepsilon \underline{\gamma}_n(t_n)) + JY(t) \int_{t_n}^{t} Y^{-1}(s)\underline{\omega}(s)\, ds + \underline{TE}^*(t) \right\} dt \right)$$

and is bounded on $[t_n, t_{n+1}]$. Although $\underline{\upsilon}(t_{n+1})$ is independent of TOL, it is not independent of ε. However, for a suitably small value of ε, $\varepsilon\underline{\gamma}_n(t_n)$ is dominated by $\underline{es}(t_n)$, from the inductive hypothesis. This means that, according to Stetter's theorem [20], the LEPUS is controlled over the step $[t_n, t_{n+1}]$ and so $\underline{ge}(t_{n+1}, \text{TOL}) \propto \varepsilon h^2$. By induction, it follows that this is true for all timesteps. Since the spatial discretisation error is proportional to h^2, it follows that for a suitably small value of ε, the spatial discretisation error will dominate the global time error.

3.3. An alternative approach. When estimates of the spatial discretisation error can be computed directly (for example, see Babuška and Rheinboldt [2]) it is possible to employ the control strategy (3.5) in a slightly different form. In this case, the local time error is controlled so that it is a fraction of the change in the spatial discretisation error over the current timestep. The error control strategy has the form

$$\|\underline{le}_{n+1}(\text{TOL})\| < \varepsilon \|\underline{es}(t_{n+1}) - \underline{es}(t_n)\|. \tag{3.10}$$

The equivalent form of (3.6) is given by

$$\|\underline{le}_{n+1}(\text{TOL})\| \leqq \varepsilon \left\| \int_{t_n}^{t_{n+1}} J\underline{\upsilon}(t) + \underline{TE}(t)\, dt \right\|.$$

If the analysis in the theorem given above is applied to this control strategy, then the local time error can also be given by (3.9). In this case, the only difference is that $\underline{\upsilon}(t_{n+1})$ will not depend on $\underline{\gamma}_n(t_n)$ and so will be independent of both ε and TOL. Using the mean value theorem, the strategy (3.10) can be written as

$$\|\underline{le}_{n+1}(\text{TOL})\| < \varepsilon k \left\| \frac{\partial \underline{es}(t)}{\partial t} \right\|$$

for $t \in (t_n, t_{n+1})$. From this, it follows that

$$\|\underline{le}_{n+1}(\text{TOL})\| < \varepsilon k h^2 \max_t \left\| \frac{\partial \bar{\omega}}{\partial t} \right\|$$

and that the LEPUS tolerance, to which the global time error is proportional, is given by

$$\text{TOL} = \varepsilon h^2 \max_t \left\| \frac{\partial \bar{\omega}}{\partial t} \right\|.$$

The advantage of this form of the LEPUS error control is that it enables the spatial and temporal errors within the method of lines to be balanced by making use of *any* estimate of the change in the spatial discretisation error over a timestep. The norm used in the time integration can be chosen so as to reflect the norm in which the spatial discretisation error is estimated.

3.4. Implementation of the new strategy. In practice, the control strategy (3.5) is difficult to apply directly since the term $\underline{E}(t_{n+1})$ is known only through the global error estimator using (2.20). Instead, we can approximate the strategy (3.5), by using equation (2.20), to substitute for $\underline{E}(t_{n+1})$, thus giving

$$\|\underline{le}_{n+1}(\mathrm{TOL})\| < \varepsilon \|M^{-1}(\underline{E}(t_n) + k_n \underline{TE}(t_{n+1})) - \underline{E}(t_n)\|,$$

which, using the definition of the matrix $M = I - k_n \gamma J$, gives

$$\begin{aligned} \|\underline{le}_{n+1}(\mathrm{TOL})\| &< \varepsilon \|M^{-1}(k_n \gamma J \underline{E}(t_n) + k_n \underline{TE}(t_{n+1}))\| \\ &= k_n \varepsilon \|M^{-1}(\gamma J \underline{E}(t_n) + \underline{TE}(t_{n+1}))\|. \end{aligned} \tag{3.11}$$

The expression (3.11) shows the form of the control strategy (3.5) that is used in practice and that the strategy is of the LEPUS form. The expression also shows how the time integration accuracy varies with the spatial truncation error and the overall error. In practice, we need to modify (3.11) to cater for a zero spatial error or for a very large spatial error. This may be done by ensuring that

$$\mathrm{TOL}_{\min} < \|M^{-1}(\gamma J \underline{E}(t_n) + \underline{TE}(t_{n+1}))\| < \mathrm{TOL}_{\max},$$

where $\mathrm{TOL}_{\min}$ is of the order of the machine unit roundoff error and $\mathrm{TOL}_{\max}$ is $O(1)$.

4. Comparison with a similar error control strategy.

4.1. The error control strategy of Schönauer, Schnepf, and Raith [16]. There have been very few attempts, up to the present, to either estimate or control the global error incurred within the method of lines. One attempt is that of Schönauer, Schnepf, and Raith [16] who present variable stepsize/variable order difference methods for the solution of parabolic pde's. The error control strategy used attempts to balance the errors (spatial and temporal for parabolic pde's) according to a prescribed tolerance. An estimate of the final error is also presented.

We have already seen that when using the method of lines to solve parabolic pde's the global error is a combination of the error in the spatial discretisation method and the error due to the integration of the ode's. The assumption used by Schönauer, Schnepf, and Raith [16] is that the space mesh is chosen initially and remains fixed throughout the calculation. This means that the error control strategy presented consists of designating the spatial error to be a *key error term* to which the time error must be adapted in order to efficiently use the method of lines. Thus, the next time stepsize (for a pth-order finite-difference method in time) is given by [16] as

$$k_n = k_{n-1}[\tfrac{1}{3} \max(\mathrm{TOL}, \|\underline{TE}(t_n)\|)/\|\underline{D}_t(t_n)\|]^{1/p}, \tag{4.1}$$

where $\underline{TE}(t_n)$ is the spatial truncation error estimate and $\underline{D}_t(t_n)$ is the temporal truncation error estimate at time t_n. This is equivalent to saying that $\underline{D}_t(t)$ is adapted to either the prescribed error tolerance TOL or to the spatial truncation error $\underline{TE}(t)$ with a safety factor of $\frac{1}{3}$.

To compare this strategy with that presented in § 3.1, it is assumed that

$$\mathrm{TOL} < \|\underline{TE}(t_n)\|.$$

Therefore the strategy given by (4.1) is equivalent to

$$\|\underline{D}_t(t_n)\|\left(\frac{k_n}{k_{n-1}}\right)^p \leqq \frac{1}{3}\|\underline{TE}(t_n)\|.$$

Suppose, also, that the time truncation error estimate is of order p, that is,

$$\|\underline{D}_t(t_n)\| \approx Ck_{n-1}^p,$$

where C is a constant which is independent of k_n, and so, the strategy (4.1) aims to ensure that

$$Ck_n^p \leqq \tfrac{1}{3}\|\underline{TE}(t_{n+1})\|, \tag{4.2}$$

or, equivalently, assuming that C does not vary rapidly over the step, it seeks to control the time truncation error $\underline{D}_t(t)$ so that

$$\|\underline{D}_t(t_{n+1})\| \leqq \tfrac{1}{3}\|\underline{TE}(t_{n+1})\|. \tag{4.3}$$

This error control strategy is obviously similar to that proposed in § 3.1, (3.2), with $\varepsilon \approx \frac{1}{3}$, providing that we can establish the relationship between $\|\underline{D}_t(t_{n+1})\|$ and $\|\widehat{\underline{le}}_{n+1}(\mathrm{TOL})\|$.

For the ode defined by (2.7) with true solution $\underline{U}$, computed solution $\underline{V}$ and local solution $\underline{y}_n$, the local truncation error of the numerical method used to solve (2.7) is given by

$$\underline{T}_{n+1}(k_n, \underline{U}) = \underline{\Psi}_p(t_n, \underline{U}(t_n))k_n^{p+1} + O(k_n^{p+2}), \tag{4.4}$$

where $\underline{\Psi}_p$ is the principal error function for the pth-order method. On the other hand, the local error at t_{n+1} can also be expressed in terms of the principal error function and the local solution $y_n(t)$ (see (2.10)) as

$$\begin{aligned}\widehat{\underline{le}}_{n+1}(\mathrm{TOL}) &= \underline{\Psi}_p(t_n, \underline{y}_n(t_n))k_n^{p+1} + O(k_n^{p+2}) \\ &= \underline{T}_{n+1}(k_n, \underline{y}_n) + O(k_n^{p+2}).\end{aligned} \tag{4.5}$$

Now, $\underline{D}_t(t_n)$ is actually an estimate of the local truncation error in the time derivative $\dot{\underline{U}}$ and so for an integration method based on the backward differentiation method,

$$\underline{D}_t(t_{n+1}) \approx \frac{1}{\gamma}\underline{\Psi}_p(t_n, \underline{U}(t_n))k_n^p + O(k_n^{p+1}), \tag{4.6}$$

where γ is the leading coefficient of the bdf formula (see Shampine [17]). Since the true solution to the ivp is not available, Schönauer, Schnepf, and Raith [16] estimate $\underline{D}_t(t)$ using the computed solution, and so

$$\|\underline{D}_t(t_{n+1})\| \approx \frac{\|\widehat{\underline{le}}_{n+1}(\mathrm{TOL})\|}{\gamma k_n}. \tag{4.7}$$

Therefore the error control strategy (4.2) is equivalent to

$$\frac{\|\widehat{\underline{le}}_{n+1}(\mathrm{TOL})\|}{\gamma k_n} < \frac{1}{3}\|\underline{TE}(t_{n+1})\|, \tag{4.8}$$

which is the same as that given by (3.2), with $\varepsilon = \gamma/3$, where the values of γ, as given by Shampine [17], vary from one at order one to 60/137 for order five.

In the numerical experiments reported in § 5, it was found that the performance of the error control strategy improved when using the more reliable form of the local error estimate given by (2.17). Thus the strategy becomes

$$\frac{\|\underline{le}_{n+1}(\mathrm{TOL})\|}{\gamma k_n} < \frac{1}{3}\|\underline{TE}(t_{n+1})\|,$$

where the value $\gamma = 1$ was used in the numerical experiments.

4.2. Neumann boundary conditions for the modified box scheme [19]. The error control strategy presented by Schönauer, Schnepf, and Raith [16], represented by (4.8), controls the LEPUS with respect to the spatial truncation error. Thus, the global time error will be $O(h^p)$ as long as the spatial truncation error is $O(h^p)$. Manteuffel and White [13] show that there are many discretisation schemes for which the spatial discretisation error is of a higher order than the spatial truncation error. This analysis applies to the modified version of the box scheme (Skeel and Berzins [19]). When there are Neumann boundary conditions, the spatial truncation error is only $O(h)$, but the spatial discretisation error remains $O(h^2)$. In this case, when the LEPUS is controlled with respect to the spatial truncation error, for example (3.2) and (4.8), the global time error will be $O(h)$ and will dominate the spatial discretisation error, which is $O(h^2)$.[1]

This difficulty does not occur when controlling the local time error according to strategy (3.5), even though Theorem 2 is based upon the assumption that the spatial discretisation and truncation errors are of the same order. This assumption is used when considering the solution of (3.8). However, the solution to this equation describes the continuous growth of the spatial discretisation error, which is approximated in the discrete form by (2.18). Since the analysis of Manteuffel and White [13] can be applied to the modified box scheme [19], the solution to (3.8) is $O(h^2)$, even if the spatial truncation error is $O(h)$ at the boundaries.

The analysis of Manteuffel and White [13] does not fully describe the situation close to the initial time $t = 0$. In this case, an error transient arises from the discontinuity between the zero initial condition and nonzero Neumann boundary conditions for the error defined by (2.18). This transient results in an $O(h)$ spatial discretisation error for the first few timesteps, after which the spatial discretisation error is $O(h^2)$. Experiments have shown that the time t^*, at which the increased order of accuracy is attained, appears to be related to the spatial element size h and the time stepsize k according to

$$t^* \propto \frac{h^2}{k^{1/2}}.$$

This behaviour of the error transient also means that, close to the initial time, the estimate of the spatial truncation error underestimates the true spatial truncation error. Hence, the global error is underestimated for the first few timesteps. This causes a slight inefficiency over the first few steps as the time integration error is unduly restricted by the strategy (3.5). As far as we are aware, this particular error behaviour does not seem to have been noted elsewhere and merits further study.

5. Numerical experiments. The following five problems were used to illustrate the performance of the global error indicator and to show the effectiveness of the LEPUS

[1] Schönauer, Schnepf, and Raith [16] use a one-sided formula with an additional point at the boundaries to ensure that the spatial discretisation and spatial truncation errors are of the same order and so ensure that (4.8) can be used to balance the space and time errors.

control strategy described in § 3. To show the latter, comparisons were made with two other local error control strategies. The five test problems are shown below.

PROBLEM 1. A convection diffusion problem, known as Burgers' equation, which is defined by

$$\frac{\partial u}{\partial t} = v\frac{\partial^2 u}{\partial x^2} - u\frac{\partial u}{\partial x}, \qquad (x, t) \in [0, 1] \times (0, 1],$$

where the value of $v = 0.015$ was used in the experiments. The solution satisfies Dirichlet boundary conditions and initial condition consistent with the analytic solution defined by

$$u(x, t) = \frac{0.1A + 0.5B + C}{A + B + C},$$

where $A = e^{(-0.05(x-0.5+4.95t)/v)}$, $B = e^{(-0.25(x-0.5+0.75t)/v)}$, and $C = e^{(-0.5(x-0.375)/v)}$.

PROBLEM 2. This problem was used by Berzins and Dew [4] to provide an example of a problem with a nonlinear source term and with nonlinear boundary conditions:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} - 2\left(\frac{\partial u}{\partial x}\right)^2 \frac{1}{u} - (2 + 4t^3 x)u^2, \qquad (x, t) \in [0, 1] \times (0, 2],$$

with boundary conditions

$$\frac{\partial u}{\partial x}(0, t) = -u^2 t^4$$

and

$$\frac{\partial u}{\partial x}(1, t) = -u^2(-2 + t^4).$$

The initial condition is consistent with the analytic solution

$$u(x, t) = \frac{1}{2 - x^2 + xt^4}.$$

PROBLEM 3. This problem provides an example of a problem with a nonlinear source term and a travelling wave solution:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + u^2(1 - u), \qquad (x, t) \in [-1, 10] \times (0, 1],$$

with Dirichlet boundary conditions and initial condition consistent with the analytic solution of

$$u(x, t) = \frac{1}{1 + e^{p(x-pt)}},$$

where $p = 0.5\sqrt{2}$.

PROBLEM 4. This problem is the heat equation with Neumann boundary conditions:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \qquad (x, t) \in [0, 1] \times (0, 0.25],$$

with the boundary conditions

$$\frac{\partial u}{\partial x}(x, t) = \pi e^{-\pi^2 t} \cos(\pi x)$$

at $x=0$ and $x=1$. The initial condition is consistent with the analytic solution

$$u(x, t)=\sin(\pi x)\, e^{-\pi^2 t}.$$

PROBLEM 5. The work of Lindberg [12] suggests that controlling the LEPUS when solving stiff problems is inefficient. This stiff problem was included to test the performance of the new error control strategy when solving stiff ode's:

$$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}+(\pi^2-1)u-pu+(p\,e^{-t}+e^{-pt})\sin \pi x, \qquad (x, t)\in[0, 1]\times(0, 1].$$

The Dirichlet boundary conditions and initial condition are consistent with the analytic solution

$$u(x, t)=(e^{-t}+e^{-pt})\sin \pi x.$$

The solution to this problem contains both a slow transient e^{-t}, and a rapid transient e^{-pt} where $p=5000$, characteristic of stiff problems.

5.1. Testing procedure. The test problems were solved using fixed, evenly spaced, space meshes containing 41, 81, 161, and 321 points. The SPGEAR (Adams/bdf methods) module of the SPRINT software [5], with the Linpack banded matrix routines, was used to solve the ode system in time obtained after discretising the pde in space using the method presented by Skeel and Berzins [19]. Three different error control strategies were used within the time integration routines.

(A) The mixed LEPS strategy used in SPRINT [5], that is, controlling the local error $\underline{le}_{n+1}(\mathrm{TOL})$ so that

$$\left\| \frac{\underline{le}_{n+1}(\mathrm{TOL})}{(\mathrm{RTOL}\cdot|\underline{Y}(t)|+\mathrm{ATOL})} \right\| <1. \tag{5.1}$$

(B) The LEPUS strategy presented in § 3, that is, controlling the local error $\underline{le}_{n+1}(\mathrm{TOL})$ so that

$$\|\underline{le}_{n+1}(\mathrm{TOL})\| \leqq k_n \varepsilon \|M^{-1}(\gamma J\underline{E}(t_n)+\underline{TE}(t_{n+1}))\|. \tag{5.2}$$

(C) The LEPUS strategy presented by Schönauer, Schnepf, and Raith [16], that is, controlling the local error $\underline{le}_{n+1}(\mathrm{TOL})$ so that

$$\|\underline{le}_{n+1}(\mathrm{TOL})\| \leqq \tfrac{1}{3} k_n \|\underline{TE}(t_{n+1})\|. \tag{5.3}$$

The use of the error control strategies (5.2) and (5.3) required a slightly modified version of the ode integrator.

The performance of the global error estimate can be measured by defining the error index

$$E_I(t)=\frac{\|\text{Estimated grid errors at time } t\|_\infty}{\|\text{Actual grid errors at time } t\|_\infty}.$$

The error index was calculated at the end of every timestep in the integration. More details about the performance of the error indicator can be found in Berzins [3].

Before comparing the performance of the three local error control strategies, the choice of the parameter ε in strategy (5.2) must be considered. Section 3 suggests that this parameter be chosen in order that the spatial discretisation error dominates the time integration error and that this will be so for some value of $\varepsilon<1$. A second aim is to choose ε in order that the work needed, when employing this error control strategy, be kept to a minimum. Obviously, a large value for ε will require fewer ode timesteps

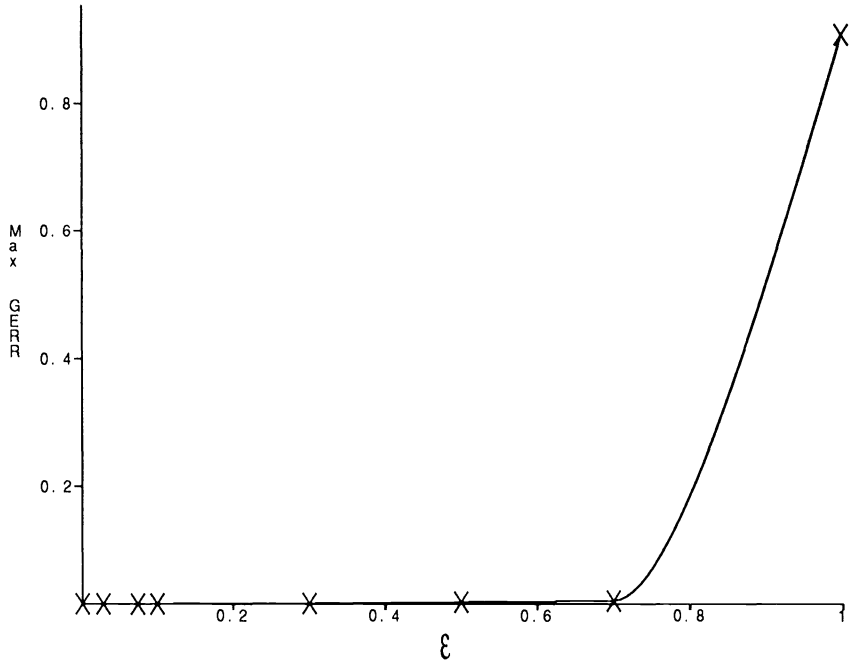


FIG. 1. *Problem* 1. *Graph showing relationship between* Max GERR *and* ε.

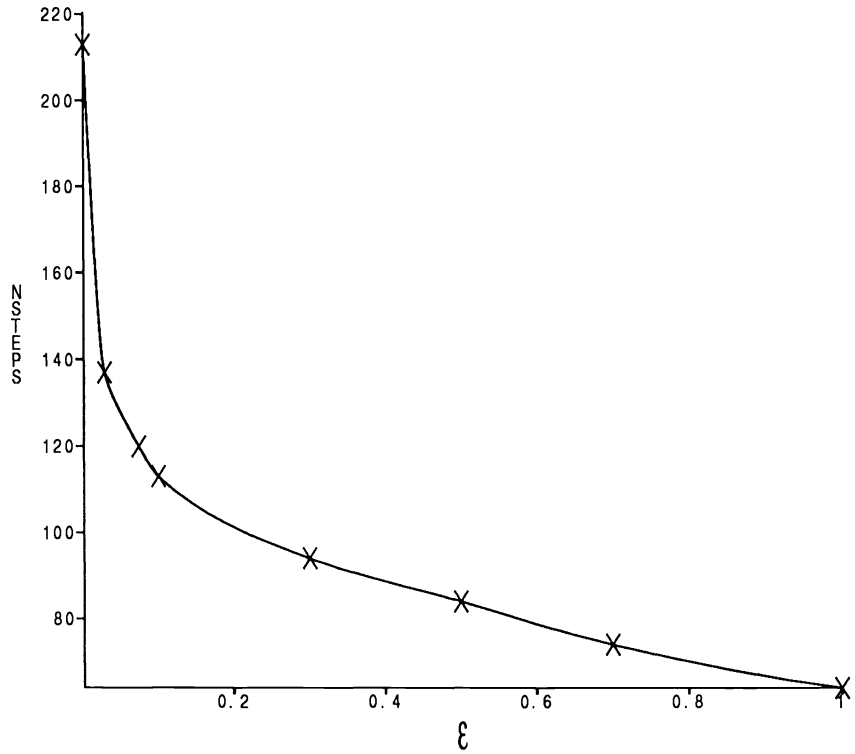


FIG. 2. *Problem* 1. *Graph showing relationship between* NSTEPS *and* ε.

TABLE 1
Results for Problem 1.

METHOD	NPTS	TOL or ε	Max GERR at time $T=0.01$	Max GERR at time $T=0.56$	Max GERR at time $T=1.0$	CPU	NSTEPS	AVG
A	41	0.5D−3	0.49D−2	0.71D−1	0.12D00	3	75	0.49
		0.1D−3	0.48D−2	0.71D−1	0.12D00	4	99	0.51
		0.5D−4	0.48D−2	0.71D−1	0.12D00	5	121	0.53
B	41	0.3	0.51D−2	0.73D−1	0.12D00	4	62	0.51
A	81	0.1D−3	0.13D−2	0.21D−1	0.25D−1	7	125	0.61
		0.5D−4	0.12D−2	0.21D−1	0.25D−1	9	144	0.65
		0.1D−4	0.12D−2	0.21D−1	0.25D−1	12	205	0.69
B	81	0.3	0.13D−2	0.21D−1	0.38D−1	8	94	0.61
A	161	0.5D−4	0.37D−3	0.54D−2	0.91D−2	16	155	0.68
		0.1D−4	0.35D−3	0.53D−2	0.91D−2	20	196	0.72
		0.5D−5	0.35D−3	0.53D−2	0.91D−2	23	217	0.74
B	161	0.3	0.35D−3	0.55D−2	0.92D−2	19	140	0.69
A	321	0.1D−4	0.88D−4	0.13D−2	0.22D−2	47	211	0.77
		0.5D−5	0.87D−4	0.13D−2	0.22D−2	53	241	0.81
		0.1D−5	0.87D−4	0.13D−2	0.22D−2	58	266	0.79
B	321	0.3	0.87D−4	0.13D−2	0.22D−2	47	185	0.73

but will lead to a larger time integration error, while a small value for ε will yield a smaller time integration error but more ode timesteps will be required. Figures 1 and 2 show, respectively, the maximum global errors incurred and the number of ode timesteps required when using values of ε in the range of 0.001 for 1.0 in solving Problem 1, Burgers' equation. These results indicate that we should set $\varepsilon \approx 0.3$, since for $\varepsilon > 0.3$ the maximum global errors increase sharply, while for $\varepsilon < 0.3$ the number of ode timesteps required is rather large. This value happens to be approximately the same as the "safety factor" used by Schönauer, Schnepf, and Raith [16], in the third error control strategy (5.3).

For the comparison of the different strategies, it is necessary to obtain solutions for which the spatial discretisation error dominates. In the case of the LEPS strategy (5.1), the global time error is not directly related to the LEPS tolerance and the spatial discretisation error domination can only be achieved by experimentation with a variety of different tolerances. To show that the results quoted are the most efficient with the spatial error dominating, Tables 1, 2(a), 3, 4(a), and 5 show the results obtained when using the LEPS control strategy (5.1) with the best accuracy tolerance in terms of efficiency, as well as with smaller and larger accuracy tolerances. The CPU *time* quoted, however, does not reflect the experimentation needed to actually find the best tolerance.

Key to Tables 1 to 4.

NPTS is the number of points used in the spatial mesh.

TOL is the value given to ATOL and RTOL used in the strategy (5.1).

Max GERR is the maximum grid error found at the specified output times.

CPU is the amount of CPU time used, measured in seconds.

TABLE 2(a)
Results for Problem 2.

METHOD	NPTS	TOL or ε	Max GERR at time $T=0.01$	Max GERR at time $T=0.89$	Max GERR at time $T=2.0$	CPU	NSTEPS	AVG
A	41	0.1D−4	0.23D−3	0.29D−3	0.76D−2	2.8	71	1.2
		0.5D−5	0.23D−3	0.28D−3	0.76D−2	3.2	83	1.2
		0.1D−5	0.23D−3	0.27D−3	0.76D−2	3.7	99	1.3
B	41	0.3	0.21D−3	0.27D−3	0.77D−2	3.3	56	1.3
A	81	0.5D−5	0.55D−4	0.76D−4	0.20D−2	4.4	75	1.0
		0.1D−5	0.58D−4	0.69D−4	0.19D−2	5.4	94	1.3
		0.5D−6	0.58D−4	0.68D−4	0.19D−2	6.4	110	1.3
B	81	0.3	0.52D−4	0.66D−4	0.20D−2	6.2	76	1.3
A	161	0.1D−5	0.14D−4	0.18D−4	0.49D−3	8.8	85	1.2
		0.5D−6	0.14D−4	0.17D−4	0.49D−3	10.4	99	1.2
		0.1D−6	0.14D−4	0.18D−4	0.50D−3	13.3	132	1.4
B	161	0.3	0.13D−4	0.17D−4	0.50D−3	10.3	80	1.4
A	321	0.5D−6	0.38D−5	0.33D−5	0.12D−3	20.8	94	1.1
		0.1D−6	0.35D−5	0.45D−5	0.12D−3	26.1	122	1.3
		0.5D−7	0.35D−5	0.44D−5	0.12D−3	27.2	127	1.5
B	321	0.3	0.34D−5	0.43D−5	0.13D−3	26.0	102	1.5

TABLE 2(b)
Results for Problem 2.

METHOD	NPTS	Max GERR at Time $T=0.01$	Max GERR at Time $T=0.89$	Max GERR at Time $T=2.0$	CPU	NSTEPS	AVG
C	41	0.20D−3	0.26D−2	0.76D−2	1.9	30	1.4
C	81	0.49D−4	0.10D−2	0.82D−2	3.4	40	1.8
C	161	0.13D−4	0.59D−3	0.79D−2	5.3	37	1.2
C	321	0.26D−5	0.13D−1	0.69D−2	12.8	48	1.6

NSTEPS is the number of timesteps used in the integration of the odes.
AVG is the average value of the error index sampled at the end of every timestep.
Method A is the strategy (5.1).
Method B is the strategy (5.2).
Method C is the strategy (5.3)
The results obtained when using method C are only shown where they differ significantly from the results obtained using method B.

5.2. Discussion of the numerical results. A comparison of the results shown in Tables 1 and 3 shows that when solving Problems 1 and 3, all three error control strategies yield solutions of comparable accuracy. The efficiency of each computation can be compared by observing the CPU times and the number of ode timesteps used. In these cases, it appears that controlling the LEPUS by using either (5.2) or (5.3) is

TABLE 3
Results for Problem 3.

METHOD	NPTS	TOL or ε	Max GERR at time $T=0.01$	Max GERR at time $T=0.56$	Max GERR at time $T=1.0$	CPU	NSTEPS	AVG
A	41	0.1D−5	0.38D−5	0.11D−3	0.17D−3	0.8	19	1.0
		0.5D−6	0.37D−3	0.11D−3	0.17D−3	0.9	21	1.0
		0.1D−6	0.35D−5	0.11D−3	0.17D−3	1.1	27	1.0
B	41	0.3	0.35D−5	0.11D−3	0.17D−3	1.5	25	1.0
A	81	0.5D−6	0.13D−5	0.29D−4	0.43D−4	1.2	20	1.1
		0.1D−6	0.94D−6	0.29D−4	0.43D−4	1.6	26	1.1
		0.5D−7	0.91D−6	0.29D−4	0.43D−4	1.7	29	1.1
B	81	0.3	0.87D−6	0.29D−4	0.42D−4	2.0	25	1.1
A	161	0.1D−6	0.30D−6	0.72D−5	0.11D−4	2.6	24	1.1
		0.5D−7	0.26D−6	0.72D−5	0.11D−4	2.7	26	1.1
		0.1D−7	0.23D−6	0.72D−5	0.11D−4	3.8	33	1.1
B	161	0.3	0.22D−6	0.71D−5	0.11D−4	3.6	29	1.1
A	321	0.5D−7	0.12D−6	0.18D−5	0.27D−5	6.1	28	1.2
		0.1D−7	0.63D−7	0.18D−5	0.27D−5	7.1	33	1.1
		0.5D−8	0.58D−7	0.18D−5	0.27D−5	7.5	35	1.1
B	321	0.3	0.55D−7	0.18D−5	0.26D−5	7.9	33	1.1

as efficient as controlling the LEPS (5.1). However, in the case of the stiff problem, Problem 5, the LEPUS strategies have failed to ensure that the spatial discretisation error dominates the time integration error. The results in Table 5 show that, for this problem, the global error estimate is larger than the actual global error. This means that the tolerance used in the LEPUS strategies is too large to ensure that the spatial discretisation error dominates. One solution to this problem is to allow the parameter ε in (5.2) to vary adaptively. Such a strategy has been presented by Lawson [22], although both this and the global error estimating algorithm require more work.

The results in Tables 2(a) and 4(a) show that by either controlling the LEPS, (5.1), or the LEPUS according to (5.2) leads to solutions of comparable accuracy. However, the results in Tables 2(b) and 4(b) show that the LEPUS strategy given by (5.3) has failed to control the local time error sufficiently so that the spatial discretisation error dominates. The reason for this (see § 4) is that Problems 2 and 4 have Neumann boundary conditions, which means that the spatial truncation error for the modified box scheme [19] is $O(h)$, whereas the spatial discretisation error is $O(h^2)$. This means that when controlling the local time error using (5.3), the global time error will also be $O(h)$. Since, however, the spatial discretisation error is $O(h^2)$, the global time error dominates. This case apart, the results show that the LEPUS strategies, in particular (5.2), yield *automatically*, solutions which are generally of the same order of accuracy as those obtained when controlling the LEPS with tolerances chosen in order that the spatial discretisation error dominates (as it does for the LEPUS strategies). The user no longer has to experiment with different accuracy tolerances to find the solution for which the spatial error is dominant.

TABLE 4(a)
Results for Problem 4.

METHOD	NPTS	TOL or ε	Max GERR at time $T=0.01$	Max GERR at time $T=0.11$	Max GERR at time $T=0.25$	CPU	NSTEPS	AVG
A	41	0.1D−4	0.33D−3	0.47D−3	0.39D−3	1.6	39	1.3
		0.5D−5	0.33D−3	0.46D−3	0.36D−3	1.8	43	1.3
		0.1D−5	0.33D−3	0.46D−3	0.36D−3	2.2	54	1.3
B	41	0.3	0.27D−3	0.34D−3	0.25D−3	2.0	32	1.5
A	81	0.5D−5	0.83D−4	0.12D−3	0.10D−3	2.4	40	1.3
		0.1D−5	0.83D−4	0.12D−3	0.91D−4	3.1	51	1.3
		0.5D−6	0.83D−4	0.12D−3	0.91D−4	3.3	55	1.3
B	81	0.3	0.69D−4	0.88D−4	0.63D−4	3.6	41	1.5
A	161	0.1D−5	0.21D−4	0.29D−4	0.23D−4	4.7	44	1.3
		0.5D−6	0.21D−4	0.29D−4	0.22D−4	5.4	51	1.3
		0.1D−6	0.21D−4	0.29D−4	0.22D−4	6.7	65	1.4
B	161	0.3	0.18D−4	0.23D−4	0.16D−4	6.6	47	1.5
A	321	0.5D−6	0.47D−5	0.67D−5	0.41D−5	10.1	45	1.4
		0.1D−6	0.51D−5	0.69D−5	0.51D−5	12.2	56	1.4
		0.5D−7	0.51D−5	0.69D−5	0.51D−5	14.5	66	1.4
B	321	0.3	0.45D−5	0.59D−5	0.41D−5	14.4	55	1.5

TABLE 4(b)
Results for Problem 4.

METHOD	NPTS	Max GERR at time $T=0.01$	Max GERR at time $T=0.11$	Max GERR at time $T=0.25$	CPU	NSTEPS	AVG
C	41	0.35D−3	0.79D−3	0.29D−3	1.3	18	1.3
C	81	0.83D−4	0.51D−3	0.14D−3	2.0	21	1.4
C	161	0.18D−4	0.17D−3	0.14D−4	3.8	25	2.0
C	321	0.58D−5	0.30D−4	0.38D−4	7.8	27	1.1

6. Conclusions and further developments. Our aim is to develop a fully automatic general-purpose algorithm for the numerical solution of parabolic equations using the method of lines. From practical experience, the local time error control strategy introduced in § 3, equation (3.5), appears to provide a promising starting point for the development of such an algorithm. This control strategy aims to maximise the efficiency of the method of lines by attempting to balance the spatial and temporal errors, although the spatial discretisation error is allowed to dominate in order that the estimation of the spatial truncation error remains valid. By computing the LEPUS accuracy tolerance at each timestep, not only have we enabled the error in the time integration to vary in relation to the spatial discretisation error, we have ensured that the method of lines is being used efficiently. The results for the stiff problem, Problem 5, indicate that although the LEPUS strategy (3.5) is automatic, it may still be more

TABLE 5
Results for Problem 5.

METHOD	NPTS	TOL or ε	Max GERR at time $T=0.01$	Max GERR at time $T=0.56$	Max GERR at time $T=1.0$	CPU	NSTEPS	AVG
A		0.1D−4	0.76D−3	0.51D−3	0.12D−3	7	73	1.7
A	41	0.5D−5	0.76D−3	0.40D−3	0.22D−3	8	78	1.7
A		0.1D−5	0.76D−3	0.44D−3	0.29D−3	24	139	1.4
B	41	0.3	0.76D−3	0.15D−3	0.11D−3	9	86	1.7
A		0.5D−5	0.19D−3	0.67D−4	0.10D−4	12	78	1.7
A	81	0.1D−5	0.19D−3	0.10D−3	0.79D−4	39	139	1.4
A		0.5D−6	0.19D−3	0.11D−3	0.69D−4	15	102	1.8
B	81	0.3	0.19D−3	0.17D−3	0.11D−3	18	120	2.0
A		0.1D−5	0.48D−4	0.21D−4	0.25D−4	69	139	1.4
A	161	0.5D−6	0.48D−4	0.27D−4	0.16D−4	26	102	1.8
A		0.1D−6	0.48D−4	0.27D−4	0.17D−4	32	123	1.8
B	161	0.3	0.48D−4	0.32D−4	0.28D−4	39	145	1.9
A		0.5D−6	0.12D−4	0.59D−5	0.28D−5	51	102	1.8
A	321	0.1D−6	0.12D−4	0.65D−5	0.42D−5	63	123	1.8
A		0.5D−7	0.12D−4	0.67D−5	0.42D−5	73	140	1.8
B	321	0.3	0.12D−4	0.35D−4	0.84D−5	73	153	2.5

efficient to control the LEPS if a suitable value for the tolerance is known a priori. The situation in the method of lines is different from standard ode systems principally because there is an error already present from the spatial discretisation of the pde. The error control approach we have taken reflects this, in contrast to standard local error control where the tolerance is supplied by the user and experimentation is needed to balance the spatial and temporal errors.

Finally, an important area of research lies in the combination of the error control strategy (3.5) with the use of adaptive or moving mesh algorithms. The use of mesh modification techniques becomes especially important if the nature of the solution changes as time elapses. One or both of two approaches is usually adopted; the meshpoints move continuously with the computed solution (e.g., [14]), or the mesh is only adapted at certain discrete times during the computation (e.g., [5], [6]). Another approach is to combine these two ideas and so we have a moving mesh scheme with local refinement (e.g., [1], [15]). All of these methods attempt to place the meshpoints to follow the changing nature of the computed solution, thus attempting to reduce the errors incurred due to the spatial discretisation method. The aim to balance the spatial and temporal errors in the method of lines relates directly to the mesh modification techniques which seek to control the spatial discretisation error. The automatic algorithm developed by Lawson and Berzins ([21] and [22]) uses the pattern recognition technique presented by Bieterman and Babuška [6] to modify the spatial mesh at discrete times in order that the spatial discretisation error be controlled with respect to some tolerance. This means that the user has only to supply the definition of the problem to be solved, an initial coarse mesh, and an error tolerance for the energy

norm of the spatial discretisation error. The algorithm controls the spatial discretisation error according to the tolerance, by suitably modifying the spatial mesh, and then ensures efficient use of the method of lines by using the strategy (3.5) in the time integrator.

Acknowledgment. The helpful comments made by the referees are appreciated.

REFERENCES

[1] S. ADJERID AND J. E. FLAHERTY, *A moving finite element method with error estimation and refinement for one-dimensional time dependent partial equations*, SIAM J. Numer. Anal., 23 (1986), pp. 778–796.

[2] I. BABUŠKA AND W. C. RHEINBOLDT, *A posteriori error estimates for the finite element method*, Internat. J. Numer. Methods Engrg., 12 (1978), pp. 1597–1615.

[3] M. BERZINS, *Global error estimation in the method of lines for parabolic equations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 687–703.

[4] M. BERZINS AND P. M. DEW, *A note on C^0 Chebyshev methods for parabolic* P.D.E.'s, IMA J. Numer. Anal., 7 (1987), pp. 15–37.

[5] M. BERZINS, P. M. DEW, AND R. M. FURZELAND, *Developing* P.D.E. *software using the method of lines and differential algebraic integrators*, Appl. Numer. Math., 5 (1989), pp. 375–397.

[6] M. BIETERMAN AND I. BABUŠKA, *An adaptive method of lines with error control for parabolic equations of the reaction-diffusion type*, J. Comput. Phys., 63 (1986), pp. 33–66.

[7] T. S. CHUA AND P. M. DEW, *The design of a variable step integrator for the simulation of gas transmission networks*, Internat. J. Numer. Methods Engrg., 20 (1984), pp. 1797–1813.

[8] W. A. COPPEL, *Stability and Asymptotic Behaviour of Differential Equations*, Heath, Boston, 1965.

[9] A. C. HINDMARSH, ODEPACK—*a systematized collection of* O.D.E. *solvers*, in Advances in Computer Methods IV, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, New Brunswick, CT, 1983.

[10] T. R. HOPKINS, *Numerical solution of quasi-linear parabolic differential equations*, Ph.D. thesis, University of Liverpool, Liverpool, U.K., 1976; Numer. Math., 49 (1986), pp. 659–683.

[11] H. B. KELLER, *A new difference scheme for parabolic problems*, in Numerical Solution of P.D.E.'s II, B. Hubbard, ed., SYNSPADE, Academic Press, London, 1970.

[12] B. LINDBERG, *Characterisation of optimal stepsize sequences for methods for stiff differential equations*, SIAM J. Numer. Anal., 14 (1977), pp. 859–887.

[13] T. A. MANTEUFFEL AND A. B. WHITE, JR., *The numerical solution of 2nd-order* B.V.P.*s on non-uniform meshes*, Math. Comp., 47 (1986), pp. 511–535.

[14] K. MILLER AND R. N. MILLER, *Moving finite elements.* I, SIAM J. Numer. Anal., 18 (1981), pp. 1019–1032.

[15] L. R. PETZOLD, *Observations on an adaptive moving grid method for 1-dimensional systems of partial differential equations*, Appl. Numer. Math., 3 (1987), pp. 347–360.

[16] W. SCHÖNAUER, E. SCHNEPF, AND K. RAITH, *Experiences in designing* P.D.E. *software with self adaptive variable step size/order difference methods*, Computing, 5 (1984), pp. 227–242.

[17] L. F. SHAMPINE, *Global error estimation for stiff* O.D.E.'*s*, in Proc. Dundee Conference on Numerical Analysis, Lecture Notes in Mathematics, 1066, Springer-Verlag, New York, 1984.

[18] ———, *Tolerance proportionality in* O.D.E. *codes*, SMU Report 87-8, Department of Mathematics, Southern Methodist University, Dallas, TX, 1987.

[19] R. D. SKEEL AND M. BERZINS, *A method for the spatial discretization of parabolic equations in one space variable*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1–32.

[20] H. J. STETTER, *Considerations concerning a theory for* O.D.E. *solvers*, in Numerical Treatment of Differential Equations, R. Bulirsch, R. D. Grigorieff, and J. Schroder, eds., Lecture Notes in Mathematics 631, Springer-Verlag, New York, 1978, pp. 188–200.

[21] J. LAWSON AND M. BERZINS, *Towards an automatic algorithm for the numerical solution of parabolic partial differential equations*, Proc. IMA Conference on Computational O.D.E.s, London, U.K., 1989, to appear.

[22] J. LAWSON, *Towards error control for the numerical solution of parabolic equations*, Ph.D. thesis, School of Computer Studies, University of Leeds, Leeds, U.K., 1989.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 595-606, May 1991

© 1991 Society for Industrial and Applied Mathematics
007

A LEAST SQUARES APPROACH TO LINEAR DISCRIMINANT ANALYSIS*

RICHARD M. CROWNOVER†

Abstract. A new approach to linear discriminant analysis that puts the burden of the computational load on solving a linear least squares problem rather than on solving a generalized eigensystem is presented. The problem being solved is that of reducing the dimension of a vector space in which several classes of objects are to have the best possible separation after the dimension reduction. Two similar algorithms using the least squares approach are developed, one based on the singular value decomposition and the other on orthogonal triangularization (QR factorization). The version using the singular value decomposition is more stable, especially in the nearly rank deficient case. The orthogonal triangularization version is faster, and is the preferred method if updating is anticipated.

Key words. Fisher's linear discriminant, generalized eigensystem, least squares, dimension reduction, recognition

AMS(MOS) subject classifications. primary 62H30; secondary 65F30, 68T10

1. Introduction. An important technique for object recognition and classification in image analysis, speech recognition, and other situations where intelligence is gleaned from comparison with training data, is to first reduce the dimension of the space in which the data is stored. In this scheme of things, object data is represented as vectors in a high-dimensional space, perhaps of dimension in the hundreds or thousands. These vectors, along with the ones to be recognized, are projected into a very low-dimensional space, even as low as one dimension, in such a way that the separation of prototypical classes is preserved. It is then anticipated that recognition algorithms will perform at their best because of the reduced computational load and reduced accumulation of roundoff errors.

Perhaps the best known of these dimension reduction techniques is *Fisher's linear discriminant* method [3], [4]. Fisher's method, described in § 4, requires the solution of a generalized eigensystem problem of the type also known as a generalized singular value decomposition. Early implementations called for computing large cross-product matrices, causing them to be ill-conditioned. They were also computationally expensive. Recent progress in solving generalized singular value problems [1], [2], [7]–[9] offers improved stability, but these methods do not take advantage of the rich structure inherent in the Fisher problem.

We present a new approach that takes advantage of the special structure of Fisher's method, is well conditioned, and reduces the computational load. The idea is to project, in the least squares sense, the vectors in a given class onto a single point in the smaller-dimensional space. First, a preliminary optimization problem must be solved to find the right projection points. Two algorithms are described in this paper. The first, which is more stable in the nearly rank deficient case, uses the singular value decomposition. The second and faster method makes use of orthogonal triangularization (QR factorization). It is the method that should be used if updating the data

* Received by the editors January 11, 1988; accepted for publication (in revised form) March 22, 1990. This research was supported in part by a contract from Emerson Electric Company, Electronics & Space Division, and Air Force Office of Scientific Research grant AFOSR-87-0226.

† Department of Mathematics, University of Missouri, Columbia, Missouri 65211 (MATHRMC@UMCVMB.BITNET).

vectors is anticipated. Like the generalized singular value techniques, these methods avoid potentially disastrous errors from calculating large cross-product matrices.

2. Preliminaries. Suppose we have k data vectors $\{\mathbf{z}_1, \cdots, \mathbf{z}_k\}$ in R^n divided into c classes or clusters, with k_1 vectors in class 1, k_2 vectors in class 2, and so on through class c, having k_c vectors. Thus

$$k = k_1 + \cdots + k_c.$$

Let Z_i be the $n \times k_i$ matrix having columns that are the vectors in class i and let

$$Z = [Z_1, \cdots, Z_c]$$

be the $n \times k$ matrix of all data vectors.

The problem we address is that of projecting the given data from R^n into a smaller-dimensional space R^p ($p < n$; usually $p = c-1$, as we shall see) in such a way that correct identification of class membership can be determined from analysis performed in the smaller-dimensional space.

A naïve way to perform the reduction of dimension is to project into the $(c-1)$-dimensional space generated by the vectors joining the mean vectors for the c classes. This can be achieved as follows:

1) Find $\mathbf{m}_1, \cdots, \mathbf{m}_c$, the mean vectors of classes $1, \cdots, c$, respectively,
2) Find the mean $\mathbf{m}$ of all the data vectors,
3) Find

$$\phi_i = \frac{\mathbf{m}_i - \mathbf{m}}{\|\mathbf{m}_i - \mathbf{m}\|_2}, \qquad i = 1, \cdots, c-1,$$

4) Let $\Phi = [\phi_1, \cdots, \phi_{c-1}]$,
5) Define the projection of the data vectors by

$$\mathbf{z}_j \mapsto \Phi^T \mathbf{z}_j, \qquad j = 1, \cdots, k,$$

or equivalently,

$$Z \mapsto \Phi^T Z.$$

In order to improve upon this naïve method it is appropriate to use certain optimization criteria for determining the projection vectors. The *total scatter* of a set of vectors $\{\mathbf{z}_1, \cdots, \mathbf{z}_k\}$ in R^n having mean $\mathbf{m}$ is

$$\text{total scatter} = \sum_{j=1}^{k} \|\mathbf{z}_j - \mathbf{m}\|_2^2.$$

Let Y be the matrix of data vectors, adjusted so their mean is zero. That is,

$$Y = [\mathbf{z}_1 - \mathbf{m}, \cdots, \mathbf{z}_k - \mathbf{m}],$$

in terms of which it is easy to see that

$$\text{total scatter} = \text{trace}\,(S_T),$$

where

$$S_T = YY^T. \tag{1}$$

S_T is called the total scatter matrix.

After performing the projection into a p-dimensional space (p is not a priori $c-1$), the reduced total scatter becomes

$$\text{reduced total scatter} = \text{trace}\,(\Phi^T YY^T \Phi) = \sum_{i=1}^{p} \phi_i^T (YY^T)\phi_i = \sum_{i=1}^{p} \phi_i^T S_T \phi_i.$$

There are two other scatter matrices of interest to us, the *within-class scatter matrix* S_W and the *between-class scatter matrix* S_B. The within-class scatter is the sum of the total scatters of the separate classes. The between-class scatter is the total scatter that results from clustering all members of classes at their respective means. The matrix S_W is given by

$$S_W = S_{W_1} + \cdots + S_{W_c},$$

where for each $i = 1, \cdots, c$, S_{W_i} is the total scatter matrix for class i alone. If X_i is the adjusted data matrix for class i, then $S_{W_i} = X_i X_i^T$. Let

$$X = [X_1, \cdots, X_c];$$

in terms of X we have

$$S_W = XX^T.$$

The matrix S_B is given by

$$S_B = \sum_{i=1}^{c} k_i(\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T.$$

Define

$$E = [E_1, \cdots, E_c] = \begin{matrix} k_1 \\ k_2 \\ \\ k_c \end{matrix} \begin{pmatrix} \tilde{1} & \tilde{0} & \cdots & \tilde{0} \\ \tilde{0} & \tilde{1} & \cdots & \tilde{0} \\ & & \ddots & \\ \tilde{0} & \tilde{0} & \cdots & \tilde{1} \end{pmatrix},$$

where

$$\tilde{1} = [1, \cdots, 1]^T \quad \text{and} \quad \tilde{0} = [0, \cdots, 0]^T.$$

Let

$$M = [\mathbf{m}_1 - \mathbf{m}, \cdots, \mathbf{m}_c - \mathbf{m}].$$

It is an easy calculation to see that

$$Y = X + ME^T \tag{2}$$

and likewise to see that

$$S_B = ME^T EM^T.$$

Define

$$K = E^T E = \begin{pmatrix} k_1 & 0 & \cdots & 0 \\ 0 & k_2 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & k_c \end{pmatrix},$$

in terms of which we get

$$S_B = MKM^T. \tag{3}$$

Next we show that

$$S_T = S_B + S_W. \tag{4}$$

For this consider

$$S_T = YY^T = XX^T + ME^TX^T + XEM^T + ME^TEM^T = S_W + 0 + 0 + S_B.$$

To see that $ME^TX^T = 0 = XEM^T$ consider

$$XEM^T = [X_1, \cdots, X_c]EM^T = [0, \cdots, 0]M^T = 0,$$

and similarly for ME^TX^T. Another identity,

$$YE = MK, \tag{5}$$

follows from (2) since $YE = XE + ME^TE = 0 + ME^TE = MK$.

An important tool in the analysis that follows is the *singular value decomposition* (SVD) of the matrix Y [5]. The form of this decomposition that we use is written as

$$Y = U\Sigma V^T, \tag{6}$$

where in terms of the rank r of Y,

U is $n \times r$ and has orthonormal columns,

$\Sigma = \text{diag}\,(\sigma_1, \cdots, \sigma_r)$, where $\sigma_1 \geqq \cdots \geqq \sigma_r > 0$ are the nonzero *singular values*,

V is $k \times r$ and has orthonormal columns.

We shall also need the *generalized inverse of* Y^T [5]. The generalized inverse of Y, denoted $Y^\dagger$, is given by

$$Y^\dagger = V\Sigma^{-1}U^T.$$

It follows that

$$Y^{T\dagger} = Y^{\dagger T} = U\Sigma^{-1}V^T.$$

An important identity for our purposes is

$$YY^TY^{\dagger T} = Y. \tag{7}$$

This follows since

$$\begin{aligned} YY^TY^{\dagger T} &= (U\Sigma V^T)(V\Sigma U^T)(U\Sigma^{-1}V^T) \\ &= U\Sigma V^T = Y. \end{aligned}$$

3. The Karhunen–Loeve method. As motivation for the methods that follow, it is appropriate to look at the Karhunen-Loeve [3] method, also known as the principal component method. In this method, the reduced total scatter is maximized subject to the constraints

$$\phi_i^T\phi_i = 1, \qquad i = 1, \cdots, p,$$

where p is the dimension of the projection space, which has not yet been determined. This constrained optimization problem can be solved by the method of Lagrange multipliers in which the objective function is

$$\sum_{i=1}^{p} \phi_i^T S_T \phi_i.$$

The Lagrange function is

$$F(\phi_1, \cdots, \phi_p, \lambda_1, \cdots, \lambda_p) = \sum_{i=1}^{p} [\phi_i^T S_T \phi_i - \lambda_i(\phi_i^T\phi_i - 1)]$$

and the normal equations are

$$\nabla_{\phi_i} F = 2S_T\phi_i - \lambda_i 2\phi_i = 0, \qquad i = 1, \cdots, p,$$

$$\frac{\partial F}{\partial \lambda_i} = \phi_i^T \phi_i - 1 = 0, \qquad i = 1, \cdots, p.$$

Thus for $i = 1, \cdots, p$, the pair λ_i, ϕ_i satisfies the eigensystem relations

$$S_T\phi_i = \lambda_i\phi_i.$$

The function F is maximized, subject to the given constraints, when $\lambda_1, \cdots, \lambda_p$ are the p largest eigenvalues of S_T. Assuming λ_i, ϕ_i are so determined the reduced total scatter becomes

$$\sum_{i=1}^{p} \phi_i^T S_T \phi_i = \sum_{i=1}^{p} \lambda_i \phi_i^T \phi_i = \sum_{i=1}^{p} \lambda_i,$$

whereas the (unreduced) total scatter is

$$\text{trace}\,(S_T) = \sum_{i=1}^{n} \lambda_i.$$

Thus the effectiveness of the reduction at preserving scatter depends on the number p of reduction vectors and on the ratio

$$\left(\sum_{i=1}^{p} \lambda_i\right) \Big/ \left(\sum_{i=1}^{n} \lambda_i\right).$$

4. Fisher's linear discriminant method. It may happen that due to the geometry of the data classes, the projected data classes are not separated when using the Karhunen-Loeve method. This can occur, for instance, when there are just two classes of data, distributed in elongated clusters along two parallel lines. Here the single projection vector will be more or less parallel to the lines of data, and the projections of the data classes may be hopelessly inseparable, thus defeating the purpose of the dimension reduction.

This situation can be remedied by maximizing the reduced total scatter $\sum_{i=1}^{p} \phi_i^T S_T \phi_i$ subject to the constraints that

$$\phi_i^T S_W \phi_i = 1, \qquad i = 1, \cdots, p.$$

The constraints say that the projections of the within-class scatter in each coordinate are bounded by one. Since $S_T = S_B + S_W$,

$$\sum_{i=1}^{p} \phi_i^T S_T \phi_i = \sum_{i=1}^{p} \phi_i^T S_B \phi_i + \sum_{i=1}^{p} \phi_i^T S_W \phi_i$$

$$= \sum_{i=1}^{p} \phi_i^T S_B \phi_i + p.$$

Thus an equivalent formulation of the problem is to maximize the reduced between-class scatter $\sum_{i=1}^{p} \phi_i^T S_B \phi_i$ subject to the same constraints. This is the *Fisher linear discriminant* problem [3], [4].

The Lagrange function for the Fisher problem is

$$F(\phi_1, \cdots, \phi_p, \mu_1, \cdots, \mu_p) = \sum_{i=1}^{p} [\phi_i^T S_B \phi_i - \mu_i(\phi_i^T S_W \phi_i - 1)].$$

Setting $\nabla_{\phi_i} F = 0$ gives

$$S_B \phi_i = \mu_i S_W \phi_i, \qquad i = 1, \cdots, p, \tag{8}$$

which is a generalized eigensystem.

We shall henceforth require that rank $(X) = n$ and hence that S_W $(= XX^T)$ is nonsingular. Under this assumption the generalized eigensystem (8) is equivalent (analytically, but not for computational purposes) to the ordinary eigensystem

$$S_W^{-1} S_B \phi_i = \mu_i \phi_i, \qquad i = 1, \cdots, p.$$

Since $S_B = MKM^T$ and since M has rank $c-1$, it follows that S_B has rank $c-1$. There are $c-1$ nonzero eigenvalues μ_i and the maximum reduced between-class scatter under the given constraints is $\sum_{i=1}^{c-1} \mu_i$. Thus the appropriate value of p, the dimension of the reduced space, is $c-1$.

One way to calculate the generalized eigenvectors of (8) would be to

1) calculate S_B and S_W from their definitions,

2) calculate the Cholesky decomposition of S_W, $S_W = LL^T$, where L is lower triangular and has positive diagonal elements,

3) use the symmetric QR algorithm to compute the spectral decomposition

$$L^{-1} S_B L^{-T} = QDQ^T,$$

where Q has orthonormal columns and D is diagonal, and

4) set $\Phi = L^{-T} Q$.

This method requires about $7n^3$ flops in addition to the $n^2 k$ flops required to compute S_W [5, p. 316]. However, it is inadvisable to compute the scatter matrices S_B and S_W as steps in solving the eigensystem (8). The computation of $S_W = XX^T$, in particular, can be unstable and even if computed exactly, its condition number $\kappa(XX^T)$ can be large since

$$\kappa(XX^T) = \kappa(X)^2.$$

This can result in large relative output errors when solving the generalized eigensystem.

A stable computation of solutions of (8), or equivalently of

$$(MK^{1/2})(MK^{1/2})^T \phi_i = \mu_i XX^T \phi_i, \qquad i = 1, \cdots, c-1, \tag{9}$$

can be carried out using one of the recent treatments of the generalized singular value decomposition [1], [2], [7]-[9]. The method of Van Loan [9], which uses the CS decomposition, tends to be computationally expensive for the Fisher problem. Roughly speaking, it requires about the same amount of work as doing a singular value decomposition for a $2n \times k$ matrix, an amount that is very large compared to using the methods we shall presently propose. Our Algorithm 1, which uses the singular value decomposition, gives high accuracy in the nearly rank deficient situation. The direct generalized singular value decomposition of Paige [7] and Bai [1], [2] comes closer to matching our Algorithm 1 for speed, but requires more storage space. The main computational load for Algorithm 1 is the SVD step, which requires about $7kn^2 + (11/3)k^3$ flops. Our Algorithm 2, which uses the QR factorization for solving the least squares step of the problem, requires about $n^2(k - n/3)$ flops. Numerical simulations comparing these methods follow the discussion of the algorithms.

5. A new approach to the Fisher method. In this section we develop a new approach to the Fisher linear discriminant problem. The main idea is to make an initial reduction in dimension by first projecting the classes onto unit basis vectors in R^c. The projections are calculated using a least squares criterion. The second step of the procedure requires solving a $c \times c$ symmetric eigensystem. Since the number of classes c is usually very small in applications, the computational load of this step is very light compared with the initial dimension reduction of the first step.

THEOREM 1. *Let $\Phi = [\phi_1, \cdots, \phi_{c-1}]$ be the matrix of projection vectors for the Fisher problem. Let $G = Y^{\dagger T}E$, i.e., let G be the solution of the least squares problem*

$$\min \| Y^T G - E \|_F, \tag{10}$$

where $\| \cdot \|_F$ is the Frobenius norm. Then for each $i = 1, \cdots, c-1$, there exists a vector $\mathbf{x}_i \in R^c$ such that

$$\phi_i = G\mathbf{x}_i.$$

Moreover, $\mathbf{x}_1, \cdots, \mathbf{x}_{c-1}$ are the eigenvectors of the $c \times c$ symmetric generalized eigensystem

$$E^T V V^T E \mathbf{x} = \lambda K \mathbf{x} \tag{11}$$

or equivalently,

$$K M^T G \mathbf{x} = \lambda K \mathbf{x}. \tag{12}$$

Proof. We first show that

$$\mathcal{R}(\Phi) = \mathcal{R}(S_W^{-1} M),$$

where $\mathcal{R}$ acting on a matrix denotes the range space of the matrix. Each ϕ_i satisfies $S_B \phi_i = \mu_i S_W \phi_i$ for some nonzero μ_i. Since $S_B = MKM^T$, we have

$$\phi_i = S_W^{-1} \left[\frac{1}{\lambda_i} MKM^T \phi_i \right] \in \mathcal{R}(S_W^{-1} M).$$

Thus $\mathcal{R}(\Phi) \subset \mathcal{R}(S_W^{-1} M)$. The rank of Φ is $c-1$. Also, the rank of M, and hence that of $S_W^{-1} M$, is $c-1$. It must therefore follow that $\mathcal{R}(\Phi) = \mathcal{R}(S_W^{-1} M)$.

Next we show that

$$\mathcal{R}(S_W^{-1} M) = \mathcal{R}(Y^{\dagger T} E).$$

Let $\mathbf{w} = Y^{\dagger T} E \mathbf{x} \in \mathcal{R}(Y^{\dagger T})$. Using identities from § 2, it follows that

$$\begin{aligned} S_W \mathbf{w} &= S_T \mathbf{w} - S_B \mathbf{w} \\ &= (YY^T) Y^{\dagger T} E \mathbf{x} - MKM^T \mathbf{w} \quad \text{by (1) and (3)} \\ &= YE\mathbf{x} - MKM^T \mathbf{w} \quad \text{by (7)} \\ &= MK\mathbf{x} - MKM^T \mathbf{w} \quad \text{by (5)} \\ &\in \mathcal{R}(M). \end{aligned}$$

Therefore,

$$\mathcal{R}(Y^{\dagger T} E) \subset \mathcal{R}(S_W^{-1} M).$$

Again we use a rank argument to prove equality. We know that rank $(S_W^{-1} M) = c-1$. Since $Y^{\dagger T} E$ is an $n \times c$ matrix, rank $(Y^{\dagger T} E) \leqq c$. Also, the rank plus the nullity of $Y^{\dagger T} E$ is equal to c. We show the nullity is one and the result follows.

Accordingly, suppose $Y^{\dagger T}E\mathbf{x}=0$. Then arguing as above,

$$YY^{T}Y^{\dagger T}E\mathbf{x}=0,$$

$$YE\mathbf{x}=0 \quad \text{by (7)}, \qquad MK\mathbf{x}=0 \quad \text{by (5)}.$$

We know that rank $(M)=c-1$, and hence that nullity $(M)=1$. Since K is nonsingular, $\mathbf{x}$ is from a one-dimensional subspace.

As regards the second assertion, equation (11), suppose the pair ϕ_i, λ_i satisfies

$$S_B\phi_i=\mu_i S_W\phi_i, \qquad \phi_i\neq 0,$$

where $\phi_i=Y^{\dagger T}E\mathbf{x}_i$. Let $\mathbf{x}\coloneqq\mathbf{x}_i$, $\mu=\mu_i$, and $\phi=\phi_i$. Applying (4) we get

$$(1+\mu)S_B\phi=\mu S_T\phi,$$

and since $S_T\phi\neq 0$, we must have $\mu\neq -1$. Therefore

$$\begin{aligned}
S_B\phi&=\lambda S_T\phi \qquad (\lambda=\mu/(\mu+1)),\\
MKM^{T}\phi&=\lambda YY^{T}\phi \quad \text{by (3) and (1)},\\
M(YE)^{T}\phi&=\lambda YY^{T}\phi \quad \text{by (5)},\\
M(YE)^{T}Y^{\dagger T}E\mathbf{x}&=\lambda YY^{T}Y^{\dagger T}E\mathbf{x},\\
ME^{T}Y^{T}Y^{\dagger T}E\mathbf{x}&=\lambda YE\mathbf{x} \quad \text{by (7)},\\
ME^{T}V\Sigma U^{T}U\Sigma^{-1}V^{T}E\mathbf{x}&=\lambda YE\mathbf{x} \quad \text{by (6)},\\
ME^{T}VV^{T}E\mathbf{x}&=\lambda MK\mathbf{x} \quad \text{by (5)},\\
M[E^{T}VV^{T}E\mathbf{x}-\lambda K\mathbf{x}]&=0.
\end{aligned}$$

Equation $S_B\phi=\lambda S_T\phi$ has exactly $c-1$ solution pairs (λ, ϕ) for nonzero λ. Thus it suffices to find $c-1$ solutions of (11) for nonzero λ.

From (11) and

$$Y^{T}Y^{\dagger T}=VV^{T} \quad \text{(shown above)},$$

we obtain

$$\begin{aligned}
E^{T}Y^{T}Y^{\dagger T}E\mathbf{x}_i&=\lambda_i K\mathbf{x}_i,\\
(MK)^{T}Y^{\dagger T}E\mathbf{x}_i&=\lambda_i K\mathbf{x}_i \quad \text{by (5)},\\
KM^{T}Y^{\dagger T}E\mathbf{x}_i&=\lambda_i K\mathbf{x}_i,
\end{aligned}$$

which is (12).

THEOREM 2. *If $c=2$ in the previous theorem, so that there is just one $\mathbf{x}_i$ (call it $\mathbf{x}$), then up to scalar multiples it can be assumed to be given by*

$$\mathbf{x}=\begin{pmatrix}-k_2\\ k_1\end{pmatrix}.$$

Proof. Since $\phi=Y^{\dagger T}E\mathbf{x}$, then by (7),

$$YY^{T}\phi=YY^{T}Y^{\dagger T}E\mathbf{x}=YE\mathbf{x}.$$

Therefore, $Y^{T}\phi=E\mathbf{x}$ modulo the nullspace of Y. If $Y^{T}\phi=E\mathbf{x}$, we have

$$E^{T}Y^{T}\phi=K\mathbf{x}, \qquad [MK]^{T}\phi=K\mathbf{x} \quad \text{by (5)}$$

so that

$$\mathbf{x} = M^T\phi = \begin{pmatrix} (\mathbf{m}_1 - \mathbf{m})^T\phi \\ (\mathbf{m}_2 - \mathbf{m})^T\phi \end{pmatrix} = \frac{1}{k_1}(\mathbf{m}_2 - \mathbf{m})^T\phi \begin{pmatrix} -k_2 \\ k_1 \end{pmatrix},$$

since $k_1(\mathbf{m}_1 - \mathbf{m})^T + k_2(\mathbf{m}_2 - \mathbf{m})^T = 0$. It may not be true that $Y^T\phi = E\mathbf{x}$, but any $\mathbf{x} \in R^2$ can be represented as

$$\mathbf{x} = \alpha \begin{pmatrix} -k_2 \\ k_1 \end{pmatrix} + \beta \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

and hence

$$\begin{aligned} \phi &= Y^{\dagger T}E\mathbf{x} \\ &= \alpha Y^{\dagger T}E \begin{pmatrix} -k_2 \\ k_1 \end{pmatrix} + \beta Y^{\dagger T}E \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ &= \alpha Y^{\dagger T}E \begin{pmatrix} -k_2 \\ k_1 \end{pmatrix} + \beta \cdot 0. \end{aligned}$$

Therefore, we can assume without loss of generality that $\beta = 0$.

6. Algorithms. The preferred method for greatest accuracy in implementing Theorem 1 is to use the singular value decomposition $Y = U\Sigma V^T$ as described in (6). The least squares problem (10) is then solved by $G = Y^{\dagger T}E = U\Sigma^{-1}V^TE$ [5, Chap. 6]. The eigensystem (12) is easily solved to produce the vectors $\mathbf{x}_i$ such that $\phi_i = G\mathbf{x}_i$.

For a faster algorithm, at the possible expense of losing some stability in the nearly rank deficient case, we can use the QR factorization of Y^T as a tool. It is also the preferred method if updating the projection vectors is anticipated. Recall that Y^T is a $k \times n$ matrix and rank $(Y^T) = n$. The QR factorization produces a $k \times n$ factor Q having orthonormal columns and an $n \times n$ nonsingular upper triangular factor R for which

$$Y^T = QR.$$

The matrix G satisfies $RG = Q^TE$, as follows from the method of [5, Chap. 6]. The eigensystem (12) is then solved to find the vectors $\mathbf{x}_i$ for which $\phi_i = G\mathbf{x}_i$.

ALGORITHM 1. SVD version.

1) Calculate Y.
2) Calculate the SVD of Y:

$$Y = U\Sigma V^T.$$

3) Calculate

$$A = V^TE.$$

4) Calculate the right singular vectors corresponding to the nonzero singular values of $AK^{-1/2}$. Let V_1 be the $c \times (c-1)$ matrix whose columns are these vectors.
5) Calculate $X = K^{-1/2}V_1$.
6) Calculate

$$\Phi = U\Sigma^{-1}AX.$$

7) Normalize ϕ_i, $i = 1, \cdots, c-1$, if desired.

The main computational load in this algorithm is step 2, the singular value decomposition, which requires about $7kn^2+(11/3)k^3$ flops.

Special note for the case $c=2$. If $c=2$, omit steps 3, 4, and 5, for in this case there is just one $\mathbf{x}_i$ (namely, $\mathbf{x}_1$) and up to scalar multiplies, it can be taken to be

$$\mathbf{x}_1=\begin{pmatrix}-k_2\\ k_1\end{pmatrix}.$$

ALGORITHM 2. QR version.

1) Calculate Y.
2) Calculate the QR factorization of Y^T:

$$Y^T=QR,$$

where Q is $k\times n$ and has orthonormal columns, and R is $n\times n$, upper triangular, and nonsingular.

3) Calculate G by
   (a) finding $B=Q^E$, and
   (b) solving $RG=B$ for G.
4) Calculate $A=K^{1/2}M^TGK^{-1/2}$.
5) Calculate the eigenvectors $\mathbf{v}_1,\cdots,\mathbf{v}_{c-1}$ of (the symmetric matrix) A corresponding to nonzero eigenvalues.
6) Calculate $\phi_j=GK^{-1/2}\mathbf{v}_j$, $j=1,\cdots,c-1$.
7) Normalize ϕ_i, $i=1,\cdots,c-1$, if desired.

The main computational load in this algorithm is step 2, the QR factorization of Y^T. It requires about $n^2(k-n/3)$ flops if done by Householder transformations.

Special note for the case $c=2$. If $c=2$, omit steps 4 and 5, for in this case there is just one $\mathbf{x}_i$ (namely, $\mathbf{x}_1$) and up to scalar multiples, it is given by

$$\mathbf{x}_1=\begin{pmatrix}-k_2\\ k_1\end{pmatrix}.$$

7. Numerical examples. We have implemented four algorithms for solving the Fisher problem. Two of these algorithms are classified as *high accuracy* methods and two are classified as *faster* methods. All programming code was written in double precision FORTRAN, using LINPACK and EISPACK subroutines where possible. The code was implemented on an IBM 4381-R14 mainframe computer with VSFORTRAN compiler.

The two high accuracy methods we used were Algorithm 1 (SVD version) and a recent generalized singular value decomposition (GSVD) of Bai [1], [2] and Paige [7]. The Bai-Paige algorithm was applied directly to (9). The two faster methods were Algorithm 2 (QR version) and the International Mathematics and Statistics Library (IMSL) subroutine ODFISH [6]. The latter computes S_W^{-1} and reduces the problem to solving the ordinary eigensystem $S_W^{-1}S_B\phi=\lambda\phi$. Of course, one of the main points of this paper and of the Bai and Paige papers on the GSVD is to not compute large cross-product matrices (such as S_W) and their inverses.

Run times for the four algorithms are presented below for data classes in spaces of dimensions 80, 90, 120, and 250. The experiments are intended to illustrate comparisons of run times only, and not comparisons of accuracy. Usually, when nearly rank deficient data is presented, only the high accuracy methods run to completion. The results presented here are typical of what we observed in general (Table 1).

TABLE 1
Data parameters.

	Example 1	Example 2	Example 3	Example 4
n	80	90	120	250
k	90	100	150	300
k_1	30	20	40	100
k_2	30	35	50	100
k_3	30	45	60	100

Time (in seconds)

	Example 1	Example 2	Example 3	Example 4
Algorithm 1	16.5	23.2	35.9	*
Bai-Paige GSVD	36.8	53.4	252.8	*
Algorithm 2	2.3	3.1	4.7	52.1
IMSL ODFISH	8.9	11.9	24.0	*

* Virtual storage exceeded.

Our high accuracy method, Algorithm 1, appears to run about 2–7 times faster than the Bai-Paige GSVD for the range of data parameters considered here. Also, the storage requirements for our FORTRAN code for Algorithm 1 were considerably less than for the code for the Bai-Paige code. (However, it is quite possible that the Bai-Paige GSVD algorithm, coded just for application to the Fisher problem, would have reduced run time and storage requirements. The version we ran was a general purpose GSVD solver.) Algorithm 2 was the fastest method we tried, and in particular, was about 3.9–5.1 times faster than the IMSL ODFISH routine for the range of data parameters considered here. It was the only method we tried that would run with Example 4 data because the others required too much storage.

8. Conclusions and comments. Our least squares approach to the Fisher linear discriminant problem offers an efficient alternative to the usual approach of calculating a (large) generalized eigensystem. The least squares steps can be performed in a number of ways, including the two we have analyzed and compared. The SVD approach has higher accuracy and is preferred if the data is nearly rank deficient. It is faster and requires less storage than the (also high accuracy) GSVD solver of Bai and Paige [1], [2], [7]. Our faster method, which uses a QR factorization in the least squares step, appears to run faster than other methods we have seen, with the exception of another QR based method suggested by an anonymous referee and which runs at about the same speed. In this latter method, the QR step is performed on the matrix X, where our method performs it on Y. There is an advantage to our method when updating the QR step is anticipated, since it is inherently simpler to update the QR factorization of Y than of X.

We are doing further research on using a neural network to do the least squares step, and thus of producing a neural network implementation of Fisher's linear discriminant method. It would be much more difficult to do this based solely on solving the generalized singular value decomposition.

Acknowledgments. The author wishes to thank the sponsors of this research for their support and encouragement. He particularly wishes to thank Gregory Hobson, James Keller, Robert McLaren, and John Wootton for many helpful conversations.

He wishes to express special thanks to Zhaozun Bai for providing software for implementing his version of the GSVD.

REFERENCES

[1] Z. BAI, *C. Paige's direct algorithm for computing the generalized singular value decomposition revisited*, preprint.

[2] ———, *The direct* GSVD *algorithm and its parallel implementation*, Ph.D. thesis, Fudan University, 1987. Also available as Report CS-TR-1901, Department of Computer Science, University of Maryland, College Park, MD, 1987.

[3] R. O. DUDA AND P. E. HART, *Pattern Classification and Scene Analysis*, John Wiley, New York, 1973.

[4] R. A. FISHER, *The use of multiple measures in taxonomic problems*, Ann. Eugenica, 7, Part II (1936), pp. 179–188.

[5] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[6] *International Mathematical and Statistical Library Manual*, IMSL Library, Ninth Edition, Reference Manual, IMSL, Inc., Houston, TX, 1982.

[7] C. C. PAIGE, *Computing the generalized singular value decomposition*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1122–1146.

[8] G. W. STEWART, *Computing the* CS *decomposition of a partitioned orthonormal matrix*, Numer. Math., 43 (1982), pp. 297–306.

[9] C. F. VAN LOAN, *Computing the* CS *and generalized singular value decompositions*, Numer. Math., 46 (1985), pp. 479–491.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 607-647, May 1991
© 1991 Society for Industrial and Applied Mathematics
008

NUMERICAL CALCULATION OF INVARIANT TORI*

LUCA DIECI†, JENS LORENZ‡, AND ROBERT D. RUSSELL§

Abstract. The problem of computing a smooth invariant manifold for a finite-dimensional dynamical system is considered. In this paper, it is assumed that the manifold can be parameterized over a torus in terms of a subset of the system variables. The approach used here then involves solving a system of partial differential equations subject to periodic boundary conditions.

The resulting numerical approach is analyzed and contrasted with some previously tested ones, and several methods of implementation are considered. Numerical results are given for the forced van der Pol equation and for a system of two linearly coupled oscillators.

Key words. invariant tori, partial differential equations with same principal part, modified leap-frog, compactification, forced van der Pol equation, coupled oscillator

AMS(MOS) subject classifications. 65L99, 34C99, 65P05

1. Introduction. As is well known, the expanding interest in dynamical systems theory is due to the increasing investigation of diverse mathematical models in science which are neither linear nor almost linear. Solution of these problems generally requires a blend of geometrical techniques in conjunction with computational techniques. Analyzing the evolution of a dynamical system can involve tracking trajectories in high-(possibly infinite-) dimensional phase space. For dissipative systems of this type, trajectory evolution in the phase space is often characterized by its settling down to a finite-dimensional attracting manifold. One of the current concerns for both finite- and infinite-dimensional systems is the characterization and location of such manifolds (e.g., see [GH], [TS], [Ha], [T]). This has, in turn, spurred recent interest in the scientific computing community in developing methods for numerically approximating these manifolds (e.g., see [KASP] and [vV1], [FJKST]).

Here, we shall be concerned with the autonomous first-order system of ODEs (ordinary differential equations)

$$\frac{d\mathbf{w}}{dt} = \mathbf{F}(\mathbf{w}), \qquad \mathbf{w} \in W \subset \mathbb{R}^n, \tag{1.1}$$

where $\mathbf{F}: \mathbb{R}^n \to \mathbb{R}^n$ is smooth. We assume the existence of a smooth manifold $M \subset W$ which is invariant under the flow of (1.1). In other words, if S^t is the solution operator of (1.1), and if we have initial data

$$\mathbf{w}(0) = \mathbf{w}_0 \in M, \tag{1.2}$$

then the solution

$$\mathbf{w}(t) = S^t \mathbf{w}_0 \in M, \qquad -\infty < t < \infty, \tag{1.3}$$

that is, it stays in M for all time. Our aim is to compute an approximation for M.

* Received by the editors June 21, 1989; accepted for publication (in revised form) March 26, 1990.

† School of Mathematics, Georgia Institute of Technology, Atlanta, Georgia 30334. The research of this author was supported in part by National Science Foundation grant DMS 8802762.

‡ Applied Mathematics Department, California Institute of Technology, Pasadena, California 91125. Present address, Department of Mathematics and Statistics, University of New Mexico, Albuquerque, New Mexico 87131. The research of this author was supported in part by the Department of Energy under contract DE-AS03-76ER72012.

§ Department of Mathematics, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6. The research of this author was supported in part by a Science and Engineering Research Council (UK) Fellowship and by Natural Sciences and Engineering Research Council of Canada grant A8781.

In this paper we restrict ourselves to systems (1.1), which are partitioned in the form

$$\begin{aligned} \frac{d\mathbf{u}}{dt} &= \mathbf{f}(\mathbf{u}, \mathbf{v}), \\ \frac{d\mathbf{v}}{dt} &= \mathbf{g}(\mathbf{u}, \mathbf{v}), \qquad \mathbf{w} = (\mathbf{u}, \mathbf{v}) \in U \times V = W, \end{aligned} \tag{1.4}$$

and we ask for a manifold

$$M = \{(\mathbf{u}, \boldsymbol{\Phi}(\mathbf{u})) : \mathbf{u} \in U\}, \tag{1.5}$$

which can be parameterized over U. Clearly, this is a severe restriction, and finding proper variables to parameterize the manifold can be one of the most difficult aspects of the problem. However, for some problems of interest, the form (1.4), (1.5) is realistic and we shall give examples. Moreover, by changing coordinates $\tilde{\mathbf{w}} = T(\mathbf{w})$, $\tilde{\mathbf{w}} = (\tilde{\mathbf{u}}, \tilde{\mathbf{v}})$, such a form can (locally) always be achieved. In future extensions of this work we plan to implement such general coordinate transformations.

Assuming that $\boldsymbol{\Phi}: U \to V$ is a smooth function, it is easy (e.g., see [DLR]) to show that the manifold (1.5) is invariant for the system (1.4) if and only if

$$\Phi'(\mathbf{u})\mathbf{f}(\mathbf{u}, \boldsymbol{\Phi}(\mathbf{u})) = \mathbf{g}(\mathbf{u}, \boldsymbol{\Phi}(\mathbf{u})), \qquad \mathbf{u} \in U, \tag{1.6}$$

where $\Phi'(\mathbf{u}) := (\partial \Phi_i(\mathbf{u})/\partial u_j)$ is the Jacobian of $\boldsymbol{\Phi}$. If dim $U = p$ and dim $V = q$, then (1.6) is a system of partial differential equations for

$$\Phi_1(u_1, \cdots, u_p), \cdots, \Phi_q(u_1, \cdots, u_p).$$

Typically, these PDEs have appropriate associated boundary conditions, arising because U is generally assumed to be compact. In our applications U is a torus, i.e., a manifold without boundary. Consequently, no boundary conditions for $\boldsymbol{\Phi}$ are necessary; numerically, the fact that U is a torus is realized by periodicity requirements for $\boldsymbol{\Phi}$.

The natural association between a solution to the PDEs (1.6) and a solution $(\mathbf{u}, \mathbf{v} := \boldsymbol{\Phi}(\mathbf{u}))$ to the dynamical system (1.4) has been used for a long time in analytical work, and applications (in apparently unrelated settings) are numerous. Meyer [M] considers special hyperbolic systems like (1.6) where the PDEs each have the *same principal part* and hence the characteristic equations are (1.4). He relates $\Phi'(\mathbf{u})$ to the solution of the Riccati equation associated to the linearized system. Not surprisingly, the system (1.6) is at the core of the *invariant embedding* techniques (and (1.6) are the so-called invariant embedding equations). In another context [GH], a center manifold is approximated by finding a series expansion for the solution to (1.6) subject to appropriate initial conditions. In a setting much closer to ours, Sacker [S1] investigates the existence of perturbed invariant tori by analyzing solvability of (1.6), which he does by so-called elliptic regularization. (For marching schemes, his type of technique is routinely used by numerical practitioners, corresponding to the introduction of (artificial) dissipation terms to solve hyperbolic-type PDEs.)

In this paper, we intend to exploit the PDE formulation (1.6) for an invariant manifold from a computational viewpoint. It is an appealing feature of the PDE formulation that it does not depend on the attractivity type of M. (The manifold can be attracting, repelling, or of mixed attractivity type as $t \to \infty$.) For this reason, we have chosen a simple discretization of (1.6) which also does not favor or utilize attractivity properties. In fact, we discretize (1.6) on a fixed grid by the leap-frog

scheme without any artificial viscosity and then solve with Newton's method. For simple model problems with constant coefficients we prove the stability of the discretization independently of attractivity properties. An extension of this result to the general case remains an open problem. Since we do not directly integrate (1.1), the success of the algorithm—in this form—is affected by geometry of the invariant manifold itself, rather than by the solution dynamics on it which are often complex for problems of interest.

In many applications, the computation of a two-dimensional invariant manifold can be reduced to the computation of an invariant curve for an associated Poincaré map (e.g., see [L]). An obvious advantage is that the dimensionality of the unknown object is reduced from two to one. In our approach we use this idea in the following way. After discretization and linearization, we obtain (large) matrix systems having special sparsity structure, which can be solved by a compactification process. Loosely speaking, the final compactified systems are discretized linearized versions of the equation for the invariant curve of the Poincaré map. Our compactification is interpretable as a (time) marching algorithm, and, as is known, for example, from shooting algorithms for ordinary BVPs, must be used cautiously, because numerical instabilities can lead to an ill-conditioned system. In fact, in general, we have to modify the leap-frog scheme (see below) when we compactify. For simple model problems (constant coefficients) and an attracting invariant manifold, we prove the numerical stability of our approach. Numerical examples show the feasibility in more general (yet attracting) situations.

Algorithms for the computation of attracting invariant curves via the Poincaré map have been devised and studied by van Veldhuizen [vV1], [vV2]. Had he used the parameterization (1.4)–(1.5), his approach would have turned out to basically correspond to solving the PDE (1.6) (appropriately linearized) by the method of characteristics. (van Veldhuizen does not use Newton's method, however.) Our motivation to base the numerical approach directly on (1.6) is its generality. The compactification process may or may not be useful, depending on the application. In contrast, any approach based on the Poincaré map can only treat a certain class of problems and it does subject the numerical algorithm to the dynamics on the manifold itself. In general, however, a comparative study of different strategies requires further investigation.

An outline of the paper is as follows. In § 2 we analyze the smoothness properties of the solution to the PDE for the important case where U is the 2-torus, V the real line, and (1.6) is linear; for this case, the method of characteristics for solving (1.6) is related to the Poincaré map for the original system. In § 3 we consider the system structure arising when the PDE (1.6) with U now a p-torus and $V = \mathbb{R}^q$, for arbitrary p and q, is discretized by leap-frog and solved by Newton's method. We also consider a model constant coefficient PDE and use Fourier analysis to show uniqueness and estimates of its solution. We show that the numerical solution with leap-frog inherits these properties and that the error is of second order in the discretization. In § 4, for the case where U is a 2-torus, we show how to compactify the linear system arising for a modified leap-frog scheme in a stable way. Our analysis for the constant coefficient case indicates that, unlike when dealing with simple leap-frog and the full system, stability of the process depends upon a proper discretization, and also upon the attractivity of the invariant manifold. Finally, to demonstrate the method and various implementation procedures, in § 5 we present numerical results for two relevant problems: the forced van der Pol oscillator and a system of two linearly coupled oscillators. Section 6 contains some conclusions and recommendations for future work.

2. The basic PDE approach. In this section we present the analytical ideas which guide our numerical approach. The reader is referred to the papers [F], [S1], [S2] for a deeper, more complete theory of invariant manifolds of dynamical systems. We consider the case in which the parameter space U (see (1.4)) is (identifiable with) the p-dimensional torus, denoted by

$$T^p = \{\boldsymbol{\theta} = (\theta_1, \cdots, \theta_p): \theta_j \in \mathbb{R} (\mathrm{mod}\, 2\pi)\}.$$

In particular, T^1 denotes the circle. Accordingly, the dynamical system (1.1) (or (1.4)) is rewritten as

$$\frac{d\boldsymbol{\theta}}{dt} = \mathbf{f}(\boldsymbol{\theta}, \mathbf{r}), \qquad \frac{d\mathbf{r}}{dt} = \mathbf{g}(\boldsymbol{\theta}, \mathbf{r}), \tag{2.1}$$

where $(\boldsymbol{\theta}, \mathbf{r}) \in T^p \times \mathbb{R}^q$ (note that the "radial" coordinates $\mathbf{r}$ are not necessarily nonnegative) and

$$\mathbf{f}: T^p \times \mathbb{R}^q \to \mathbb{R}^p, \qquad \mathbf{g}: T^p \times \mathbb{R}^q \to \mathbb{R}^q \tag{2.2}$$

are assumed to be smooth. (As usual, we can also consider $\mathbf{f}$ and $\mathbf{g}$ as smooth functions from $\mathbb{R}^{p+q}$ to $\mathbb{R}^p, \mathbb{R}^q$, respectively, which are 2π-periodic in $\theta_1, \cdots, \theta_p$.) Thus, from (1.6), an invariant manifold M of the form

$$M = \{(\boldsymbol{\theta}, \mathbf{r}(\boldsymbol{\theta})): \boldsymbol{\theta} \in T^p\} \subset T^p \times \mathbb{R}^q \tag{2.3}$$

can be determined by solving the PDE system

$$\sum_{j=1}^{p} f_j(\boldsymbol{\theta}, \mathbf{r}(\boldsymbol{\theta})) \frac{\partial \mathbf{r}}{\partial \theta_j}(\boldsymbol{\theta}) = \mathbf{g}(\boldsymbol{\theta}, \mathbf{r}(\boldsymbol{\theta})), \qquad \boldsymbol{\theta} \in T^p \tag{2.4}$$

subject to the periodic boundary conditions

$$\begin{aligned} r_i(\theta_1, \cdots, \theta_{k-1}, 0, \theta_{k+1}, \cdots, \theta_p) &= r_i(\theta_1, \cdots, \theta_{k-1}, 2\pi, \theta_{k+1}, \cdots, \theta_p), \\ & i = 1, \cdots, q, \qquad k = 1, \cdots, p, \end{aligned} \tag{2.5}$$

for the unknown (smooth) function $\mathbf{r}: T^p \to \mathbb{R}^q$. For simplicity, consider first the case $p = 2$ and $q = 1$, for which the PDE system (2.4)–(2.5) becomes

$$f_1(\boldsymbol{\theta}, r) r_{\theta_1} + f_2(\boldsymbol{\theta}, r) r_{\theta_2} = g(\boldsymbol{\theta}, r), \qquad \boldsymbol{\theta} \in [0, 2\pi] \times [0, 2\pi], \tag{2.6a}$$

$$r(0, \theta_2) = r(2\pi, \theta_2), \qquad r(\theta_1, 0) = r(\theta_1, 2\pi). \tag{2.6b}$$

From a numerical viewpoint, we solve (2.6) by discretizing and using Newton's method. To investigate solvability requirements and conditioning of the linearized problem, consider the (formal) Newton process for the continuous problem (2.6). If $r^0 = r^0(\boldsymbol{\theta})$ is a smooth approximate solution which satisfies (2.6b) exactly, then the next Newton iterate r^1 is the solution of (2.6b) and

$$b_1(\boldsymbol{\theta}) r^1_{\theta_1} + b_2(\boldsymbol{\theta}) r^1_{\theta_2} + c(\boldsymbol{\theta}) r^1 = q(\boldsymbol{\theta}), \tag{2.7a}$$

where

$$\begin{aligned} b_1(\boldsymbol{\theta}) &:= f_1(\boldsymbol{\theta}, r^0), \qquad b_2(\boldsymbol{\theta}) := f_2(\boldsymbol{\theta}, r^0), \\ c(\boldsymbol{\theta}) &:= -g_r(\boldsymbol{\theta}, r^0) + f_{1r}(\boldsymbol{\theta}, r^0) r^0_{\theta_1} + f_{2r}(\boldsymbol{\theta}, r^0) r^0_{\theta_2}, \\ q(\boldsymbol{\theta}) &:= g(\boldsymbol{\theta}, r^0) - [g_r(\boldsymbol{\theta}, r^0) - f_{1r}(\boldsymbol{\theta}, r^0) r^0_{\theta_1} - f_{2r}(\boldsymbol{\theta}, r^0) r^0_{\theta_2}] r^0. \end{aligned} \tag{2.7b}$$

The functions in (2.7b) are 2π-periodic in θ_1, θ_2. In general, there is a severe theoretical difficulty for convergence of this Newton process because of a loss of smoothness

during the iteration. In fact, if f_{1r}, f_{2r} are nonzero and $r^0 \in C^k$, then the coefficients $c, q \in C^{k-1}$, and from (2.7a) we can only expect $r^1 \in C^{k-1}$. This difficulty can be overcome by adding a dissipation term in (2.7a) whose coefficient is sent to zero during the iteration [S2]. In our numerical experiments with Newton's method on a fixed grid, we did not observe such a difficulty, and obtained quadratic convergence throughout.

Next, we investigate the solution's smoothness properties when (2.6) is linear. To simplify notation, let

$$\theta_1 = \theta, \qquad \theta_2 = t, \qquad r = r(\theta, t),$$

and consider the *special case* of (2.6) given by

$$r_t + b(\theta, t) r_\theta + c(\theta, t) r = q(\theta, t), \qquad (\theta, t) \in T^2. \tag{2.8}$$

(Note that this form is obtained from (2.7) if we can divide by $b_2(\boldsymbol{\theta})$, and it is then representative of the linear problems to be solved.)

To analyze (2.8), let us for the moment ignore the periodicity requirement in t (but retain the 2π-periodicity in θ), and suppose we are given periodic initial data

$$r(\theta, 0) = R(\theta), \qquad R(\theta) \equiv R(\theta + 2\pi).$$

Then we can solve the initial value problem for r using the method of characteristics: If $\theta(\alpha, t)$ is the solution of

$$\frac{d\theta}{dt} = b(\theta, t), \qquad \theta(0) = \alpha,$$

then at $t = 2\pi$

$$r(\gamma(\alpha), 2\pi) = \tilde{H}(\alpha) R(\alpha) + \tilde{Q}(\alpha), \qquad \alpha \in T^1, \tag{2.9}$$

where

$$\begin{aligned} &\gamma(\alpha) := \theta(\alpha, 2\pi), \qquad \tilde{H}(\alpha) := \exp\left(-\int_0^{2\pi} c(\theta(\alpha, \tau), \tau)\, d\tau \right), \\ &\tilde{Q}(\alpha) := \tilde{H}(\alpha) \int_0^{2\pi} q(\theta(\alpha, s), s) \exp\left(\int_0^s c(\theta(\alpha, \tau), \tau)\, d\tau \right) ds. \end{aligned} \tag{2.10}$$

The function $\partial\theta/\partial\alpha = \theta_\alpha$ solves the linear problem

$$\frac{d\theta_\alpha}{dt} = b_\theta(\theta, t)\theta_\alpha, \qquad \theta_\alpha(0) = 1,$$

thus

$$\frac{d\gamma}{d\alpha}(\alpha) = \theta_\alpha(\alpha, 2\pi) = \exp\left(\int_0^{2\pi} b_\theta(\theta(\alpha, \tau), \tau)\, d\tau \right) > 0.$$

We conclude that the circle map[1] $\gamma: T^1 \to T^1$ is invertible, and its inverse $\chi = \gamma^{-1}$ is as smooth as γ. From (2.9), we find

$$r(\theta, 2\pi) = H(\theta) R(\chi(\theta)) + Q(\theta), \qquad \theta \in T^1$$

with

$$H(\theta) := \tilde{H}(\chi(\theta)), \qquad Q(\theta) := \tilde{Q}(\chi(\theta)). \tag{2.11}$$

[1] Considered as a function from $\mathbb{R}$ to $\mathbb{R}$, the map γ satisfies $\gamma(\alpha + 2\pi) = \gamma(\alpha) + 2\pi$.

Thus, in order to obtain $r = r(\theta, t)$ 2π-periodic in t, the initial data $r(\theta, 0) = R(\theta)$ must satisfy the functional equation

$$R(\theta) = H(\theta)R(\chi(\theta)) + Q(\theta), \qquad \theta \in T^1. \tag{2.12}$$

LEMMA 2.1. *Suppose that the coefficients of* (2.8) *have k continuous derivatives, i.e., b, c, q* $\in C^k(T^2)$, $k \geqq 1$. *Then the functions* $\chi = \gamma^{-1}$, H, Q *defined in* (2.10), (2.11) *are in* $C^k(T^1)$. *If* (2.12) *has a solution* $R \in C^k(T^1)$, *then there is a corresponding solution* $r \in C^k(T^2)$ *of* (2.8).

Proof. It is clear that $\theta = \theta(\alpha, t)$ is a C^k-function and $\theta(\alpha + 2\pi, t) = \theta(\alpha, t) + 2\pi$. Thus the statement about χ, H, and Q follows. If $R \in C^k(T^1)$ solves (2.12), we obtain a solution $r(\theta, t) \in C^k(T^1 \times [0, 2\pi])$ of

$$r_t + br_\theta + cr = q, \qquad r(\theta, 0) = r(\theta, 2\pi).$$

The periodicity condition $r_t(\theta, 0) = r_t(\theta, 2\pi)$ follows from the PDE, and periodicities of r_{tt}, etc., then follow by differentiating. □

Remark. The functional equation (2.12) can also be derived via the Poincaré map P for the system

$$\frac{d\theta}{dt} = b(\theta, t), \qquad \frac{dr}{dt} = -c(\theta, t)r + q(\theta, t), \qquad t \in \mathbb{R} \bmod 2\pi.$$

In fact, for given initial data $\theta(0) = \alpha$, $r(0) = \rho$, the solution values at $t = 2\pi$ are

$$P(\alpha, \rho) := (\theta(2\pi), r(2\pi)) = (\gamma(\alpha), \tilde{H}(\alpha)\rho + \tilde{Q}(\alpha)),$$

and if we request invariance of the curve $(\alpha, R(\alpha))$, $\alpha \in T^1$, under the Poincaré map P, we arrive at (2.12) with $\chi = \gamma^{-1}$, $H = \tilde{H} \circ \chi$, $Q = \tilde{Q} \circ \chi$. We have emphasized the derivation of (2.12) via the PDE (2.8) because the discrete analogues of (2.8) are fundamental in our numerical approach.

The following result regarding the smoothness properties of the solution $R(\theta)$ to the functional equation (2.12) is immediate.

THEOREM 2.2. (i) *Suppose that* $\chi : T^1 \to T^1$, $H, Q : T^1 \to \mathbb{R}$ *satisfy* $\chi, H, Q \in C(T^1)$ *and that* $|H|_\infty := \max\{|H(\theta)| : \theta \in T^1\} < 1$. *Then* (2.12) *has a unique solution* $R^* \in C(T^1)$.

(ii) *If, in addition,* $\chi, H, Q \in C^k(T^1)$ *and*

$$\left| H \left(\frac{d\chi}{d\theta} \right)^k \right|_\infty < 1, \tag{2.13}$$

then $R^* \in C^k(T^1)$.

Proof. Define the linear operator $L : C(T^1) \to C(T^1)$ by

$$(LR)(\theta) = H(\theta)R(\chi(\theta)), \qquad \theta \in T^1.$$

Then, (2.12) reads $(I - L)R = Q$, so if $|L|_\infty = |H|_\infty < 1$, part (i) follows because

$$R^*(\theta) = \sum_{j=0}^{\infty} L^j Q(\theta) = \sum_{j=0}^{\infty} H(\theta)H(\chi(\theta)) \cdots H(\chi^{j-1}(\theta))Q(\chi^j(\theta))$$

(where $\chi^0(\theta) := \theta$, $\chi^1(\theta) := \chi(\theta)$, $\chi^2(\theta) := \chi(\chi(\theta)), \cdots$). Part (ii) would follow if we could successively differentiate the relation $(I - L)R = Q$ (e.g., $R' - H'R - HR'\chi' = Q'$ implies $(I - H\chi')R' = Q' + H'R$, and (2.13) gives (ii) for $k = 1$). To show this last fact, we prove that L is a contraction on $C^k(T^1)$ with respect to the norm $|R|_{k,\varepsilon} := \varepsilon |d^k R / d\theta^k|_\infty + |R|_\infty$, if $\varepsilon > 0$ is sufficiently small. We have $(d^k/d\theta^k)(LR)(\theta) =$

$H(\theta)(d\chi/d\theta)^k(d^kR/d\theta^k)+S(\theta)$, where $S(\theta)=\sum_{j=0}^{k-1} a_j(\theta)(d^jR/d\theta^j)$ and the a_j's are independent of R. Therefore, given any $\eta>0$, there is a constant $C(\eta)$ (depending on η but not on R) for which $|S|_\infty \leqq \eta|d^kR/d\theta^k|_\infty + C(\eta)|R|_\infty$. Thus we obtain $|LR|_{k,\varepsilon} \leqq \varepsilon(|H(d\chi/d\theta)^k|_\infty+\eta)|d^kR/d\theta^k|_\infty+(\varepsilon C(\eta)+|H|_\infty)|R|_\infty$. If now we first choose $\eta>0$ so that $|H(d\chi/d\theta)^k|_\infty+\eta<1$, and then choose $\varepsilon>0$ so that $|H|_\infty+\varepsilon C(\eta)<1$, contractivity of L follows, and the theorem is proven. □

A very special case occurs if $b(\theta, t)$ **is constant**, say $b(\theta, t)=b_0$. Then, for the original angular coordinates θ_1 and θ_2, we have (for appropriate constants θ_{10}, θ_{20})

$$\theta_1(t)=b_0t+\theta_{10}, \qquad \theta_2(t)=t+\theta_{20}, \tag{2.14}$$

i.e., the flow of the differential equation projected onto the torus T^2 is parallel. If b_0 is rational, the motion (2.14) is periodic; if b_0 is irrational, it is quasi-periodic. From (2.10), $\gamma(\alpha)=2\pi b_0+\alpha(\mathrm{mod}\, 2\pi)$, and therefore $\chi(\theta)=\theta-2\pi b_0$, $d\chi/d\theta=1$. Furthermore, $|H|_\infty=|\tilde{H}|_\infty<1$ if and only if

$$\int_0^{2\pi} c(\tau b_0+\alpha, \tau)\, d\tau>0, \qquad \alpha\in T^1. \tag{2.15}$$

To summarize, we obtain Corollary 2.3 from Lemma 2.1 and Theorem 2.2.

COROLLARY 2.3. *Suppose that* $b(\theta, t)=b_0$ *is constant and* c, $q\in C^k(T^2)$, $k\geqq 1$. *Then* (2.8) *has a unique solution* $r\in C^k(T^2)$ *if* (2.15) *holds.* □

In the case of **variable** $b(\theta, t)$, in general, $(d\gamma/d\alpha)(\alpha)=\exp(\int_0^{2\pi} b_\theta(\theta(\alpha, \tau), \tau)\, d\tau)$ takes on values less than 1, so $d\chi/d\theta>1$, and condition (2.13) may be violated for $k>0$ even if $|H|_\infty<1$. It is possible that χ, H, and $Q\in C^\infty(T^1)$, but the solution R^* of (2.12) is not smooth. For example, the equation

$$R(\theta)=bR(a\theta)+\cos\theta, \qquad \theta\in\mathbb{R},$$

with a, $b\in\mathbb{R}$, $|b|<1$, has the unique continuous solution $R(\theta)=\sum_{j=0}^{\infty} b^j\cos(a^j\theta)$, which is nowhere differentiable if a is an odd integer and $ab>1+3\pi/2$ [Hi]. (This function was first given by Weierstrass in 1875 as an example of a continuous nowhere differentiable function.)

In summary, for the simple system

$$\frac{d\theta}{dt}=b(\theta, t), \qquad \theta, t\in\mathbb{R}(\mathrm{mod}\, 2\pi),$$

$$\frac{dr}{dt}=-c(\theta, t)r+q(\theta, t),$$

we obtain an invariant manifold $M=\{(\theta, t, r(\theta, t)): (\theta, t)\in T^2\}$ if $r: T^2\to\mathbb{R}$ solves (2.8). If we let b, c, $q\in C^k(T^2)$ and, for definiteness, $c(\theta, t)>0$, then $r\in C^k(T^2)$ is guaranteed only as long as (2.13) holds (which it is if $b(\theta, t)$ is constant). If (2.13) is violated, we can expect that r may lose its smoothness.

In practice, of course, it is not easy to verify (2.13), even when a problem of interest does arise in the simple form (2.8). However, for the first example in § 5 (the van der Pol oscillator), the underlying linear PDE to solve at each Newton iteration is of this form; in that case, we have not observed any difficulty corresponding to loss of smoothness for r. In the second example, we instead get a PDE like (2.8) for a two-dimensional vector $\mathbf{r}$ at each Newton iteration, and observe a loss of smoothness in the solution (see Figs. 17, 19), which could very well be due to the inadequacy of the global parameterization.

3. Leap-frog analysis. The numerical scheme we use for solving the PDE (2.4) is the *leap-frog* discretization. We use constant stepsize h_j in each coordinate direction θ_j, so by letting

$$\mathbf{h}=(h_1, h_2, \cdots, h_p), \quad h_j=\frac{2\pi}{N_j}, \quad N_j \geqq 1 \quad (j=1, \cdots, p), \quad \text{and} \quad N=\prod_{j=1}^{p} N_j, \tag{3.1}$$

we obtain the mesh with N points (representing the torus T^p):

$$\Omega_h=\{(n_1h_1, n_2h_2, \cdots, n_ph_p): n_j \in \mathbb{Z}(\text{mod } N_j), j=1, \cdots, p\}. \tag{3.2}$$

The leap-frog scheme for (2.4) consists of replacing $\partial/\partial\theta_j$ by the standard centered difference operator along θ_j,

$$(D_jf)(\cdots, n_jh_j, \cdots)=\frac{1}{2h_j}[f(\cdots, (n_j+1)h_j, \cdots)-f(\cdots, (n_j-1)h_j, \cdots)] \tag{3.3}$$

and solving the nonlinear system

$$\sum_{j=1}^{p} f_j(\boldsymbol{\theta}, \mathbf{r}_h(\boldsymbol{\theta}))D_j\mathbf{r}_h(\boldsymbol{\theta})=\mathbf{g}(\boldsymbol{\theta}, \mathbf{r}_h(\boldsymbol{\theta})), \qquad \boldsymbol{\theta} \in \Omega_h, \tag{3.4}$$

for grid function $\mathbf{r}_h: \Omega_h \to \mathbb{R}^q$. To solve (3.4) for the qN unknowns $\mathbf{r}_h(\boldsymbol{\theta})$, $\boldsymbol{\theta} \in \Omega_h$, we use Newton's method: If $\mathbf{r}_h^0$ is an initial approximation to $\mathbf{r}_h(\boldsymbol{\theta})$, then the next Newton iterate $\mathbf{r}_h^1$ satisfies ((2.5) implicitly and)

$$\sum_{j=1}^{p} f_j^0 D_j\mathbf{r}_h^1+\left\{\sum_{j=1}^{p} (D_j\mathbf{r}_h^0)\nabla f_j^0\right\}(\mathbf{r}_h^1-\mathbf{r}_h^0)=\mathbf{g}^0+G^0(\mathbf{r}_h^1-\mathbf{r}_h^0), \tag{3.5}$$

where

$$\begin{aligned} &f_j^0 := f_j(\boldsymbol{\theta}, \mathbf{r}_h^0), \qquad \nabla f_j^0 := \left(\frac{\partial f_j}{\partial r_1}, \cdots, \frac{\partial f_j}{\partial r_q}\right)(\boldsymbol{\theta}, \mathbf{r}_h^0), \\ &\mathbf{g}^0 := \mathbf{g}(\boldsymbol{\theta}, \mathbf{r}_h^0), \quad G^0 := \begin{bmatrix} \nabla g_1^0 \\ \vdots \\ \nabla g_q^0 \end{bmatrix}, \quad \nabla g_j^0 := \left(\frac{\partial g_j}{\partial r_1}, \cdots, \frac{\partial g_j}{\partial r_q}\right)(\boldsymbol{\theta}, \mathbf{r}_h^0). \end{aligned} \tag{3.6}$$

Therefore, the linear system for $\mathbf{r}_h^1$ has the form

$$\sum_{j=1}^{p} \omega_j D_j\mathbf{r}_h+\bar{C}^0\mathbf{r}_h=\boldsymbol{\psi}_h, \qquad \mathbf{r}_h: \Omega_h \to \mathbb{R}^q, \tag{3.7}$$

where $\omega_j \in \mathbb{R}$, $\bar{C}^0 \in \mathbb{R}^{q\times q}$, and $\boldsymbol{\psi}_h \in \mathbb{R}^q$ are known grid functions depending on $\mathbf{h}$ and $\mathbf{r}_h^0$.

We analyze the stability of the leap-frog discretization for the case in which (2.1) has the form

$$\begin{aligned} &\frac{d\theta_j}{dt}=\omega_j(\boldsymbol{\theta}), \qquad j=1, \cdots, p, \\ &\frac{d\mathbf{r}}{dt}=-\bar{C}(\boldsymbol{\theta})\mathbf{r}+\boldsymbol{\psi}(\boldsymbol{\theta}), \end{aligned} \tag{3.8}$$

for which (2.4) is the linear PDE

$$\omega_1\frac{\partial\mathbf{r}}{\partial\theta_1}+\cdots+\omega_p\frac{\partial\mathbf{r}}{\partial\theta_p}+\bar{C}\mathbf{r}=\boldsymbol{\psi}, \tag{3.9}$$

and the leap-frog discretization over Ω_h gives the $(Nq \times Nq)$ system

$$\omega_1 D_1 \mathbf{r}_h + \cdots + \omega_p D_p \mathbf{r}_h + \bar{C}\mathbf{r}_h = \boldsymbol{\psi}_h. \tag{3.10}$$

Below we derive a stability estimate for the discrete solution and relate it to a similar estimate for the continuous solution for the constant coefficient case, i.e.,

$$\boldsymbol{\omega} = (\omega_1, \cdots, \omega_p)^T \in \mathbb{R}^p, \qquad \bar{C} \in \mathbb{R}^{q\times q}, \qquad \boldsymbol{\psi}(\boldsymbol{\theta}) : T^p \to \mathbb{R}^q. \tag{3.11}$$

In this restricted case, we can study (3.9) and (3.10) by a classical Fourier analysis.

Let $(\mathbf{u}, \mathbf{v}) := \int_0^{2\pi} \cdots \int_0^{2\pi} \langle \mathbf{u}(\boldsymbol{\theta}), \mathbf{v}(\boldsymbol{\theta})\rangle \, d\boldsymbol{\theta}$ denote the L_2-inner product for L_2 functions, $\mathbf{u}, \mathbf{v}: T^p \to \mathbb{C}^q$, $\langle \mathbf{u}, \mathbf{v}\rangle := \sum_{j=1}^q \bar{u}_j v_j$ be the Euclidean inner product for vectors of $\mathbb{C}^q$, and $\|\mathbf{u}\| := (\mathbf{u}, \mathbf{u})^{1/2}$, $|\mathbf{u}| := \langle \mathbf{u}, \mathbf{u}\rangle^{1/2}$ be the respective induced norms. Assume further that

$$c_m \neq 0, \qquad c_m := \min\{|\mathrm{Re}\,(\lambda)|, \lambda \text{ an eigenvalue of } \bar{C}\}, \tag{3.12}$$

where $\mathrm{Re}\,(\lambda)$ indicates the real part of λ. Then we can prove Lemma 3.1.

LEMMA 3.1. *Let* (3.11) *and* (3.12) *hold. Then* (3.9) *has a unique solution* $\mathbf{r}(\boldsymbol{\theta})$ *which is as smooth as* $\boldsymbol{\psi}$ *is. Furthermore, there is a positive constant* c_0 *such that*

$$\|\mathbf{r}\| \leqq c_0 \|\boldsymbol{\psi}\|. \tag{3.13}$$

Proof. Taking a Fourier expansion in (3.9) in terms of the orthonormal basis of $L_2(T^p, \mathbb{C})$ given by $\{e^{i\langle \mathbf{k}, \boldsymbol{\theta}\rangle}/(2\pi)^{p/2}\}_{\mathbf{k}\in\mathbb{Z}^p}$, we obtain

$$(i\langle \boldsymbol{\omega}, \mathbf{k}\rangle I + \bar{C})\hat{\mathbf{r}}(\mathbf{k}) = \hat{\boldsymbol{\psi}}(\mathbf{k}), \qquad \mathbf{k} \in \mathbb{Z}^p, \tag{3.14}$$

where we have set $\boldsymbol{\psi}(\boldsymbol{\theta}) = 1/(2\pi)^{p/2} \sum_{\mathbf{k}\in\mathbb{Z}^p} \hat{\boldsymbol{\psi}}(\mathbf{k})\, e^{i\langle \mathbf{k}, \boldsymbol{\theta}\rangle}$, and thus $\hat{\boldsymbol{\psi}}(\mathbf{k}) \in \mathbb{C}^q$ has components

$$\hat{\psi}_j(\mathbf{k}) = \frac{1}{(2\pi)^{p/2}} (e^{i\langle \mathbf{k}, \boldsymbol{\theta}\rangle}, \psi_j(\boldsymbol{\theta})).$$

For simplicity, assume $\bar{C}$ diagonalizable (see Remark (iii) below). If $S^{-1}\bar{C}S = \Lambda = \mathrm{diag}\,(\lambda_i)$ is the diagonal reduction of $\bar{C}$, then, from (3.12), $\Lambda + i\alpha I$ is invertible for any $\alpha \in \mathbb{R}$, and

$$|(\Lambda + i\alpha I)^{-1}| \leqq \frac{1}{c_m}. \tag{3.15}$$

Therefore, for any $\mathbf{k} \in \mathbb{Z}^p$,

$$|\hat{\mathbf{r}}(\mathbf{k})| \leqq c_0 |\hat{\boldsymbol{\psi}}(\mathbf{k})|, \qquad c_0 = \frac{\mathrm{cond}\,(S)}{c_m}, \tag{3.16}$$

where $\mathrm{cond}\,(S) := |S||S^{-1}|$. Finally, letting $\mathbf{r}(\boldsymbol{\theta})$ be the function with Fourier coefficients $\hat{\mathbf{r}}(\mathbf{k})$, then (3.13) follows from (3.16) by using Parseval's identity for $\mathbf{r}$ and $\boldsymbol{\psi}$ (e.g., $\|\boldsymbol{\psi}\|^2 = \sum_{\mathbf{k}\in\mathbb{Z}^p} |\hat{\boldsymbol{\psi}}(\mathbf{k})|^2$). L_2 estimates of derivatives of $\mathbf{r}$ in terms of derivatives of $\boldsymbol{\psi}$ follow in the same way. □

Remarks. (i) Assumption (3.12) is needed if no requirements are to be made on the frequency vector $\boldsymbol{\omega} \in \mathbb{R}^p$.

(ii) The constant c_0 depends on the eigenstructure of $\bar{C}$, but not on the function $\boldsymbol{\psi}$, nor on $\mathbf{k}$ or $\boldsymbol{\omega}$.

(iii) Diagonalizability of $\bar{C}$ is unnecessary. Replacing $\bar{C}$'s diagonal form by its Jordan form J, instead, and then transforming J to $\bar{J} = D_\varepsilon^{-1} J D_\varepsilon$, with $D_\varepsilon := \mathrm{diag}\,(1, \varepsilon, \varepsilon^2, \cdots, \varepsilon^{q-1})$, $\varepsilon > 0$, then each Jordan block gets transformed to one with ε above the diagonal. Thus, we can immediately get (as in (3.15)) $|(\bar{J} + i\alpha I)^{-1}| \leqq 2/c_m$

if ε is chosen small enough. We get a different c_0 than in (3.16), but still independent of ψ, $\mathbf{k}$, and $\boldsymbol{\omega}$. In any case, c_0 is larger the closer c_m is to zero.

We proceed in a similar way to bound the discrete solution, now taking a discrete Fourier transform of (3.10). Let $(\mathbf{u}_h, \mathbf{v}_h)_h := [(2\pi)^p/(N_1 \cdots N_p)] \sum_{\boldsymbol{\theta}\in\Omega_h} \langle \mathbf{u}_h(\boldsymbol{\theta}), \mathbf{v}_h(\boldsymbol{\theta})\rangle$ denote the discrete L_2 inner product for grid functions $\mathbf{u}_h, \mathbf{v}_h : \Omega_h \to \mathbb{C}^q$ and $\|\mathbf{u}_h\|_h := (\mathbf{u}_h, \mathbf{u}_h)_h^{1/2}$ be the induced norm. For simplicity, assume that $N_j = 2M_j + 1$, $j = 1, \cdots, p$, are odd, so that $\{e^{i\langle \mathbf{k},\boldsymbol{\theta}\rangle}/(2\pi)^{p/2}, \mathbf{k} \in \mathbb{Z}^p : -M_j \leqq k_j \leqq M_j, j = 1, \cdots, p\}$ is an orthonormal basis for grid functions $\mathbf{v}_h : \Omega_h \to \mathbb{C}$. Then we have Lemma 3.2.

LEMMA 3.2. *Under the assumption of Lemma* 3.1, (3.10) *has a unique solution* $\mathbf{r}_h$. *Furthermore, there is a positive constant* c_0 *such that*

$$\|\mathbf{r}_h\|_h \leqq c_0 \|\boldsymbol{\psi}_h\|_h. \tag{3.17}$$

Proof. The proof is similar to the one in Lemma 3.1. The discrete Fourier transform of (3.10) gives

$$\left[i\left(\frac{\omega_1}{h_1} \sin k_1 h_1 + \cdots + \frac{\omega_p}{h_p} \sin k_p h_p\right) I + \bar{C} \right] \hat{\mathbf{r}}_h(\mathbf{k}) = \hat{\boldsymbol{\psi}}_h(\mathbf{k}), \tag{3.18}$$

$$\mathbf{k} \in \mathbb{Z}^p : -M_j \leqq k_j \leqq M_j, \qquad 1 \leqq j \leqq p,$$

where

$$\boldsymbol{\psi}_h(\boldsymbol{\theta}) = \frac{1}{(2\pi)^{p/2}} \sum_{\substack{\mathbf{k}\in\mathbb{Z}^p \\ -M_j \leqq k_j \leqq M_j}} \hat{\boldsymbol{\psi}}_h(\mathbf{k})\, e^{i\langle \mathbf{k},\boldsymbol{\theta}\rangle},$$

and thus $\hat{\boldsymbol{\psi}}_h(\mathbf{k})$ has components $(\boldsymbol{\psi}_h(\mathbf{k}))_j = (e^{i\langle \mathbf{k},\boldsymbol{\theta}\rangle}, (\boldsymbol{\psi}_h(\boldsymbol{\theta}))_j)_h/(2\pi)^{p/2}$. From (3.15) we immediately get

$$|\hat{\mathbf{r}}_h(\mathbf{k})| \leqq c_0 |\hat{\boldsymbol{\psi}}_h(\mathbf{k})|, \tag{3.19}$$

for $c_0 = \text{cond}\,(S)/c_m$, and letting $\mathbf{r}_h$ be the grid function with coefficients $\hat{\mathbf{r}}_h$, (3.17) follows from Parseval's identity for $\mathbf{r}_h$ and $\boldsymbol{\psi}_h$ (e.g., $\|\boldsymbol{\psi}_h\|_h^2 = \sum_{\mathbf{k}\in\mathbb{Z}^p, -M_j \leqq k_j \leqq M_j} |\hat{\boldsymbol{\psi}}_h(\mathbf{k})|^2$). □

Remarks. (i) Remark (iii) following Lemma 3.1 applies here too. Also, since $c_0 > 0$ is independent of $\mathbf{h} = (h_1, \cdots, h_p)$, the stability estimate (3.17) does not require anything like a CFL condition. This is not a contradiction to the finite speed of propagation, since we are globally discretizing a periodic problem.

(ii) It is worth remarking that assumption (3.12) does not require attractivity of the invariant manifold (in positive time) which would be the case had we instead required the $\text{Re}\,(\lambda_i) > 0$ for all eigenvalues of $\bar{C}$ (see (3.9)).

Given the stability estimate (3.17), it is possible to obtain convergence in the standard ways if $\mathbf{r} = \mathbf{r}(\boldsymbol{\theta})$ is smooth enough. In fact, if $\mathbf{r} \in C^3(T^p)$, by standard arguments we obtain the global error estimate

$$\|\mathbf{r}|_h - \mathbf{r}_h\|_h \leqq c_0 \|\mathbf{r}^{(3)}\|_\infty \max_{1\leqq j\leqq p} h_j^2, \tag{3.20}$$

where $\mathbf{r}|_h$ denotes the restriction of $\mathbf{r}$ to Ω_h, c_0 is given in Lemma 3.1, and $\mathbf{r}^{(3)}$ is the third derivative of $\mathbf{r}$.

An estimate such as (3.20), while showing that the leap-frog discretization is second-order accurate, could be regarded as nonoptimal, in that it requires a bound on the third derivative of the solution for a second-order scheme. This is a standard

shortcoming of a classical finite differences analysis, and it might suggest looking to a finite-element approach (e.g., see [JNP]). Regardless, it is important to understand how to modify the assumption (3.12) for the variable coefficients PDE (3.9), and how the modified assumption relates to the original dynamics. We intend to investigate these various aspects in the future.

4. The compactification algorithm and a modified leap-frog scheme. In this section we consider a modified leap-frog scheme. We study its numerical stability characteristics, and contrast them to those of leap-frog, when a compactification procedure is used to solve the resulting linear systems.

Consider the case $p=2$ and q arbitrary (cf. (3.1)-(3.7)); generalization of the results herein to arbitrary p is possible. Thus, Ω_h is simply the periodic square $\Omega_h=\{(n_1h_1, n_2h_2)\colon n_j\in\mathbb{Z}(\text{mod } N_j),\ j=1,2\}$, on which we introduce a natural vertical ordering:

$$\begin{aligned}\Omega_h=\{&(0,0),(0,h_2),\cdots,(0,(N_2-1)h_2),\\ &(h_1,0),\cdots,(h_1,(N_2-1)h_2),\cdots,((N_1-1)h_1,(N_2-1)h_2)\}.\end{aligned}\tag{4.1}$$

Accordingly, we order $\mathbf{r}_h$ and $\boldsymbol{\psi}_h$ by grouping together the components corresponding to the same vertical strip of Ω_h: $\mathbf{r}_h=(\mathbf{r}_0,\cdots,\mathbf{r}_{N_1-1})$, $\boldsymbol{\psi}_h=(\boldsymbol{\psi}_0,\cdots,\boldsymbol{\psi}_{N_1-1})$. Then (3.10) takes the form

$$S\mathbf{r}_h=\boldsymbol{\psi}_h,\tag{4.2}$$

where S has the "periodic tridiagonal" block structure

$$S=\begin{bmatrix} A_0 & G_0 & & & & & B_0\\ B_1 & A_1 & G_1 & & & & 0\\ & B_2 & A_2 & G_2 & & & \\ & 0 & \ddots & \ddots & \ddots & & \\ & & & B_{N_1-2} & A_{N_1-2} & G_{N_1-2} & \\ G_{N_1-1} & & & & B_{N_1-1} & A_{N_1-1} & \end{bmatrix},\tag{4.3}$$

and each A_i, B_i, $G_i\in\mathbb{R}^{qN_2\times qN_2}$ (and B_i, G_i are diagonal for (3.7)), with the A_i having the same block structure as S, i.e.,

$$A_i=\begin{bmatrix} \tilde{A}_0 & H_0 & & & E_0\\ E_1 & \tilde{A}_1 & H_1 & & 0\\ & \ddots & \ddots & \ddots & \\ & 0 & E_{N_2-2} & \tilde{A}_{N_2-2} & H_{N_2-2}\\ H_{N_2-1} & & & E_{N_21} & \tilde{A}_{N_2-1} \end{bmatrix}.\tag{4.4}$$

where now each $\tilde{A}_j$, E_j, $H_j\in\mathbb{R}^{q\times q}$, with E_j, H_j diagonal and the $\tilde{A}_j$ (generally) full for (3.7). (When $p>2$ and Ω_h is ordered similarly, then S has the same structure as in (4.3), with the A_i's as in (4.4), but now the $\tilde{A}_j$'s also have the same structure as the A_i's and so forth for p stages.)

There are several ways to exploit the sparseness in solving (4.2) efficiently. We could use a sparse Gaussian elimination algorithm directly on (4.3), or reorder the block-unknowns as $1, N_1, 2, N_1-1,\cdots,\lfloor N_1/2\rfloor,\ \lfloor N_1/2\rfloor+1$ (which gives a block-pentadiagonal permuted version of S) and directly apply elimination methods for

banded matrices, or possibly investigate cyclic odd-even reduction methods (e.g., see [He]) or an iterative technique. Here, we have analyzed and implemented a compactification procedure for solving systems like (4.2), but it must be emphasized that the motivation for such a procedure has mainly come from dynamical systems considerations (recall the remark following Lemma 2.1 about the Poincaré map) and not from purely algebraic ones. The procedure consists of a systematic block elimination of unknowns along the θ_1-direction.

Assuming that the matrices G_i are invertible (see later), we have

$$\begin{bmatrix} \mathbf{r}_{i+1} \\ \mathbf{r}_i \end{bmatrix} = \begin{bmatrix} -G_i^{-1}A_i & -G_i^{-1}B_i \\ I & 0 \end{bmatrix} \begin{bmatrix} \mathbf{r}_i \\ \mathbf{r}_{i-1} \end{bmatrix} + \begin{bmatrix} G_i^{-1}\boldsymbol{\psi}_i \\ \mathbf{0} \end{bmatrix}, \qquad i = 0, \cdots, N_1 - 1, \tag{4.5}$$

and the periodicity condition is

$$\begin{bmatrix} \mathbf{r}_{N_1} \\ \mathbf{r}_{N_1-1} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{r}_{-1} \end{bmatrix}. \tag{4.6}$$

Letting

$$\mathbf{R}_i := \begin{bmatrix} \mathbf{r}_i \\ \mathbf{r}_{i-1} \end{bmatrix}, \qquad \boldsymbol{\phi}_i := \begin{bmatrix} G_i^{-1}\boldsymbol{\psi}_i \\ \mathbf{0} \end{bmatrix}, \qquad Q_i := \begin{bmatrix} -G_i^{-1}A_i & -G_i^{-1}B_i \\ I & 0 \end{bmatrix}, \tag{4.7a}$$

repeated use of (4.5) gives

$$\mathbf{R}_{i+1} = \left[\prod_{j=0}^{i} Q_{i-j} \right] \mathbf{R}_0 + \left[\sum_{j=0}^{i-1} \left(\prod_{k=0}^{i-j-1} Q_{i-k} \right) \boldsymbol{\phi}_j + \boldsymbol{\phi}_i \right], \qquad i = 0, 1, \cdots, N_1 - 1, \tag{4.7b}$$

so that with (4.6) we obtain

$$\Gamma \mathbf{R}_0 = \boldsymbol{\gamma},$$

$$\Gamma := I - \prod_{j=0}^{N_1-1} Q_{N_1-1-j} \in \mathbb{R}^{2qN_2 \times 2qN_2}, \qquad \boldsymbol{\gamma} := \sum_{j=0}^{N_1-2} \left(\prod_{k=0}^{N_1-2-j} Q_{i-k} \right) \boldsymbol{\phi}_j + \boldsymbol{\phi}_{N_1-1}. \tag{4.8}$$

After solving the (dense) system (4.8) for $\mathbf{R}_0$ we use (4.7b) to recover $\mathbf{r}_h$. For this compactification procedure, the two basic issues to deal with are feasibility and numerical stability, and we address these next. A positive answer depends upon certain (natural) assumptions and a proper modification of the leap-frog scheme.

When the PDE (2.4) is discretized with leap-frog and solved with Newton's method, then for an initial $\mathbf{r}_h^0 \in \Omega_h$, the next iterate $\mathbf{r}_h^1$ satisfies (3.5)–(3.7). Instead, taking a formal linearization of the PDE (2.4) around a continuous approximation $\mathbf{r}^0 (= \mathbf{r}^0(\boldsymbol{\theta}))$ gives for the next iterate $\mathbf{r}^1$

$$\begin{aligned} &\sum_{j=1}^{p} f_j(\boldsymbol{\theta}, \mathbf{r}^0) \frac{\partial \mathbf{r}^1}{\partial \theta_j} + \left\{ \sum_{j=1}^{p} \left(\frac{\partial \mathbf{r}^0}{\partial \theta_j} \right) \left[\left(\frac{\partial f_j}{\partial r_1}, \cdots, \frac{\partial f_j}{\partial r_q} \right) (\boldsymbol{\theta}, \mathbf{r}^0) \right] \right\} (\mathbf{r}^1 - \mathbf{r}^0) \\ &= \mathbf{g}(\boldsymbol{\theta}, \mathbf{r}^0) + \begin{bmatrix} \frac{\partial g_1}{\partial r_1}, \cdots, \frac{\partial g_1}{\partial r_q} \\ \cdots \\ \frac{\partial g_q}{\partial r_1}, \cdots, \frac{\partial g_q}{\partial r_q} \end{bmatrix}_{(\boldsymbol{\theta}, \mathbf{r}^0)} (\mathbf{r}^1 - \mathbf{r}^0). \end{aligned} \tag{4.9}$$

Discretizing (4.9) with leap-frog (and approximating $\partial \mathbf{r}^0 / \partial \theta_j$ as $D_j \mathbf{r}_h^0$ as well), we obtain exactly (3.7), thus showing that, for leap-frog, discretization and linearization commute.

With same notation as in (3.5)-(3.7), let

$$\mathbf{r}_h^0(\boldsymbol{\theta}) \in V \subseteq \mathbb{R}^q, \qquad \omega_j(\boldsymbol{\theta}) := f_j(\boldsymbol{\theta}, \mathbf{r}_h^0), \qquad \boldsymbol{\theta} \in \Omega_h,$$

$$\mathrm{G}(\boldsymbol{\theta}, \mathbf{r}) := \begin{bmatrix} \nabla g_1 \\ \vdots \\ \nabla g_q \end{bmatrix}, \quad \nabla g_j := \left(\frac{\partial g_j}{\partial r_1}, \cdots, \frac{\partial g_j}{\partial r_q}\right)(\boldsymbol{\theta}, \mathbf{r}), \quad \boldsymbol{\theta} \in T^p, \quad G^0 = G(\boldsymbol{\theta}, \mathbf{r}_h^0), \quad \boldsymbol{\theta} \in \Omega_h,$$

(4.10)

$$F(\boldsymbol{\theta}, \mathbf{r}) := \sum_{j=1}^{p} \frac{\partial \mathbf{r}}{\partial \theta_j}\left[\left(\frac{\partial f_j}{\partial r_1}, \cdots, \frac{\partial f_j}{\partial r_q}\right)(\boldsymbol{\theta}, \mathbf{r})\right], \qquad \bar{C}(\boldsymbol{\theta}, \mathbf{r}) := F - G,$$

$$F^0 := \sum_{j=1}^{p} (D_j \mathbf{r}_h^0) \nabla f_j^0, \qquad \bar{C}^0 := F^0 - G^0.$$

To guarantee feasibility of the compactification procedure for the leap-frog scheme, we shall need the following assumption:

(A1) $\quad f_1(\boldsymbol{\theta}, \mathbf{r}) \geqq c_1 > 0, \qquad \boldsymbol{\theta} \in T^2, \qquad \mathbf{r} \in V.$

Remark on (A1). (i) (A1) can be relaxed by requiring that some $f_j(\boldsymbol{\theta}, \mathbf{r})$ does not vanish, at least for $\mathbf{r}$ sufficiently close to the invariant manifold solution ($\mathbf{r}(\boldsymbol{\theta}) \in V$). Although (A1) is verified for our examples in § 5 it is not clear that such an assumption could not be violated for physically relevant problems.

LEMMA 4.1. *Let* (A1) *hold. If* (2.4) *is discretized with leap-frog and then solved with Newton's method, then the compactification procedure* (4.5)-(4.8) *is well defined.*

Proof. The discretization equations (3.7) have the form

$$\frac{\omega_1(\theta_{ij})}{2h_1}(\mathbf{r}(\theta_{i+1,j}) - \mathbf{r}(\theta_{i-1,j})) + \frac{\omega_2(\theta_{i,j})}{2h_2}(\mathbf{r}(\theta_{i,j+1}) - \mathbf{r}(\theta_{i,j-1})) + \bar{C}(\theta_{ij})\mathbf{r}(\theta_{ij}) = \boldsymbol{\psi}(\theta_{ij}),$$

$$\theta_{ij} = (ih_1, jh_2) \in \Omega_h,$$

and with the ordering in (4.1) the matrices in (4.3), (4.4) are (see (4.10))

$$H_j = -E_j = \frac{\omega_2(\theta_{ij})}{2h_2} I_q, \qquad \tilde{A}_j = \bar{C}(\theta_{ij}),$$

(4.11)

$$G_i = -B_i = \frac{1}{2h_1}\begin{bmatrix} \omega_1(\theta_{i,0}) I_q & & 0 \\ & \ddots & \\ 0 & & \omega_1(\theta_{i,N_2-1}) I_q \end{bmatrix}.$$

The well-definedness of (4.5)-(4.8) is guaranteed by invertibility of G_i, via (A1). □

To address the stability issue, first note that the above compactification procedure in the θ_1 direction amounts to essentially treating θ_1 like a time variable, and the repeated use of (4.5) is a marching, or integration, scheme with stepsize h_1 (see [B] and references therein for similar considerations). Therefore, we are in some sense solving the periodic PDE system (2.4) (or (4.9)) by integrating a time-dependent Cauchy problem. For the latter, it is well known that the zero-order term in the PDE can lead to growing spurious modes for the simple leap-frog scheme [K]; as a result, (4.8) can be highly ill conditioned, as demonstrated by the following analysis and numerical results of § 5. In his analysis for a scalar, linear, homogeneous, time-dependent PDE, Kreiss [K] proposes a modified leap-frog scheme, in order to overcome the numerical instability problems of leap-frog. We have extended his choice to nonlinear PDEs like (2.4) in the following way. Let K_1 denote the averaging operator

between two θ_1 levels defined by

$$(K_1 f)(n_1 h_1, n_2 h_2) = \tfrac{1}{2}[f((n_1+1)h_1, n_2 h_2) + f((n_1-1)h_1, n_2 h_2)]. \tag{4.12}$$

Then the leap-frog scheme (3.4) is replaced by the modified scheme:

$$f_1(\boldsymbol{\theta}, \mathbf{r}_h(\boldsymbol{\theta}))D_1\mathbf{r}_h(\boldsymbol{\theta}) + f_2(\boldsymbol{\theta}, \mathbf{r}_h(\boldsymbol{\theta}))D_2\mathbf{r}_h(\boldsymbol{\theta}) = K_1\mathbf{g}(\boldsymbol{\theta}, \mathbf{r}_h(\boldsymbol{\theta})), \qquad \boldsymbol{\theta} \in \Omega_h. \tag{4.13}$$

As before, to solve the nonlinear system (4.13) we use Newton's method as follows. Given an initial approximation $\mathbf{r}_h^0(\boldsymbol{\theta})$, to obtain $\mathbf{r}_h^1(\boldsymbol{\theta})$ we solve (see (3.5), (3.6))

$$f_1^0 D_1\mathbf{r}_h^1 + f_2^0 D_2\mathbf{r}_h^1 + \left\{\sum_{j=1}^{2} (D_j\mathbf{r}_h^0)\nabla f_j^0\right\}(\mathbf{r}_h^1 - \mathbf{r}_h^0) - K_1 G^0(\mathbf{r}_h^1 - \mathbf{r}_h^0) = K_1\mathbf{g}^0. \tag{4.14}$$

Therefore, the linear system for $\mathbf{r}_h^1 = \mathbf{r}_h$ has the form (cf. (3.7))

$$\sum_{j=1}^{2} \omega_j D_j\mathbf{r}_h + F^0\mathbf{r}_h - K_1(C\mathbf{r}_h) = F^0\mathbf{r}_h^0 + K_1(\mathbf{g}_h^0 - C\mathbf{r}_h^0) =: \boldsymbol{\psi}_h, \qquad \mathbf{r}_h : \Omega_h \to \mathbb{R}^q, \tag{4.15}$$

where F^0 and $C = G^0 \in \mathbb{R}^{q\times q}$, $\boldsymbol{\psi}_h \in \mathbb{R}^q$, and $\omega_j \in \mathbb{R}$ are all known on the grid Ω_h. This is the scheme we use in our computations in § 5; the observed rate of convergence is quadratic throughout.

The matrix associated with the linear system of equations for modified leap-frog has the same periodic tridiagonal block structure as for leap-frog. Hence, we will use the same notation given in (4.2)-(4.4). However, the blocks, in particular the blocks G_i, are now no longer characterized by (4.11), and to guarantee feasibility of the compactification procedure we make the following assumption (recall (4.10) and (A1)):

(A2) Re $\chi_i(\boldsymbol{\theta}, \mathbf{r}) \leqq c_M$, $\chi_i(\boldsymbol{\theta}, \mathbf{r})$ eigenvalue of $G(\boldsymbol{\theta}, \mathbf{r})$, $i = 1, \cdots, q$, $\boldsymbol{\theta} \in T^2$, $\mathbf{r} \in V$; also, either $c_M \leqq 0$ or, if $c_M > 0$, $h_1 c_M < c_1$.

Remark on (A2). (i) (A2) has been formulated in the weakest form needed to guarantee feasibility of the numerical procedure. In practice (see later), (A2) is satisfied with $c_M < 0$, in which case it becomes a more typical assumption (essentially already in [S1]). These are also the cases in which the compactification procedure is of numerical interest, as we will see; otherwise (A2) could force an undesired restriction on h_1.

LEMMA 4.2. *Suppose* (A1) *and* (A2) *hold. Then the scheme* (4.13) *coupled with Newton's method gives a linear system* (4.3), *for which the compactification procedure* (4.5)-(4.8) *is well defined.*

Proof. Rewriting (4.15) as

$$\frac{\omega_1(\theta_{ij})}{2h_1}(\mathbf{r}(\theta_{i+1,j}) - \mathbf{r}(\theta_{i-1,j})) + \frac{\omega_2(\theta_{ij})}{2h_2}(\mathbf{r}(\theta_{i,j+1}) - \mathbf{r}(\theta_{i,j-1})) + F(\theta_{ij})\mathbf{r}(\theta_{ij})$$
$$-\frac{1}{2}C(\theta_{i+1,j})\mathbf{r}(\theta_{i+1,j}) - \frac{1}{2}C(\theta_{i-1,j})\mathbf{r}(\theta_{i-1,j}) = \text{R.H.S.},$$

with the ordering of (4.1), we obtain (4.3), (4.4) with

$$H_j = -E_j = \frac{\omega_2(\theta_{ij})}{2h_2} I_q, \qquad \tilde{A}_j = F(\theta_{ij}),$$

$$G_i = \operatorname{diag}\left(\frac{\omega_1(\theta_{ij})}{2h_1} I_q - \frac{1}{2}C(\theta_{i+1,j}), j = 0, \cdots, N_2 - 1\right)$$

$$B_i = \operatorname{diag}\left(-\frac{\omega_1(\theta_{ij})}{2h_1} I_q - \frac{1}{2}C(\theta_{i-1,j}), j = 0, \cdots, N_2 - 1\right), \tag{4.16}$$

$$i = 0, \cdots, N_1 - 1, \quad j = 0, \cdots, N_2 - 1.$$

The well posedness of (4.5)-(4.8) results if the G_i's are invertible, which is guaranteed by the hypotheses since the real parts of the eigenvalues of the diagonal blocks of G_i are greater than or equal to $c_1/2h_1 - \frac{1}{2}c_M > 0$. □

We now consider the numerical stability properties of the compactification procedure (4.5)-(4.8) for the leap-frog and modified leap-frog schemes. Its interpretation as a marching algorithm is easily highlighted by showing how the matrix (4.8) is computed in practice. In fact, letting $P_0 := I$, $P_{i+1} := Q_i P_i$, $i = 0, 1, \cdots, N_1 - 1$ then, from (4.7), (4.8) we get

$$\Gamma = I - P_{N_1}. \tag{4.17}$$

Thus, we can compute Γ one column at the time (in principle for all columns in parallel) by using (4.7b) with all $\boldsymbol{\phi}_i$'s equal to zero and $\mathbf{R}_0$ a column of the identity matrix, and we can recover $\boldsymbol{\gamma}$ by using (4.7b) with $\mathbf{R}_0 = 0$. This procedure is essentially a *simple shooting* process, with the P_i's being discrete fundamental solution matrices.

For numerical stability, we want the spectral radii of the matrices Q_i to be bounded by 1 and Γ to be well conditioned. In this respect, the leap-frog scheme and its modification behave quite differently. To show it, we restrict consideration to the *constant coefficient* case. (The variable coefficient case is much more involved.) In this case, the PDE and its leap-frog discretization are given by (3.9) and (3.10), respectively (with $p = 2$), and the modified leap-frog scheme is explicitly given by

$$\begin{aligned}&\left(\frac{\omega_1}{2h_1} I + \frac{1}{2}\bar{C}\right)\mathbf{r}((l+1)h_1, jh_2) - \left(\frac{\omega_1}{2h_1} I - \frac{1}{2}\bar{C}\right)\mathbf{r}((l-1)h_1, jh_2) + \omega_2 D_2 \mathbf{r}(lh_1, jh_2)\\ &\quad = \frac{1}{2}\{\boldsymbol{\psi}((l+1)h_1, jh_2) + \boldsymbol{\psi}((l-1)h_1, jh_2)\} =: \boldsymbol{\phi}(lh_1, jh_2).\end{aligned} \tag{4.18}$$

Also, (A1) is simply $\omega_1 > 0$, and the Q_i's in (4.7a) are the same matrix, call it Q: thus (4.17) gives $\Gamma = I - Q^{N_1}$. The following result is elementary.

LEMMA 4.3. *The modified leap-frog scheme* (4.18) *is second-order accurate in the truncation error.*

Proof. Let the second-order centered-difference operator in θ_1 be defined by

$$\begin{aligned}&\Delta_1(\mathbf{f}_h) := \Delta_1(\mathbf{f}(n_1 h_1, n_2 h_2))\\ &\quad = \frac{\mathbf{f}((n+1)h_1, n_2 h_2) - 2\mathbf{f}(n_1 h_1, n_2 h_2) + \mathbf{f}((n_1 - 1)h_1, n_2 h_2)}{2h_1^2}.\end{aligned} \tag{4.19}$$

Then, we have

$$K_1(\mathbf{f}_h) = \mathbf{f}_h + h_1^2 \Delta_1(\mathbf{f}_h), \tag{4.20}$$

from which the result follows by comparing the simple leap-frog scheme and the modified one and using (3.20). □

The study of numerical stability of leap-frog and its modification in a marching situation was originally given by Kreiss [K], for a simple scalar constant coefficient problem. Our analysis is in fact an extension of his work to systems. To this end, we need to have (A2) strengthened as follows:

$$(\overline{\text{A2}}) \qquad \text{Re}\,\lambda_i \geqq c_m > 0, \qquad \lambda_i \text{ eigenvalue of } \bar{C}, \qquad i = 1, \cdots, q.$$

Remarks on ($\overline{\text{A2}}$). (i) ($\overline{\text{A2}}$) is exactly (A2) in the constant coefficients case, with $c_m = -c_M$ and $c_M < 0$.

(ii) Comparing ($\overline{\text{A2}}$) with (3.12) and recalling Remark (ii) after Lemma 3.2, we realize that ($\overline{\text{A2}}$) expresses *attractivity* of the invariant manifold. In practice, if ($\overline{\text{A2}}$)

is not severely violated, the procedure can still be profitably used (see § 5). However, if we desire to deal with genuinely mixed attractivity problems, it is probably better not to apply the compactification procedure and to solve (4.2)-(4.3) by other means.

Assume for simplicity that $N_2 = 2M+1$ and consider the discrete Fourier representation of a grid function $\mathbf{r}(\theta_2)$ (with similar notation to that in § 3),

$$\mathbf{r}(\theta_2) = \frac{1}{\sqrt{2\pi}} \sum_{k=-M}^{M} \hat{r}(k)\, e^{ik\theta_2}, \qquad \theta_2 = jh_2, \qquad (i^2 = -1). \tag{4.21}$$

Therefore, discrete Fourier transformation of (4.18) gives at each level $\theta_1 = lh_1$

$$\left(\frac{\omega_1}{2h_1} I + \frac{1}{2}\bar{C}\right)\hat{\mathbf{r}}_{l+1}(k) - \left(\frac{\omega_1}{2h_1} I - \frac{1}{2}\bar{C}\right)\hat{\mathbf{r}}_{l-1}(k) + i\frac{\omega_2}{h_2}\sin(kh_2)\hat{\mathbf{r}}_l(k) = \hat{\boldsymbol{\phi}}_l(k),$$

$$-M \leqq k \leqq M.$$

By letting

$$\mathbf{R}_l(k) := \begin{bmatrix} \hat{\mathbf{r}}_l(k) \\ \hat{\mathbf{r}}_{l-1}(k) \end{bmatrix}, \qquad \boldsymbol{\phi}_l(k) := \begin{bmatrix} \left(\frac{\omega_1}{2h_1} I + \frac{1}{2}\bar{C}\right)^{-1} \hat{\boldsymbol{\phi}}_l(k) \\ 0 \end{bmatrix},$$

we obtain the discrete Fourier transformation of the marching scheme (4.5),

$$\mathbf{R}_{l+1}(k) = \hat{Q}(k)\mathbf{R}_l(k) + \boldsymbol{\phi}_l(k), \qquad -M \leqq k \leqq M, \tag{4.22a}$$

with

$$\hat{Q}(k) = \begin{bmatrix} -2i\frac{\omega_2}{h_2}\sin(kh_2)\left(\frac{\omega_1}{h_1} I_q + \bar{C}\right)^{-1} & \left(\frac{\omega_1}{h_1} I_q + \bar{C}\right)^{-1}\left(\frac{\omega_1}{h_1} I_q - \bar{C}\right) \\ I_q & 0 \end{bmatrix}. \tag{4.22b}$$

Next, we want to study the spectral properties of $\hat{Q}(k)$. Here $|\hat{Q}| := (\rho(\hat{Q}\hat{Q}^*))^{1/2}$ denotes the spectral norm of $\hat{Q}$ (ρ is the spectral radius and $\hat{Q}^*$ the conjugate transpose of $\hat{Q}$), and $\|\Gamma\| := (\rho(\Gamma\Gamma^*))^{1/2}$ is the spectral norm of Γ. With Parseval's identity, Lemma 4.4 follows immediately.

LEMMA 4.4. *For the matrix* $\Gamma = (I - Q^{N_1})$ *we have*

$$\|\Gamma\|^2 = \max_{-M \leqq k \leqq M} |I - \hat{Q}(k)^{N_1}|^2.$$

Moreover, Γ^{-1} *exists if and only if* $I - \hat{Q}(k)^{N_1}$ *are invertible, and*

$$\|\Gamma^{-1}\|^2 = \max_{-M \leqq k \leqq M} |(I - \hat{Q}(k)^{N_1})^{-1}|^2. \qquad \square$$

Thus, to guarantee invertibility of Γ we show how to enforce $\rho(\hat{Q}(k)) < 1$. Letting

$$\eta = \sin(kh_2), \qquad \tau = \frac{h_1}{\omega_1}, \qquad \gamma = \frac{h_1\omega_2}{h_2\omega_1}, \tag{4.23a}$$

(4.22b) rewrites as

$$\hat{Q}(k) = \begin{bmatrix} -2i\gamma\eta(I + \tau\bar{C})^{-1} & (I + \tau\bar{C})^{-1}(I - \tau\bar{C}) \\ I & 0 \end{bmatrix}. \tag{4.23b}$$

We will need the following lemma.

LEMMA 4.5. *Let $0<\zeta<1$ be a fixed number and consider the matrix*

$$B_0 := \begin{bmatrix} (-2i\gamma\eta)/(1+\tau\lambda) & (1-\tau\lambda)/(1+\tau\lambda) \\ 1 & 0 \end{bmatrix}, \tag{4.24}$$

where $\gamma\in\mathbb{R}$, $\eta\in\mathbb{R}$: $|\eta|\leqq 1$, $\lambda := \lambda_R + i\lambda_I$, $\lambda_R = \mathrm{Re}\,(\lambda)>0$, $\tau>0$. If

$$0<\gamma^2\leqq 1-\zeta^2<1, \tag{4.25}$$

then we have

$$\rho(B_0) = e^{-\lambda_R\tau} + O(\tau^2|\lambda|^2). \tag{4.26}$$

Furthermore, if $\tau|\lambda|$ is sufficiently small (in particular, if $\lambda_I=0$ we need $\tau\lambda_R\leqq\zeta<1$), B_0 can be diagonalized by a similarity transformation $\hat{T}_0$ whose condition number is bounded by a constant which depends only on ζ:

$$\hat{T}_0^{-1}B_0\hat{T}_0 = \begin{bmatrix} x_1 & 0 \\ 0 & x_2 \end{bmatrix}, \qquad \mathrm{cond}\,(\hat{T}_0) = |\hat{T}_0||\hat{T}_0^{-1}| \leqq \mathrm{const.} \tag{4.27}$$

Proof. The eigenvalues, x, of B_0 satisfy $(1+\tau\lambda)x^2+2xi\gamma\eta-(1-\tau\lambda)=0$. If $\lambda_I=0$, then $\tau\lambda_R\leqq\zeta<1$ and (4.25) implies

$$|x_{1,2}|^2 \leqq \left|\frac{1-\tau\lambda_R}{1+\tau\lambda_R}\right| = \frac{1-\tau\lambda_R}{1+\tau\lambda_R} < 1 \qquad \left(\text{and } \frac{1-\tau\lambda_R}{1+\tau\lambda_R} = e^{-2\tau\lambda_R} + O((\tau\lambda_R)^2)\right).$$

In general, the eigenvalues x of B_0 satisfy $\hat{x}^2+2\hat{x}i\gamma\eta\, e^{-\tau\lambda} - e^{-2\tau\lambda}=0$ to within terms of $O(\tau^2|\lambda|^2)$. By letting $\hat{x}=e^{-\tau\lambda}\tilde{x}$, we have $\tilde{x}_{1,2} = -i\gamma\eta \pm (1-(\gamma\eta)^2)^{1/2}$. Then $|\tilde{x}_{1,2}|=1$, and thus $|\hat{x}_{1,2}| = e^{-\tau\lambda_R}$, $|\hat{x}_1-\hat{x}_2| \geqq 2\, e^{-\tau\lambda_R}\zeta$. Therefore, if $\tau|\lambda|$ is sufficiently small, the eigenvalues of B_0 stay separated and (4.26), (4.27) follow. □

We can now show Theorem 4.6.

THEOREM 4.6. *Suppose* (A1) *and* ($\overline{\text{A2}}$) *hold. Let $0<\zeta<1$ be a fixed number and let λ_M be the eigenvalue of $\bar{C}$ of largest modulus. If $\tau|\lambda_M|$ is sufficiently small and* (4.25) *holds, then*

$$\rho(\hat{Q}(k)) \leqq e^{-c_m\tau} + O((\tau|\lambda_M|)^2). \tag{4.28}$$

Let T be the transformation matrix achieving the reduction of $\bar{C}$ to Jordan form. Then, with respect to the spectral norm, we have for Γ

$$\mathrm{cond}\,(\Gamma) := \|\Gamma\|\cdot\|\Gamma^{-1}\| \leqq \mathrm{const.}\frac{[\mathrm{cond}\,(T)\cdot\mathrm{cond}\,(\hat{T})]^2}{1-e^{-2\pi c_m}} \leqq \mathrm{const.}, \tag{4.29}$$

with $\hat{T}$ defined by (4.30).

Proof. Consider the transformation for $\hat{Q}(k)$: $T_1 = \left[\begin{smallmatrix} T & 0 \\ 0 & T \end{smallmatrix}\right]P$, where P is a permutation matrix such that $T_1^{-1}\hat{Q}(k)T_1$ is a block upper triangular matrix $B(k)$, with the following (2×2) blocks along the diagonal

$$B_j(k) = \begin{bmatrix} -2i\gamma\eta/(1+\tau\lambda_j) & (1-\tau\lambda_j)/(1+\tau\lambda_j) \\ 1 & 0 \end{bmatrix}, \qquad j=1,\cdots,q.$$

Thus $I-\hat{Q}(k)^{N_1} = T_1(I-B^{N_1}(k))T_1^{-1}$, $(I-\hat{Q}(k)^{N_1})^{-1} = T_1(I-B^{N_1}(k))^{-1}T_1^{-1}$, and we have that $I-B^{N_1}(k)$ and $(I-B^{N_1}(k))^{-1}$ are both (block) upper triangular matrices whose (block) diagonals are given by the matrices $I_2-B_j^{N_1}(k)$ and $(I_2-B_j^{N_1}(k))^{-1}$, $j=1,\cdots,q$, respectively. Now, from Lemma 4.5, we have that

$$I_2-B_j^{N_1}(k) = \hat{T}_j^{-1}\begin{bmatrix} 1-\lambda_{j_1}^{N_1} & 0 \\ 0 & 1-\lambda_{j_2}^{N_1} \end{bmatrix}\hat{T}_j, \qquad |\lambda_{j_{1,2}}| = e^{-\tau\,\mathrm{Re}\,(\lambda_j)} + O((\tau|\lambda_j|)^2),$$

$$\mathrm{cond}\,(\hat{T}_j) \leqq \mathrm{const.},$$

and thus (4.28) follows at once. By using Lemma 4.4 we also get

$$\max_k |I - \hat{Q}(k)^{N_1}| \max_k |(I - \hat{Q}(k)^{N_1})^{-1}| \leqq (|T_1| \cdot |T_1^{-1}|)^2 \max_k |I - B^{N_1}(k)| \cdot \max_k |(I - B^{N_1}(k))^{-1}|,$$

so that by letting

$$\hat{T} = \operatorname{diag}(\hat{T}_j, j = 1, \cdots, q), \tag{4.30}$$

(4.29) follows immediately. □

We can easily modify the above reasoning to study stability of the compactification procedure for the simple leap-frog scheme. Using the notation of (4.23a) we now have

$$\hat{Q}(k) = \begin{bmatrix} -2i\gamma\eta - 2\tau\bar{C} & I \\ I & 0 \end{bmatrix}. \tag{4.31}$$

Reducing $\bar{C}$ to Jordan form and reasoning as in the proof of Theorem 4.6, the spectral properties of $\hat{Q}(k)$ are given by studying the eigenvalues of

$$B := \begin{bmatrix} -2(i\gamma\eta + \tau\lambda) & 1 \\ 1 & 0 \end{bmatrix} \quad (\lambda \text{ eigenvalue of } \bar{C}).$$

These eigenvalues are $x_{1,2} = -(i\gamma\eta + \tau\lambda) \pm \sqrt{(\gamma\eta)^2 + (\tau\lambda)^2 + 1 + 2i(\gamma\eta)\tau\lambda}$, so, using (4.25), we get

$$x_{1,2} = (-i\gamma\eta \pm \sqrt{1 - (\gamma\eta)^2}) \left(1 \mp \frac{\tau\lambda}{\sqrt{1 - (\gamma\eta)^2}}\right) + O((\tau|\lambda|)^2). \tag{4.32}$$

Thus, $\rho(B) > 1$ unless $\tau\lambda = 0$, reflecting the fact that the compactification procedure in this case is prone to instabilities (confirming the results of [K]).

Remarks. (i) The degree of instability for the compactification scheme with simple leap-frog depends upon how large $c_M = \max_{1 \leqq j \leqq q} \{\operatorname{Re}(\lambda_j), \lambda_j \text{ eigenvalue of } \bar{C}\}$ is. If c_M is small, the procedure could still be reasonably accurate, but in other cases the accuracy is destroyed and Γ becomes numerically singular (see § 5).

(ii) Stability of the compactification procedure for modified leap-frog (4.18) requires that (4.25) be satisfied. This condition is similar to the CFL condition in our setting (recall (4.23a)), and is the price we pay to have introduced a preferred marching direction in the problem. Whether or not (4.25) poses a severe restriction on h_1 and h_2 depends upon the ratio $|\omega_2/\omega_1|$. In the constant coefficients case, it is not really restrictive because we can always reorder the unknowns to have $|\omega_2/\omega_1| \leqq 1$. In the general variable coefficients case, with modified leap-frog given by (4.15), we consider the worst possible ratio, $\max_{\theta \in \Omega_h} |\omega_2(\theta)/\omega_1(\theta)|$, and attempt to satisfy (4.25) with this replacing $|\omega_2/\omega_1|$. The other key assumption of Theorem 4.6 is that $\tau|\lambda_M|$ be small, which could put a severe restriction on the stepsize h_1 for even only moderate values of $|\lambda_M|$. For example, if $\omega_1 = \omega_2$ and all λ_j are real (and positive), then we need $h_1\lambda_M \leqq \zeta < 1$ and $(h_1/h_2)^2 \leqq 1 - \zeta^2$. If all $\lambda_j = 100$, say, then we need $h_1 < 1/100$, but h_2 is essentially unrestricted. If we want two digit accuracy, then discretization error considerations would imply that we only need N_1 and $N_2 \simeq 30$, whereas the above analysis for compactification would still force $N_1 \geqq 630$. The situation is analogous to the one of solving stiff IVPs for ODEs by a scheme with restricted stability region (and the condition $\tau|\lambda_M| < 1$ is reminiscent of the one for Euler's method). Recalling the remarks on the assumptions (A2), (A2), this theoretical restriction on h_1 is more severe

the stronger the attractivity exercised by the invariant manifold itself. All of these considerations are borne out in practice in § 5.

5. Numerical results: van der Pol oscillator and a system of two coupled oscillators. In this section we report on some experimentation aimed at assessing the feasibility of our PDE approach and illustrating the various procedures introduced in the previous sections. Our main concern is in understanding the numerical aspects of this PDE approach, not so much in the most efficient implementation.

As already remarked, we are interested in invariant manifolds parameterized over the 2-torus T^2. We consider the cases of one and two radial coordinates as functions on T^2, representing the invariant manifold. It is worth emphasizing that the technique we propose extends very naturally to several radial coordinates parameterized over several angular coordinates, although of course the dimension (and complexity) of the discretized problem increases accordingly. We present two problems: the forced van der Pol oscillator and a system of two linearly coupled oscillators. Both of these problems are of considerable interest in their own right and have received recent attention from a numerical viewpoint [vV1], [vV2], [ADO]. All computations were performed in single precision on a CDC 855 (machine unit roundoff is about 10^{-14}). The three-dimensional pictures have been produced by using the package DISSPLA. We have not exploited vectorization nor parallelizability in our programs, which will be the subject of future work.

5.1. The van der Pol oscillator. Our first case study is the driven van der Pol oscillator

$$\ddot{x}+\alpha(x^2-1)\dot{x}+x=\beta\cos(\omega t). \tag{5.1}$$

The asymptotic behavior of the solution as $\alpha\to 0$ is clearly analyzed in the book by Guckenheimer and Holmes [GH]. For $\alpha>0$, van Veldhuizen [vV1] presents a numerical study of invariant curves (transverse sections of invariant tori) for several values of the parameters $k=\beta/2\alpha$ and $\sigma=(1-\omega^2)/\alpha$ (note that there is a misprint both in [vV1] and the first edition of [GH]). Without repeating the analysis, we present the results for the invariant torus obtained with our approach when $\alpha=k=\sigma=2/5$.

We first rewrite (5.1) as a first-order system as in [GH]:

$$\begin{aligned} p(x)&=\frac{x^3}{3}-x, \qquad y=\dot{x}+\alpha p(x),\\ \dot{x}&=y-\alpha p(x), \qquad \dot{y}=-x+\beta\cos(\omega t), \end{aligned} \tag{5.2}$$

and then rewrite (5.2) in terms of polar coordinates ($x=r\cos\theta_2$, $y=r\sin\theta_2$), let $\theta_1:=\omega t$, and thus obtain the autonomous dynamical system

$$\begin{aligned} \dot{\theta}_1&=\omega(=\sqrt{.84})=:f_1(\theta_1,\theta_2,r),\\ \dot{\theta}_2&=-1+\frac{1}{r}[\alpha p(r\cos\theta_2)\sin\theta_2+\beta\cos\theta_2\cos\theta_1]=:f_2(\theta_1,\theta_2,r),\\ \dot{r}&=-\alpha\cos\theta_2 p(r\cos\theta_2)+\beta\sin\theta_2\cos\theta_1=:g(\theta_1,\theta_2,r). \end{aligned} \tag{5.3}$$

Assuming that a smooth invariant torus exists and can be globally represented as $r(\boldsymbol{\theta})$ ($\boldsymbol{\theta}:=(\theta_1,\theta_2)\in T^2$), it must be a solution of the scalar nonlinear PDE (2.4),

$$\omega r_{\theta_1}+f_2(\theta_1,\theta_2,r)r_{\theta_2}-g(\theta_1,\theta_2,r)=0. \tag{5.4}$$

Linearization of (5.4) around an approximation $r^0(\theta)$ gives (4.9) for the update $r^1(\theta)$ where we now have (see (4.10) and (5.2), (5.3)):

$$F(\theta_1, \theta_2, r^0) = \frac{\partial r^0}{\partial \theta_2}\left(\frac{2}{3}\alpha r^0 \cos^3\theta_2 \sin\theta_2\right) - \frac{\beta}{(r^0)^2}\cos\theta_2 \cos\theta_1, \tag{5.5}$$

$$G(\theta_1, \theta_2, r^0) = \alpha (r^0)^2 \sin^2\theta_2 \cos^2\theta_2, \qquad \bar{C} = F - G.$$

Note that, from (5.5), in the assumption (A2) of § 4 we have that $c_M > 0$ and thus satisfying (A2) and guaranteeing feasibility of compactification for the modified leap-frog scheme requires restricting h_1 appropriately. (This would not be a severe restriction in this example.) However, the leap-frog scheme used with the compactification algorithm is well defined for any h_1 (see Lemma 4.1 and (5.3)), although there are potential stability difficulties, as seen from the analysis of § 4. With respect to the notation of (4.10) we should generally expect $\bar{C}(\theta)$ to change sign; for the solution trajectory computed below, $\bar{C}(\theta)$ in fact is roughly between $-.1$ and $+2$.

Table 1 gives some results obtained with the leap-frog scheme and compactification along the θ_1 direction to solve (5.4). We have attempted to enforce the CFL-like condition $[h_1\omega_2(\theta)/h_2\omega_1(\theta)]^2 \leqq 1-\zeta^2$; violation of it can produce an extremely ill-conditioned matrix Γ and severely affect the accuracy (see entries 1 and 2 in Table 1). As usual, N_1 and N_2 are the number of discretization points in the θ_1 and θ_2 directions

TABLE 1

N_1	N_2	CPU	NITS	COND	(0, 0)
30	30	5.6″	8(&)	$5 * 10^7$	1.97435784
50	50	30.3″	(*)	$2 * 10^{13}$	1.97691695
40	24	4.4″	8	$3 * 10^3$	1.95472527
50	30	8.5″	8	$2 * 10^3$	1.97133348
100	30	16.3″	8	$6 * 10^2$	1.96688319
400	30	62.9″	8	$4 * 10^2$	1.96255821
1600	30	249.8″	8	10^3	1.96235427
100	60	59.9″	8	$2 * 10^3$	1.97696878
200	120	360″	8	$5 * 10^3$	1.97774020

(&) := iteration stopped at 10^{-8}.
(*) := no improvement after five iterations, and error 10^{-2}.

($h_j = 2\pi/N_j, j = 1, 2$). The Newton iteration was always started with the constant profile $r^0 \equiv 2$, and the iteration stopped when the error (discrete ∞-norm) between two successive iterates differs by less than TOLIT (here, TOLIT is 10^{-12}). Convergence of the Newton's iterates was observed to be quadratic throughout, NITS is the number of required iterations, COND is an estimate for the condition number of the compactified matrix Γ (LINPACK style), and (0, 0) is the approximate value for $r(\theta_1 = 0, \theta_2 = 0)$.

Leap-frog and the compactification algorithm perform adequately for this problem, as long as the condition $((h_1/h_2\omega_1)\max_\theta |\omega_2(\theta)|)^2 < 1$ is satisfied, which it is essentially for all entries of Table 1 after the first two (along our discrete solution trajectory, $\omega_2(\theta)$ is approximately between -1.5 and 0). Further refinement in the θ_1 direction does not improve COND (as expected, refinement in the θ_1 direction increases the cost only linearly in N_1). Finally, the process appears to be clearly second order in the discretization (as seen by comparing the (0, 0) column for the third, fourth, eighth, and ninth entry).

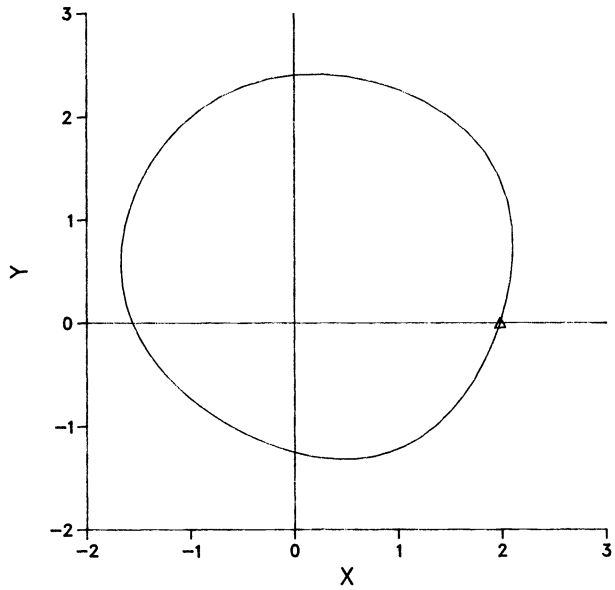


FIG. 1. *Intersection of the invariant torus with* $\mathbb{R}^2$ *for the van der Pol oscillator* (5.2): $\alpha = .4$, $\beta = .32$, $\omega = \sqrt{.84}$.

For the original coordinates x and y of (5.2), the intersection of the computed invariant torus with $\mathbb{R}^2$ (that is, the invariant torus at $t = 0, 2\pi, \cdots$) is given in Fig. 1. This curve is obtained with $N_1 = 100$, $N_2 = 60$, but it is virtually identical to the one obtained when $N_1 = N_2 = 30$ (aside from interpolation between gridpoints). Our results are in close agreement with those in [vV1], although to obtain them van Veldhuizen's algorithm needs to integrate the DE (5.1) 336 times at each iteration. One drawback of direct integration of (5.1) is that it subjects the method to the complex trajectories dynamics on the manifold itself. For example, for the same α and σ and $k = 0.4125$, van Veldhuizen's algorithm necessitates 608 integrations at each iteration step (see [vV1] for details), but our PDE approach is not at all affected by this change in k, e.g., we solve the problem as before with $N_1 = 100$, $N_2 = 60$, starting with $r^0 = 2$, in eight iterations and 60″ of CPU time. Of course, the algorithm of van Veldhuizen presents some advantages over our approach (most notably, it does not need an explicit parameterization) and some other drawbacks (for example, it does not appear immediately extendable to higher-dimensional tori). Still, it is not our intention to compare these two implementations in any detail, but simply to illustrate the potential of the PDE approach.

5.2. System of two coupled oscillators. Our second case study is a system of two linearly coupled nonlinear oscillators. (For the physical significance of this problem, see [ADO].) The problem is modelled by the equations

$$
\begin{aligned}
\dot{x}_1 &= \alpha_1 x_1 + \beta_1 y_1 - (x_1^2 + y_1^2)x_1 - \delta(x_1 + y_1 - x_2 - y_2),\\
\dot{y}_1 &= -\beta_1 x_1 + \alpha_1 y_1 - (x_1^2 + y_1^2)y_1 - \delta(x_1 + y_1 - x_2 - y_2),\\
\dot{x}_2 &= \alpha_2 x_2 + \beta_2 y_2 - (x_2^2 + y_2^2)x_2 + \delta(x_1 + y_1 - x_2 - y_2),\\
\dot{y}_2 &= -\beta_2 x_2 + \alpha_2 y_2 - (x_2^2 + y_2^2)y_2 + \delta(x_1 + y_1 - x_2 - y_2),
\end{aligned}
\tag{5.6}
$$

with $\alpha_i, \beta_i > 0$ and coupling parameter $\delta \geqq 0$. We refer to [ADO] on why the form of the cubic nonlinearity of each oscillator is an appropriate one and why we can consider

linear coupling. When $\delta = 0$, there are two distinct oscillators, having stable, attracting, periodic limit cycles given by $x_i^2 + y_i^2 = \alpha_i$ and time period of the orbit $T_i = 2\pi/\beta_i$ $(i = 1, 2)$. Therefore, for $\delta = 0$ the system (5.6) has a stable, attracting, invariant torus given by the cartesian product of these two circles. Such a torus persists for weak coupling (e.g., see [F]), but disappears for strong coupling [ADO]. As usual, it is the deformation of the torus for the intermediate range of values of δ which is of interest.

To apply the PDE approach to this problem, we represent (5.6) in terms of a polar coordinate representation for the planes (x_i, y_i). That is, we let

$$x_i = r_i \cos\theta_i, \qquad y_i = -r_i \sin\theta_i \quad (i = 1, 2) \tag{5.7}$$

(for convenience, using $-\theta$ rather than $+\theta$), and thus we obtain a system of the form (2.1) with $p = q = 2$:

$$\dot{\boldsymbol{\theta}} = \mathbf{f}(\boldsymbol{\theta}, \mathbf{r}), \qquad \dot{\mathbf{r}} = \mathbf{g}(\boldsymbol{\theta}, \mathbf{r}), \qquad \boldsymbol{\theta} = (\theta_1, \theta_2), \qquad \mathbf{r} = (r_1, r_2), \tag{5.8a}$$

where

$$\mathbf{f}(\boldsymbol{\theta}, \mathbf{r}) := \begin{bmatrix} \beta_1 + \delta\left\{\cos 2\theta_1 - \dfrac{r_2}{r_1}[\sin(\theta_1 - \theta_2) + \cos(\theta_1 + \theta_2)]\right\} \\ \beta_2 + \delta\left\{\cos 2\theta_2 - \dfrac{r_1}{r_2}[\cos(\theta_1 + \theta_2) + \sin(\theta_2 - \theta_1)]\right\} \end{bmatrix},$$

$$\mathbf{g}(\boldsymbol{\theta}, \mathbf{r}) := \begin{bmatrix} r_1(\alpha_1 - r_1^2) - \delta\{r_1(1 - \sin 2\theta_1) + r_2[\sin(\theta_1 + \theta_2) - \cos(\theta_1 - \theta_2)]\} \\ r_2(\alpha_2 - r_2^2) - \delta\{r_1[\sin(\theta_1 + \theta_2) - \cos(\theta_1 - \theta_2)] + r_2(1 - \sin 2\theta_2)\} \end{bmatrix}. \tag{5.8b}$$

We then seek the invariant torus parameterized over $\boldsymbol{\theta}$ and $\mathbf{r} = \mathbf{r}(\boldsymbol{\theta})$ for which (2.4), (2.5) leads to the problem

$$f_1(\boldsymbol{\theta}, \mathbf{r}(\boldsymbol{\theta}))\begin{bmatrix} \dfrac{\partial r_1}{\partial \theta_1} \\ \dfrac{\partial r_2}{\partial \theta_1} \end{bmatrix} + f_2(\boldsymbol{\theta}, \mathbf{r}(\boldsymbol{\theta}))\begin{bmatrix} \dfrac{\partial r_1}{\partial \theta_2} \\ \dfrac{\partial r_2}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} g_1(\boldsymbol{\theta}, \mathbf{r}(\boldsymbol{\theta})) \\ g_2(\boldsymbol{\theta}, \mathbf{r}(\boldsymbol{\theta})) \end{bmatrix}, \tag{5.9}$$

$$r_i(0, \theta_2) = r_i(2\pi, \theta_2), \qquad r_i(\theta_1, 0) = r_i(\theta_1, 2\pi), \qquad i = 1, 2,$$

with f_i, g_i defined from (5.8b).

Actually, we find it convenient to further introduce new coordinates z_i, $i = 1, 2$ (also proposed, but not used, in [ADO]) defined by the relation

$$r_i = \sqrt{\alpha_i} + \delta z_i, \qquad i = 1, 2. \tag{5.10}$$

Under this linear change of coordinates we obtain the new system

$$\dot{\boldsymbol{\theta}} = \mathbf{f}(\boldsymbol{\theta}, \mathbf{z}), \qquad \dot{\mathbf{z}} = \mathbf{g}(\boldsymbol{\theta}, \mathbf{z}), \tag{5.11a}$$

where

$$\mathbf{f}(\boldsymbol{\theta}, \mathbf{z}) := \begin{bmatrix} \beta_1 + \delta\left\{\cos 2\theta_1 - \dfrac{\sqrt{\alpha_2} + \delta z_2}{\sqrt{\alpha_1} + \delta z_1}[\sin(\theta_1 - \theta_2) + \cos(\theta_1 + \theta_2)]\right\} \\ \beta_2 + \delta\left\{\cos 2\theta_2 - \dfrac{\sqrt{\alpha_1} + \delta z_1}{\sqrt{\alpha_2} + \delta z_2}[\cos(\theta_1 + \theta_2) + \sin(\theta_2 - \theta_1)]\right\} \end{bmatrix},$$

$$\mathbf{g}(\boldsymbol{\theta}, \mathbf{z}) := \begin{bmatrix} -2\alpha_1 z_1 - \delta z_1^2(\delta z_1 + 3\sqrt{\alpha_1}) \\ \quad -\{(\sqrt{\alpha_1} + \delta z_1)(1 - \sin 2\theta_1) + (\sqrt{\alpha_2} + \delta z_2)[\sin(\theta_1 + \theta_2) - \cos(\theta_1 - \theta_2)]\} \\ -2\alpha_2 z_2 - \delta z_2^2(\delta z_2 + 3\sqrt{\alpha_2}) \\ \quad -\{(\sqrt{\alpha_1} + \delta z_1)[\sin(\theta_1 + \theta_2) - \cos(\theta_1 - \theta_2)] + (\sqrt{\alpha_2} + \delta z_2)(1 - \sin 2\theta_2)\} \end{bmatrix}, \tag{5.11b}$$

and the invariant manifold condition translates into the PDE system

$$f_1(\boldsymbol{\theta}, \mathbf{z}(\boldsymbol{\theta}))\frac{\partial \mathbf{z}}{\partial \theta_1}+f_2(\boldsymbol{\theta}, \mathbf{z}(\boldsymbol{\theta}))\frac{\partial \mathbf{z}}{\partial \theta_2}=\mathbf{g}(\boldsymbol{\theta}, \mathbf{z}(\boldsymbol{\theta})), \tag{5.12}$$

$$z_i(0, \theta_2)=z_i(2\pi, \theta_2), \qquad z_i(\theta_1, 0)=z_i(\theta_1, 2\pi), \qquad i=1, 2.$$

(For the new "radial" coordinates z_i, there is no guarantee in general that $z_i \geqq 0$.) There are several symmetries and other coordinate choices which can be used in this problem. In [ADO], where a very detailed bifurcation study is made for various values of α_i, β_i, and δ, such symmetries are more carefully exploited; in particular, they use a coordinate representation relative to a symmetric/antisymmetric splitting of the solution space (see eq. (28) in [ADO]), which might also be interesting with our approach. However, here we consider only the system (5.12) and the parameter values

$$\alpha_1=\alpha_2=1, \qquad \beta_1=\beta_2=\beta=.55, \qquad \delta \geqq 0 \tag{5.13}$$

(for which a numerical bifurcation study is done in [ADO]), and do not directly enforce any symmetry in the numerical algorithm.

Numerically, we have found no relevant difference between solving (5.9) or (5.12), but analytically, for $\delta=0$ (5.12) gives two linear, decoupled PDEs which can be solved exactly. Specifically, we have Lemma 5.1.

LEMMA 5.1. *If* (5.13) *holds with* $\delta=0$, *then the* PDE (5.12) *has the solution*

(5.14)

$$z_1(\boldsymbol{\theta})=-\left[\frac{1-\cos\bar{\theta}}{2}+\frac{2\beta\cos 2\theta_1-2\sin 2\theta_1+2\sin(2\theta_1+\bar{\theta})-2\beta\cos(2\theta_1+\bar{\theta})}{4+4\beta^2}\right],$$

$$z_2(\boldsymbol{\theta})=-\left[\frac{1-\cos\bar{\theta}}{2}\right.$$

$$\left.+\frac{2\beta\cos(2(\theta_1+\bar{\theta}))-2\sin(2(\theta_1+\bar{\theta}))+2\sin(2\theta_1+\bar{\theta})-2\beta\cos(2\theta_1+\bar{\theta})}{4+4\beta^2}\right]$$

along lines $\theta_2=\theta_1+\bar{\theta}$.

Proof. When $\delta=0$, (5.12) decouples into the two linear PDEs

$$\frac{\partial z_i}{\partial \theta_1}+\frac{\partial z_i}{\partial \theta_2}=-\frac{2}{\beta}z_i-\frac{1}{\beta}(1-\sin 2\theta_i)-\frac{1}{\beta}[\sin(\theta_1+\theta_2)-\cos(\theta_1-\theta_2)], \qquad i=1, 2.$$

Thus, for both $i=1, 2$, the characteristics are given by

$$\frac{d\theta_2}{d\theta_1}=1, \qquad \text{or } \theta_2=\theta_1+\bar{\theta} \quad (\bar{\theta}\in[0, 2\pi]).$$

Along these curves, for z_1 (and similarly for z_2) we simply solve

$$\frac{dz_1}{d\theta_1}=-\frac{2}{\beta}z_1-\frac{1}{\beta}(1-\sin 2\theta_1)-\frac{1}{\beta}[\sin(2\theta_1+\bar{\theta})-\cos\bar{\theta}_1],$$

which, using periodicity, gives the solution (5.14). □

Remarks. (i) When $\bar{\theta}_1=0$ we have $z_1=z_2=0$; also, from (5.14) z_1 and z_2 are bounded by $\frac{1}{2}$ (say).

(ii) For $\delta>0$ we have no explicit solution, but as long as the solution is unique, by symmetry

$$z_1=z_2=0 \quad \text{along } \theta_1=\theta_2, \qquad \delta \geqq 0, \tag{5.15}$$

which we use for checking the accuracy of our numerical results.

Linearization of (5.12) around a trajectory $\mathbf{z}^0(\boldsymbol{\theta})$ gives the equation (4.9) (with $\mathbf{z}$ replacing $\mathbf{r}$), where

$$f_1(\boldsymbol{\theta}, \mathbf{z}) = \beta + \delta\left\{\cos 2\theta_1 - \frac{1+\delta z_2^0(\boldsymbol{\theta})}{1+\delta z_1^0(\boldsymbol{\theta})}[\sin(\theta_1-\theta_2)+\cos(\theta_1+\theta_2)]\right\}$$

$$F = (F_{ij})_{i,j=1,2}, \qquad G = (G_{ij})_{i,j=1,2}, \qquad \bar{C} = F - G, \tag{5.16}$$

$$F_{ij} = \left(\frac{\partial z_1^0}{\partial \theta_1}\frac{\partial f_1}{\partial z_j} + \frac{\partial z_1^0}{\partial \theta_2}\frac{\partial f_2}{\partial z_j}\right)(\mathbf{z}^0(\boldsymbol{\theta})), \qquad G_{ij} = \left(\frac{\partial g_i}{\partial z_j}\right)(\mathbf{z}^0(\boldsymbol{\theta})),$$

with

$$\frac{\partial f_1}{\partial z_1} = \delta^2\frac{(1+\delta z_2)}{(1+\delta z_1)^2}[\sin(\theta_1-\theta_2)+\cos(\theta_1+\theta_2)],$$

$$\frac{\partial f_1}{\partial z_2} = -\frac{\delta^2}{1+\delta z_2}[\sin(\theta_1-\theta_2)+\cos(\theta_1+\theta_2)], \tag{5.17}$$

$$\frac{\partial g_1}{\partial z_1} = -2-\delta[1-\sin 2\theta_1+6z_1+3\delta z_1^2], \qquad \frac{\partial g_1}{\partial z_2} = -\delta[\sin(\theta_1+\theta_2)-\cos(\theta_1-\theta_2)],$$

and f_2, $\partial f_2/\partial z_1$, $\partial f_2/\partial z_2$, $\partial g_2/\partial z_1$, and $\partial g_2/\partial z_2$ of the same form as f_1, $\partial f_1/\partial z_2$, $\partial f_1/\partial z_1$, $\partial g_1/\partial z_2$, and $\partial g_1/\partial z_1$, except the indices 1 and 2 are interchanged. It follows immediately (with (5.13)) that for δ sufficiently small (A1) and (A2) with $c_M < 0$ are satisfied for arbitrary $\mathbf{z}$. Thus, modified leap-frog with compactification is well defined for small δ by Lemma 4.2. For increasing values of δ, we can numerically check for validity of the assumptions (A1), (A2) along the computed solution trajectory (using (5.16), (5.17)).

Unlike for the PDE (5.4) where θ_1 is essentially a time coordinate, for the PDE (5.12) there is no preferred direction along which to compactify. Here, compactification is done along the θ_2 direction (interpretation of the results of § 4 should be done with the obvious interchange of indices). The results reported in the tables below are all relative to using modified leap-frog and this compactification.

Table 2 gives the results for $\delta = 0$, with initial guess $z_1^0 = z_2^0 = 0$. (This is done to check the accuracy of the procedure.) N_1, N_2, and COND are as before, and E_1 and E_2 are the errors in the ∞-norm computed using (5.15). Clearly, convergence of the Newton iteration only requires one iteration. The results clearly show the second-order convergence in $\mathbf{h} = (h_1, h_2)$. Figures 2 and 3 show plots of z_1 and z_2 on a rectangular grid $0 \leqq \theta_1, \theta_2 \leqq 2\pi$, from the viewpoint $(\theta_1, \theta_2, z) = (2, -5, 1)$, obtained when $N_1 = 60$, $N_2 = 120$. Even with $\delta = 0$, leap-frog with compactification gives very poor results as expected from § 4; in particular, with $N_1 = 30$ and $N_2 = 60$, Γ has a condition number of about 10^{11}, and when $N_1 = 60$ and $N_2 = 120$, Γ is numerically singular.

For the case of $\delta = 0$, we have (see (5.16)) that $\omega_1 = \omega_2 = \beta$ and thus the conditions needed for Theorem 4.6 (for compactification in the θ_2-direction) are that

TABLE 2

N_1	N_2	COND	E_1	E_2
30	30	3	.0014	.0099
15	30	1	.00252	.01227
30	60	1	.00635	.00293
60	120	1	.00159	.000726

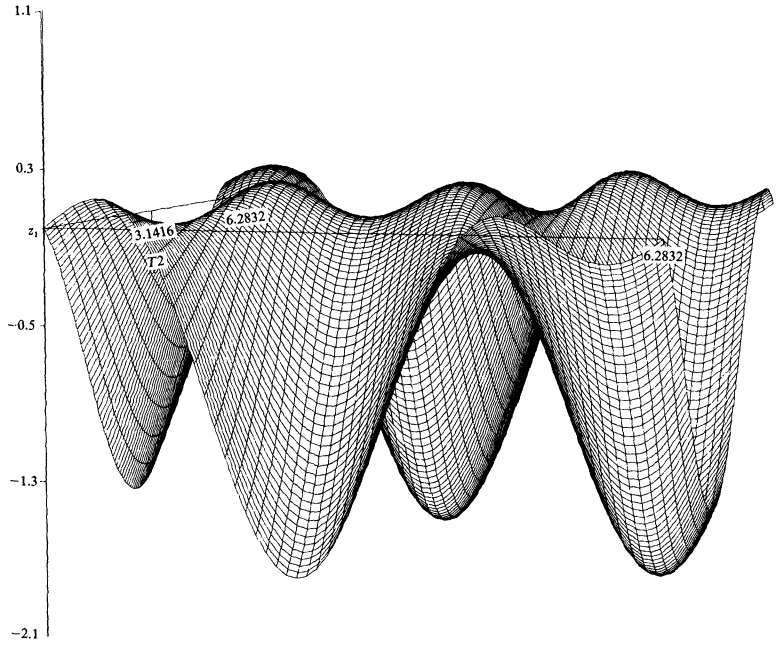


FIG. 2. *Coupled oscillator.* $\delta = 0$, *view*: $(\theta_1, \theta_2, z_1) = (2, -5, 1)$.

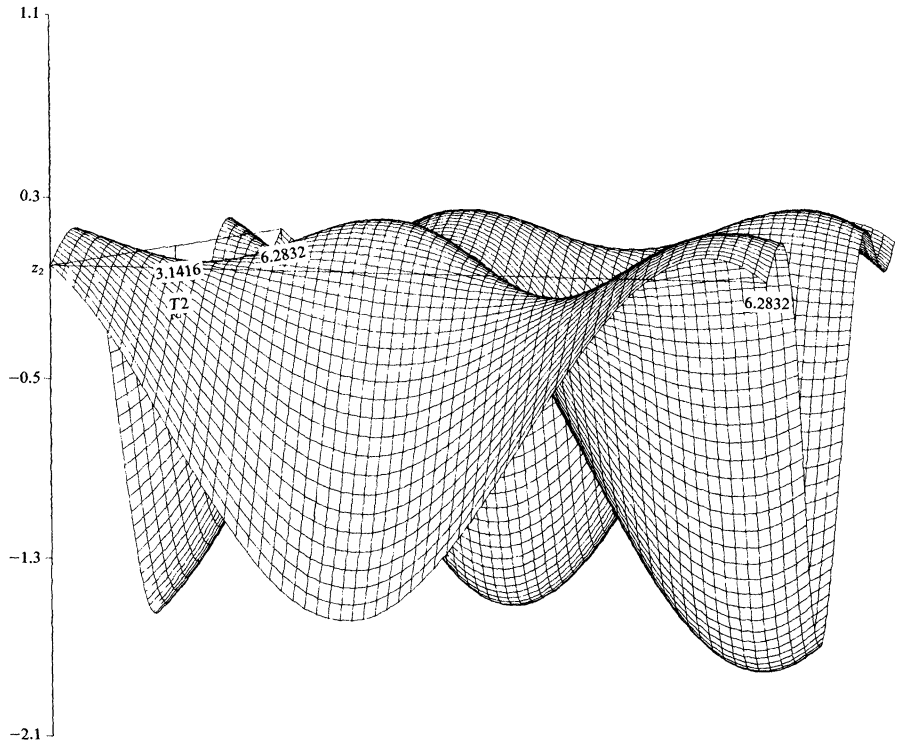


FIG. 3. *Coupled oscillator* $\delta = 0$, *view*: $(\theta_1, \theta_2, z_2) = (2, -5, 1)$.

$(h_2\omega_1/h_1\omega_2)^2 < 1-\zeta^2$ (cf. (4.25)) and that $\lambda_j \cdot h_2/\omega_2$ is small for eigenvalues λ_j of $\bar{C}$. By using (5.13), (5.16), (5.17) these amount to requiring

$$\frac{23}{N_2} < \zeta < 1, \qquad \left(\frac{N_1}{N_2}\right)^2 < 1-\zeta^2,$$

both of which are satisfied for the last three entries in Table 2.

When $\delta > 0$, we will attempt to satisfy these conditions, but since ω_1 and ω_2 are not constants, we replace ω_1/ω_2 in the inequality by its largest value on the grid; i.e., we attempt enforcing

$$\left(\frac{N_1}{N_2}\mu\right)^2 < 1-\zeta^2, \qquad \mu := \max_{\theta} \left|\frac{\omega_1(\theta)}{\omega_2(\theta)}\right| \tag{5.18}$$

all along the current solution trajectory. Table 3 shows results for various δ's. The initial profile is the solution at the previous value of δ, and the iteration stopped when the two most recent iterates differed by less than 10^{-12}. Convergence of the Newton iterates was quadratic throughout. As usual, NITS is the number of required iterations, COND is a measure of the condition number of Γ, E_1 and E_2 are estimates of the errors in z_1 and z_2 along $\theta_1 = \theta_2$ (obtained from (5.15)), and μ is obtained on the grid at convergence.

TABLE 3

N_1	N_2	δ	COND	NITS	E_1	E_2	μ
60	120	10^{-4}	1	3	.001	.001	<2
60	120	10^{-2}	1	3	.001	.001	<2
60	120	.1	1	5	.002	.001	2.15
60	120	.15	$9 * 10^{11}$	$4^{(1)}$	.003	.002	3.4
30	90	.15	1	$7^{(2)}$	.002	.001	3.4

(1) := iteration stopped at 10^{-7}.
(2) := initial guess here is $z_1^0 = z_2^0 = 0$.

For $\delta = 10^{-2}$, Figs. 4 and 5 show r_1 and r_2 (see (5.10)), the viewpoint being as for Figs. 2 and 3. From Table 3, we see that (5.18) is severely violated when $N_1 = 60$, $N_2 = 120$, and $\delta = .15$; this is reflected in Γ being very ill conditioned and in large computational errors (not detected along the lines $\theta_1 = \theta_2$), as can be seen by comparing the computed z_1 and z_2 in this case to approximations obtained when satisfying (5.18). Interestingly, when (5.18) is not severely violated (as when $N_1 = 30$, $N_2 = 90$, $\delta = .15$), no major difficulties arise. The results which follow are obtained by strictly enforcing (5.18). In order to continue using simple continuation in δ (results obtained at $\delta = \delta_1$ are used to start the iteration at $\delta = \delta_2$, $\delta_1 < \delta_2$) when grid sizes change, we use (linear) interpolation on the previous grid. For several δ values, runs have also been made by starting the iteration with the initial profile $z_1^0 = z_2^0 = 0$, and in all cases the same results were obtained; such robustness with respect to the initial guess seems to be an advantage of this PDE approach.

Table 4 gives results for $\delta \leqq .2527$. When we attempt to continue to $\delta = .2528$, the estimate for μ is about 50 and we stop after the first iteration. In fact, the global coordinate representation (5.7) becomes invalid and the procedure unfeasible in these coordinates. To illustrate this, we summarize in Table 5 the results from computing

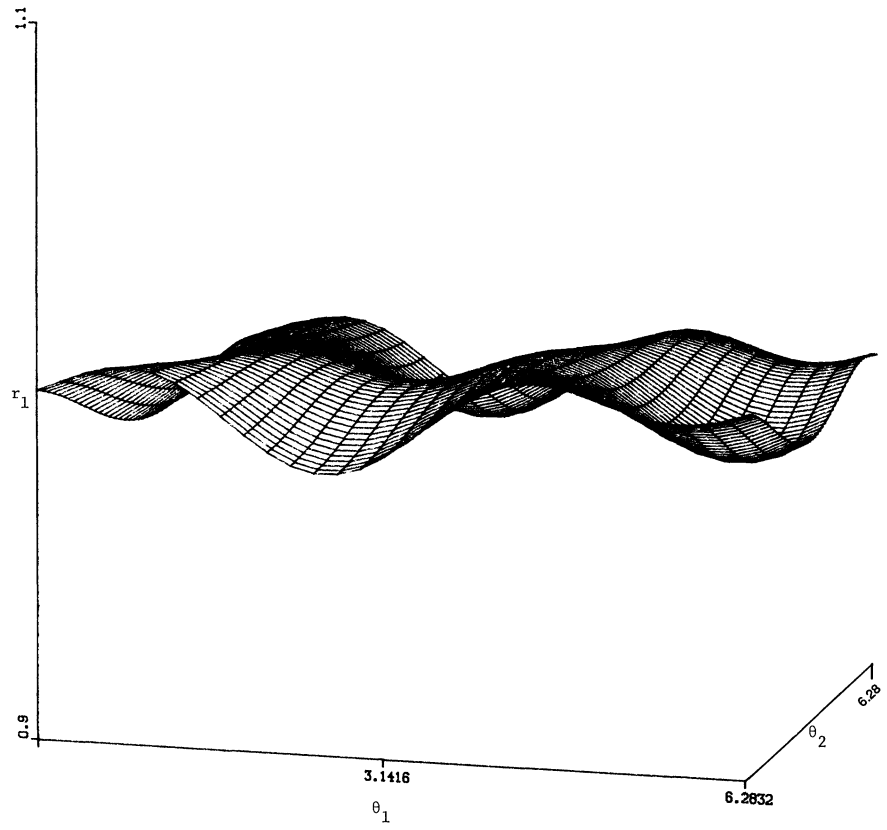


FIG. 4. *Coupled oscillator.* $\delta = 10^{-2}$, *view*: $(\theta_1, \theta_2, r_1) = (2, -5, 1)$.

z_1, z_2 and then forming r_1, r_2 via (5.10) for various δ's, from which it is clear that the representation (5.7) breaks down. Figures 6–13 are pairwise plots of z_1, z_2 for $\delta = .2$, .23, .2525, .2527 from the viewpoint $(\theta_1, \theta_2, z) = (-7, -7, 1)$. The deformation of the tori is quite dramatic. For $\delta = .2527$, Figs. 14, 15 show z_1 and z_2 from the viewpoint $(\theta_1, \theta_2, z) = (2, -5, 1)$ and Fig. 16 shows z_1 from the viewpoint (13, 3, 1). It appears from these figures that the torus is losing smoothness, as can be observed from the development of peeklike singularities. The interplay of this with the loss of validity for the global representation (5.7) is appreciated by looking at Figs. 17 and 18, where the intersection of $r_1(\theta_1, \theta_2)$ with the planes $\theta_1 = 0$, $\theta_1 = 2\pi/15$, respectively, are given for several values of δ. Figures 19 and 20 show enlargements of part of these plots. The surface $r_2(\theta_1, \theta_2)$ is much better behaved, as Table 5 suggests and as graphically confirmed by Figs. 21, 22 where we show the tori $r_2(\theta_1, \theta_2)$ (in toroidal coordinates with internal radius = 2) for $\delta = .2525$, .2527, respectively. What the graphics seem to imply is that there are tori bifurcating from the original torus (see also [ADO]). We leave this investigation for future work.

We conclude this section by pointing out that once the invariant tori have been computed, it is an easy matter to put the flow on them, without integrating the differential system. This can be simply done (recall (2.1)–(2.3)) by drawing the vector field of $\dot{\theta} = f(\theta, r(\theta))$. In Fig. 23 we show this, in the periodic (θ_1, θ_2)-plane, for $r_2(\theta_1, \theta_2)$ and $\delta = .2527$. The symmetric and periodic behavior are obviously quite well depicted.

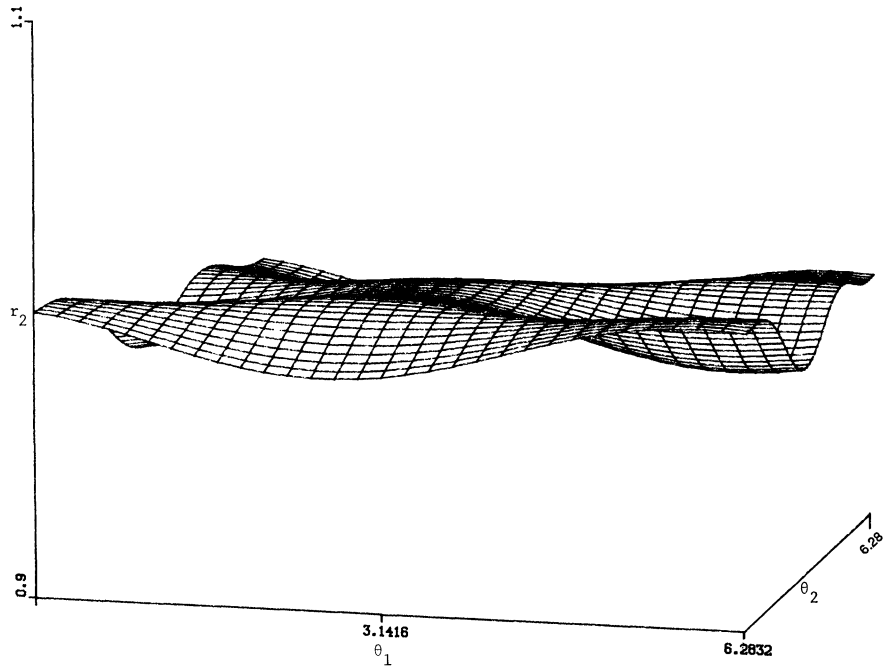


FIG. 5. *Coupled oscillator.* $\delta = 10^{-2}$, *view*: $(\theta_1, \theta_2, r) = (2, -5, 1)$.

TABLE 4

N_1	N_2	δ	COND	NITS	E_1	E_2	μ
30	120	.2	1.01	$8^{(1)}$	.05	.05	5.95
30	300	.2	1.0004	$3^{(2)}$	.05	.05	5.96
30	600	.23	1.006	6	.01	.02	8.45
30	600	.25	1.1	$8^{(3)}$	.03	.04	10.3
30	600	.252	1.18	6	.07	.08	10.4
30	600	.2525	1.2	6	.07	.08	10.5
30	600	.2527	3.4	7	.06	.07	17.8
30	1200	.2527	2.9	$4^{(4)}$	.06	.07	17.8

(1) := with initial guess $z_1^0 = z_2^0 = 0$.
(2) := initial profile from $\delta = .2$, $N_1 = 30$, $N_2 = 120$.
(3) := initial guess from $\delta = .2$.
(4) := initial profile from $\delta = .2527$, $N_1 = 30$, $N_2 = 600$.

TABLE 5

δ	min r_1	min r_2
.250	.1957	.2774
.252	.1027	.2562
.2525	.05039	.2512
.2527	.01761	.2490
$.2528^{(1)}$	.00646	.2481
$.253^{(1)}$	−.01301	.2461

(1) := after one iteration.

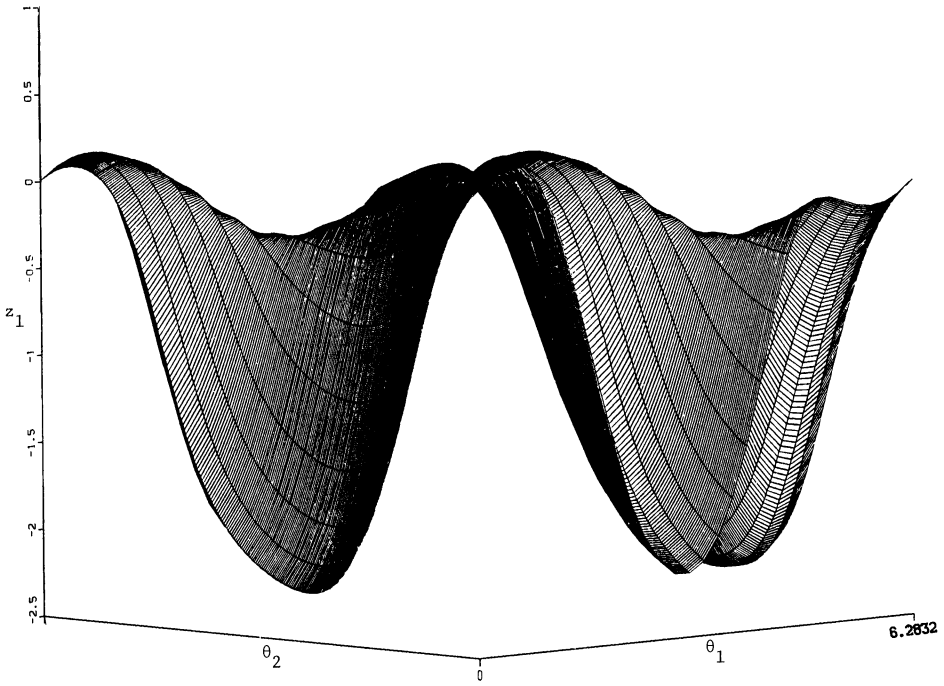


FIG. 6. *Coupled oscillator.* $\delta = .2$, *view*: $(\theta_2, \theta_2, z_1) = (-7, -7, 1)$.

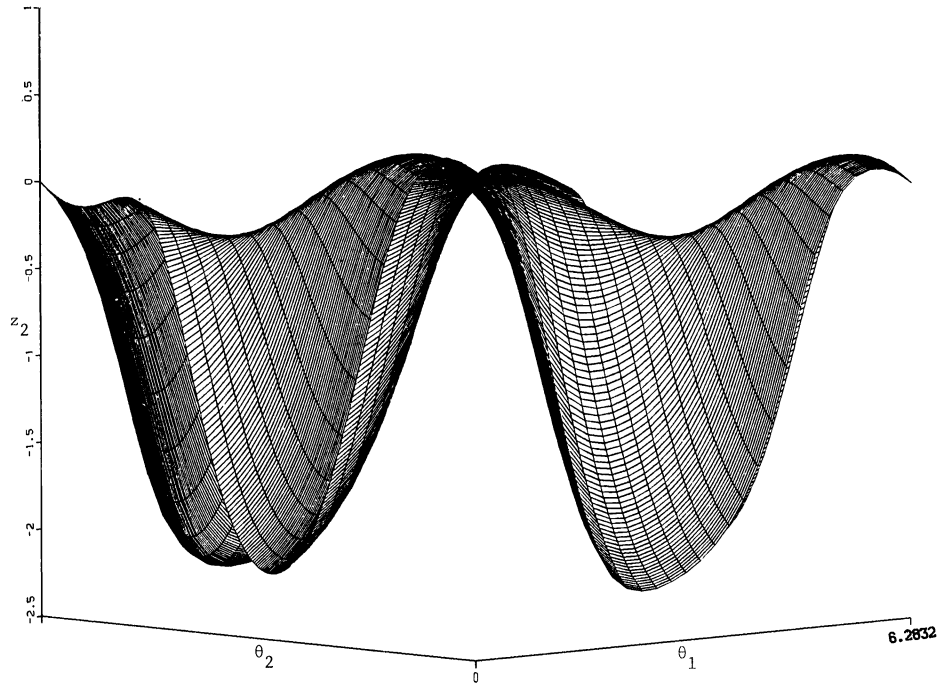


FIG. 7. *Coupled oscillator.* $\delta = .2$, *view*: $(\theta_1, \theta_2, z_2) = (-7, -7, 1)$.

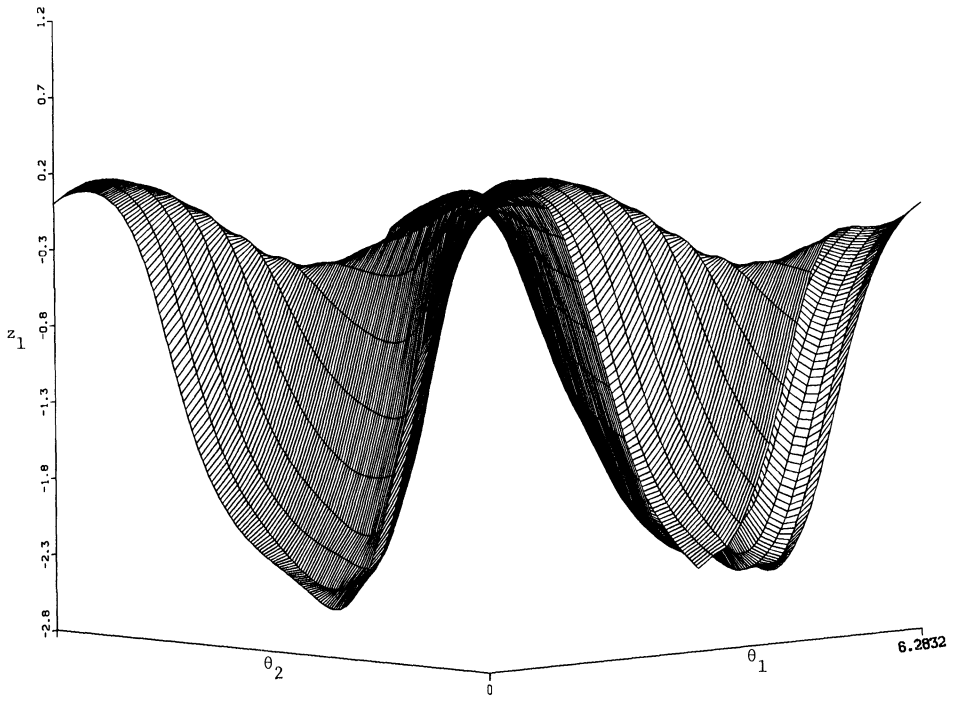


FIG. 8. *Coupled oscillator.* $\delta = .23$, *view*: $(\theta_1, \theta_2, z_1) = (-7, -7, 1)$.

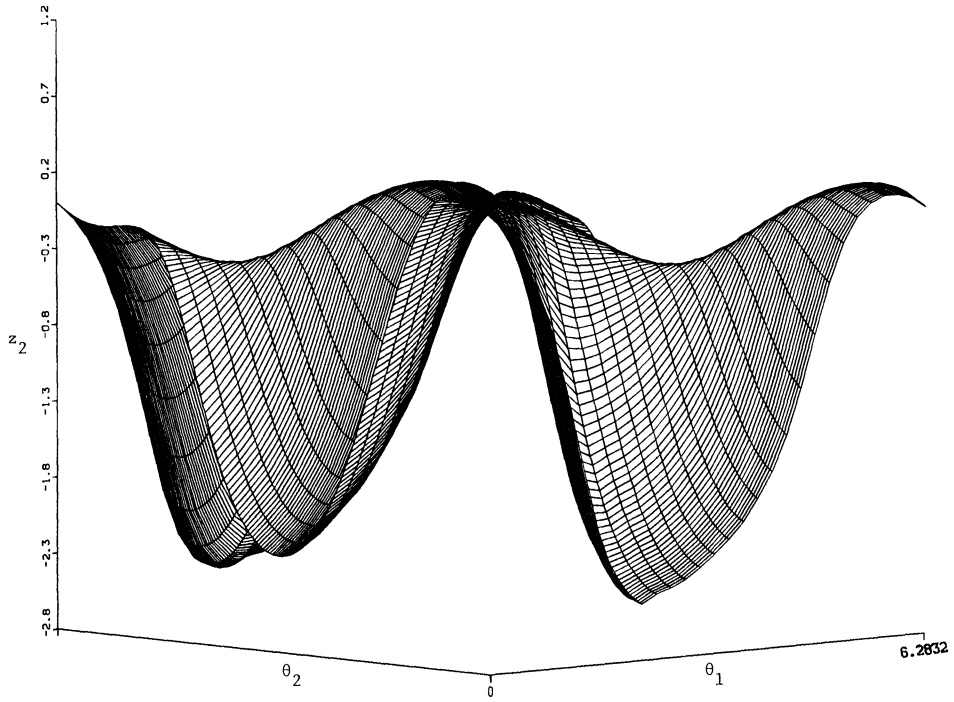


FIG. 9. *Coupled oscillator.* $\delta = .23$, *view*: $(\theta_1, \theta_2, z_2) = (-7, -7, 1)$.

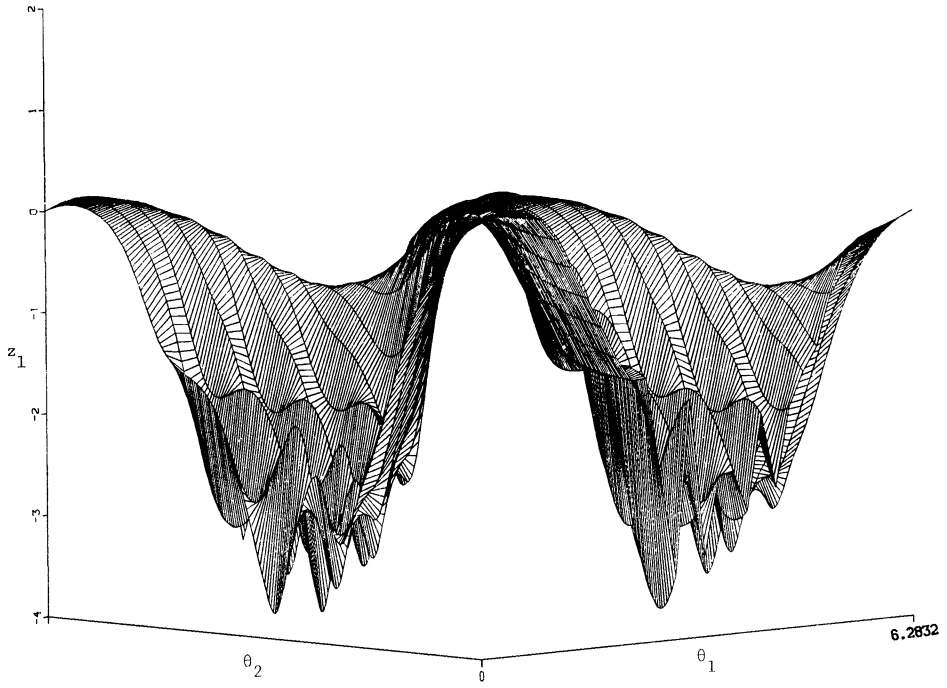


FIG. 10. *Coupled oscillator.* $\delta = .2525$, *view*: $(\theta_1, \theta_2, z_1) = (-7, -7, 1)$.

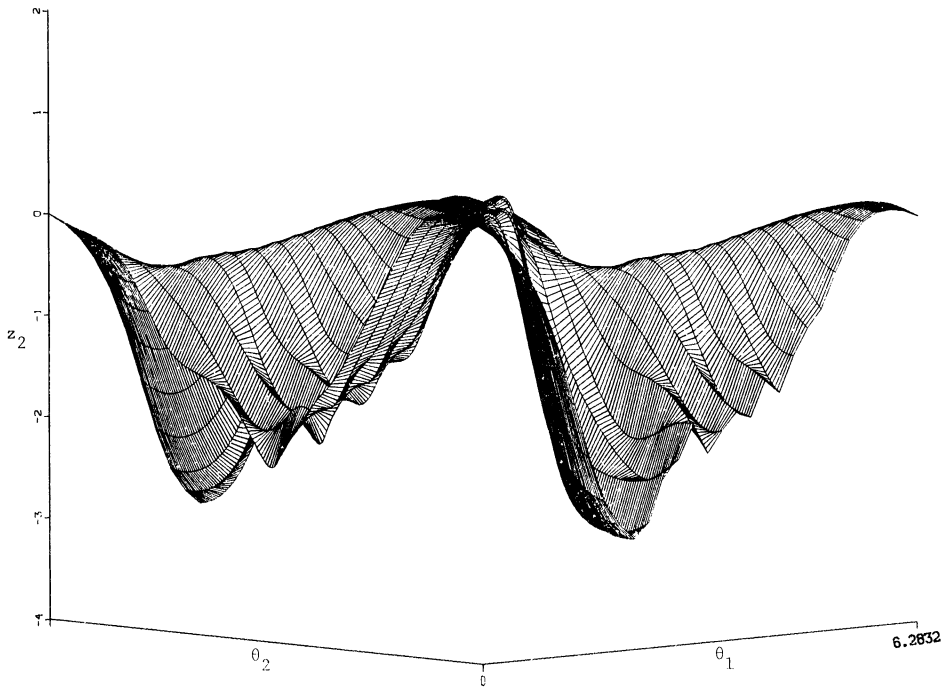


FIG. 11. *Coupled oscillator.* $\delta = .2525$, *view*: $(\theta_1, \theta_2, z_2) = (-7, -7, 1)$.

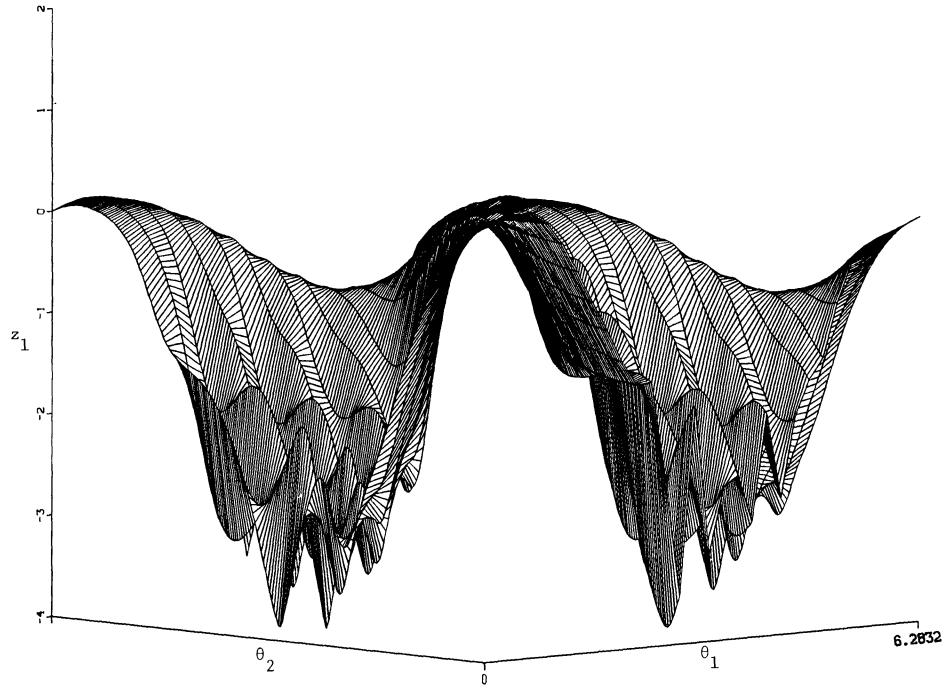


FIG. 12. *Coupled oscillator.* $\delta = .2527$, *view*: $(\theta_1, \theta_2, z_1) = (-7, -7, 1)$.

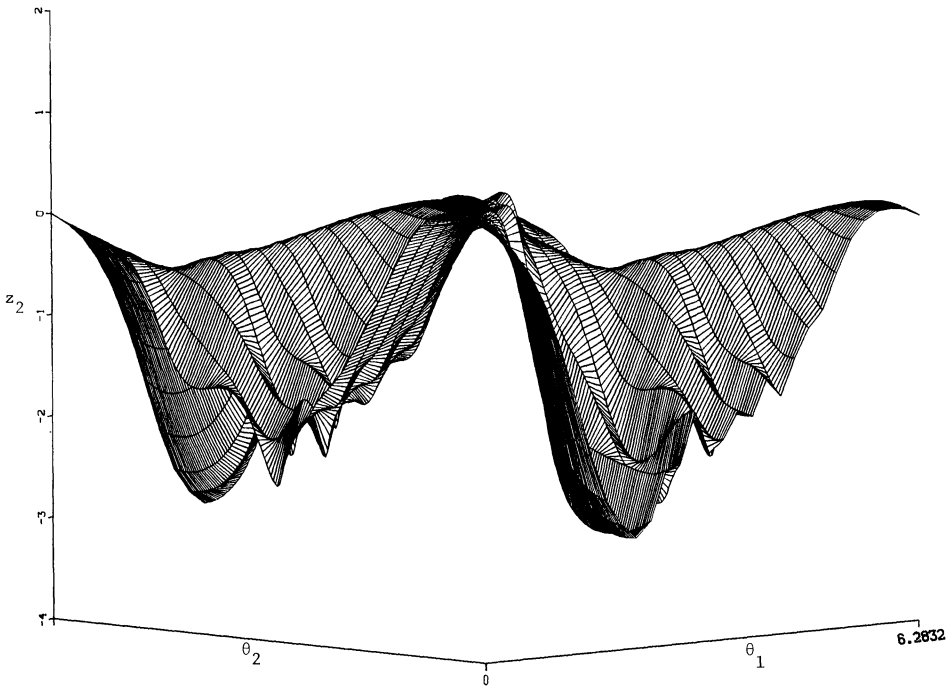


FIG. 13. *Coupled oscillator.* $\delta = .2527$, *view*: $(\theta_1, \theta_2, z_2) = (-7, -7, 1)$.

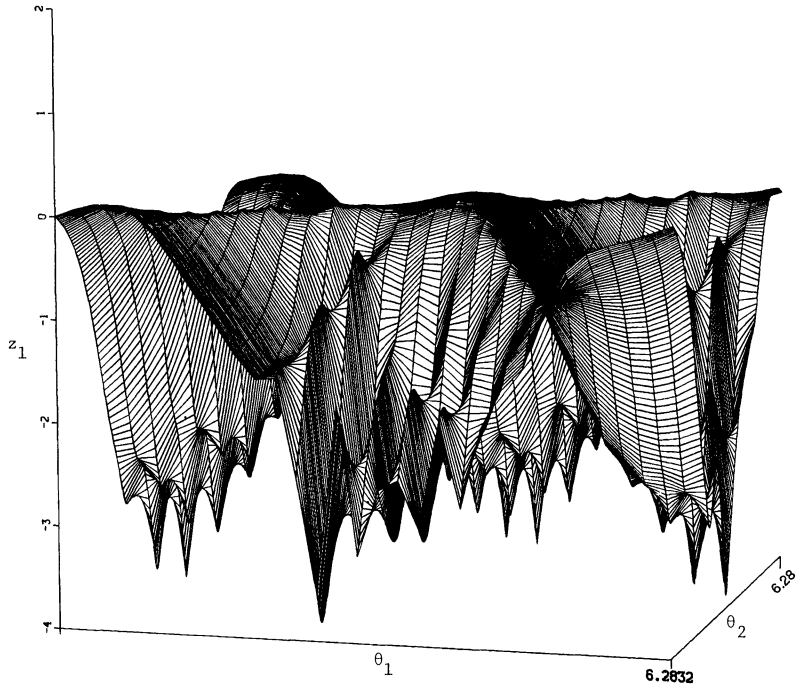


FIG. 14. *Coupled oscillator.* $\delta = .2527$, *view*: $(\theta_1, \theta_2, z_1) = (2, -5, 1)$.

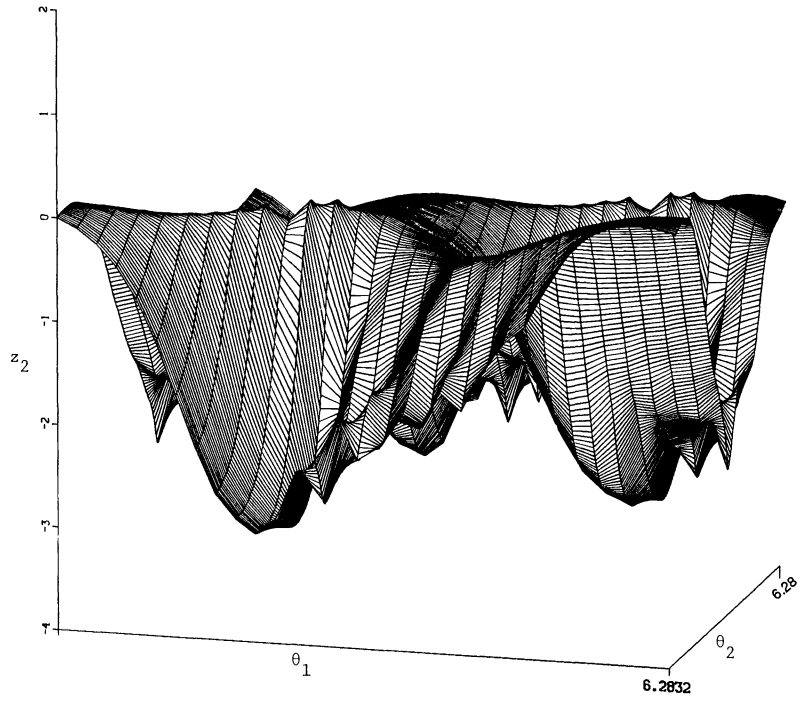


FIG. 15. *Coupled oscillator.* $\delta = .2527$, *view*: $(\theta_1, \theta_2, z_2) = (2, -5, 1)$.

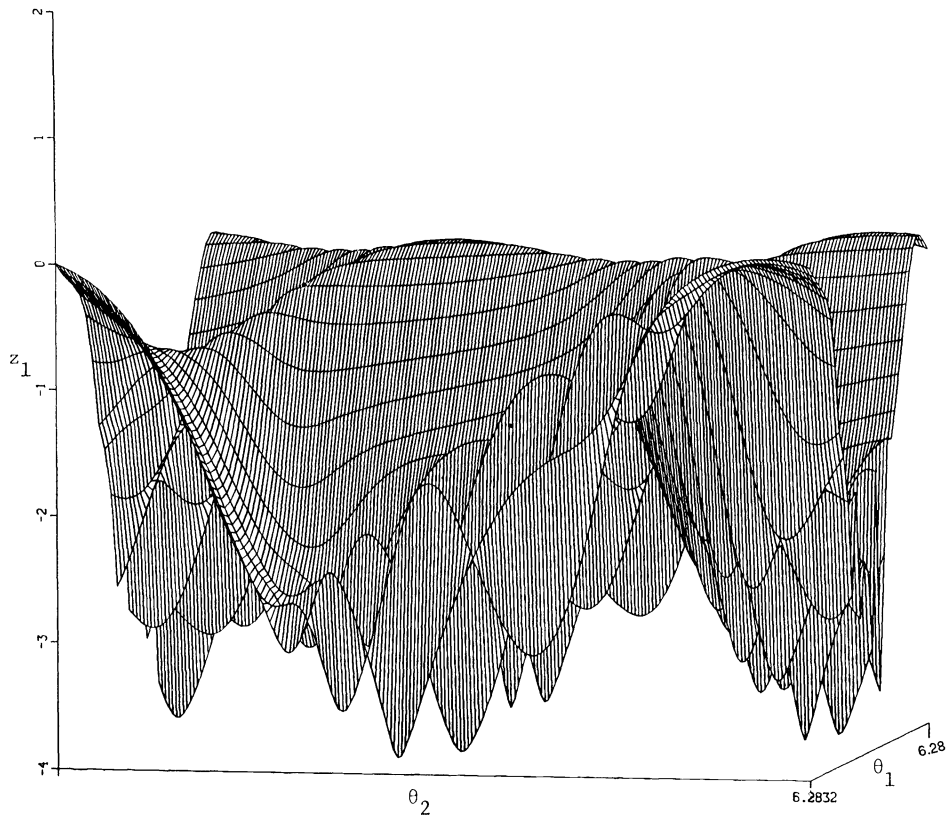


FIG. 16. *Coupled oscillator.* $\delta = .2527$, *view*: $(\theta_1, \theta_2, z_1) = (13, 3, 1)$.

6. Conclusions. In this paper we have considered the problem of computing smooth invariant tori for autonomous first-order systems of ODEs of the form (1.4). Under certain assumptions, a parameterization can be found by solving the first-order PDE system (1.6). We have attempted to exploit this fact computationally and thus focused on solving (1.6) numerically. As far as we know, this work represents the first numerical study of the PDE approach to invariant manifold approximation. We have discretized the PDE with the leap-frog scheme on a fixed grid and then solved with Newton's method. For model constant coefficients problems we have shown stability (and convergence) of the discretization, independently of the attractivity properties of the manifold. In the case of the 2-torus, we have also analyzed a compactification procedure for solving the linear systems arising at each Newton iteration. This is closely connected to computing an invariant curve under a suitable Poincaré map of the original system. The study of numerical stability for this compactification led us (see also [K]) to propose a modified leap-frog discretization. For model constant coefficient problems we give stability proofs showing that now the use of compactification requires attractivity of the manifold. Finally, we have given numerical results for the forced van der Pol equation and for a system of two coupled oscillators.

Our conclusion from this work is that the PDE formulation for the invariant manifold is of great appeal not only analytically, as has long been recognized, but also computationally. Sensitivity of the numerical method depends more upon the geometry

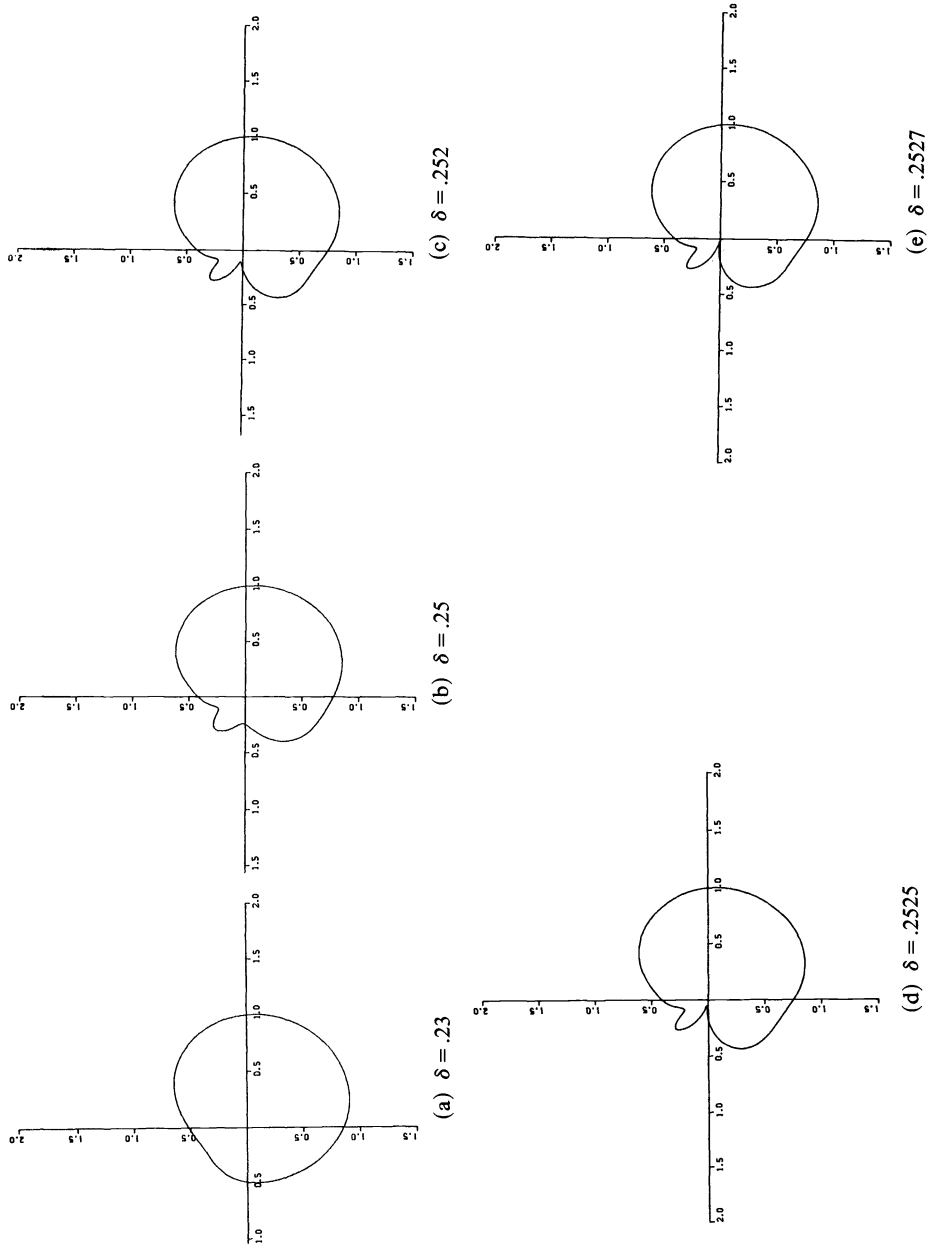
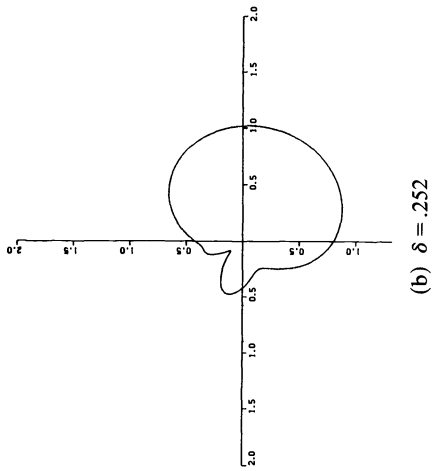
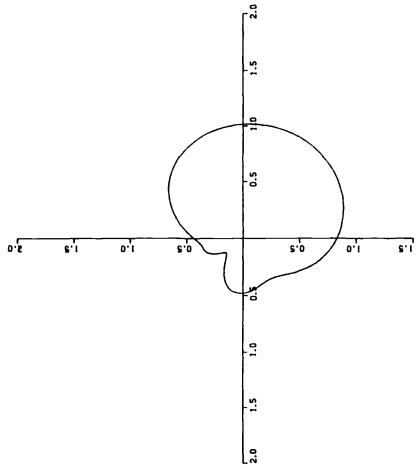


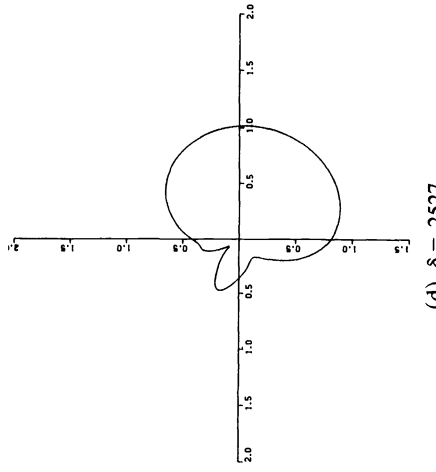
FIG. 17. *Coupled oscillator.* $r_1(0, \theta_2)$.



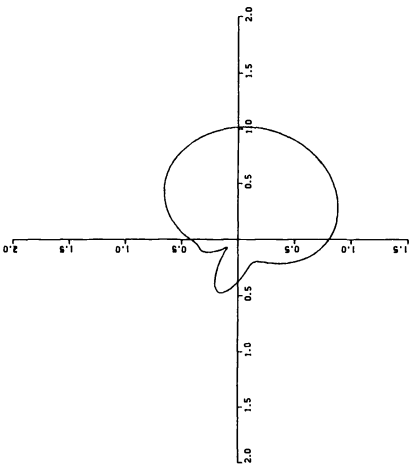
(b) $\delta = .252$



(a) $\delta = .25$



(d) $\delta = .2527$



(c) $\delta = .2525$

FIG. 18. *Coupled oscillator.* $r_1(\pi/15, \theta_2)$.

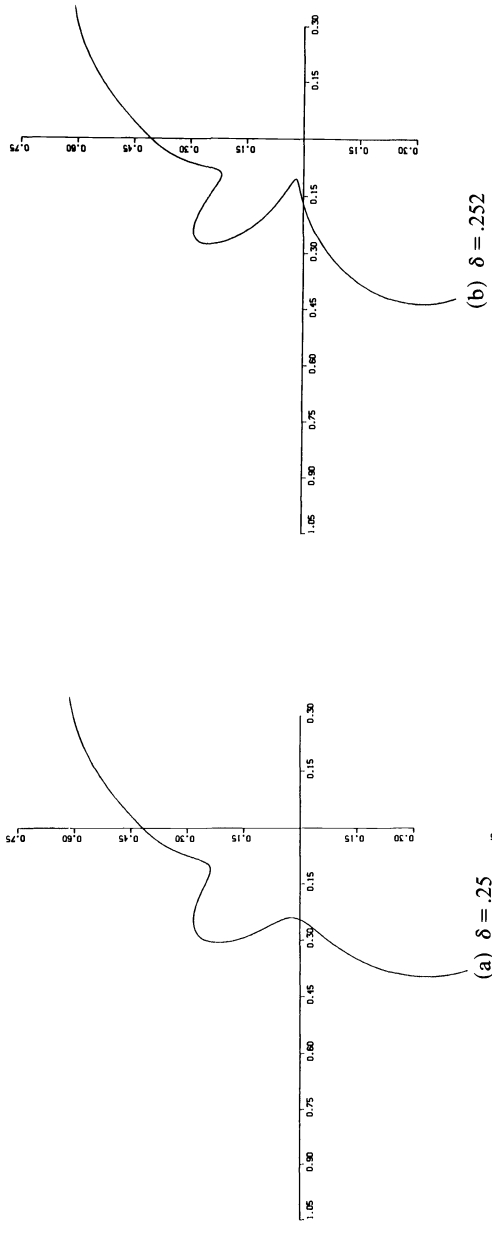


FIG. 19. *Coupled oscillator.* $r_1(0, \theta_2)$: *blow-up.*

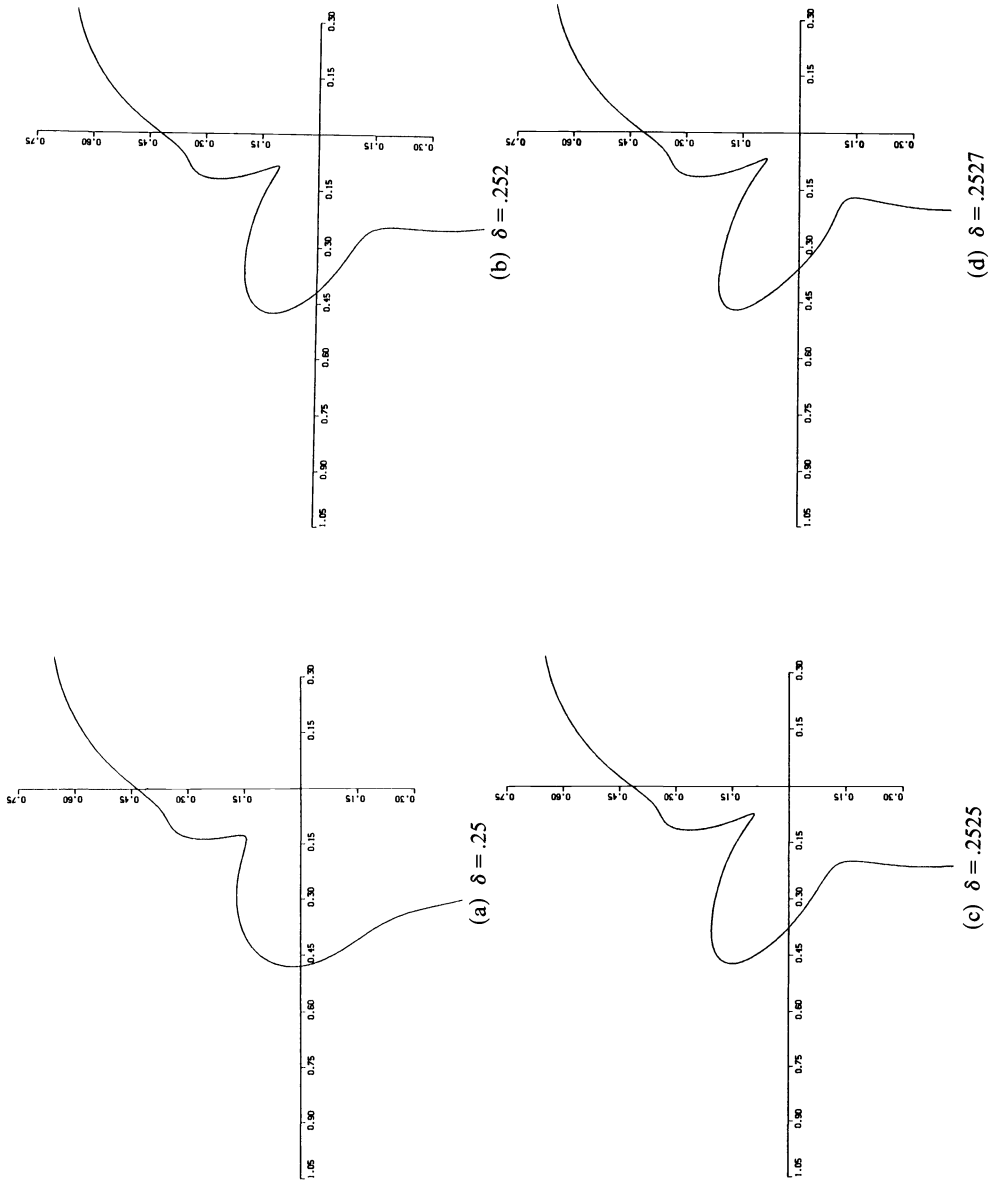


FIG. 20. *Coupled oscillator.* $r_1(\pi/15, \theta_2)$: *blow-up.*

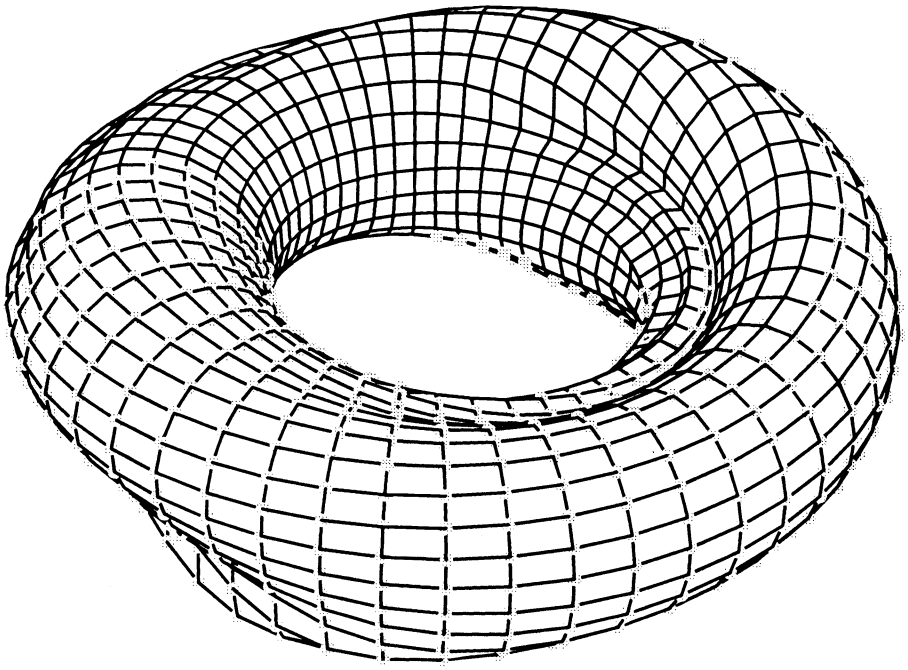


FIG. 21. *Coupled oscillator.* $r_2(\theta_1, \theta_2)$, $\delta = .2525$.

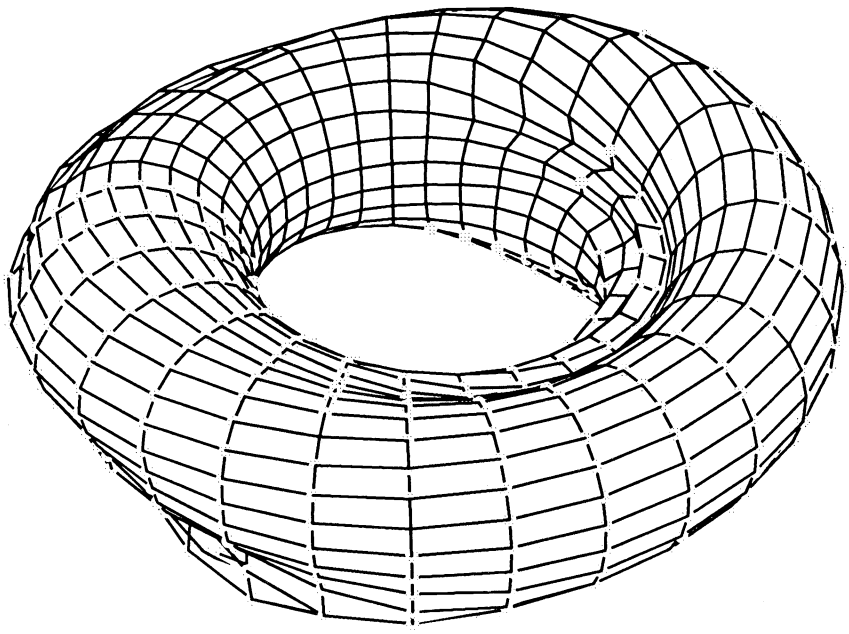


FIG. 22. *Coupled oscillator.* $r_2(\theta_1, \theta_2)$, $\delta = .2527$.

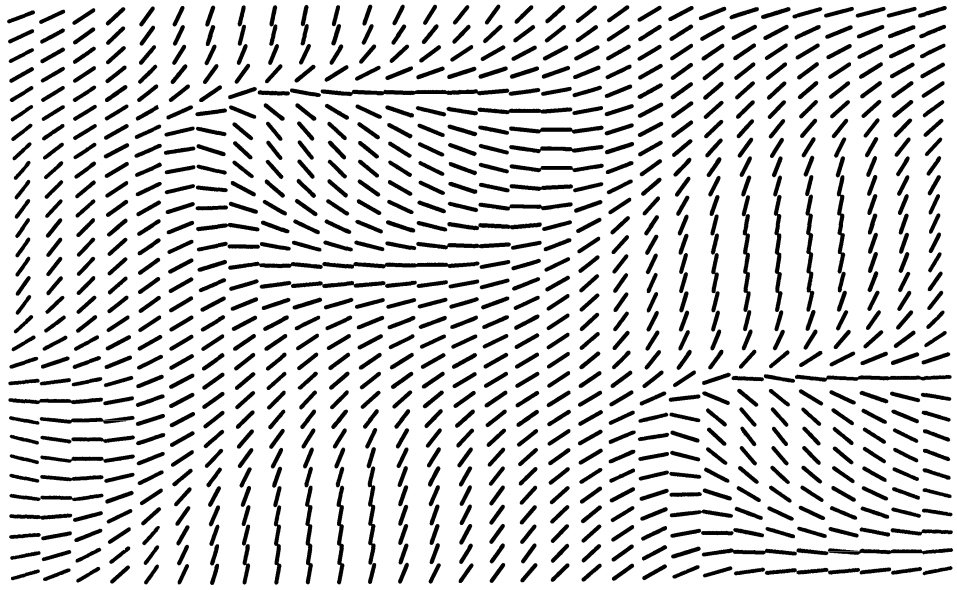


FIG. 23. *Coupled oscillator.* $r_2(\theta_1, \theta_2)$, $\delta = .2527$. *Streamlines.*

of the invariant torus than upon the dynamics of the original system of ODEs, and in this respect preserves the spirit of many of the analytical results (such as that of Levinson [L]). Interestingly, although the technique of elliptic regularization (e.g., see [S1], [S2], [FLS]) has proven very fruitful in analytical work, here it has been unnecessary to apply it to (1.6) computationally. Study of different discretizations for the PDE (1.6), as well as of different strategies for solving the resulting linear systems, is certainly warranted. Regardless, the main limitation to this PDE approach is that it is necessary to use a subset of the dependent variables in (1.6) to parameterize the manifold. We intend to remove this restriction in future extensions of this work. The extension of the analysis from constant to variable coefficients problems also needs to be done. Finally, comparison of this PDE approach to altogether different approaches (see especially [KASP], [vV1]) would also be of interest.

Acknowledgments. We wish to thank Erik van Vleck and Tim Kelley for their help in drawing Figs. 21-23 with Mathematica.

REFERENCES

[ADO] D. G. ARONSON, E. J. DOEDEL, AND H. G. OTHMER, *An analytical and numerical study of the bifurcations in a system of linearly-coupled oscillators*, Phys. D, 25 (1987), pp. 20-104.

[B] R. E. BANK, *Marching algorithms and block Gaussian elimination*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 293-307.

[DLR] L. DIECI, J. LORENZ, AND R. D. RUSSELL, *Decoupling of dynamical systems using boundary value techniques*, LCCR Tech. Report 88-9, Department of Mathematics, Simon Fraser University, Burnaby, British Columbia, Canada, 1988.

[FLS] E. FABES, M. LUSKIN, AND G. R. SELL, *Construction of inertial manifolds by elliptic regularization*, IMA Preprint Series #459, University of Minnesota, Minneapolis, MN, 1988.

[F] N. FENICHEL, *Persistence and smoothness of invariant manifolds for flows*, Indiana Univ. Math. J., 21 (1971), pp. 193-226.

[FJKST] C. FOIAS, M. S. JOLLY, I. G. KEVREKIDIS, G. R. SELL, AND E. S. TITI, *On the computation of inertial manifolds*, Phys. Lett. A, 131 (1988), pp. 433-436.

[GH] J. GUCKENHEIMER AND P. HOLMES, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations in Vector Fields*, Applied Mathematical Sciences 42, Springer-Verlag, New York, 1983.

[Ha] J. HALE, *Asymptotic Behavior of Dissipative Systems*, Math. Surveys and Monographs, Vol. 25, American Mathematical Society, Providence, RI, 1988.

[He] D. HELLER, *Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems*, SIAM J. Numer. Anal., 13 (1976), pp. 484-496.

[Hi] E. HILLE, *Analytic Function Theory*, Vol. 2, Chelsea, New York, 1962.

[JNP] C. JOHNSON, U. NÄVERT, AND J. PITKÄRANTA, *Finite element methods for linear hyperbolic problems*, Comput. Methods Appl. Mech. Engrg., 45 (1984), pp. 285-312.

[KASP] I. G. KEVREKIDIS, R. ARIS, L. D. SCHMIDT, AND S. PELIKAN, *Numerical computations of invariant circles of maps*, Phys. D, 16 (1985) pp. 243-251.

[K] H. O. KREISS, *Numerical methods for solving time-dependent problems for partial differential equations*, Les Presses de l'Université de Montréal, Montréal, Québec, Canada, 1978.

[L] N. LEVINSON, *Small periodic perturbations of an autonomous system with a stable orbit*, Ann. of Math., 52 (1950), pp. 727-738.

[M] G. H. MEYER, *Initial Value Methods for Boundary Value Problems*, Academic Press, New York, 1973.

[S1] R. SACKER, *A new approach to the perturbation theory of invariant surfaces*, Comm. Pure Appl. Math., 18 (1965), pp. 717-732.

[S2] ———, *A perturbation theorem for invariant manifolds and Hölder continuity*, J. Math. Mech., 18 (1969), pp. 705-762.

[T] R. TEMAM, *Infinite-dimensional dynamical systems in mechanics and physics*, Applied Mathematical Sciences 68, Springer-Verlag, New York, 1988.

[TS] J. M. T. THOMPSON AND H. B. STEWART, *Nonlinear Dynamics and Chaos*, John Wiley, Chichester, U.K., 1986.

[vV1] M. VAN VELDHUIZEN, *A new algorithm for the numerical approximation of an invariant curve*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 951-962.

[vV2] ———, *Convergence results for invariant curve algorithms*, Math. Comp., 51 (1987), pp. 677-697.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 648-667, May 1991
© 1991 Society for Industrial and Applied Mathematics
009

HIGH DEGREE POLYNOMIAL INTERPOLATION IN NEWTON FORM*

HILLEL TAL-EZER†

Abstract. Polynomial interpolation is an essential subject in numerical analysis. Dealing with a real interval, it is well known that even if $f(x)$ is an analytic function, interpolating at equally spaced points can diverge [P. J. Davis, *Interpolation and Approximation*, Dover, New York, 1975]. On the other hand, interpolating at the zeros of the corresponding Chebyshev polynomial will converge. Using the Newton formula, this result of convergence is true only on the theoretical level. It is shown that the algorithm which computes the divided differences is numerically stable only if: (1) The interpolating points are arranged in a certain order. (2) The size of the interval is four. Stabilizing Newton interpolation is described in greater generality of interpolation in the complex plane.

High degree polynomial interpolation is employed in approximation of functions of matrices. Some applications of the results described in the paper for this purpose are given.

Key words. polynomial interpolation, Newton form, Fejér points

AMS(MOS) subject classifications. 65D05, 65F99

1. Introduction.

Let $f(x)$ be a real function defined on the interval $[a, b]$, and let $\{x_i\}_{i=0}^m$ be a set of $m+1$ points in $[a, b]$; then the general formulation of Newton interpolating polynomial of degree m is

$$P_m(x) = \sum_{k=0}^{m} a_k R_k(x) \tag{1.1}$$

where a_k are the divided differences

$$a_k = f[x_0, \cdots, x_k] \tag{1.2}$$

and

$$R_0(x) = 1, \tag{1.3}$$

$$R_k(x) = \prod_{i=0}^{k-1} (x - x_i), \qquad k = 1, \cdots, m. \tag{1.4}$$

If $E_m(\bar{x})$ is the error at point $\bar{x} \in [a, b]$, then we have Theorem 1.1 [Davi75].

THEOREM 1.1. *Let $f(x)$ be a real-valued function defined on $[a, b]$ and $m+1$ times differentiable on $[a, b]$. If $P_m(x)$ is the polynomial of degree less than or equal to m which interpolates $f(x)$ at $m+1$ distinct points $x_0, \cdots, x_m$ in $[a, b]$. Then for all $\bar{x} \in [a, b]$, there exists $\xi = \xi(\bar{x}) \in (a, b)$ such that*

$$E_m(\bar{x}) = f(\bar{x}) - P_m(\bar{x}) = \frac{f^{(m+1)}(\xi)}{(m+1)!} R_{m+1}(\bar{x}). \tag{1.5}$$

It is well known that if $x_0, \cdots, x_m$ are equally spaced points, then the local maxima of $R_{m+1}(\bar{x})$ increases as we move from the middle of the interval towards its ends, and this increase becomes more pronounced with increasing m. Moreover, if $f(x)$ has singularities in the complex plane, divergence can occur (Runge phenomenon).

* Received by the editors May 31, 1988; accepted for publication (in revised form) February 25, 1990. This research was supported by National Aeronautics and Space Administration contract NAS1-18107 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia 23665, and Air Force Office of Scientific Research grant 85-0303.

† School of Mathematical Sciences, Tel Aviv University, Ramat Aviv, 69978 Tel Aviv, Israel.

A well-known remedy for these two drawbacks is to interpolate the function at the zeros of the corresponding scaled and translated Chebyshev polynomial

$$x_i = \frac{1}{2}\left[(b-a)\cos\frac{(2i+1)\pi}{2m+2} + b + a\right], \qquad i = 0, \cdots, m \tag{1.6}$$

and if $f(x)$ is analytic in $[a, b]$, then $P_m(x)$ converges to $f(x)$.

In the following pages we will show that this result of convergence is true only in theory. Practically, an interpolating polynomial based on (1.6) *will not* converge to $f(x)$ because of the finite accuracy of the computer. The numerical instability can be traced to two sources:

(1) The algorithm which computes the divided differences is very vulnerable to roundoff errors and overflow. The supersensitivity to roundoff errors is explained by the fact that for m large, the first k points ($k \ll m$) are very close to each other ($\min |x_{i+1} - x_i| \sim 1/m^2$). But even if roundoff errors were eliminated we still would face overflow. It results from having the first k points ($k \ll m$) concentrated on one side of the interval. This distribution will lead to nonuniformity of $R_k(x)$ and $E_k(x)$. It is this nonuniformity which will cause overflow at intermediate stages of the interpolation process.

(2) Eliminating the first source of numerical instability (by taking the points in a different order (§ 2)) results in "almost" uniformity of $R_k(x)$ for $k < m$ but still

$$\limsup_{k\to\infty} \left(\max_x |R_k(x)|\right)^{1/k} = \frac{1}{2}. \tag{1.7}$$

Hence we see that $\max_x |R_k(x)|$ decreases exponentially fast as $k \to \infty$. When the interpolated function is such that high degree interpolation is needed by accuracy considerations, it is clear that since (1.1), the coefficients should grow (at intermediate stages) in order to counterbalance the exponential decrease of $R_k(x)$ and we can get overflow. It will be shown that we can eliminate this source of instability by a simple change of variables

$$\hat{x} = 2x$$

resulting in $[-2, 2]$ as the domain of interpolation. For this interval (or any real interval of size four) the right-hand side of (1.7) is one.

We are going to approach this phenomenon in the more general context of interpolation in the complex plane. Background material is given in § 2. Based on the theory, we show that by:

(1) Arranging the interpolating points in a certain order, and

(2) Making a simple change of variables,

we get

$$\limsup_{m\to\infty} \left(\max_{z\in D} |R_m(z)|\right)^{1/m} = 1 \tag{1.8a}$$

where D is the domain of interpolation in the complex plane. Algorithm 2.1, presented in § 2, generates m points, known as Fejér points, such that

$$\limsup_{m\to\infty} \left(\max_{z\in D} |R_k(z)|\right)^{1/k} = 1, \qquad k = [\beta m], \qquad 0 < \beta < 1. \tag{1.8b}$$

Algorithm 2.1 is useful when the degree of the interpolating polynomial is given. Sometimes, we do not know m ahead and we would like to be able to add points, if

necessary, without restarting the interpolation process. It is well known that the Newton algorithm has, in potential, this feature. Adding another point results in computing only one additional term. But by using Chebyshev zeros (1.6) we would not capitalize on this property since increasing m results in changing all the previous points. To this end we construct Algorithm 2.2, which enables us to add interpolating points such that (1.8) is satisfied. Observe that since (1.1) and (1.8), the a_k's behave asymptotically like the interpolation error. Thus we can use the algorithm, which computes the divided differences as a numerical device for estimating the degree of the polynomial needed to achieve a given accuracy. It can be useful when the mathematical expression of the error (1.5) is difficult to analyze.

The need for high-degree interpolation is confronted in approximating a finite operator which can be presented as $f(A)$ where A is another finite operator. (The case $f(z)=\exp(z)$ is a popular example [MoVa78].) Approximating $f(A)$ can be reduced to a problem of approximating $f(z)$ where z belongs to a domain D in the complex plane which includes all the eigenvalues of A. A possible approach is to expand the function as a sum of orthogonal polynomials. In [Tale86] and [Tale85] we have used this approach for the function $f(z)=\exp(tz)$ where the domains were $[-iR, iR]$, $[-R, 0]$ $(R>0)$, respectively. The algorithm which results make use of the three-term recurrence relations satisfied by the polynomials (scaled and translated Chebyshev). For more complicated domains, the related polynomial of degree k satisfies a k-term recurrence relation and therefore the expansion approach is not suitable, since it results in a highly space-consuming algorithm. An alternative way is to use interpolation. A brief description of the method is given in § 4. A particular and very important case of polynomial approximation of $f(A)$ is an iterative solution to a linear system $Ax=b$. Finding optimal parameters α_k which result in numerically stable Richardson algorithm

$$x^{k+1}=x^k-\alpha_k(Ax^k-b) \tag{1.9}$$

can be achieved by considering polynomial interpolation to the function $f(z)=1/z$. When D is one side of the real line, this method is widely treated in the numerical analysis literature (Chebyshev acceleration [HaYo81]). It is also known that having to decide on the number of iterations before starting the algorithm reduces its efficiency. The Richardson process which uses points generated by Algorithm 2.2 is free of this disadvantage. (For more elaborate discussion and numerical examples, see [Tale87].) Section 5 concludes the paper by giving some numerical results.

2. Interpolation in the complex plane. Let D be bounded continuum in C such that D^c = the complement D is simply connected in the extended plane and contains the point at infinity. Considering interpolation in the domain D, we are faced with the problem of which are the "good" interpolating points? To answer this question we need some preliminaries from complex approximation theory.

For D satisfying the above there exists a conformal mapping $\psi(w)$, which maps the complement of the unit disc onto D^c [Wals56]

$$z=\psi(w)=\rho\omega+b_0+\frac{b_1}{w}+\cdots \qquad (\rho>0) \tag{2.1}$$

(ρ is the *logarithmic capacity* of D).

Let $z_i^{(m)}\in D$ and

$$M_m=\max_{z\in D}(|R_m(z)|) \tag{2.2}$$

where

$$R_m(z) = \prod_{i=0}^{m-1} (z - z_i^{(m)}); \tag{2.3}$$

then we have Definition 2.1.

DEFINITION 2.1. The points $z_i^{(m)}$ are *uniformly distributed* on D if

$$\limsup_{m\to\infty} M_m^{1/m} = \rho. \tag{2.4}$$

Let $\phi(z)$ be the inverse of $\psi(w)$, and let

$$B_r = \{z: |\phi(z)| = r\} \qquad (r > 1) \tag{2.5}$$

denote the level curves in D^c. The connection between uniformly distributed points and convergence of the interpolating polynomial is given by the following theorem.

THEOREM 2.1. *Suppose $r > 1$ is the largest number such that $f(z)$ is analytic inside B_r. The interpolating polynomials $P_m(z)$ with interpolating points $z_i^{(m)}$ that are uniformly distributed on D then satisfy*

$$\limsup_{m\to\infty} \left(\max_{z\in D} |f(z) - P_m(z)| \right)^{1/m} = \frac{1}{r}. \tag{2.6}$$

A series of interpolating polynomials satisfying (2.6) is said to *converge maximally* to $f(z)$ [Wals56].

Hence we have the solution to the question presented at the beginning of this section—the interpolating points should be uniformly distributed on D. We now turn to the problem of obtaining these points.

Fejér points. If $\psi(w)$ has continuous extension to $\{w: |w| \geqq 1\}$ then the points

$$z_j^{(m)} = \psi(e^{2\pi ij/(m+1)}), \qquad j = 0, \cdots, m \tag{2.7}$$

are called *Fejér points* of order m on D. We have Theorem 2.2.

THEOREM 2.2 (Fejér, 1918). *The Fejér points are uniformly distributed on D.*

Another possible set of uniformly distributed points are the zeros of the corresponding Faber polynomial. Let D and $\phi(z)$ be as defined previously. We have [Mark77]

$$[\phi(z)]^m = c_m z^m + \cdots + c_0 + c_{-1}/z + c_{-2}/z^2 + \cdots. \tag{2.8}$$

Faber polynomial of degree m related to D is the polynomial part of (2.8)

$$F_m(z) = c_m z^m + \cdots + c_0. \tag{2.9}$$

Interpolating at the zeros of $F_m(z)$ satisfies (2.6) [Mark77]. When

$$D = \{z \,|\, -1 \leqq \operatorname{Re} z \leqq 1,\ I_m z = 0\}, \tag{2.10}$$

then

$$F_m(z) = T_m(z) = \cos(m \cos^{-1}(z)),$$

and the m zeros are Chebyshev points (1.6).

For a general domain in the complex plane, finding the zeros of $F_m(z)$ for large m can be troublesome. Thus, it is simpler to use Fejér points since just $\psi(w)$ is required.

Assume now that $z_j^{(m)}$, $j = 0, \cdots, m$ is a set of Fejér points arranged according to (2.7). Because of (2.4) it is obvious that in order to satisfy (1.8) we must scale the domain such that $\rho = 1$. This is achieved by doing a simple change of variables

$$\hat{z} = z/\rho. \tag{2.11}$$

This modification by itself is not enough. Observe that in (2.7), the first k points

$$k=[\beta m], \qquad 0<\beta \ll 1, \tag{2.12}$$

are clustered in a small region of the boundary of the domain. Hence, $M_k^{1/k}$ would be far from satisfying (2.4). Moreover, due to the superfine spacing between adjacent points, computing the divided differences will suffer from huge roundoff errors. Consequently, we will face overflow. Algorithm 2.1 presented below reorders (2.7) such that for every j, $z_1^{(m)} \cdots z_j^{(m)}$, are "almost" evenly distributed on the boundary of D. The new distribution is not vulnerable to roundoff errors and it can be shown that (1.8) is satisfied.

ALGORITHM 2.1.

$$\theta_1=0, \qquad j=l=1, \qquad \delta\theta=2\pi/m,$$

Find the largest k such that k is power of two and $k\delta\theta<\pi\leqq 2k\delta\theta$.

(1) For $i=1$ until l do

$$\eta=\theta_i+k\delta\theta$$
$$\textit{if } (\eta\geqq\pi) \textit{ go to } 2$$
$$j=j+1$$
$$\theta_j=\eta$$

(2) end do.

$$l=j$$
$$k=k/2$$
$$\textit{if } (k<1) \textit{ stop}$$
$$\text{go to } 1$$

This part of the algorithm produces arguments of points on the upper part of the unit circle (includes 1 but not -1). Thus,

$$\begin{aligned} &z_0=\psi(1), \qquad z_1=\psi(-1), \\ &z_{2j}=\psi(e^{i\theta_{j+1}}), \qquad z_{2j+1}=\bar{z}_{2j}, \qquad 1\leqq j\leqq(m-1)/2. \end{aligned} \tag{2.13}$$

As mentioned in the Introduction, we would like to have an interpolating process which will allow us to add points if desired. To this end we have the next algorithm. It generates an infinite set of uniformly distributed points and is equivalent to the one given in [FiRe87] (based on bit-reversed binary representation of the points count). Using this algorithm we do not have to decide on the degree of the polynomial ahead.

ALGORITHM 2.2.

$$\delta\theta=\pi$$
$$k=1$$
$$z_0=\psi(1)$$
$$\theta_1=0$$

(1) For $i=1$ until k do

$$\theta_{k+i}=\theta_i+\delta\theta$$
$$z_{k+i-1}=\psi(e^{i\theta_{k+i}})$$

end do.

$$\delta\theta = \delta\theta/2$$

$$k = 2k$$

go to 1

Comparing Algorithms 2.1 and 2.2, we see that for Algorithm 2.1, the points are evenly distributed on the unit circle. The arguments of two adjacent points differ by π/m. On the other hand, in Algorithm 2.2, the even distribution is achieved only when m is power of two. In other cases, the arguments differ either by $\pi/2^k$ or $\pi/2^{k-1}$ where $2^{k-1} < m < 2^k$. The uneven distribution of points in Algorithm 2.2 is the price we have to pay for gaining the ability to add points. In general, we can expect the interpolation based on Algorithm 2.1 to be more accurate.

Remark. In order to implement the algorithms presented in this section, we need the conformal mapping function. For some domains we do have an analytic expression of this function. In more complicated situations, we must resort to numerical techniques. When D is a polygon, $\psi(w)$ is a Schwarz-Cristoffel transformation. Numerical routines for this case, written by Trefethen (based on [Tref80]), are available through Netlib facilities.

The routines described in [Tref80] map the interior of the unit disc on the interior of the polygon. Since, in our case, the mapping of the exteriors is needed, the routines should be modified accordingly. When the domain D is symmetric around the real axis we can still use the original interior's mapping routines without any modification [Tref80]. The idea is explained in Appendix A.

3. Interpolation on the real line. Let

$$D = [-b, b]. \tag{3.1}$$

The conformal mapping $\psi(w)$, which maps the complement of the unit disc on the complement of D, is [Mark77]

$$\psi(w) = \frac{b}{2}(w + w^{-1}). \tag{3.2}$$

Since (2.1) we have

$$\rho = \frac{b}{2}. \tag{3.3}$$

Therefore we should make the following change of variables:

$$\hat{x} = \frac{2x}{b} \tag{3.4}$$

in order to get

$$\hat{D} = [-2, 2] \tag{3.5}$$

with capacity

$$\hat{\rho} = 1. \tag{3.6}$$

Similarly, if

$$D = [a, b], \tag{3.7}$$

then

$$\hat{x} = \frac{4x}{b-a}. \tag{3.8}$$

Thus, without loss of generality, we assume that

$$D=[-2,2]. \tag{3.9}$$

Using (3.2), Fejér points are

$$x_j = w_j + w_j^{-1} \tag{3.10}$$

where w_j can be generated by the algorithms given in § 2. Observe that the x_j's are double interpolation points (except for 2 and -2).

Faber polynomials which correspond to D are scaled Chebyshev polynomials [Mark77]

$$\tilde{T}_m(x)=\cos\left(m\cos^{-1}\left(\frac{x}{2}\right)\right), \tag{3.11}$$

whose zeros are

$$y_j = 2\cos\frac{(2j+1)\pi}{2m}, \qquad j=0,\cdots,m-1. \tag{3.12}$$

Using the arguments θ_j produced by Algorithm 2.1, we can get the zeros of $\tilde{T}_m(x)$, arranged as follows:

$$x_j = 2\cos\left(\theta_{j+1}+\frac{\delta\theta}{2}\right), \qquad j=0,\cdots,m-1. \tag{3.13}$$

Observe that (3.13) does not include the boundary points. A popular set of uniformly distributed points which does include the boundaries is [Rivl74].

$$x_j=\cos\left(\frac{j\pi}{m}\right), \quad 0\leqq j\leqq m. \tag{3.14}$$

Using the Algorithm 2.1, the rearrangement of the points results in

$$x_0=2, \qquad x_1=-2, \tag{3.15a}$$

$$x_j=2\cos\theta_j, \qquad 2\leqq j\leqq m. \tag{3.15b}$$

4. Polynomial approximation of a function of a matrix. Let A be an $N\times N$ matrix, and let $F(z)$ be a function analytic in a domain D in the complex plane which includes all the eigenvalues of A. Let w be the vector which results from operating with $F(A)$ on a vector v

$$w=[F(A)]v. \tag{4.1}$$

Getting an approximation of w is a problem frequently confronted in applied mathematics. One feasible way to tackle it is to develop an algorithm based on polynomial approximation of the function $F(z)$. An elaborate description of this approach is given in [Tale87]. Here we will give a short summary.

Let w_m be an approximation of w which results from approximating the function $F(z)$ by the polynomial $P_m(z)$

$$w_m=[P_m(A)]v. \tag{4.2}$$

When A is diagonalizable, the relative error vector e_m

$$e_m=\frac{\|w-w_m\|}{\|v\|} \tag{4.3}$$

satisfies

$$e_m \leqq \|T\| \, \|T^{-1}\| \max_{z \in D} |F(z) - P_m(z)| \tag{4.4}$$

where T is the matrix which diagonalizes A. Observing (4.4), we would like to choose $P_m(z)$ such that $\max_{z \in D} |F(z) - P_m(z)|$ will be small. Thus the original problem has been reduced to a problem of approximating analytic function in a domain D in the complex plane. Assuming that D satisfies the requirements stated in § 2, and by using the results described there, we can develop a stable polynomial algorithm for approximating the vector w. If A is a normal matrix, then $\|T\| \, \|T^{-1}\| = 1$. Hence

$$e_m \leqq \max_{z \in D} |F(z) - P_m(z)|. \tag{4.5}$$

The right-hand side of (4.5) can be approximated numerically by computing the interpolation error at different checkpoints. By repeating this process for increasing m we can get a good approximation of the degree of the polynomial needed to achieve a given accuracy. Knowing m, we can use Algorithm 2.1 as points generator. Hence, for a given D let $z_0, \cdots, z_m$ be $m+1$ interpolating points; then the interpolating polynomial in Newton form is

$$P_m(z) = \sum_{k=0}^{m} a_k R_k(z) \tag{4.6}$$

where a_k are the divided differences

$$a_k = f[z_0, \cdots, z_k] \tag{4.7}$$

and

$$R_0(z) = 1, \tag{4.8}$$

$$R_k(z) = \prod_{i=0}^{k-1} (z - z_i), \qquad k = 1, \cdots, m. \tag{4.9}$$

Using (4.2) and (4.6)-(4.9), w_m is computed by the following algorithm.

ALGORITHM 4.1.

$$r = v$$

$$u = a_0 r.$$

For $i = 1, \cdots, m$ do

$$r = (A - z_{i-1} I) r$$

$$u = u + a_i r.$$

end do

$$w_m = u.$$

When A is not normal, since $\|T\| \, \|T^{-1}\| \neq 1$, we cannot know ahead of time the degree of the polynomial needed for given accuracy. Therefore we will use Algorithm 2.2 as generator of interpolating points. The criterion for stopping will be the relative norm of the last added vector. We have Algorithm 4.2.

ALGORITHM 4.2.

$$r = v$$
$$u = a_0 r.$$

For $i = 1, \cdots$ do

$$r = (A - z_{i-1} I) r$$
$$u = u + a_i r.$$
$$\text{if } \frac{\|a_i r\|}{\|u\|} < \delta \quad \text{go to } 1$$

end do

$$1 \qquad w_m = u.$$

δ is an error tolerance given by the user.

When A is a real matrix and $f(z)$ accepts real values for z real, the algorithms have to be modified in order to eliminate complex arithmetic (see [Tale87] for proofs). The modified version of Algorithm 4.1 is Algorithm 4.3.

ALGORITHM 4.3.

$$u = [a_0 I + a_1 (A - z_0 I)] v$$
$$r = (A - z_0 I)(A - z_1 I) v.$$

For $i = 1, \cdots, (m-1)/2$ do

$$\tilde{r} = (A - z_{2i}^R I) r$$
$$u = u + a_{2i}^R r + a_{2i+1}^R \tilde{r}$$
$$r = (A - z_{2i}^R I) \tilde{r} + (z_{2i}^I)^2 r$$

end do

$$w_m = u.$$

(The superscripts R, I stand for the real and imaginary parts, respectively.)

Using Algorithm 2.2, we will order the points as follows: $z_0, z_1, z_2, \bar{z}_2, \cdots$, where z_{2i} are generated by the algorithm. The modified version of Algorithm 4.2 will read as follows.

ALGORITHM 4.4.

$$u = [a_0 I + a_1 (A - z_0 I)] v$$
$$r = (A - z_0 I)(A - z_1 I) v.$$

For $i = 1, \cdots$ do

$$\tilde{r} = (A - z_{2i}^R I) r$$
$$u = u + a_{2i}^R r + a_{2i+1}^R \tilde{r}$$
$$\text{if } \frac{\|a_{2i}^R r + a_{2i+1}^R \tilde{r}\|}{\|u\|} < \delta \quad \text{go to } 1$$
$$r = (A - z_{2i}^R I) \tilde{r} + (z_{2i}^I)^2 r$$

end do

$$1 \qquad w_m = u.$$

An iterative solution to a general linear system

$$Ax = b \tag{4.10}$$

is a particular case of approximating $f(A)$ where $f(z)=1/z$. Applying Algorithms 4.1–4.4 to this problem can be shown to be equivalent to a Richardson type process [Tale87]. Having an initial approximation x^0, we have that Algorithms 4.1–4.4 will result in Algorithms 4.5–4.8, respectively, where the relaxation parameters satisfy

$$\alpha_j = \frac{1}{z_j}. \tag{4.11}$$

ALGORITHM 4.5.
For $j=0$ until m do

$$x^{j+1} = x^j - \alpha_j(Ax^j - b)$$

end do

ALGORITHM 4.6.
For $j=0,\cdots$ do

$$r = Ax^j - b$$

$$x^{j+1} = x^j - \alpha_j r$$

$$\text{if } \frac{\|r\|}{\|b\|} < \delta \quad \text{stop}$$

end do

ALGORITHM 4.7.

$$x^1 = x^0 - \alpha_0(Ax^0 - b)$$

$$x^2 = x^1 - \alpha_1(Ax^1 - b)$$

For $j=1$ until $(m-1)/2$ do

$$r^{2j} = Ax^{2j} - b$$

$$\tilde{r}^{2j} = [|\alpha_j|^2 A - 2\alpha_j^R]r^{2j}$$

$$x^{2j+1} = x^{2j-1} + \tilde{r}^{2j}$$

end do.

ALGORITHM 4.8.

$$x^1 = x^0 - \alpha_0(Ax^0 - b)$$

$$x^2 = x^1 - \alpha_1(Ax^1 - b)$$

For $j=1$ until satisfied do

$$r^{2j} = Ax^{2j} - b$$

$$\tilde{r}^{2j} = [|\alpha_j|^2 A - 2\alpha_j^R]r^{2j}$$

$$x^{2j+1} = x^{2j-1} + \tilde{r}^{2j}$$

$$\text{if } \frac{\|\tilde{r}^{2j}\|}{\|b\|} < \delta \quad \text{stop}$$

end do.

Thus for this particular function we do not have to compute the divided differences. Therefore, the change of variables (2.11) is not significant here. But still rearranging the points is important. Otherwise, we can face numerical instability due to the nonuniformity of the error at intermediate stages. This phenomenon is also treated in [AnGo72], [LeFi76], and [FiRe87]. The algorithm proposed in [FiRe87] results in the same set of parameters generated by Algorithm 2.2. Hence, Algorithms 4.6 and 4.8 are essentially identical to the ones described there.

5. Numerical results. All the numerical experiments reported in this section were performed in double precision on Gould NP1.

Let $P_m(z)$ be the interpolating polynomial; then the relative maximal error

$$E = \frac{\max_{z\in D} |f(z) - P_m(z)|}{\max_{z\in D} |f(z)|} \tag{5.1}$$

is approximated by

$$E_{ap} = \frac{\max_{z_i^j} |f(z_i^j) - P_m(z_i^j)|}{\max_{z\in D} |f(z)|} \tag{5.2}$$

where z_i^j are checkpoints. In Tables 1-4, $D = [-1, 1]$ and the $10(m-1)$ checkpoints are

$$z_i^j = \left(1 - \frac{j}{11}\right) \cos\left(\frac{\pi(i-1)}{m-1}\right) + \frac{j}{11} \cos\left(\frac{\pi i}{m-1}\right), \qquad 1 \leqq i \leqq m-1, \qquad 1 \leqq j \leqq 10. \tag{5.3}$$

In the first two experiments, the interpolated function is

$$f(z) = \frac{1}{(z^2+\varepsilon)(z^2-z+0.25+\varepsilon)(z^2+z+0.25+\varepsilon)} \tag{5.4}$$

where

$$\varepsilon = 5.00E-04.$$

By the following change of variables

$$y = zb \tag{5.5}$$

we get

$$f(z) = g(y) = \frac{1}{((y/b)^2+\varepsilon)((y/b)^2-y/b+0.25+\varepsilon)((y/b)^2+y/b+0.25+\varepsilon)} \qquad -b \leqq y \leqq b. \tag{5.6}$$

The interpolating points are distributed on $[-b, b]$ in two different orderings:

$$y_j^1 = b \cos(\pi j/m), \qquad 0 \leqq j \leqq m, \tag{5.7}$$

and y_j^2 are generated by Algorithm 2.1.

In the first experiment, the polynomial is of degree 200. Selected divided differences and the relative maximal error (E_{ap}) are presented in Table 1.

Table 1 verifies the fact that y_j^1 are useless for high degree Newton interpolation. Since no overflow (underflow) has occurred in computing the divided differences, the error related to y_j^2 is the same in all the three cases. However, observe the "nice" behavior of the coefficients which correspond to $b = 2$. Next, we took $m = 600$. The results are presented in Table 2.

Now, we observe overflow or underflow in all the cases except for y_j^2 and $b = 2$.

TABLE 1
($m = 200.$)

	$b=1$		$b=2$		$b=3$	
k	$\lvert a_k(y^1)\rvert$	$\lvert a_k(y^2)\rvert$	$\lvert a_k(y^1)\rvert$	$\lvert a_k(y^2)\rvert$	$\lvert a_k(y^1)\rvert$	$\lvert a_k(y^2)\rvert$
1	1.77E+00	1.77E+00	1.77E+00	1.77E+00	1.77E+00	1.77E+00
26	5.25E+31	1.63E+09	6.33E+23	4.87E+01	1.34E+20	1.92E−03
51	1.59E+51	2.88E+17	1.33E+35	2.55E+02	1.78E+27	4.00E−07
76	1.12E+60	1.67E+24	7.52E+34	4.42E+01	1.58E+23	2.74E−12
101	6.74E+63	5.13E+31	5.11E+33	4.05E+01	1.22E+16	9.96E−17
126	7.39E+67	2.73E+38	1.74E+30	6.41E+00	1.69E+08	6.25E−22
151	1.45E+67	1.13E+46	1.02E+22	7.97E+00	3.93E−05	3.07E−26
176	7.59E+63	2.83E+52	1.59E+11	5.91E−01	2.42E−20	9.03E−32
198	4.77E+59	3.37E+59	2.37E+00	1.68E+00	4.85E−35	3.42E−35
199	2.38E+59	2.38E+59	5.94E−01	5.94E−01	8.09E−36	8.09E−36
200	1.05E+47	2.23E+49	3.80E−14	1.80E−11	1.44E−48	1.55E−46
E_{ap}	1.24E+71	2.32E−02	1.64E+47	2.33E−02	2.12E+42	2.33E−02

TABLE 2
($m = 600.$)

	$b=1$		$b=2$		$b=3$	
k	$\lvert a_k(y^1)\rvert$	$\lvert a_k(y^2)\rvert$	$\lvert a_k(y^1)\rvert$	$\lvert a_k(y^2)\rvert$	$\lvert a_k(y^1)\rvert$	$\lvert a_k(y^2)\rvert$
1	1.77E+00	1.77E+00	1.77E+00	1.77E+00	1.77E+00	1.77E+00
76	7.23E+75	7.48E+23	7.23E+75	1.98E+01	7.23E+75	1.23E−12
151	7.23E+75	1.82E+46	7.23E+75	1.27E+01	7.23E+75	4.93E−26
226	7.23E+75	4.44E+67	5.84E+75	8.24E−01	3.89E+75	1.98E−40
301	7.21E+75	3.30E+75	3.61E+75	5.24E−01	5.48E+63	7.81E−54
376	5.22E+75	3.45E+75	2.27E+65	2.39E−02	2.36E+29	2.21E−68
451	4.23E+75	3.33E+75	5.90E+43	1.51E−02	1.98E−17	0.00E+00
526	3.75E+75	3.39E+75	7.47E+16	5.60E−04	3.02E−63	0.00E+00
598	7.98E+75	5.08E+75	1.39E−09	4.49E−06	0.00E+00	0.00E+00
599	3.99E+75	1.02E+75	3.48E−10	3.38E−06	0.00E+00	0.00E+00
600	7.15E+63	3.81E+75	2.09E−17	4.17E−11	0.00E+00	0.00E+00
E_{ap}	2.13E+71	1.47E−02	2.27E+71	3.01E−06	2.27E+71	5.93E−04

The numerical instability presented in Tables 1 and 2 has nothing to do with the fact that $f(z)$ has singular points in the complex plane. We will get similar instability for entire functions as demonstrated in Table 3 where

$$f(z) = \sin(460z), \qquad -1 \leqq z \leqq 1.$$

Next, we interpolated the function

$$f(z) = \cos(2{,}000z), \qquad -1 \leqq z \leqq 1. \tag{5.8}$$

In this case we need a super high degree polynomial. We took $m = 2{,}100$. The divided differences are presented in Table 4.

TABLE 3
($m = 480.$)

	$b=1$		$b=2$		$b=3$	
k	$\lvert a_k(y^1)\rvert$	$\lvert a_k(y^2)\rvert$	$\lvert a_k(y^1)\rvert$	$\lvert a_k(y^2)\rvert$	$\lvert a_k(y^1)\rvert$	$\lvert a_k(y^2)\rvert$
60	7.23E+75	7.23E+16	7.23E+75	1.25E−01	7.23E+75	5.12E−12
120	7.23E+75	4.64E+34	7.23E+75	7.04E−02	7.23E+75	−7.18E−23
180	7.23E+75	4.98E+52	5.89E+75	6.49E−02	3.93E+75	1.96E−33
240	7.23E+75	1.29E+70	3.63E+75	1.86E−02	4.29E+63	1.18E−44
300	5.24E+75	9.62E+75	7.64E+67	1.84E−02	1.50E+34	4.20E−55
360	4.24E+75	1.32E+75	1.35E+49	9.33E−03	2.19E−04	4.46E−66
420	8.22E+74	4.13E+74	6.94E+22	2.49E−03	5.53E−47	0.00E+00
480	3.62E+75	1.05E+75	8.97E−04	8.77E−04	0.00E+00	0.00E+00
E_{ap}	6.88E+75	5.36E+00	7.24E+75	2.68E−03	7.23E+75	1.25E+02

TABLE 4
($m = 2{,}100.$)

k	$\lvert a_k\rvert$	k	$\lvert a_k\rvert$	k	$\lvert a_k\rvert$	k	$\lvert a_k\rvert$
1	3.67E−01	801	2.36E−02	1601	1.71E−02	2031	1.99E−05
201	8.52E+00	1001	3.80E−03	1801	4.47E−02	2051	1.11E−06
401	3.78E−01	1201	1.51E−02	2001	5.13E−02	2099	1.05E−11
601	5.69E−03	1401	1.22E−02	2011	2.78E−03	2100	2.79E−11

The numerical error is

$$E_{ap} = 7.17E-11. \tag{5.9}$$

Observe that the asymptotic decay of the coefficients starts when k crosses 2,000. We encounter a similar behavior of coefficients while expanding (5.8) in Chebyshev polynomials.

$$\cos(2{,}000z) = \sum_{k=0}^{\infty} b_k T_k(z), \qquad -1 \leqq z \leqq 1. \tag{5.10}$$

It can be shown [AbSt72] that

$$b_k = \begin{cases} 0 & k \text{ odd}, \\ 2J_k(2{,}000) & k \text{ even} \end{cases} \tag{5.11}$$

where J_k are Bessel functions of order k. By asymptotic analysis of Bessel functions we know that $J_k(R) \to 0$ exponentially fast only when $k > R$. This similarity between a_k and b_k could be anticipated since $T_k(z)$ satisfies

(1) $|T_k(z)| \leqq 1$, $|z| \leqq 1$

(2) Its zeros are uniformly distributed.

Where $R_k(z)$ satisfies

(1) $|R_k(z)| \sim 1$, $|z| \leqq 2$

(2) Its zeros are uniformly distributed.

In the third numerical experiment we have interpolated the function $\exp(z)$ in a diamond-shape domain in the complex plane whose vertices are $(-3, 0)$, $(0, 3)$, $(3, 0)$,

$(0, -3)$. Computing the logarithmic capacity numerically using (2.3), (2.4) with $z = (0., 0.)$, we obtained $\rho = 2.504$. The interpolating points were generated by the subroutine given in Appendix A. The first 20 points are:

k	z_k	k	z_k
0	(3.000, 0.000)	10	(−0.568, 2.432)
1	(−3.000, 0.000)	11	(−0.568, −2.432)
2	(0.000, 3.000)	12	(0.568, 2.432)
3	(0.000, −3.000)	13	(0.568, −2.432)
4	(1.500, 1.500)	14	(−2.432, 0.568)
5	(1.500, −1.500)	15	(−2.432, −0.568)
6	(−1.500, 1.500)	16	(2.796, 0.204)
7	(−1.500, −1.500)	17	(2.796, −0.204)
8	(2.432, 0.568)	18	(−0.204, 2.796)
9	(2.432, −0.568)	19	(−0.204, −2.796)

Table 5 presents the numerical results where

$$E_{ap} = \frac{\max_i |f(z_i) - P_k(z_i)|}{\max_{z \in D} |f(z)|}, \qquad 1 \leqq i \leqq 1{,}000 \tag{5.12}$$

and z_i are checkpoints distributed evenly on the boundary of D.

In the next numerical experiment, reported in Table 6, we have interpolated the function $1/z$ in the rectangular whose vertices are $(1, 1)$, $(50, 1)$, $(50, -1)$, $(1, -1)$. We

TABLE 5

k	E_{ap}	$\lvert a_k \rvert$
2	5.784E+00	8.361E+00
4	3.129E+00	2.871E+00
6	2.320E+00	1.341E+00
8	6.831E−02	1.234E−01
10	3.144E−02	1.816E−03
12	1.973E−03	8.338E−04
14	1.799E−04	3.372E−05

TABLE 6

k	$E_{ap}(2.1)$	$E_{ap}(2.2)$	$\frac{E_{ap}(2.2)}{E_{ap}(2.1)}$	k	$E_{ap}(2.1)$	$E_{ap}(2.2)$	$\frac{E_{ap}(2.2)}{E_{ap}(2.1)}$
4	5.79E−01	5.79E−01	1	44	1.66E−03	5.66E−03	3.41
8	4.20E−01	4.20E−01	1	48	9.20E−04	1.79E−03	1.98
12	2.54E−01	3.54E−01	1.39	52	4.90E−04	1.53E−03	3.12
16	1.37E−01	1.37E−01	1	56	2.66E−04	7.27E−04	2.74
20	5.98E−02	1.19E−01	1.99	60	1.40E−04	5.54E−04	3.94
24	3.45E−02	7.11E−02	2.06	64	7.36E−05	7.36E−05	1
28	2.21E−02	5.69E−02	2.57	68	3.88E−05	6.50E−05	1.67
32	1.19E−02	1.19E−02	1	72	2.06E−05	4.46E−05	2.16
36	5.98E−03	1.05E−02	1.75	76	1.09E−05	3.70E−05	3.37
40	3.09E−03	6.88E−03	2.23	80	5.87E−06	1.30E−05	2.21

were interested in comparing Algorithms 2.1 and 2.2. As could be predicted, when the number of points is not power of two, Algorithm 2.1 results in higher accuracy. E_{ap} is given by (5.12).

Polynomial approximation of a function of a matrix. Let us have the following parabolic partial differential equation:

$$U_t = U_{xx} + U_{yy}, \qquad |x| < 1, \qquad |y| < 1, \tag{5.13a}$$

$$U(x, 0) = \sum_{j=1}^{4} \sin(\pi j x) \sin(\pi j y), \tag{5.13b}$$

$$U = 0 \quad \text{on the boundaries of the unit square.}$$

Using the following space discretization:

$$U_{ij} = U(x_i, y_j, t), \tag{5.14a}$$

$$x_i = -1 + i\Delta x, \qquad 1 \leqq i \leqq N, \tag{5.14b}$$

$$y_j = -1 + j\Delta y, \qquad 1 \leqq j \leqq N, \tag{5.14c}$$

$$\Delta x = \Delta y = \frac{2}{N+1} \tag{5.14d}$$

where N is the number of gridpoints at each direction, we approximate the space derivatives by the standard second-order central differencing:

$$U_{xx} \sim (U_{i+1,j} - 2U_{ij} + U_{i-1,j})/\Delta x^2, \tag{5.15a}$$

$$U_{yy} \sim (U_{i,j+1} - 2U_{ij} + U_{i,j-1})/\Delta y^2. \tag{5.15b}$$

Thus we get the following semidiscrete representation of (5.13):

$$(U_N)_t = G_N U_N, \tag{5.16a}$$

$$U_N^0 = U^0 \tag{5.16b}$$

where U_N is a vector of dimension N^2, G_N is an $N^2 \times N^2$ matrix which results from the space discretization and

$$U_{ij}^0 = u(x_i, y_j, 0), \qquad 1 \leqq i, \qquad j \leqq N. \tag{5.17}$$

The formal solution of (5.16) is

$$U_N = \exp(tG_N) U_N^0. \tag{5.18}$$

Now, by using polynomial approximation of $\exp(tG_N)$, we get

$$U_N^k(t) = P_k(tG_N) U_N^0 \tag{5.19}$$

where $P_k(tz)$ approximates $\exp(tz)$ in the domain which includes all the eigenvalues of G_N. The numerical solution was computed at $t = 0.1$. A simple Fourier analysis

shows that $D=[a, b]$ where

$$a=-2(N+1)^2, \qquad b=(N+1)^2\left[\cos\left(\frac{2}{N+1}\right)-1\right]\Big/2.$$

Therefore the interpolating points are

$$y_j=\tfrac{1}{2}[(b-a)x_j+b+a]$$

while x_j are given by (3.13). The exact solution of (5.13) is

$$U(x, y, t)=\sum_{j=1}^{4}\exp\left(-(j\pi)^2 t\right)\sin(j\pi x)\sin(j\pi y). \tag{5.20}$$

Tables 7 and 8 present the dependence of three factors on k, the degree of the interpolating polynomial.

$|a_k|$ is the absolute value of the maximal divided difference.

$l_2r=\|a_k r\|/\|w\|$ is the relative residual which represents the time accuracy.

$l_2w=\|U-w\|/\|w\|$ is the relative error which represents the space+time accuracy.

r and w are as defined in Algorithm 4.1 (w is actually U_N^k).

For $N=8$, D is

$$D=\{x\,|\,-162\leqq x\leqq -0.9\} \tag{5.21}$$

and the results are given in Table 7.

For $N=16$, D is

$$D=\{x\,|\,-578\leqq x\leqq -0.9\} \tag{5.22}$$

and the results are given in Table 8.

TABLE 7
($N=8$.)

k	$\|a_k\|$	l_2r	l_2w	k	$\|a_k\|$	l_2r	l_2w
2	2.45E−01	7.79E+00	7.01E+00	12	2.73E−03	1.85E−02	8.30E−02
4	1.79E−01	5.42E+00	3.27E+00	14	2.25E−05	1.92E−03	8.23E−02
6	8.29E−02	2.32E+00	1.74E+00	16	1.11E−06	1.48E−04	8.23E−02
8	1.31E−02	5.81E−01	2.54E−01	18	6.15E−08	1.77E−06	8.23E−02
10	2.40E−03	9.18E−02	1.06E−01	20	2.97E−09	1.94E−07	8.23E−02

TABLE 8
($N=16$.)

k	$\|a_k\|$	l_2r	l_2w	k	$\|a_k\|$	l_2r	l_2w
3	1.67E−01	2.44E−01	9.19E+00	18	9.50E−04	3.28E−02	9.76E−02
6	2.88E−02	1.67E−01	2.64E+00	21	6.32E−05	1.80E−03	2.28E−02
9	2.29E−02	5.19E−01	3.59E+00	24	1.64E−05	3.05E−03	2.29E−02
12	4.08E−02	1.35E+00	2.36E−01	27	5.92E−06	3.71E−04	2.26E−02
15	6.50E−03	3.05E−01	2.46E−01	30	2.56E−07	4.96E−05	2.26E−02

Observing Tables 7 and 8, we see that for $N=8$ we need 14 iterations to recover the space accuracy and for $N=16$ we need 27 iterations. This result compares favorably with standard explicit algorithms like forward Euler or Runge-Kutta, where the number of iterations (timesteps) is multiplied by four (due to stability limitations) whenever we increase the space resolution by two. In [Tale85] we describe an algorithm which is spectral in time. It is shown there that for parabolic problems, the number of matrix-vector multiplications depends almost linearly on the space resolution parameter. Since the algorithm used in the present paper can also be considered as spectral in time, it explains this improvement in the number of matrix-vector multiplications needed to march the solution to a given time level.

More realistic geophysical problems solved by this interpolation approach is described in [TaCK89].

Appendix A. In order to map the exterior of the unit disc to the exterior of a polygon, symmetric about the real axis, we can treat the upper half of the exterior of the polygon as an interior of a polygon with vertex at infinity. Then, the desired mapping function is interior Schwarz-Cristoffel transformation composed with $f(u)$ which maps the upper half of the exterior of the unit disc on the interior of the unit disc. $f(u)$ can be written as $f(u)=(f_2\circ f_1)(u)$ where

$$f_1(u)=\frac{1}{2}\left(u+\frac{1}{u}\right), \tag{A1}$$

$$f_2(u)=u_a\frac{u-\bar{u}_b}{u-u_b}. \tag{A2}$$

u_a is a point on the unit circle and $\bar{u}_b$ is a point in the upper halfplane. Fixing u_a and u_b will be made clear by the following example. Let D be a rectangle with vertices (taken in counterclockwise direction): $(1,-2)$, $(1,2)$, $(-4,2)$, $(-4,-2)$. Using the Schwarz-Cristoffel routines we can map the interior of a polygon with the following vertices (counterclockwise):

$$\tilde{z}_1=(1,0),\quad \tilde{z}_2=\infty,\quad \tilde{z}_3=(-4,0),\quad \tilde{z}_4=(-4,2),\quad \tilde{z}_5=(1,2)$$

on the interior of the unit disc.

Let u_i be the images of $\tilde{z}_i$

$$u_i=\tilde{\varphi}(\tilde{z}_i),\qquad 1\leqq i\leqq 5. \tag{A3}$$

($\tilde{\varphi}(z)$ is the interiors Schwarz-Cristoffel mapping function.) In order that ∞ will be mapped to ∞, u_a must satisfy

$$u_a=u_2. \tag{A4}$$

u_b is given by satisfying the two equations

$$f(1)=u_1, \tag{A5a}$$

$$f(-1)=u_3, \tag{A5b}$$

or

$$u_2\frac{1-\bar{u}_b}{1-u_b}=u_1, \tag{A6a}$$

(A6b) $$u_2 \frac{-1-\bar{u}_b}{-1-u_b} = u_3.$$

Solving (A6) we get

(A7) $$u_b = \frac{2u_2 - u_1 - u_3}{u_3 - u_1}.$$

Let

(A8) $$w_j = x_j + iy_j$$

be a set of uniformly distributed points on the upper half of the unit circle; then the desired interpolating points z_i on the upper boundary of the polygon are given by

(A9) $$z_j = \tilde{\psi}\left(u_2 \frac{x_j - \bar{u}_b}{x_j - u_b}\right)$$

where $\tilde{\psi}$ is the inverse of $\tilde{\varphi}$.

The following subroutine carries out the mapping $\tilde{\psi}$.

```
      SUBROUTINE SCPNTS (W1,NVRT,X,ZW,NN)
C
C  PURPOSE
C
C       SCPNTS COMPUTES FEJER POINTS ON THE UPPER PART
C       OF A POLYGON USING SCPACK ROUTINES (TREF80).
C
C  INPUT PARAMETERS
C
C       W1     VECTOR WHICH CONTAINS THE UPPER VERTICES
C              OF THE POLYGON SUCH THAT W1(1)=(A,0),···,W1(NVRT)=(B,0)
C              AND A<B
C       NVRT   DIMENSION OF W1.
C       X      UNIFORMLY DISTRIBUTED POINTS ON [-1,1].
C       NN     DIMENSION OF X AND ZW.
C
C  OUTPUT PARAMETERS
C
C       ZW    FEJER POINTS ON THE UPPER PART OF THE POLYGON.
C
      IMPLICIT COMPLEX*16(C,W,Z)
      IMPLICIT DOUBLE PRECISION (A-B,D-H,O-V,X-Y)
      DIMENSION Z(20),W(20),BETAM(20),QWORK(1000),ZW(NN)
      1,X(NN),W1(20)
      ZERO=(0.D0,0.D0)
      ZI=(0.D0,1.D0)
      H=0.D0
      DO 99 I=1,NVRT
      W(I)=W1(I)
      HH=DIMAG(W(I))
      IF(HH.GT.H)H=HH
   99 CONTINUE
```

```
      H2=2*H
      IF(H2.LT.10.D0)H2=10.D0
      RL=(W(1)+W(NVRT))/2.D0
      WC=DCMPLX(RL,H2)
      N=NVRT+1
      ZZ1=W(1)
      W(1)=W(NVRT)
      DO 98 I=3,N-1
   98 W(N+3-I)=W(N+1-I)
      W(2)=(99.D0,99.D0)
      W(3)=ZZ1
      NPTSQ=12
      PI=2.D0*DASIN(1.D0)
      XN=DREAL(W(N))
      YN=DIMAG(W(N))
      X1=W(1)
      X3=W(3)
      X4=DREAL(W(4))
      Y4=DIMAG(W(4))
      AB=XN-X1
      BETAM(1)=-.5D0
      IF(AB.NE.0.D0)BETAM(1)=DATAN(YN/AB)/PI
      IF(AB.GT.0.D0)BETAM(1)=-1.D0+BETAM(1)
      BETAM(2)=-2.D0
      AB=X3-X4
      BETAM(3)=-.5D0
      IF(AB.NE.0.D0)BETAM(3)=DATAN(Y4/AB)/PI
      IF(AB.GT.0.D0)BETAM(3)=-1.D0+BETAM(3)
      CALL ANGLES(N,W,BETAM)
      CALL QINIT(N,BETAM,NPTSQ,QWORK)
      IPRINT=0
      IGUESS=0
      TOL=1.D-10
      CALL SCSOLV(IPRINT,IGUESS,TOL,ERREST,N,C,Z,WC,W,BETAM
      1,NPTSQ,QWORK)
      ARG=PI/(NN-1)
      ZOC=(2.D0*Z(2)-Z(1)-Z(3))/(Z(3)-Z(1))
      ZO=CONJG(ZOC)
      DO 33 I=1,NN
      ZZ=Z(2)*(X(I)-ZO)/(X(I)-ZOC)
      CALL NEARZ(ZZ,ZN,WN,KN,N,Z,WC,W,BETAM)
      ZW(I)=WSC(ZZ,0,ZN,WN,KN,N,C,Z,BETAM,NPTSQ,QWORK)
   33 CONTINUE
      RETURN
      END
```

Remark. While this article was in the refereeing process, a paper by B. Fischer and L. Reichel addressing Newton interpolation was published [FiRe89]. The interested reader can find there some mathematical proofs to the instability phenomenon decribed in both papers.

REFERENCES

[AbSt72] M. ABRAMOWITZ AND I. A. STEGUN, *Handbook of Mathematical Functions*, Dover, New York, 1972.

[AnGo72] R. S. ANDERSSEN AND G. H. GOLUB, *Richardson's non-stationary matrix iterative procedure*, Res. Report STAN-CS-72-304, Department of Computer Science, Stanford University, Stanford, CA, 1972.

[CodB82] S. D. CONTE AND C. DE BOOR, *Elementary Numerical Analysis*, McGraw-Hill International Student Edition, McGraw-Hill, New York, 1982.

[Davi75] P. J. DAVIS, *Interpolation and Approximation*, Dover, New York, 1975.

[FiRe87] B. FISCHER AND L. REICHEL, *A stable Richardson iteration method for complex linear systems*, Preprint 10, Institut für Angewandte Mathematik der Universität Hamburg, Hamburg, FRG, 1987.

[FiRe89] ———, *Newton interpolation in Fejér and Chebyshev points*, Math. Comp., 53 (1989), pp. 265–278.

[HaYo81] L. HAGEMAN AND D. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.

[LeFi76] V. I. LEBEDEV AND S. A. FINOGENOV, *Utilization of ordered Chebyshev parameters in iterative methods*, U.S.S.R. Comput. Math. and Math. Phys., 16 (1976), pp. 70–83.

[Mark77] A. I. MARKUSHEVICH, *Theory of Functions of a Complex Variable*, Chelsea, New York, 1977.

[MoVa78] C. B. MOLER AND C. F. VAN LOAN, *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Rev., 20 (1978), pp. 801–836.

[SmLe68] V. I. SMIRNOV AND N. A. LEBEDOV, *Functions of a Complex Variable*, ILIFFE Books Ltd., London, U.K., 1968.

[Rivl74] T. J. RIVLIN, *The Chebyshev Polynomials*, John Wiley, New York, 1974.

[Tale86] H. TAL-EZER, *Spectral methods in time for hyperbolic equations*, SIAM J. Numer. Anal., 23 (1986), pp. 11–26.

[Tale85] ———, *Spectral methods in time for parabolic problems*, SIAM J. Numer. Anal., 26 (1989), pp. 1–11.

[Tale87] ———, *Polynomial approximation of functions of matrices and its application to the solution of a general system of linear equations*, J. Sci. Comput. 4 (1989), pp. 25–60.

[TaCK89] H. TAL-EZER, J. M. CARCIONE, AND D. KOSLOFF, *An accurate and efficient scheme for wave propagation in linear viscoelastic media*, Geophysics, submitted.

[Tref80] L. N. TREFETHEN, *Numerical computation of the Schwarz-Christoffel transformation*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 82–102.

[Wals56] J. L. WALSH, *Interpolation and Approximation by Rational Functional Functions in the Complex Domain*, American Mathematical Society, Providence, RI, 1956.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 668-680, May 1991
© 1991 Society for Industrial and Applied Mathematics
010

FOURIER ANALYSIS OF RELAXED INCOMPLETE FACTORIZATION PRECONDITIONERS*

TONY F. CHAN†

Abstract. Fourier analysis is used to study the behavior of a class of incomplete factorization preconditioners for elliptic problems, which blends the classical ILU and MILU preconditioners via a scalar relaxation parameter $\alpha \in [0, 1]$. An expression is obtained for the eigenvalues of the preconditioned system for a model Poisson problem with periodic boundary conditions, which yields information on both the condition number $K(\alpha)$ and the eigendistribution of the preconditioned system. An optimal value is derived for α and it is shown that $K(\alpha_{\text{opt}}) = O(h^{-1})$. The Fourier results agree extremely well with numerical results for the model Poisson problem with *Dirichlet* boundary conditions, even though the Fourier analysis is not exact for this problem. For example, they predict the sensitive behavior near $\alpha = 1$ (MILU). Finally, it is shown that the relaxed methods are closely related to the classical "modified" ILU (MILU) method.

Key words. incomplete factorizations, preconditioners, Fourier analysis, conditioner numbers

AMS(MOS) subject classifications. 65N20, 65F10

1. Introduction. The class of incomplete factorization methods (ILU, MILU) [6], [8] has been used very successfully as preconditioners for linear systems of the form $Ax = b$, especially when A corresponds to discretizations of elliptic partial differential equations. The ILU method computes an approximate LU factorization M of A based on Gaussian elimination in which fillins at the (i, j)th element is dropped if $(A)_{ij} = 0$. In the MILU method, the dropped fillins are added back to the diagonal entry plus an additional term ch^2, where c is a constant and h is the meshsize. For matrices A arising from discretizations of second-order elliptic problems, it can be proved [5], [10], [6], [9] that the condition number $K(M^{-1}A)$ is bounded by $O(h^{-2})$ and $O(h^{-1})$ for ILU and MILU ($c \neq 0$), respectively. Tests in [10] also showed that the number of PCG iterations for MILU varies very little with c. While these theoretical results may indicate a clear superiority for the MILU method, it has been observed in some practical problems that the MILU method (especially with $c = 0$) does not work as well as the ILU method [7]. This phenomenon led some researchers to consider a relaxed version of the two methods, which is similar to the MILU method except that only a fraction, say $\alpha \in [0, 1]$, of the dropped fillins are added back to the diagonal [2], [3], [7]. Thus, when $\alpha = 0$ we get the ILU method; and when $\alpha = 1$ we get the MILU method with $c = 0$. We shall denote this relaxed method by RILU (α).[1] Numerical experiments in [1]–[3], [7] indicate that the number of iterations needed to reduce an initial residual by a fixed amount (when accelerated by the preconditioned conjugate gradient method) typically behaves as in Fig. 1 (for a more accurate plot, see [7]), which shows an extremely sensitive behavior near $\alpha = 1$, especially for three-dimensional problems with badly behaved coefficients. This phenomenon has not been well understood. In [7], it is attributed to "an early loss of orthogonality in the CG iteration, which spoils the convergence behavior." A value of $\alpha = 0.95$ is recommended as a safe cure.

* Received by the editors November 14, 1988; accepted for publication (in revised form) May 29, 1990. This work was supported in part by Department of Energy contract DE-FG03-87ER25037, National Science Foundation contract NSF-DMS87-14612, and Army Research Office contract DAAL03-88-K-0085.

† Department of Mathematics, University of California, Los Angeles, California 90024.

[1] If A is symmetric positive definite, the corresponding preconditioner has been called the relaxed incomplete Cholesky (RIC) preconditioner in [2].

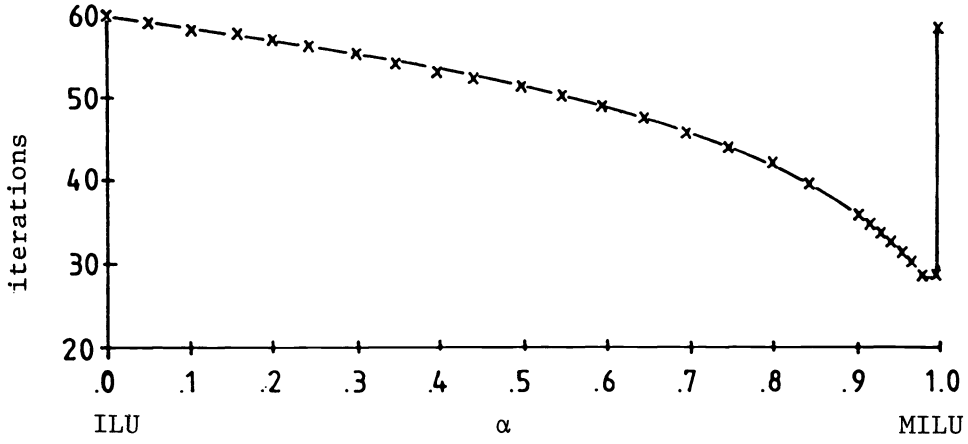


FIG. 1. *Convergence behavior of the* RILU (α) *method for a three-dimensional problem.*

The main purpose of this paper is to analyze (in § 2) the convergence behavior of the RILU (α) method. The tool we shall use is Fourier analysis (following the framework in [4]), which is similar to the von Neumann stability analysis for finite-difference schemes for time-dependent problems. It is theoretically applicable only for constant coefficient problems with periodic boundary conditions, but the results in Chan and Elman [4] indicate that it can successfully predict the behavior of most classical iterative methods and preconditioners even for more general classes of problems with other boundary conditions. In particular, the results for the periodic problem with a meshsize h_p can be used to predict the convergence behavior of the corresponding Dirichlet boundary condition problem with meshsize $h_d = 2h_p$. In addition to deriving the condition number $K(M^{-1}A)$, the Fourier analysis also gives the eigenvalue distribution, which is extremely valuable in understanding the convergence behavior of the method. Moreover, we can easily derive optimal values for parameters of the method (such as α in the RILU (α) method).

For the RILU (α) method applied to the model Poisson problem, the Fourier analysis (§ 3) predicts an optimal value of $\alpha_{\text{opt}} = 1 - 8\sin^2 \pi h$ and $K(\alpha_{\text{opt}}) = h^{-1}2\pi + O(1)$. The estimate for $K(\alpha_{\text{opt}})$ agrees with the result derived earlier in [10] for the MILU method. The Fourier analysis also successfully predicts the behavior of the spectrum of $M^{-1}A$ as α varies. In § 4, we present some numerical results for the Dirichlet model problem which seem to verify these predictions. However, while there is excellent general agreement, there is still some discrepancy for the special case $\alpha = 1$. Finally in § 5, we show that there is a close relationship between the RILU (α) method and the classical MILU (c) method.

2. Fourier analysis of the RILU (α) method. We shall restrict our attention to the model problem in two dimensions:

$$-\Delta u = f$$

with periodic boundary conditions on the unit square $(0, 1) \times (0, 1)$ discretized by the standard second-order five-point stencil on a uniform $n \times n$ mesh with size $h \equiv 1/(n+1)$. The matrix A can then be represented by the stencil in Fig. 2. The RILU (α) method computes a preconditioner M of a form represented by the product of the stencils in Fig. 3.

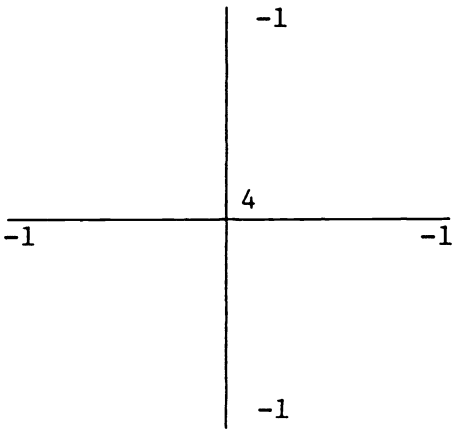


FIG. 2. *Stencil for A.*

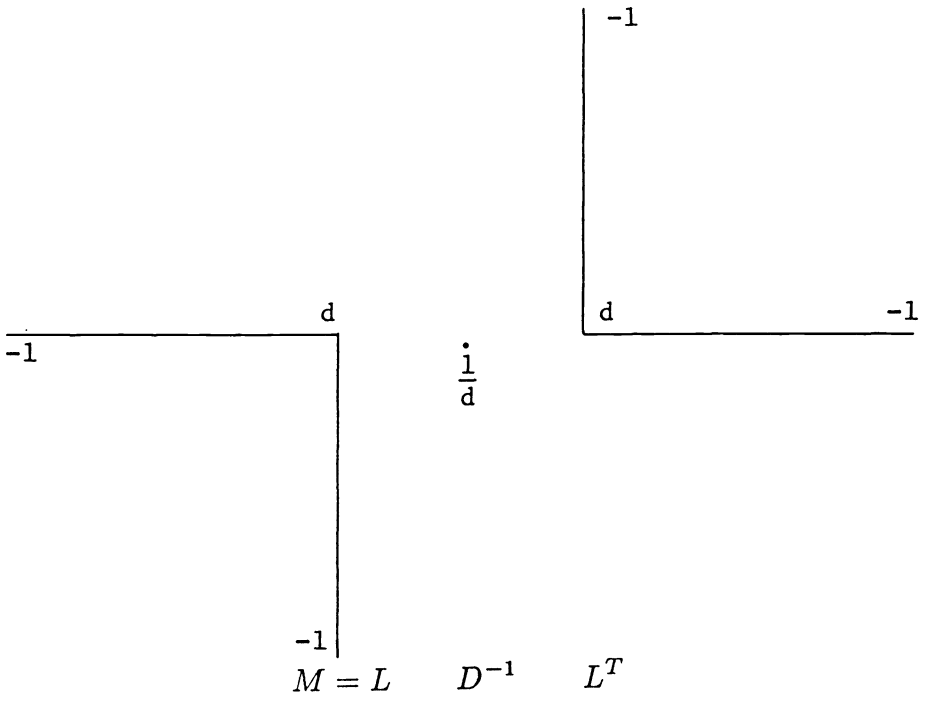


FIG. 3. *Stencils for factors of* RILU (α).

Note that due to the periodic boundary conditions, the matrix corresponding to L is not lower triangular (as in the classical preconditioner for Dirichlet boundary conditions) but circulant. It can easily be verified that the product M is given by Fig. 4. The fillins correspond to the two $1/d$ terms, a fraction of which, $\alpha(2/d)$, when added to the diagonal $d+2/d$, should match the diagonal entry of A. This gives the following equation for d:

$$d+\frac{2}{d}+\frac{2\alpha}{d}=4, \tag{1}$$

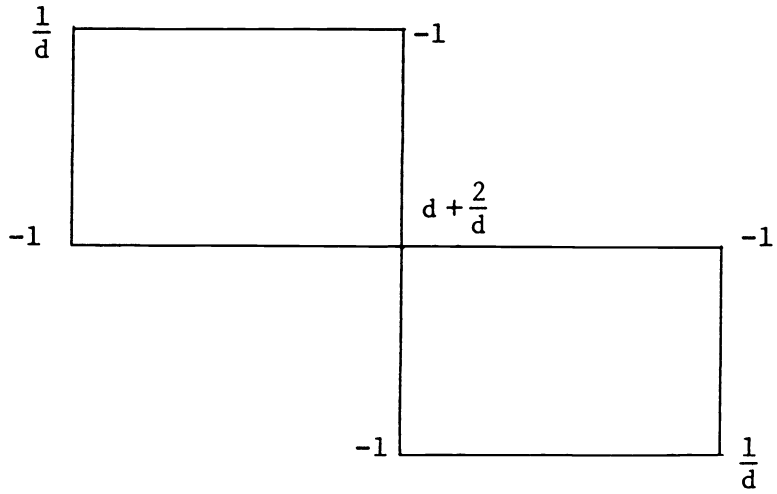


FIG. 4. *Stencil for M of* RILU (α).

which gives

$$d = 2 + \sqrt{2(1-\alpha)}. \tag{2}$$

The Fourier analysis proceeds by computing the action of the operator $M^{-1}A$ on the eigenvectors represented by the Fourier modes whose value at the (j, k)th gridpoint is given by

$$\varphi_{st} = e^{ij\theta_s} e^{ik\phi_t} \tag{3}$$

where

$$i \equiv \sqrt{-1}, \qquad \theta_s = 2\pi sh, \qquad \phi_t = 2\pi th, \qquad 1 \leqq s, \qquad t \leqq n. \tag{4}$$

(Following the framework in [4], we have excluded the indices $s = 0$ and $t = 0$, which correspond to φ_{st} being constant in one or both of the coordinates.) For example, the Fourier transform $\lambda_{st}(A)$ of A is defined by

$$A\varphi_{st} = \lambda_{st}(A)\varphi_{st}$$

and can be computed to be

$$\begin{aligned} \lambda_{st}(A) &= 4 - e^{i\theta_s} - e^{-i\theta_s} - e^{i\phi_t} - e^{-i\phi_t} \\ &= 4\left(\sin^2\frac{\theta_s}{2} + \sin^2\frac{\phi_t}{2}\right). \end{aligned} \tag{5}$$

Similarly, the Fourier transform $\lambda_{st}(M)$ of M is

$$\begin{aligned} \lambda_{st}(M) &= [d - e^{-i\theta_s} + e^{-i\phi_t})]\frac{1}{d}[d - (e^{i\theta_s} + e^{i\phi_t})] \\ &= 4\left(\sin^2\frac{\theta_s}{2} + \sin^2\frac{\phi_t}{2}\right) + \frac{2}{d}(\cos(\theta_s - \phi_t) - \alpha). \end{aligned} \tag{6}$$

Thus the Fourier transform μ_{st} of $M^{-1}A$ (i.e., the eigenvalues of $M^{-1}A$) is given by

$$\mu_{st}(\alpha) = \frac{4\left(\sin^2\frac{\theta_s}{2} + \sin^2\frac{\phi_t}{2}\right)}{4\left(\sin^2\frac{\theta_s}{2} + \sin^2\frac{\phi_t}{2}\right) + \left(\frac{2}{2+\sqrt{2(1-\alpha)}}\right)(\cos(\theta_s - \phi_t) - \alpha)}. \tag{7}$$

The condition number $K(\alpha)$ is defined by

$$K(\alpha)=\frac{\max_{s,t}\mu_{st}(\alpha)}{\min_{s,t}\mu_{st}(\alpha)}. \tag{8}$$

From (7), $K(\alpha)$ can be easily computed numerically for a given meshsize h. In Fig. 5, $K(\alpha)$ is plotted versus α for the case $n=40$. The sensitive behavior near $\alpha=1$ is evident.

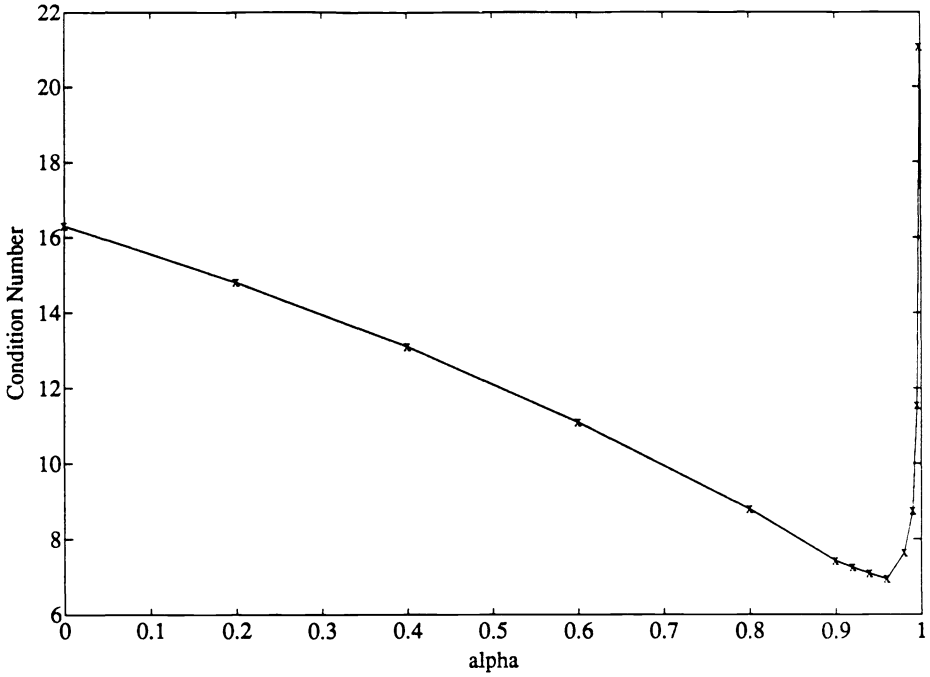


FIG. 5. *Condition number of* RILU (α) *versus* α, $n=40$.

In Figs. 6(a)-6(e), the eigenvalue distribution for five representative values of α are plotted in a three-dimensional perspective. In Fig. 7, the same eigenvalues are plotted in a linear fashion. (Similar plots for the Dirichlet model problem are given in [2].) In Fig. 8, the maximum and minimum eigenvalues of $M^{-1}A$ are plotted as a function of α. We can make several observations from these plots:

1. In all cases, there is a clustering of eigenvalues around one.

2. For $\alpha\approx 0$ (ILU) most of the eigenvalues are less than or equal to one, while for $\alpha\approx 1$ (MILU) most are greater than or equal to one.

3. The three-dimensional plots show clearly the transition from ILU to MILU. The two "humps" near $(s, t)=(1, n)$ and $(n, 1)$ shoot up steeply when α rises past the optimal value of ≈ 0.96. These account for the large eigenvalues of $M^{-1}A$. The small eigenvalues of A occur near $(s, t)=(1, 1)$, (n, n) and rise smoothly from ≈ 0 to ≈ 1 as α goes from zero to one. Thus the sensitive behavior in Figs. 1 and 5 can be traced to the sharp rise of the large eigenvalues of $M^{-1}A$ as α approaches one.

4. The condition number $K(\alpha)$ is substantially lower at the optimal value of $\alpha(\approx 0.96)$ than at either $\alpha=0$ or $\alpha=1$.

3. Optimal relaxation parameter. In this section, we shall derive the optimal value for α which minimizes the condition number $K(\alpha)$. To do this, we first find the minimum and maximum eigenvalues as a function of α.

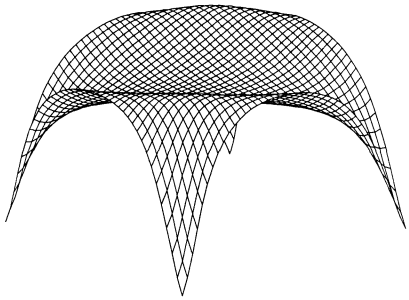


FIG. 6(a). *Fourier eigenvalues for* RILU, $\alpha = 0$.

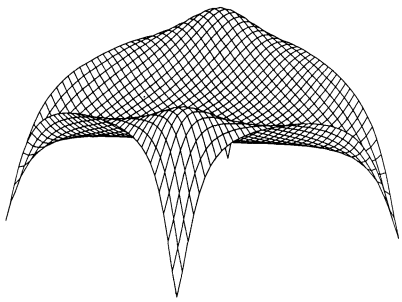


FIG. 6(b). *Fourier eigenvalues for* RILU, $\alpha = 0.6$.

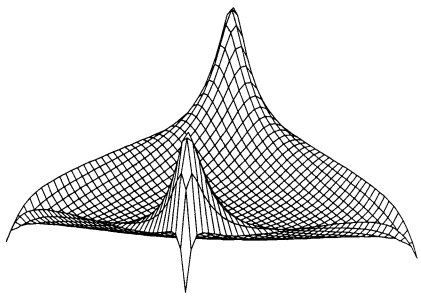


FIG. 6(c). *Fourier eigenvalues for* RILU, $\alpha = 0.96$.

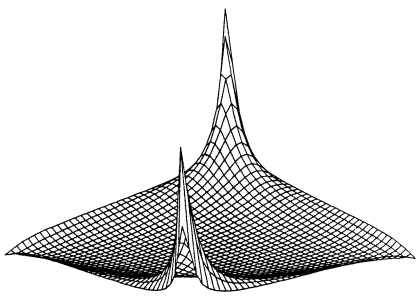


FIG. 6(d). *Fourier eigenvalues for* RILU, $\alpha = 0.995$.

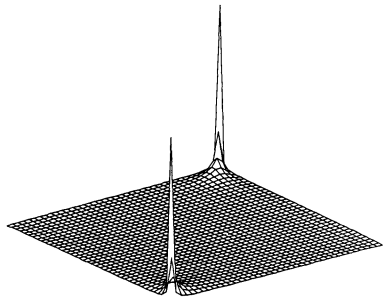


FIG. 6(e). *Fourier eigenvalues for* RILU, $\alpha = 1$.

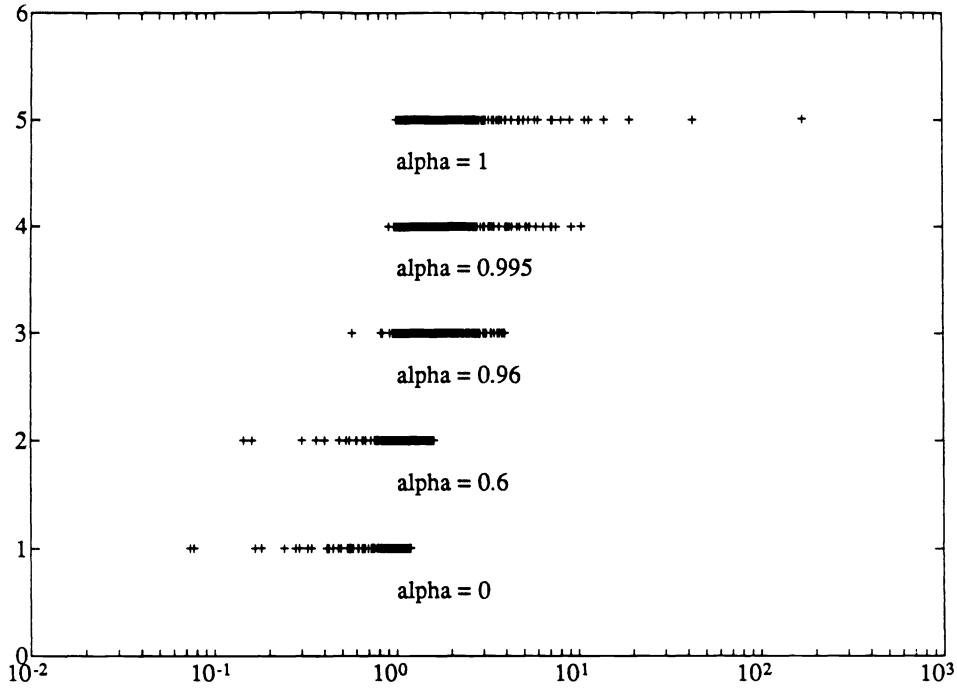


FIG. 7. *Distribution of Fourier eigenvalues for* RILU.

First we show that the maximum eigenvalue occurs at $(s, t) = (1, 1)$. This follows from the inequalities

$$\begin{aligned} \mu_{s,t} &\geqq \frac{g_{s,t}}{g_{s,t} + 2(1-\alpha)/(2+\sqrt{2(1-\alpha)})} \\ &\geqq \frac{g_{1,1}}{g_{1,1} + 2(1-\alpha)/(2+\sqrt{2(1-\alpha)})} \\ &= \mu_{1,1} \end{aligned} \tag{9}$$

where

$$g_{s,t} \equiv 4\left(\sin^2\frac{\theta_s}{2} + \sin^2\frac{\phi_t}{2}\right).$$

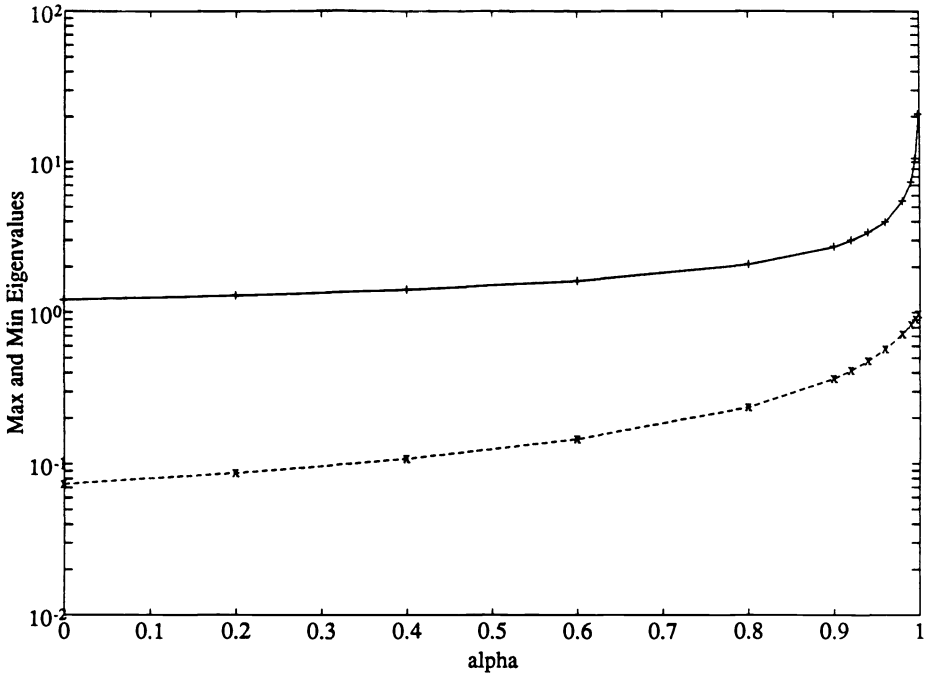


FIG. 8. Max *and* min *Fourier* E.V. *versus* α *for* RILU, $n = 40$.

To simplify the algebra, we define

$$\varepsilon^2 \equiv 2(1-\alpha),$$

with $\varepsilon \in [0, \sqrt{2}]$.

Thus, we obtain from (7) that

$$\mu_{\min}(\varepsilon) = \mu_{11}(\varepsilon) = \frac{\beta}{\beta + \varepsilon^2/8(2+\varepsilon)}, \tag{10}$$

where

$$\beta \equiv \sin^2 \pi h. \tag{11}$$

Next we shall find the maximum eigenvalue. We shall treat the variables θ_s and ϕ_t as continuous variables θ and ϕ, which allows us to use differential calculus to find the maximum. This is justified in the sense that any maximum found this way is achievable in the discrete case for small enough h.

Due to the symmetries

$$\mu(\theta, \phi) = \mu(\phi, \theta)$$

and

$$\mu(\theta, \phi) = \mu(2\pi - \phi, 2\pi - \theta),$$

we can restrict our search for the maximum to the triangular region T defined by

$$T = \{(\theta, \phi) \mid \phi \geqq 0,\ \theta + \phi \leqq 2\pi,\ \theta \geqq \phi\}.$$

By differentiating $\mu(\theta, \phi)$ in (7) with respect to the variable $\theta + \phi$ while keeping $\theta - \phi$ fixed, it can be verified that μ is nonincreasing or nondecreasing along any $\theta - \phi =$ constant line in T. Therefore, the maximum of μ must occur on the lines $\phi = 0$ or $\theta + \phi = 2\pi$.

To obtain the maximum eigenvalue on the line $\theta+\phi=2\pi$, we eliminate the variable ϕ and obtain

$$\mu(\theta)=\frac{8\sin^2\frac{\theta}{2}}{8\sin^2\frac{\theta}{2}+\left(\frac{2}{2+\varepsilon}\right)\left(\cos 2\theta-1+\frac{\varepsilon^2}{2}\right)}. \tag{12}$$

Noting that

$$\cos 2\theta-1=-8\sin^2\frac{\theta}{2}\left(1-\sin^2\frac{\theta}{2}\right) \tag{13}$$

and defining

$$x\equiv\sin^2\frac{\theta}{2}, \tag{14}$$

we can simplify (12) to obtain

$$\mu(x)=\frac{(2+\varepsilon)x}{\varepsilon x+2x^2+\varepsilon^2/8}. \tag{15}$$

Since $\mu(\theta=0)=\mu(\theta=2\pi)=0$ and $\mu(\theta)$ is symmetric about $\theta=\pi$, it follows that the maximum must occur at an interior point where $\mu'(x)=0$. We find $\mu'(x)=0$ when

$$\varepsilon x+2x^2+\frac{\varepsilon^2}{8}=x(\varepsilon+4x),$$

which gives the value of x where $\mu_{\max}$ occurs:

$$x_{\max}=\frac{\varepsilon}{4}. \tag{16}$$

Substituting the value of $x_{\max}$ into (15), we get

$$\mu^{(1)}_{\max}(\varepsilon)=\frac{1}{2}+\frac{1}{\varepsilon}. \tag{17}$$

For the maximum on the line $\phi=0$, it can be easily verified that it occurs at $\theta=\pi$ with

$$\mu^{(2)}_{\max}(\varepsilon)=\frac{4}{4+2/(2+\varepsilon)(\varepsilon^2/2-2)}.$$

A simple calculation shows that

$$\mu^{(1)}_{\max}(\varepsilon)\geqq\mu^{(2)}_{\max}(\varepsilon)$$

whenever $\varepsilon\leqq 2$. Since $\varepsilon\leqq\sqrt{2}$, it follows that $\mu_{\max}=\mu^{(1)}_{\max}$.

Therefore, from (10) and (17) we get

$$K(\varepsilon)=\frac{\left(\frac{1}{2}+\frac{1}{\varepsilon}\right)}{\left(\beta\Big/\left(\beta+\frac{\varepsilon^2}{8(2+\varepsilon)}\right)\right)}=\frac{8\beta(2+\varepsilon)+\varepsilon^2}{16\beta\varepsilon}.$$

To find the minimum value of $K(\varepsilon)$, we find that $K'(\varepsilon)=0$ when

$$(8\beta+2\varepsilon)\varepsilon=8\beta(2+\varepsilon)+\varepsilon^2,$$

which gives

$$\varepsilon_{\text{opt}}=4\sqrt{\beta}\equiv 4\sin\pi h \tag{18}$$

or equivalently

$$\alpha_{\text{opt}}=1-8\sin^2\pi h. \tag{19}$$

Since $\varepsilon_{\text{opt}}\in[0,\sqrt{2}]$ for small enough h and $K(\varepsilon)$ is unimodal, the minimum must be interior and occurs at ε_{opt} as given in (18). After some manipulations, we obtain

$$K(\alpha_{\text{opt}})=\frac{1+\sin\pi h}{2\sin\pi h}. \tag{20}$$

Therefore

$$K(\alpha_{\text{opt}})=\frac{1}{2\pi}h^{-1}+O(1). \tag{21}$$

Remarks. 1. Note that $\alpha_{\text{opt}}=1-O(h^2)$. For $h=1/41$, $\alpha_{\text{opt}}\approx 0.953$, which is the value recommended in [7]. For smaller h, the value 0.95 will be an underestimate of the optimal value.

2. The optimal blending gives

$$K(\alpha_{\text{opt}})=O(h^{-1})$$

which is similar to that for the classical MILU method [5], [11].

3. From (17), we see that the value of $\mu_{\max}$ grows like $1/\varepsilon$ near $\varepsilon=0$ (i.e., $\alpha=1$) but is bounded below by $\frac{1}{2}+1/\sqrt{2}$ for any ε. From (10), we see that $\mu_{\min}$ grows from $O(h^2)$ at $\varepsilon=\sqrt{2}$ (i.e., $\alpha=0$) to one at $\varepsilon=0$ (i.e., $\alpha=1$).

4. Due to the sharp rise in $K(\alpha)$ as $\alpha\to 1$, it is better in practice to underestimate α_{opt} than to overestimate it.

5. From (17), it may appear that $\mu_{\max}$ becomes unbounded as $\varepsilon\to 0$. However, this results from the assumption that in (16), $x_{\max}$ can take on any value in $(0,1)$. However, for finite h, x must lie in $[\sin^2\pi h, 1)$. Therefore,

$$\mu_{\max}(\varepsilon=0)=\frac{1}{\sin^2\pi h}. \tag{22}$$

Since $\mu_{\min}(\varepsilon=0)=1$, this implies that

$$K(\varepsilon=0)=\frac{1}{\sin^2\pi h}=O(h^{-2}). \tag{23}$$

This agrees with the Fourier results in [4] for MILU with $c=0$, although for the model problem with Dirichlet boundary conditions, it has been observed experimentally that, for MILU, even with $c=0$, the condition number grows like $O(h^{-1})$. We note that the theory in [5] cannot predict this behavior either.[2] In [4], a plausible explanation is given, based on the fact that the $O(h^{-2})$ behavior in the Fourier case results from a very delicate cancellation in the denominator in (7) at $c=0$, which may not occur in the Dirichlet model problem. The Fourier results, together with the experimental results

[2] A more detailed analysis for the model problem carried out by Beauwen [11] does explain this phenomenon.

reported in [7] suggest that for practical problems the $O(h^{-2})$ behavior can indeed occur (unlike for the model problem) and therefore it is not a good idea to use $c=0$ in MILU (i.e., $\alpha=1$ in RILU) for more general problems.

4. Comparison with the Dirichlet model problem. We have carried out some numerical experiments to verify how well the Fourier results predict the behavior of the RILU preconditioner as applied to the model Poisson problem with *Dirichlet* boundary conditions.

To this end, we consider the model Dirichlet problem with the forcing function:

$$f(x, y) = x(x-1)y(y-1)\, e^{xy}.$$

We solve the discrete problem by the conjugate gradient method preconditioned by the Dirichlet RILU method (with nonconstant diagonals) using zero as an initial guess. We stop the iteration when the square of the two-norm of the residual has been reduced by a factor of 10^{-7}. The results are given in Table 1.

In Table 2, we give the values of

$$\alpha_{\text{opt}} = 1 - 8 \sin^2\left(\frac{\pi h_d}{2}\right),$$

which is obtained from (19) by taking $h = h_d/2$ (where $h_d = 1/(n_d+1)$) and the corresponding number of PCG iterations using the optimal values.

By comparing Tables 1 and 2, we see that:

1. Due to the discrete nature of the number of PCG iterations, it is difficult to locate precisely where the optimal value of α is from Table 1. Nonetheless, the Fourier α_{opt} is consistent with the data in Table 1. We note that Axelsson and Lindskog [2]

TABLE 1
Model Dirichlet problem: number of PCG *iterations.*

	Grid size $n_d \times n_d$			
α	15×15	31×31	63×63	127×127
0.0	10	19	37	74
0.3	9	17	34	67
0.6	8	15	29	58
0.8	8	13	25	50
0.9	8	12	21	42
0.95	8	12	19	36
0.98	8	12	17	30
0.99	8	12	17	26
1.0	9	13	20	30

TABLE 2
Number of PCG *iterations with* α_{opt}.

Grid	α_{opt}	PCG iters
15×15	0.923	8
31×31	0.981	12
63×63	0.995	17
127×127	0.9988	25

suggested that $\alpha_{\text{opt}} = 1 - O(h)$ on the basis of their experimental results. However, on careful examination, the α_{opt} obtained from our Fourier analysis is also consistent with the numerical results in [2], to the resolution allowed.

2. The number of PCG iterations using α_{opt} grows like $O(h_d^{-1/2})$, which is consistent with the well-known asymptotic behavior of the number of PCG iterations growing like $(K(\alpha_{\text{opt}}))^{1/2}$ with $K(\alpha_{\text{opt}}) = O(h_d^{-1})$.

The results above show that the Fourier analysis does successfully predict α_{opt} and $K(\alpha_{\text{opt}})$ for the Dirichlet problem.

5. Relation between RILU (α) and MILU (c). In this section, we show that the RILU (α) method can really be viewed as an MILU (c) method in which the parameter c, which in the original method is constrained to be independent of h, is now allowed to vary with h.

The MILU (c) method has the same stencils as in Figs. 3 and 4. The only differences are that (1) the dropped terms (rather than a fraction of them) are added to the diagonal, and (2) in addition a term ch^2 is added to the diagonal, where c is a constant independent of h.

Thus the diagonal d for MILU (c) satisfy the relationship (see [4])

$$d + \frac{2}{d} + \frac{2}{d} = 4 + ch^2, \tag{24}$$

which gives

$$d = 2 + \frac{ch^2}{2} + \frac{1}{2}\sqrt{8ch^2 + (ch^2)^2}. \tag{25}$$

The RILU (α) method is *identical* (for the periodic problem) to the MILU (c) method if their diagonals d are the same, i.e., if

$$2 + \sqrt{2(1-\alpha)} = 2 + \frac{ch^2}{2} + \frac{1}{2}\sqrt{8ch^2 + (ch^2)^2} \tag{26}$$

or

$$\varepsilon = \frac{ch^2}{2} + \frac{1}{2}\sqrt{8ch^2 + (ch^2)^2}, \tag{27}$$

or equivalently,

$$c = \left(\frac{\varepsilon^2}{2+\varepsilon}\right) h^{-2}. \tag{28}$$

Therefore, there is a 1-1 correspondence between the two methods. (This is strictly true only for the periodic case, but the asymptotic values of the coefficients of the corresponding Dirichlet preconditioners in the interior of the computational domain should also be equal.) Of course, if $\varepsilon \neq O(h)$, then (28) implies that $c \neq O(1)$, which violates the constraint on c in the original MILU method. However, as we can see from (18), $\varepsilon_{\text{opt}} = O(h)$ and therefore the corresponding $c_{\text{opt}} = O(1)$. In fact, from (18) and (28), we have

$$c_{\text{opt}} = \left(\frac{8 \sin^2 \pi h}{1 + 2 \sin \pi h}\right) h^{-2}, \tag{29}$$

giving

$$\lim_{h\to 0} c_{\text{opt}} = 8\pi^2, \tag{30}$$

which agrees with the result derived in [4] for MILU (c).

In a sense, the RILU (α) method is slightly more general than the MILU (c) method in that the corresponding value of c is not constrained to be $O(1)$. In fact, the results in this paper suggest that the addition of the term ch^2 with $c = 0(1)$ is indeed optimal and we cannot improve on the $O(h^{-1})$ bound for K within the class of RILU (α) or MILU (c) methods.

Finally, concerning the use of the two methods in practice, it seems better to use MILU (c) because the behavior is much less sensitive to c than that of RILU (α) to α, as long as c is bounded away from zero. We get the RILU (α) methods by allowing c to grow with h as in (28). In some sense, c is a local scaled parameter around the region $\alpha = \alpha_{\text{opt}}$ in Fig. 5.

Acknowledgments. The author acknowledges helpful discussions with Prof. van der Vorst on the subject and constructive comments from three referees. The computations for § 4 were performed by Mr. Charles Tong.

REFERENCES

[1] C. ASCHCRAFT AND R. GRIMES, *On vectorizing incomplete factorizations and* SSOR *preconditioners*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 122–151.

[2] O. AXELSSON AND G. LINDSKOG, *On the eigenvalue distribution of a class of preconditioning methods*, Numer. Math., 48 (1986), pp. 479–498.

[3] ———, *On the rate of convergence of the preconditioned conjugate gradient method*, Numer. Math., 48 (1986), pp. 499–523.

[4] T. F. CHAN AND H. ELMAN, *Fourier analysis of iterative methods for elliptic problems*, SIAM Rev., 31 (1989), pp. 20–49.

[5] T. DUPONT, R. P. KENDALL, AND H. H. RACHFORD, JR., *An approximate factorization procedure for solving self-adjoint elliptic difference equations*, SIAM J. Numer. Anal., 5 (1968), pp. 559–573.

[6] I. GUSTAFSSON, *A class of 1st order factorization methods*, BIT 18 (1978), pp. 142–156.

[7] H. A. VAN DER VORST, *High performance preconditioning*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1174–1185.

[8] J. A. MEYERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[9] O. AXELSSON AND V. A. BARKER, *Finite Element Solution of Boundary Value Problems*, Academic Press, New York, 1984.

[10] O. AXELSSON, *A generalized* SSOR *method*, BIT, 13 (1972), pp. 443–467.

[11] R. BEAUWENS, *On Axelsson's perturbations*, Linear Algebraic Appl., 68 (1985), pp. 221–242.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, pp. 681-694, May 1991
© 1991 Society for Industrial and Applied Mathematics
011

MULTIGRID METHODS FOR THE SOLUTION OF POISSON'S EQUATION IN A THICK SPHERICAL SHELL*

S. R. KARPIK† AND W. R. PELTIER†

Abstract. Two multigrid methods designed for the solution of a finite-element discretization of Poisson's equation in spherical geometry are presented and compared. One of these, based upon an approximate local least-squares inverse (LS_1), has been previously reported by J. R. Baumgardner and P. O. Frederickson [*SIAM J. Numer. Anal.*, 22 (1985), pp. 1107-1115]; the other, developed by the authors, employs a "mass lumped" line Jacobi smoothing procedure that is entirely new. The new method is shown to be both more economical and considerably more robust than that based upon the LS_1 smoothing iteration.

Key words. multigrid methods, elliptic partial differential equations, finite-element methods

AMS(MOS) subject classifications. 65F10, 65N20, 65N30

1. Introduction. In numerical studies of physical phenomena that are important in fields as diverse as astrophysics, geophysics, and the atmospheric sciences, we frequently face the task of solving a scalar Poisson equation of the form

$$-\nabla^2 u = f, \tag{1.1}$$

in circumstances in which the domain is either a thick spherical shell or can be mapped onto this shape by a suitable coordinate transformation. In the majority of cases of practical interest, (1.1) will be supplemented by boundary conditions of the form:

$$\alpha_1 \frac{\partial u(r_1)}{\partial r} + \beta_1 u(r_1) = \gamma_1, \qquad \alpha_2 \frac{\partial u(r_2)}{\partial r} + \beta_2 u(r_2) = \gamma_2, \tag{1.2}$$

in which r is the radial coordinate; the inner and outer radii of the shell are r_1 and r_2, respectively; and α_1, β_1, γ_1, α_2, β_2, and γ_2 are usually constants although, more generally, they might be functions of the angular coordinates ϑ and ϕ.

In principle, there are numerous approaches that we could employ to solve (1.1); however, only a small subset of these are worthy of implementation. Because of difficulties associated with treatment of the polar regions, especially in the context of hydrodynamics models (see, for example, [1]), we are well advised to avoid the use of the finite differences expressed in terms of spherical coordinates. To circumvent these difficulties and to exploit the spherical character of the underlying problem, many investigators in the area of computational fluid dynamics have demonstrated preference for the use of pseudospectral techniques which represent the ϑ-ϕ dependence of the solution as the sum of a truncated series of spherical harmonics but which evolve the nonlinear advection terms in gridpoint space (for further details on the pseudospectral method, see, for example, [12], [11], and references therein). An alternative approach, which is the subject of the present paper, appeals to the flexibilty and power of the finite-element method. In the context of analyses similar to those of interest to us here, Baumgardner and Frederickson [5] devised an almost uniform triangularization of the two-sphere based upon a generalization to spherical triangles of the well-known piecewise linear triangular element for planar regions (see [7]). In

* Received by the editors August 9, 1989; accepted for publication (in revised form) February 16, 1990. This work was supported by an award from Cray Research, Inc., and by Natural Sciences and Engineering Research Council of Canada grant A9627.

† Department of Physics, University of Toronto, Toronto, Ontario, Canada M5S 1A7.

subsequent research (see [4]), Baumgardner employed a three-dimensional variant of this discretization to construct a model of creeping flow in the earth's mantle.

In the present paper we shall examine and compare two multigrid algorithms FAPIN3S and MG3DS for solving Poisson's equation in a thick spherical shell for which the discretized equations follow from Baumgardner and Frederickson's spherical finite-element formulation. Employed as a central component of Baumgardner's FEM-based mantle convection model (see [3], [4]), FAPIN3S uses a least-squares approximate local inverse called LS_1 (see [6]), to annihilate high-frequency components of the defect on a nested sequence of finite-element approximation spaces. MG3DS, developed by the authors, with its smoother that, as we shall see, is similar to a semi-iterative line Jacobi procedure with Chebyshev acceleration, will be shown to be both more robust and economical than LS_1. Unlike the LS_1 approximate inverse, the new smoother does not require the estimation of parameters to optimize its performance.

2. Computational grid. Perhaps the first to recognize the value of a nearly uniform triangularization of the sphere for finite-element calculations, Cullen [8] employed such a discretization to integrate the shallow water equations on a two-sphere. To construct his grid, Cullen made regular subdivisions in a latitude–longitude plane of the 20 triangles whose vertices are coincident with those of an imbedded regular icosahedron. Later, Cullen and Hall [9] employed the same discretization to construct a five-level model of the atmospheric general circulation. This methodology has not, however, been generally accepted as competitive with the conventional pseudospectral algorithm mentioned previously. Consequently, most atmospheric general circulation models (GCMs) are constructed around the use of the pseudospectral method.

In [5], Baumgardner and Frederickson improve on the mesh generation procedure of Cullen by constructing their grid on the actual surface of the two-sphere rather than by working with the projection of the two-sphere onto a latitude-longitude plane. As a consequence of this innovation, they were able to define, in a manner completely analogous to area coordinates for planar triangular elements (see [15]), a local system of barycentric coordinates. These local barycentric coordinates afford a very convenient basis for evaluating element level contributions to the global stiffness and mass matrices.

The actual procedure for constructing Baumgardner and Frederickson's spherical triangular covering of the two-sphere is relatively simple and straightforward. The first step involves projecting the 12 vertices of a regular icosahedron onto the sphere; these 12 points define a mesh of 20 spherical triangles whose edges are segments of great circles of arc. Each of these triangles can be subsequently divided into four subtriangles by connecting the midpoints of the three sides by geodesic arcs. This procedure can be repeated ad infinitum, thus generating successively finer meshes—the first four of which are depicted in Figs. 1(a)–1(d). For convenience, let us call the coarsest triangularization $\mathbf{T}_0$ and let $\mathbf{T}_1$, $\mathbf{T}_2$, etc., denote successively finer coverings. At this point, we should emphasize an important, if perhaps obvious, feature of this discretization; namely, that all vertices associated with a particular grid, say $\mathbf{T}_m$, are also vertices of $\mathbf{T}_{m+1}$.

Construction of a fully three-dimensional grid is realized, first, by stacking layers of the two-sphere triangularization, say $\mathbf{T}_k$, at increasing radii, and then by joining nodes which lie along common radial vectors. The resulting grid is very nearly uniform over the surface of the sphere but can have any arbitrary distribution of nodes in the radial direction. A radial section of the three-dimensional grid is depicted in Fig. 2.

Let us denote by Ω_k the three-dimensional grid to be constructed from the two-sphere mesh $\mathbf{T}_k$. For further convenience, we require that this grid have 2^k+1

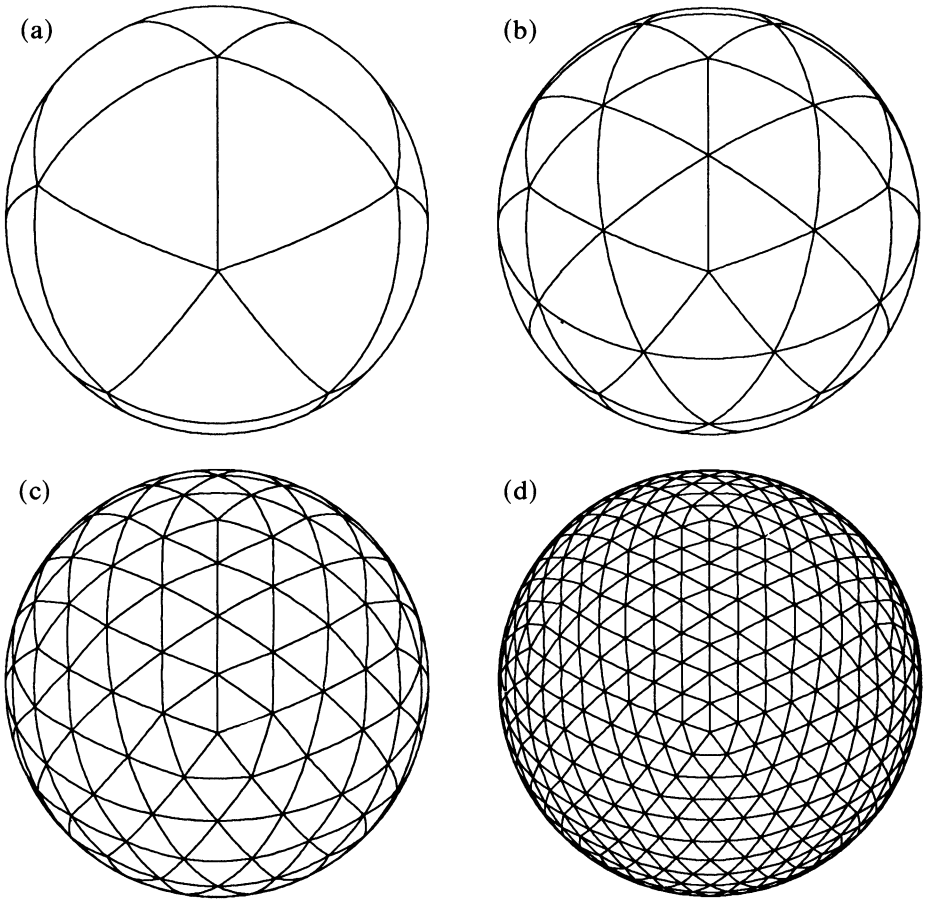


FIG. 1. *First four dyadic refinements of the icosahedral grid for the two-sphere.*

radial levels. A series of coarser grids, all of which are proper subsets of Ω_k, can easily be constructed by successive standard (or uniform) coarsenings (see [14, pp. 58–59]). Thus, for example, the grid Ω_{k-1} has $2^{k-1}+1$ radial levels which correspond to every second interior layer of Ω_k, and the surface grid for Ω_{k-1} is the two-sphere covering $\mathbf{T}_{k-1}$. This procedure can be repeated in sequence thus creating all grids Ω_m, $m = k-1, k-2, \cdots, 1, 0$. The final grid in this sequence, Ω_0, has just two radial levels, with each level covered by 20 identical spherical triangles.

3. Finite-element discretization. Rather than derive a finite-element discretization of Poisson's equation for the grid described in § 2 under the assumption of perfectly general boundary conditions such as those expressed by (1.2), let us instead consider (1.1) subject to the homogenous boundary conditions

$$u(r_1) = u(r_2) = 0. \tag{3.1}$$

By substituting (3.1) for (1.2), we avoid having to discuss complicating details associated with the more general case that, although they may be of some special interest, would only serve to detract from the primary focus of this paper. As a preliminary step towards generating the discrete equations, we transform (1.1) along with (3.1) to an

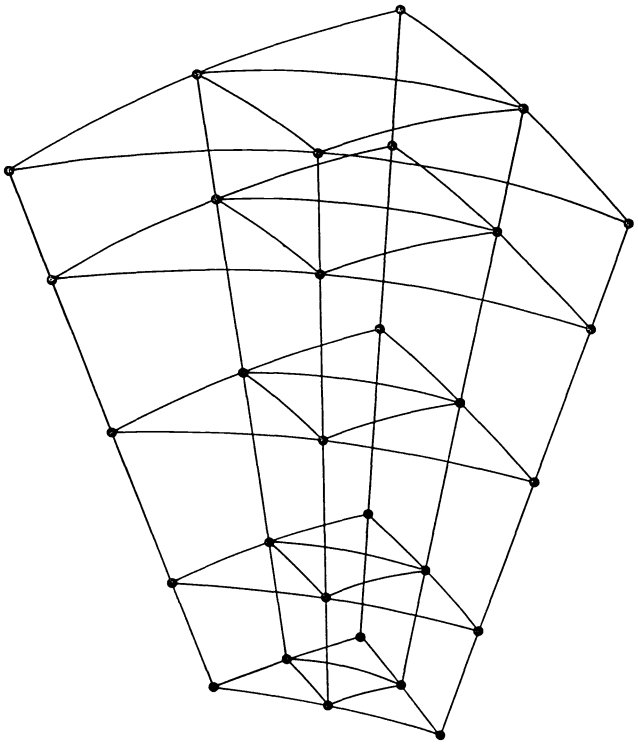


FIG. 2. *Grid cross section showing five radial levels.*

equivalent weak form (see [2, Chap. 3])

$$(\nabla u, \nabla v) = (f, v), \qquad u \in \mathcal{H} = H_0^1 \quad \text{for all } v \in \mathcal{H} \text{ and } f \in \mathcal{H}', \tag{3.2}$$

in which by $(\cdot, \cdot)$ we mean the integral of the arguments over the volume of the spherical shell. The primary advantage to be gained by working with (3.2) rather than with (1.1) in combination with (3.1), relates to the fact that (3.2) can be approximated by finite elements that are C^0 continuous whereas, because of second derivative terms present in (1.1), computationally expensive C^1 basis functions must be employed.

Given the nested character of the icosahedral mesh described in § 2, it is quite natural to consider approximations to the solution of (3.2) formulated in terms of an hierarchy of finite-element spaces $\{\mathcal{H}_l: l = 0, 1, \cdots, k\}$ that conform to the following inclusion relations:

$$\mathcal{H}_0 \subset \mathcal{H}_1 \subset \mathcal{H}_2 \subset \cdots \subset \mathcal{H}_{k-1} \subset \mathcal{H}_k \subset \mathcal{H}. \tag{3.3}$$

Although, in principle, we could employ basis functions of any arbitrary order that satisfy (3.3), we shall, for reasons of economy, follow Baumgardner [3] and restrict ourselves to the following choice for $\mathcal{H}_l$:

$$\begin{aligned} \mathcal{H}_l = \text{SPAN}\{N_i^l(\eta, r)&: N_i^l(\eta, r) = L_i^l(\eta) M_i^l(r),\ L_i^l(\eta) \in \mathcal{L}^l, \\ &M_i^l(r) \in \mathcal{M}^l\ \forall i = (2 + 10j^2)(j+1), j = 2^l\} \end{aligned} \tag{3.4}$$

where in (3.4), η is an element-level spherical barycentric coordinate (see [5]); $\mathcal{M}^l$ is the space of one-dimensional, piecewise continuous linear finite elements defined for the lth refinement of the radial grid; and $\mathcal{L}^l$ is a space of linear, piecewise continuous

functions with compact support defined with respect to the lth triangularization of the two-sphere. As an aid to understanding, it is fruitful to regard $\mathscr{H}_l$ as a straightforward generalization to the sphere of Zienkiewicz's linear serendipity element for triangular prisms (see [15, pp. 174–176]).

Using the function spaces defined above, we can approximate the solution to (3.2) on Ω_k by the following series expansion:

$$\hat{u}(x) = \sum_{i=1}^{m} u_i N_i^k(\eta, r) \tag{3.5}$$

where $m = (2+10n^2)(n+1)$ and $n = 2^k$. Similarly, f can be expanded in terms of the basis functions $N_i^k(\eta, r) \in \mathscr{H}_k$:

$$\hat{f}(x) = \sum_{i=1}^{m} f_i N_i^k(\eta, r). \tag{3.6}$$

After substituting (3.5) and (3.6) into (3.2) and transforming to spherical coordinates, we get the following set of simultaneous linear equations for the coefficients u_i:

$$\sum_{j=1}^{m} A_{ij}^k u_j = b_i \quad \forall i = 1, 2, \cdots, m \tag{3.7}$$

where

$$A_{ij}^k = \left(\frac{1}{r} M_i^k, \frac{1}{r} M_j^k\right)(\nabla L_i^k, \nabla L_j^k) + \left(\frac{\partial}{\partial r} M_i^k, \frac{\partial}{\partial r} M_j^k\right)(L_i^k, L_j^k) \tag{3.8}$$

and

$$b_i = (N_i^k, N_j^k) f_j. \tag{3.9}$$

The various terms belonging to (3.8) and (3.9) are evaluated by first calculating, to second or higher order, like expressions on a much finer grid, say on Ω_{k+4}, and then by summing these contributions to get the needed values. This approach, which lends itself to an easy and efficient computer implementation, provides for the evaluation of the required terms that, for our purposes, is essentially exact. For further details and useful recursion formulae, the interesested reader is directed to [3] or [5].

Since we are ultimately concerned with multigrid algorithms that can be used to solve (3.2), we must, in conjunction with (3.5) to (3.9), speak of discretizations of the operator $(\nabla u, \nabla v)$ on grids that are coarser than the finest grid Ω_k. Let the prolongation operator,

$$I_{j-1}^j \in [\mathscr{H}_{j-1}, \mathscr{H}_j], \qquad \ker (I_{j-1}^j) = \{0\}, \tag{3.10}$$

and its close relative, the restriction,

$$I_j^{j-1} \in [\mathscr{H}_j, \mathscr{H}_{j-1}], \qquad I_j^{j-1} = (I_{j-1}^j)^T \tag{3.11}$$

be optimally selected with respect to the order of approximation. For the hierarchy of spaces discussed above, this means that we choose the prolongation mapping to be a linear interpolation of grid function values defined on a coarse grid, say, Ω_j, onto the next finer grid Ω_{j+1}. When chosen according to (3.10) and (3.11), the following relations hold:

$$A^{j-1} = I_j^{j-1} A^j I_{j-1}^j \quad \forall j = k, k-1, \cdots, 2, 1 \tag{3.12}$$

where the A^l's are matrices of the coefficients representing the projection of the Laplacian operator onto the hierarchy of finite-dimensional spaces $\{\mathscr{H}_l : l = 0, 1, \cdots, k\}$.

4. Multigrid algorithms.

4.1. FAPIN3S. The algorithm called FAPIN3S represents an extension to three-dimensional, spherical geometries of the original FAPIN (**F**ast **AP**proximate **IN**version of large sparse linear systems) procedure described in [10]. Like its predecessor, FAPIN3S is constructed along the lines of the simplest possible multigrid solution strategy: namely, the sawtooth cycle wherein the residual is first projected recursively from the finest grid to the coarsest one without any smoothing, after which there follows a nested series of coarse grid corrections with smoothing that ultimately produce an improved iterate on the finest grid. What particularly distinguishes FAPIN3S from other multigrid algorithms is its choice of smoothing iteration. FAPIN3S employs a modified least-squares approximate one-local inverse called LS_1 (see [6] and [3]) to smooth high-frequency components of the residual on each level of the discretization.

Originally conceived of as a stand-alone iterative procedure for solving large, sparse systems of linear equations like

$$\mathbb{A}\phi = b$$

where $\mathbb{A}$ is a q-local operator of rank N (say) defined with respect to some discretization Ω_d of a domain $\Omega \subseteq R^n$, the LS_q approximate inverse of $\mathbb{A}$ is that q-local operator B that minimzes the l_2 norm of the diagonal of the matrix $B\mathbb{A} - E$, where E is the identity matrix. Following the nomenclature employed in [6], an operator, say $\mathbb{A}$, is said to be q-local if the value of $\mathbb{A}\phi$ at some node i belonging to Ω_d depends only on the values of ϕ in a q-neighborhood of i. Thus, for example, a nine-point Laplacian operator in two dimensions is said to be a one-local operator. After some straightforward but tedious algebra (see Appendix I) we can show that the coefficients of the matrix B must satisfy the following set of relations:

$$2\mathbb{A}_{pq} = \sum_{i,j=1}^{N} \mathbb{A}_{qi}\mathbb{A}_{ji}B_{pj} + B_{pi}\mathbb{A}_{ij}\mathbb{A}_{qj} \quad \forall p, q = 1, 2, \cdots, N. \tag{4.1}$$

For the FEM discretization under discussion in this paper, (4.1) requires solving a rank 21 linear system at each node belonging to the computational mesh in order to calculate the LS_1 approximate inverse operator. In actual practice, the approximate inverse employed by FAPIN3S is not exactly that which follows from the solution of an equation like (4.1); rather B is determined by solving the closely related set of equations

$$2\mathbb{A}_{pq} + \varepsilon\delta_{pq} = \sum_{i,j=1}^{N} \mathbb{A}_{qi}\mathbb{A}_{ji}B_{pj} + B_{pi}\mathbb{A}_{ij}\mathbb{A}_{qj} \tag{4.2}$$

where δ_{pq} is the Kronecker delta function and ε is a small positive constant (or, for better performance but at greater cost, ε can be a function of radius) chosen to optimize the smoothing effect of B. Although ε is usually small, that is, $\varepsilon \ll \|A\|$, the improvement gained by using (4.2) rather than (4.1) to calculate B can be large. However, the practical advantage of (4.2) over (4.1) is diminished by the fact that there exists no theoretical justification for the term $\varepsilon\delta_{pq}$ present in (4.2) that differentiates it from (4.1). Consequently, for all but the simplest cases that yield constant coefficient discretizations (like Poisson's equation on a regular mesh in a boxlike domain with periodic boundary conditions), determination of the constant ε in (4.2) by strictly analytical means is essentially impossible. Thus, ε must be estimated by some numerical optimization procedure. A heuristically reasoned method for determining good values for ε follows immediately. It turns out that the values of ε determined through

application of such a procedure depend both on the grid discretization level l and on the radial distribution of gridpoints.

Hackbusch has shown that the quantity

$$\rho_L(\nu) = \sup_{l \geqq 1} \|A^l \mathcal{S}_l^\nu\| / \|A^l\| \tag{4.3}$$

which he calls the *smoothing number* (see [14, pp. 36-37]) is a good, although not rigorously correct, estimate of the smoothing efficiency ρ_ν of an iteration $\mathcal{S}_l$. Ideally, we would like to optimize our choice of ε_l in (4.2) with respect to minimizing ρ_ν; however, the quantity ρ_ν is not easily calculated. Instead, we compute what turn out to be near optimum values for the set of constants $\{\varepsilon_l : l = 1, 2, \cdots, k\}$ by requiring that $\|A^l \mathcal{S}_l^\nu\| / \|A^l\|$ be minimized. By focusing on the problem of minimizing ρ_L rather than ρ_ν, the computational burden is greatly reduced but, unfortunately, remains substantial since the optimization process must be performed for each different radial grid distribution.

To demonstrate this procedure, we shall consider three distributions of radial gridpoints: one uniform in radius, another that squeezes the grid spacing near both bounding surfaces according to a cosine distribution, and, finally, a grid spacing that is exponentially expanding with radius. Expressed more concisely, these distributions are

$$r^j = r_1 + (j-1)(r_2 - r_1)/2^k \qquad \text{(uniform)},$$

$$r^j = \left[(r_2 + r_1) - (r_2 - r_1) \cos\left(\frac{(j-1)\pi}{2^k} \right) \right] \Big/ 2 \qquad \text{(cosine)},$$

$$r^j = r_1 (r_2 / r_1)^{(j-1)/2^k}, \qquad \text{(exponential)}$$

where $j = 1, 2, \cdots, 2^k, 2^{k+1}$. Optimal values for ε_l were found to be largely independent of the value for ν when ν ranged from one to three; however, these optimal values were seen to depend strongly on the radial gridpoint distribution. Estimated optimal values of ε_l for the uniform, cosine, and exponential gridpoint distribution are presented in Table 1 for the case $\nu = 1$. A graph illustrating the sensitivity of $\|A^l \mathcal{S}_l^\nu\| / \|A^l\|$ to variations in ε_l, again for $\nu = 1$ but only for the cosine grid, is presented in Fig. 3.

TABLE 1
Optimal ε values for $\nu = 1$.

Radial grid description	ε			
	$l=1$	$l=2$	$l=3$	$l=4$
Uniform	1.1×10^{-1}	3.5×10^{-2}	1.1×10^{-2}	2.5×10^{-3}
Exponential	8.8×10^{-2}	2.3×10^{-2}	6.1×10^{-3}	1.9×10^{-3}
cosine	1.1×10^{-1}	6.5×10^{-2}	3.6×10^{-2}	2.4×10^{-2}

In light of the results presented in Table 1 and the degree of sensitivity that the smoothing number shows to variations in ε_l (see Fig. 3), we cannot hope to estimate a single set of values $\{\varepsilon_l : l = 1, 2, \cdots, k\}$ for a given radial gridpoint distribution and then expect those same values to suffice for a broad range of other possible grid distributions. In fact, values of ε_l that are nearly optimal for one distribution of gridpoints may cause the multigrid algorithm to converge very slowly (or not at all) when used for another distribution.

4.2. MG3DS. A primary objective in creating our new MG3DS algorithm (**Multi-Grid Method for 3-D**imensional **S**pherical Problems) was to produce a new method

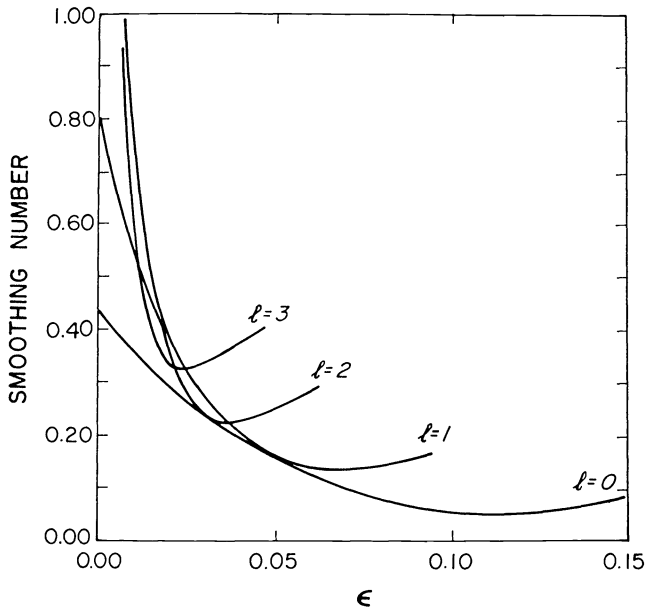


FIG. 3. *Senstivity of the smoothing number ρ_ν to variations in ε for the cosine grid.*

for solving (3.7) that, like FAPIN3S, was quickly converging but, unlike FAPIN3S, did not require the potentially costly step of estimating parameters to optimize its performance. We were especially interested in having an algorithm that was very efficient for cases in which the radial grid spacing was small compared to the edge-length of corresponding triangles on the two-sphere. Rather than complicate the job of making meaningful comparisons by introducing too many differences between the algorithms (for example, a *V*- or *W*-cycle could be substituted for the sawtooth cycle), it was decided to focus initially on the choice of smoothing algorithm. Thus, MG3DS differs significantly from FAPIN3S only in that the two algorithms employ different smoothers.

The following pseudocode procedure provides a concise description of the MG3DS algorithm:

procedure MG3DS (k, ν)
if $k = 0$ **then** $u^0 := (A^0)^{-1} b$ **else begin**
  $r^k := A^k u^k - b$;
  while $\|r^k\| > \varepsilon_{\text{tol}}$ **do begin**
    for $j := k-1$ **step** -1 **until** 0 **do** $r^j := I^j_{j+1} r^{j+1}$;
    $\delta u^0 := -(A^0)^{-1} r^0$;
    for $j := 1$ **step** 1 **until** k **do begin**
      $r^j := r^j + I^j_{j-1} A^j \delta u^{j-1}$;
      $\delta u^j := \mathcal{S}^\nu_j(r^j)$;
    end;
    $u^k := u^k + \delta u^k$;
    $r^k := A^k u^k - b$;
  end;
end;

In the above, r^j is the residual (or its projection) at the jth grid level and the symbol $\mathcal{S}^\nu_j(\cdot)$ represents ν applications of a smoothing step $\mathcal{S}_j$ to some grid function on the jth grid.

As mentioned in the Introduction, MG3DS employs a "mass lumped" (see [13] for a discussion of "mass lumping") semi-iterative, line Jacobi procedure to smooth the residual. Derivation of the relevant equations is not difficult; however, it does require that we introduce some additional notation. Here we will concentrate on deriving the functional form of the smoothing iteration on the finest grid Ω_k; the extension of these ideas to allow construction of the smoother on coarser grids should be obvious. Rather than regard A^k as a sparse matrix as we did in § 3, it is preferable instead to visualize the geometric structure or graph of the computational molecule. Centred at each point i in the computational lattice is a 21-node operator constructed in three radial layers; each layer is comprised of seven nodes arranged in the shape of hexagon around a central node. Let A^k_{ipq} be the value of the coefficient corresponding to the node (p, q) for the gridpoint i. Here the ordered pair (p, q) refers to an offset from the central point where $p = -1, 0$ and $+1$ refers, respectively, to the lower, middle, and upper radial layer; q ranges from zero to six with $q = 0$ corresponding to the point at the center of the hexagon and $q = 1, 2, \cdots, 6$ denoting points on the perimeter (see Fig. 4). It is fruitful now to decompose A^k_{ipq} into a pair of terms A^{kt}_{ipq} and A^{kr}_{ipq} that represent, respectively, the tangential and radial derivative contributions to A^k_{ipq}. Clearly, A^{kt}_{ipq} and A^{kr}_{ipq} can be identified with the terms $(M^k_i/r, M^k_j/r)(\nabla L^k_i, \nabla L^k_j)$ and $(\partial M^k_i/\partial r, \partial M^k_j/\partial r)(L^k_i, L^k_j)$ of (3.8). Just as we can construct by mass lumping a matrix that is in some sense equivalent to the mass matrix (see [13]), we can form the operator

$$\tilde{A}^k_{ipq} = 2\delta_{q0} \sum_{j=-1}^{1} A^{kt}_{ijq} + \sum_{j=0}^{6} A^{kr}_{ipj}, \tag{4.4}$$

which is similarly equivalent to A^k. Since $\tilde{A}^k$ is tridiagonal, it can be easily inverted; thus, (4.4) is suggestive of the following smoothing iteration:

$$r^k_{j+1} = \mathcal{S}^\nu_k(r^k_j) = \left(\prod_{\mu=1}^{\nu} (E - \omega_\mu A^k (\tilde{A}^k)^{-1}) \right) r^k_j \tag{4.5}$$

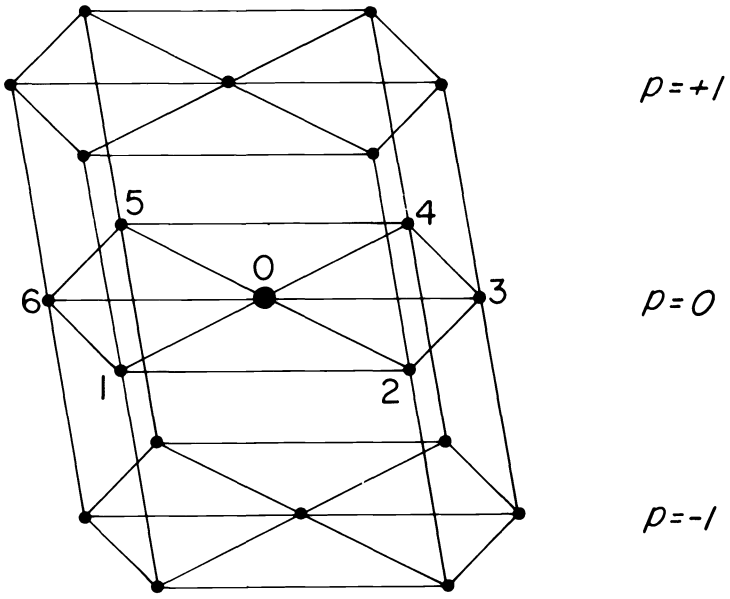


FIG. 4. *Labelling convention for nodes belonging to the computational molecule centered on the point indicated by the large dot* (•).

where the parameters ω_μ are chosen to optimize the smoothing effect of (4.5). Effective smoothing rates are realized by choosing

$$\omega_\mu = \left(1+\cos\frac{\pi}{2\nu+1}\right)\Big/\left(\cos\frac{\pi}{2\nu+2}-\cos\frac{(2\mu+1)\pi}{2\nu+1}\right). \tag{4.6}$$

This particular choice for ω_μ, which follows from minimizing a Chebyshev polynomial over the range of high-frequency eigenmodes (see [14, pp. 131-132]), yields asymptotic smoothing rates that are proportional to ν^2 rather than being linear in ν as is the case if we, for instance, chose $\omega_\mu = \text{constant}$.

5. Numerical results. In this section, we present, in summary form, results from a series of numerical experiments designed to evaluate the relative performance capabilities of the solution methods outlined in § 4. Single precision FORTRAN versions of both algorithms were compiled in CFT77 for execution on a single head of the Ontario Centre for Large Scale Computation's Cray X-MP/24 at the University of Toronto. Care was taken to ensure that the FORTRAN code for both methods was equally well optimized for vector performance so as to facilitate comparisons of computational effort.

As a model problem, we considered the solution of (3.2) on the grid Ω_4 (43,554 d.o.f.) for the uniform, exponential and cosine radial gridpoint distributions. In all cases, the ratio r_2/r_1 was fixed at two (near that appropriate to the mantle convection problem discussed by Baumgardner [4]). The value of the initial iterate u^0 was chosen to vary randomly across the grid within the range ± 1 and the function f in (3.2) was taken to be zero everywhere. Although it may seem that this choice, $f(x)=0$, is in some sense special, and thus not truly representative of realistic problems, the authors have found that asymptotic convergence rates for both algorithms are largely unaffected by the choice for f. Effectively, in (3.7), we have a right-hand side equal to $f-Au^0$ with which to begin the iterative process, and whether f is zero or not is of no real consequence to these algorithms because they only "see" the quantity $f-Au^0$, never f in isolation. The factor most strongly affecting the overall convergence history of these algorithms is not the functional form assumed for f but rather the initial value of the defect u^0-u^*, where u^0 is the initial guess for u and u^* is the value of u corresponding to the stationary point of the iteration. However, even this effect was limited to the rate at which the residual decreased for the first three or four iterations; the asymptotic convergence rate was seen to be largely independent of u^0-u^*.

In Figs. 5-7, the l_2 norm of the residual for both of the FAPIN3S (solid lines) and MG3DS (dashed lines) iterations applied to the solution of the model problems described above, is plotted as a function of iteration number. Asymptotic convergence rates for both algorithms as measured by the ratio $\|r_{j+1}^k\|/\|r_j^k\|$ in the limit $j \gg 1$, are also summarized in Table 2. The outstanding feature of the MG3DS algorithm is the degree to which its performance is independent of the particular radial gridpoint distribution. FAPIN3S, on the other hand, performs well for the uniform and exponential distributions but for the cosine case, the convergence rate is relatively poor for all values of ν.

To this point, no mention has been made of the computational cost associated with either algorithm. In Table 3, startup times and the cpu time/iteration are presented for both algorithms. Startup costs associated with FAPIN3S which include, for example, the time required to calculate the LS_1 approximate inverses on each of the grid levels, are over three times more expensive than the equivalent operations for MG3DS. This differential is due to two factors: first, compared to MG3DS, a greater number of

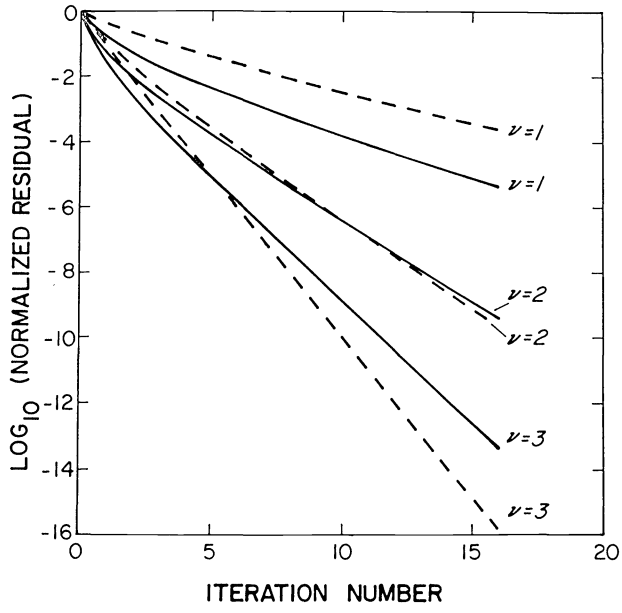


FIG. 5. *Convergence history for* FAPIN3S (*solid lines*) *and* MG3DS (*dashed lines*) *for the uniform grid.*

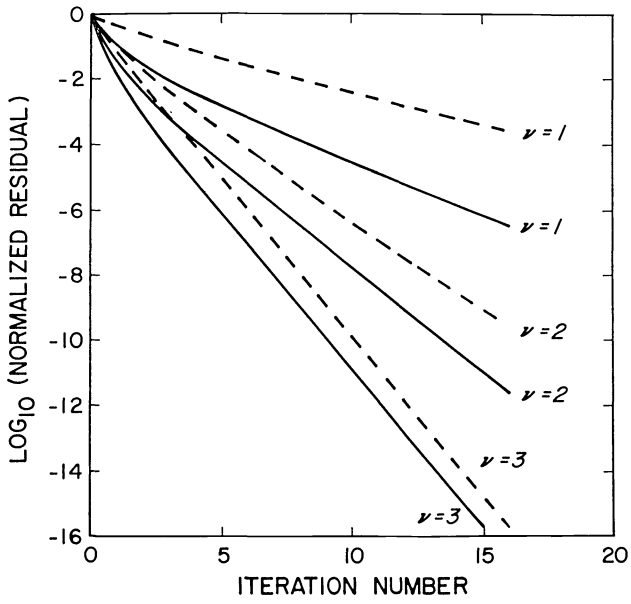


FIG. 6. *Convergence history for* FAPIN3S (*solid lines*) *and* MG3DS (*dashed lines*) *for the exponential grid.*

arithmetic operations are required to initialize coefficients and matrices used by FAPIN3S; and second, for FAPIN3S, a sizable fraction of the required operations, such as the solution of (4.2), either do not vectorize or vectorize with very short loop lengths. Returning again to Table 3, we see as well that per iteration costs favour MG3DS over FAPIN3S by about a factor of 1.6 for all values of ν. Thus, MG3DS can be used with three smoothing steps at about the same cost per iteration as FAPIN3S

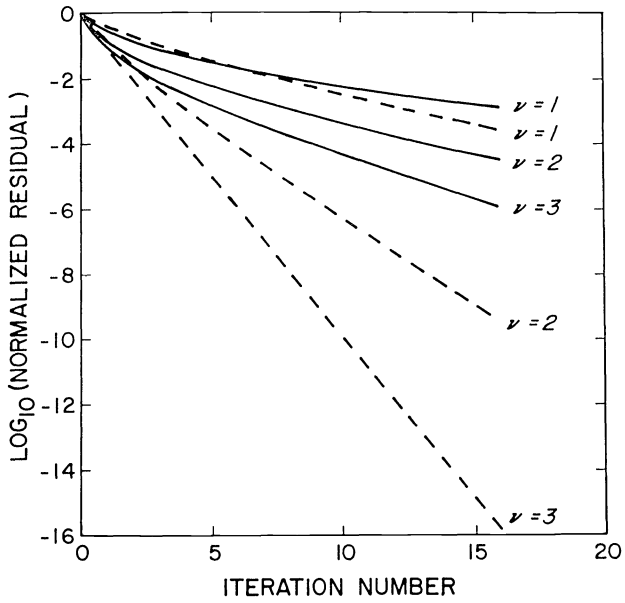


FIG. 7. *Convergence history for* FAPIN3S (*solid lines*) *and* MG3DS (*dashed lines*) *for the cosine grid.*

TABLE 2
Residual reduction factors.

Radial grid description	FAPIN3S $\nu=1$	FAPIN3S $\nu=2$	FAPIN3S $\nu=3$	MG3DS $\nu=1$	MG3DS $\nu=2$	MG3DS $\nu=3$
Uniform	0.493	0.284	0.163	0.603	0.261	0.103
Exponential	0.421	0.206	0.099	0.603	0.262	0.103
Cosine	0.682	0.559	0.463	0.603	0.264	0.102

TABLE 3
Comparison of startup and cpu/*iteration costs.*

Solver	Startup time (sec)	cpu/iteration (sec) $\nu=1$	cpu/iteration (sec) $\nu=2$	cpu/iteration (sec) $\nu=2$
FAPIN3S	3.58	0.056	0.112	0.167
MG3DS	1.14	0.034	0.068	0.103

when just two smoothing passes are employed. These facts combined with the asymptotic convergence rates displayed by both algorithms demonstrate that MG3DS is a more economical and numerically more robust alternative to FAPIN3S for the solution of Poisson's equation on a sphere.

6. Conclusion. Based on a "mass lumped" line Jacobi smoother with Chebyshev acceleration MG3DS has, for the cases reported here along with others studied by the authors, proven itself as an efficient means for solving Poisson's equation on a FEM-based icosahedral discretization of the sphere. The other algorithm tested, FAPIN3S, performed well for the majority of cases reported; however, it lacked the robustness

of MG3DS. Also, the smoother used by FAPIN3S requires the numerical estimation of parameters which are problem dependent to optimize its performance.

One feature not discussed thus far in the present paper is the capability of FAPIN3S to solve systems of vector-coupled elliptic equations such as the Stokes equations of viscous hydrodynamics (see [3], [4]). Work is proceeding on extensions to MG3DS that will allow it to similarly accommodate such increased complexity and we shall report this further development of our multigrid solver in a future paper.

Appendix I. Derivation of the LS_q local approximate inverse. In this Appendix, an overview of the major steps required to derive the functional form of the least-squares local approximate inverse (LS_q) is provided. Further details and example calculations can be found in [6] and [3].

Let $\mathbb{A}$ be a q-local operator of rank N. The LS_q approximate inverse of $\mathbb{A}$ is that q-local operator B that minimizes the functional

$$f(B) = \sum_{i=1}^{N} \operatorname{diag}[(E - B\mathbb{A})(E - B\mathbb{A})^T] \tag{A1}$$

where E is the identity matrix. Clearly, the matrix B that minimizes (A1) also satisfies:

$$g_{mn}(B) = \lim_{\varepsilon \to 0} \frac{1}{\varepsilon}[f(B - \varepsilon E^{mn}) - f(B)] \tag{A2}$$

where E^{mn} is the matrix which has unit value at the row and column location (m, n) and zero everywhere else. After some manipulations, we can transform (A2) into the more convenient and useful form

$$g_{mn}(B) = \lim_{\varepsilon \to 0} [-2\varepsilon \mathbb{A}_{nm} + \varepsilon \mathbb{A}_{ni}\mathbb{A}_{ji}B_{mj} + \varepsilon B_{mp}\mathbb{A}_{pq}\mathbb{A}_{nq} + \mathcal{O}(\varepsilon^2)]. \tag{A3}$$

In the above, repeated subscripts are meant to imply summation over the range one to N. From (A3), it is clear that $f(B)$ is minimized when the matrix B satisfies

$$2\mathbb{A}_{pq} = \sum_{i,j=1}^{N} \mathbb{A}_{qi}\mathbb{A}_{ji}B_{pj} + B_{pi}\mathbb{A}_{ij}\mathbb{A}_{qj} \quad \forall p, q = 1, 2, \cdots, N. \tag{A4}$$

Therefore, B, calculated according to (A4), is the required least-squares local approximate inverse.

Acknowledgments. The authors are greatly indebted to Drs. John Baumgardner and Paul Frederickson for providing them with a version of their FAPIN3S code and for sharing with them their experience in its use.

REFERENCES

[1] A. AKIO AND V. R. LAMB, *Computational design of the basic dynamical processes of the* UCLA *general circulation model*, Methods Comput. Phys., 17 (1977), pp. 173–265.

[2] O. AXELSON AND V.A. BARKER, *Finite Element Solution of Boundary Value Problems*, Academic Press, New York, 1984.

[3] J. R. BAUMGARDNER, *A three-dimensional finite element model for mantle convection*, Ph.D. thesis, University of California, Los Angeles, CA, 1983.

[4] ———, *Three-dimensional treatment of convective flow in the earth's mantle*, J. Statist. Phys., 39 (1985), pp. 501–511.

[5] J. R. BAUMGARDNER AND P. O. FREDERICKSON, *Icosahedral discretization of the two-sphere*, SIAM J. Numer. Anal., 22 (1985), pp. 1107–1115.

[6] M. W. BENSON AND P. O. FREDERICKSON, *Iterative solution of large sparse linear systems arising in certain multidimensional problems*, Utilitas Math., 22 (1980), pp. 127–140.

[7] R. COURANT, *Variational methods for the solution of problems of equilibrium and vibrations*, Bull. Amer. Math. Soc., 49 (1943), pp. 1-23.

[8] M. J. P. CULLEN, *Integrations of the primitive equations on a sphere using the finite element method*, Quart. J. Roy. Meteorol. Soc., 100 (1974), pp. 555-562.

[9] M. J. P. CULLEN AND C. D. HALL, *Forecasting and general circulation results from finite element models*, Quart. J. Roy. Meteorol. Soc., 105 (1979), pp. 571-592.

[10] P. O. FREDERICKSON, *Fast approximate inversion of large sparse linear systems*, Mathematics Report #7-75, Lakehead University, Thunder Bay, Canada, 1975.

[11] G. A. GLATZMAIER, *Numerical simulations of stellar convective dynamos, Part* I, J. Comput. Phys., 55 (1983), pp. 519-544.

[12] D. GOTTLIEB AND S. A. ORSZAG, *Numerical Analysis of Spectral Methods: Theory and Applications*, CBMS-NSF Regional Conference Series in Applied Mathematics 26, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1977.

[13] P. M. GRESHO, R. L. LEE, AND R. L. SANI, *Advection dominated flows, with emphasis on the consequences of mass lumping*, Finite Elements in Fluids, Vol. 3, 1978.

[14] W. HACKBUSCH, *Multi-grid Methods and Applications*, Springer-Verlag, Berlin, 1980.

[15] O. C. ZIENKIEWICZ, *The Finite Element Method*, Third Edition, McGraw-Hill, New York, 1977.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 3, p. 695–723, May 1991
©1991 Society for Industrial and Applied Mathematics
012

VARIABLE ORDER ADAMS–BASHFORTH PREDICTORS WITH AN ERROR-STEPSIZE CONTROL FOR CONTINUATION METHODS*

BRUCE N. LUNDBERG† AND AUBREY B. POORE‡

Abstract. Variable order Adams–Bashforth predictors with an error-stepsize control are developed and used for a class of predictor-corrector continuation methods which trace solution paths of an underdetermined nonlinear system $F(w) = 0$, where $F : \mathbb{R}^{n+1} \to \mathbb{R}^n$. The predictor method developed here employs consecutive order Adams–Bashforth predictors to determine the stepsize by estimating the local truncation error in the lower order formula, a comparison of three consecutive order methods to choose the order that yields the largest stepsize, and local extrapolation to improve accuracy. Unlike many differential equation algorithms, the stepsize is changed at each step to give more uniformity in the predictor error and the global error is controlled at each step through the use of a Newton-like correction back to the path after each prediction. The philosophy of following the path closely with only one or two corrections per step is shown to yield an efficient and robust algorithm which is fairly insensitive to parameter settings. The key to the success and robustness of these multistep predictors is the approximation of arclength along the path at an accuracy commensurate with that of the tangents to the path, which is achieved by a local parameterization suggested by the corrector equations. To demonstrate the efficiency and robustness of the resulting continuation algorithm, several numerical comparisons are made with state-of-the-art software packages HOMPACK and PITCON.

Key words. predictor-corrector continuation, continuation, Adams–Bashforth predictors, error-stepsize control, Davidenko equation, arclength parameterization

AMS(MOS) subject classifications. 65H10, 34A50, 65L05

1. Introduction.

Predictor-corrector methods for tracing solution paths of an underdetermined system of nonlinear equations $F(w) = 0$, where $F : \mathbb{R}^{n+1} \to \mathbb{R}^n$, have proved to be effective numerical procedures for various problems ranging from homotopy methods for nonlinear equations and fixed point problems to parametric investigations of dynamical systems and nonlinear eigenvalue problems. These methods are now in wide use and are considered to be reasonably robust but slow and computationally intensive, primarily due to the extensive linear algebra requirements in both the predictor and corrector phases. To describe these procedures, we assume that $F(w)$ is continuously differentiable, that $F(w) = 0$ has a smooth solution path $P = \{w \in \mathbb{R}^{n+1} : w = \Psi(t), t \in I\}$, where I is an interval of real numbers, and that the path is nonsingular in that $\text{rank}[D_w F|_{w \in P}] = n$. Most path following algorithms generate a sequence of points $\{(t_k, w_k)\}_{k=0}^N$, where w_k is on or near the path and w_0 is a known solution of $F(w) = 0$. To go from a point w_k to a point w_{k+1}, we first predict a new point, say, wp_{k+1}, using past and present information, and then correct back to the path using a Newton or Newton-like method. Besides these prediction and correction phases, there is usually an error-stepsize control to improve efficiency and a method for computing special points on the path such as singularities or, in case of

*Received by the editors December 27, 1988; accepted for publication (in revised form) March 12, 1990. This work was partially supported by National Science Foundation grant DMS-87-04679 and by Air Force Office of Scientific Research grant AFOSR-88-0059.

†Department of Mathematics, Grand Canyon University, 3300 W. Camelback, Phoenix, Arizona 85017. The work of the first author was also partially supported by National Aeronautics and Space Administration grant NGT-06-002-802.

‡Department of Mathematics, Colorado State University, Fort Collins, Colorado 80523.

a homotopy problem, the solution corresponding to a specific value of the homotopy parameter. Of course, the subject can become more complex when branching switching [11], [14], [16], [19], [33] at singularities and the ability to follow singularities in parameter space are added; however, the concentration in this work is on the predictor phase and an appropriate error-stepsize control with the goal being to add efficiency and robustness to these methodologies.

Several suggestions have been made for the prediction phase. Certainly we can combine points on the curve with polynomial or rational function extrapolation to produce wp_{k+1}. The use of ordinary differential equation techniques arises from the observation that dw/dt formally satisfies $[D_w F(w)]dw/dt = 0$, which is obtained by differentiating $F(w(t)) = 0$ with respect to t. Thus the solution $w = \Psi(t)$ locally satisfies the initial value problem

$$\frac{dw}{dt} = T(w), \qquad w(t_0) = w_0, \tag{1.1}$$

where $F(w_0) = 0$ and $T(w)$ is a properly oriented nonzero null vector of $D_w F(w)$ which can be computed in various ways [1], [16], [19], [30], [42]. (A rigorous justification of a local solution of (1.1) can be based on the assumed smoothness of $F(w)$, the full rank condition above, and the implicit function theorem.) The scaling of $T(w)$ depends on the choice of the parameterization of P; for example, if $||T(w)||_2 = 1$, then t is arclength. (The celebrated Davidenko differential equation $D_u F(u,\lambda)du/d\lambda + D_\lambda F(u,\lambda) = 0$ is obtained by formally differentiating $F(u,\lambda) = 0$ with respect to λ and is a special case of (1.1) in which $w^T = (u^T, \lambda)$ and λ is the independent parameter. A tangent in this case is $T(w)^T = \pm(du/d\lambda^T, 1)$.) Another possibility is to combine solution values w and tangent values $T(w)$ in an extrapolation technique such as Hermite extrapolation. Each of these possibilities has been suggested and investigated in the literature [5], [12], [24], [32], [44], [47], and more recently, Taylor and Padé predictors based on finite-difference approximation of higher derivatives have been used effectively by Schwetlick and Cleve [36]. However, Allgower and Georg [1], [14], [15] recommend the use of Adams–Bashforth predictors since they do not demand the computation of higher derivatives and are *usually* less sensitive to variations in the accuracy of the computed solution values, w, than are methods which involve the solution values in the extrapolation process. (The example given in the next paragraph shows that this need not always be the case.) From the various techniques for initial value problems arising in ordinary differential equations, we generally recommend the use of a multistep method when function evaluations are extremely expensive. We were thus motivated to investigate these Adams–Bashforth predictors.

The design philosophies of error-stepsize control for curve following algorithms are also many and varied. At one end of the spectrum is the actual use of computer software for initial value problems [3], [14], [45] to solve the differential equation (1.1). The difficulty with this as a general continuation procedure is that the solution of the initial value problem (1.1) or the Davidenko equation may be stable or unstable and the equation may even be stiff. An artificial but simple illustration of this is given by the problem

$$F(x,y,\lambda) = \begin{pmatrix} xe^{a\lambda} \\ ye^{b\lambda} \end{pmatrix} = 0,$$

whose corresponding Davidenko equation is

$$\frac{d}{d\lambda}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -a & 0 \\ 0 & -b \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}.$$

This equation can be stiff or nonstiff and the solution $(x, y) = (0, 0)$ may be stable or unstable, depending on the relative magnitudes and signs of the parameters a and b. Watson [45, p. 305] has, however, shown that under certain hypotheses the use of initial value techniques for ordinary differential equations yields an algorithm that is "inherently stable near the fixed point" for a class of probability one homotopy mappings used for the solution of fixed point problems. For other problem classes, particularly natural parameter continuation, curve following over any distance without using extreme error tolerances can lead to highly inaccurate solutions, and thus the stepsizes are required to be relatively small to maintain a reasonable global error. The use of a Newton correction back to the path P after a prediction resolves the global error problem, but the small stepsize problem still remains. At the other end of the spectrum are those methods which employ a low-order predictor with a stepsize chosen to be as large as possible while giving a prediction within the domain of convergence of Newton's method. This is an appealing philosophy, but it is not without its deficiencies: the number of iterations in the corrector process may be large, the actual estimation of the domain of convergence of Newton's method must itself be extrapolated, and robustness can be sacrificed due to curve jumping. Indeed, we have found that we often need to make several runs at different error tolerances to achieve success in continuation codes, especially when we are trying to achieve efficiency. Thus the question is whether a predictor-corrector method can be designed wherein the predicted point remains "close" to the curve for robustness and yet the efficiency of the methodology is maintained or increased by reducing the amount of work required to solve a given problem to a specified accuracy.

In this work, the Adams–Bashforth predictors are investigated with the objective being to develop a sufficiently accurate predictor so that only one or two corrector iterations back to the curve are required at each step. The error-stepsize control developed here employs consecutive order Adams–Bashforth predictors to determine the stepsize by estimating the local truncation error in the lower order formula, a comparison of three consecutive order methods to choose the order that yields the largest stepsize, and local extrapolation to improve accuracy. This methodology differs significantly from the Adams methods used in the numerical solution of ordinary differential equations, including the Adams predictors used in HOMPACK [42], [44]. Specifically, the stepsize is changed at each step to give more uniformity in the predictor error and larger stepsizes. The error-stepsize control and order selection are based on consecutive order Adams–Bashforth (explicit) formulas rather than the Adams–Moulton corrector (implicit) formulas. (As will be discussed further in §4, this is a fundamental difference with the ordinary differential equations methodology.) Finally, the error controlled at each step is the actual (global) error rather than the local error or error per unit step. This allows the use of larger error tolerances, which also results in larger stepsizes. The key, however, to the success and robustness of these multistep predictors is the approximation of arclength along the path at an accuracy commensurate with that of the tangents to the path, and this is achieved by a local parameterization suggested by the corrector equations.

In the design of a predictor-corrector continuation procedure there is a delicate balance between robustness, which is generally achieved by following the path closely, and the goal of minimizing the amount of work required to solve a given problem to a specified accuracy. To minimize the work, one can design the procedure to minimize the number of matrix factorizations, the number of function evaluations, the number of linear systems solved, or the overall work (cpu seconds) required. This design is, however, problem size dependent. On medium scale dense problems the number of

matrix factorizations may well dominate the overall work required, while on very small problems the overhead, other than the solution of linear systems, may dominate the computations. If the system has large sparse Jacobians and iterative methods are used to solve the linear systems, then the number of linear systems solved may be an appropriate measure of the work involved. Furthermore, the problem objective may differ depending on the usage. We may wish to follow a homotopy path wherein the goal is to traverse the path as cheaply as possible and to compute the solution on the path at a special value of the homotopy parameter. In this case the path itself is of little interest and only serves as a safety net. (This problem class is sometimes called artificial parameter continuation [45].) Another class of problems is the natural parameter continuation that arises, for example, in nonlinear eigenvalue problems and parametric studies of dynamical systems. Here the details of the curve itself are of the utmost importance in that the objective is to detect changes in structure and number of the solutions over a parameter range. Thus the design objective is not a simple issue and we report the operational counts for several of these performance measures in §7. In reporting computational results for comparative purposes, we have also found that the required parameter settings in such state-of-the-art software as HOMPACK [44] and PITCON [30], [31], [33], as well as our own ABCON, are not always clear. Thus we may need to make several runs on a given problem before it succeeds in computing the final answer properly or in computing the details of the path. This raises the issue of whether one should report only the most successful run or some measure of the total effort required to "solve" a problem. We have chosen to do both by reporting, in §7, the number of runs made to obtain success and statistics for this successful run. (We actually search for the largest tolerances which lead to success.) Finally, we mention three references in which the minimum work as measured by the number of Jacobian factorizations (Newton steps) [35], [40] or the number of linear systems solved [33] was the design criterion. In the first the author himself claims the results are nonconstructive; the second applies only to homotopy methods via classical continuation (correction with λ fixed) and requires knowledge of global parameters which must be estimated in some way; in the latter, PITCON Full was designed to minimize the number of linear systems solved and indeed the code works quite well in this aspect. In our opinion, both theoretical and practical implementation issues remain open in the trade-offs between robustness and minimum work requirements.

The remaining sections of this paper are organized as follows: In §2 the formulas for the Adams–Bashforth predictors are summarized. The computation of arclength along the path is developed in §3, and the error-stepsize control and order selection is developed in §4. Section 5 contains a discussion of the correction phase, methods for dealing with corrector failures, and the computation of the tangent. These procedures are summarized in a high-level pseudocode in §6. To give a comparison of the efficiency and robustness of these Adams–Bashforth predictors, several test problems from the literature are solved and the results are compared with those of HOMPACK [44] and PITCON [31] in §7.

2. The Adams–Bashforth predictors. For completeness, we state in this section the Adams–Bashforth formulas for the initial value problem (1.1). These variable stepsize formulas are due to Krogh [20] and have particularly good roundoff error properties [37]. With the exception of a shifted index these formulas can be found in Shampine and Gordon [37].

Let $P = \{w \in \Re^{n+1} : w = \Psi(t), t \in I\}$, where I is an interval of real numbers,

denote a parameterization of the solution path of $F(w) = 0$ which is assumed to be nonsingular in that $[D_w F|_{w \in P}]$ has full rank on the path P. An integration of the differential equation (1.1) yields

$$w(t_k + \Delta t) = w_k + \int_{t_k}^{t_k + \Delta t} T(w(\sigma)) d\sigma \tag{2.1}$$

Assume $w_i = w(t_i)$ and $T(t_i) = T(w(t_i))$ have been computed for $i = 0, 1, \cdots, k$, and let $P_{k,m}(t)$ be the vector of polynomials, of degree not exceeding m, which interpolates T at $t_k, \cdots, t_{k-m}$. Employing the backward Newton form for this interpolating polynomial, we have

$$T(w(t)) = P_{k,m}(t) + \Gamma_{k,m}(t) T[t_k, \cdots, t_{k-m}, t], \tag{2.2}$$

where $P_{k,m}(t) = \sum_{i=0}^{m} \{\Gamma_{k,i-1}(t) T[t_k, \cdots, t_{k-i}]\}, \Gamma_{k,i}(t) = \prod_{j=0}^{i} (t - t_{k-j})$ for $i \geq 0, \Gamma_{k,-1}(t) = 1$, and $T[t_k, \cdots, t_{k-m}]$ denotes the (vector) Newton divided difference [37], [38]. Equations (2.1) and (2.2) yield

$$w(t_k + \Delta t) = w_k + \Delta t d_{k,m}(\Delta t) + E_{k,m}(\Delta t), \tag{2.3}$$

where

$$\begin{aligned} d_{k,m}(\Delta t) &= \frac{1}{\Delta t} \int_{t_k}^{t_k + \Delta t} P_{k,m}(\sigma) d\sigma, \\ E_{k,m}(\Delta t) &= \int_{t_k}^{t_k + \Delta t} \Gamma_{k,m}(\sigma) T[t_k, \cdots, t_{k-m}, \sigma] d\sigma. \end{aligned} \tag{2.4}$$

The predictor of $w(t_k + \Delta t)$ is taken to be

$$wp(k+1, m, \Delta t) = w_k + \Delta t d_{k,m}(\Delta t), \tag{2.5}$$

which has an error $E_{k,m}(\Delta t)$. Using the following notation

$$\begin{aligned} &\psi_{k,i} = \sum_{j=0}^{i} \Delta t_{k-j} = t_{k+1} - t_{k-i}, \quad \text{where } \Delta t_k = t_{k+1} - t_k = \Delta t \\ &\beta_{k,i} = \begin{cases} 1, & i = 0, \\ \prod_{j=0}^{i-1} \frac{\psi_{k,j}}{\psi_{k-1,j}}, & i \geq 1, \end{cases} \\ &\alpha_{k,i} = \Delta t_k / \psi_{k,i} \quad \text{and} \quad \phi_{k,i} = \prod_{j=0}^{i-1} \psi_{k-1,j} T[t_k, \cdots, t_{k-i}], \end{aligned} \tag{2.6}$$

the direction $d_{k,m}(\Delta t)$ can be expressed as

$$d_{k,m}(\Delta t) = \sum_{i=0}^{m} \phi_{k,i} \beta_{k,i} g_{i1}, \tag{2.7}$$

and the predicted tangent at w_{k+1} as

$$Tp_{k+1,m}(\Delta t) = P_{k,m}(t_k + \Delta t) = \sum_{i=0}^{m} \phi_{k,i} \beta_{k,i}, \tag{2.8}$$

where $g_{i1}, \phi_{k,i}$, and $\beta_{k,i}$ can be efficiently computed from the following recurrence relationships:

$$
\begin{aligned}
& g_{0j} = \frac{1}{j} \quad \text{and} \quad g_{1j} = \frac{1}{[j \cdot (j+1)]}, \\
& g_{ij} = g_{i-1,j} - \alpha_{k,i-1} g_{i-1,j+1}, \qquad \begin{pmatrix} i = 2, \cdots, m \\ j = 1, \cdots, m-i+1 \end{pmatrix}, \\
& \phi_{k+1,i+1} = \phi_{k+1,i} - \beta_{k,i} \phi_{k,i} \quad (i = 0, \cdots, m) \quad \text{where } \phi_{k+1,0} = T(t_{k+1}), \\
& \psi_{k,i} = \psi_{k-1,i-1} + \triangle t, \qquad i \geq 1, \\
& \beta_{k,i} = \beta_{k,i-1} \frac{\psi_{k,i-1}}{\psi_{k-1,i-1}}, \qquad i \geq 1.
\end{aligned} \tag{2.9}
$$

3. The local and global parameterizations of the path: The estimation of arclength. The use of arclength as a global continuation parameter is recommended and used by many authors [2], [16], [17], [21], [22], [25], [26], [42], [44], especially as a way of dealing with limit or fold point singularities. In some of these a local parameterization such as Keller's pseudo-arclength [16] is actually used instead of arclength itself. Watson, Billups, and Morgan [44], [45], on the other hand, use arclength as the actual independent parameter in the ordinary differential equation method HOMPACK (DF and DS), but this is the arclength of the path tracked by the solution of the differential equation (1.1) versus true solution path as defined by the equation $F(w) = 0$. (This particular method does not perform a Newton-like correction back to solution path of $F(w) = 0$, but is appropriate for his intended problem class.) Arclength has also been suggested as an adaptive mesh selection technique for boundary value problems for ordinary differential equations [34] in that a uniform meshsize in arclength tends to place the needed meshpoints "appropriately" along an interval of interest. While the computation of the stepsizes in arclength may be too expensive for a general mesh selection scheme for boundary value problems, its use here is warranted in view of the extreme expense of the function evaluations, which requires the solution of several linear systems. The use of arclength along the solution path tends to give stepsizes that are more or less equal in arclength except in extremely sharp turns, where we generally reduce the multistep method to Euler's method. These more or less equal stepsizes add robustness to the predictors and the error-stepsize control as well as efficiency to the methodology.

When a Newton or Newton-like correction to the true path, defined by $F(w) = 0$, is used in the corrector phase and when a multistep method involving more than two steps is used as a predictor, an accurate estimation of arclength along the solution path of $F(w) = 0$ is required. In the absence of this accurate approximation we have found that Hermite and Adams–Bashforth multistep predictors with more that two steps degenerate severely in that the stepsizes become "too small" to be efficient. This more accurate arclength computation is not needed in either HOMPACK or PITCON: the ODE method in HOMPACK (DF and DS) does not involve a Newton-like correction back to the solution of $F(w) = 0$; at most two-step Hermite predictors are used in the other two methods in HOMPACK; and Rheinboldt's PITCON uses an Euler predictor. Thus the key to the efficiency of the Adams–Bashforth predictors, as developed in this work, is the estimation of arclength along the solution path at an accuracy commensurate with that of the predictors. The next goal is to explain how this more accurate approximation is made. In this derivation, the letter "t" will continue to be used as the independent variable in the generic parameterization for

(1.1), the letter "s" will denote arclength along the path P, and "h" will be a local variable as explained later.

Once a prediction $wp(k+1,m,h_k) = w_k + h_k d_{k,m}(h_k)$ is made, the corrected point, w_{k+1}, on the solution path is obtained by solving an augmented system of equations

$$\begin{bmatrix} F(w) \\ N(w) \end{bmatrix} = 0, \tag{3.1}$$

where $N : \mathbb{R}^{n+1} \to \mathbb{R}$.

The predictor steplength, h_k, must also be "corrected" to Δs_k and this depends on the particular augmenting equation $N(w)$. Three important examples are

$$N_1(w) = (w - wp(k+1,m,h))^T d_{k,m}(h), \tag{3.2}$$

which confines the correction to a hyperplane orthogonal to the predictor direction $d_{k,m}(h)$,

$$N_2(w) = \|w - w_k\|_2^2 - h^2 \|d_{k,m}(h)\|_2^2, \tag{3.3}$$

which confines the correction to a sphere centered at w_k, and, for $w^T = (u^T, \lambda)$ with the parameter being in the $(n+1)$st position

$$N_3(w) = e_{n+1}^T (w - wp(k+1,m,h)), \tag{3.4}$$

which corresponds to the classical correction with the parameter λ being held fixed during the correction process. Equation (3.2) corresponds to tangent [16], (3.3) to secant [25], and (3.4) to natural (λ) parameterizations of the solution path. The first two schemes allow effective computation around fold (limit, turning) point singularities and define a local parameterization [4] of the curve in the following way: given h, compute $wp(k+1,m,h)$ and then correct back to the curve to obtain a locally unique solution $w(h)$. (Some correction processes, such as Georg's normal flow algorithm [1], [2], [14], do not fit this characterization.) The following propositions make these statements precise and allow effective computation of arclength along the solution path P. In the first theorem functional and derivative information needed in the computation of arclength is established, while the next two theorems establish existence and certain properties of the local parameterization for the augmenting equation $N_1(w)$; however, similar results hold for $N_2(w)$. Since the proof of Theorem 1 follows by direct computation and the other two theorems follow easily from the implicit function theorem [9], the proofs are omitted.

THEOREM 1. *Let $F : \mathbb{R}^{n+1} \to \mathbb{R}^n$ be continuously differentiable in an open neighborhood of $P = \{w \in \mathbb{R}^{n+1} : w = \Psi(t), F(\Psi(t)) = 0, t \in I\}$, where I is an interval of real numbers and assume that the path P is nonsingular in that $[D_w F|_{w \in P}]$ is of full rank. Let $T_k = T(w(t_k))$, where $T(w(t))$ denotes the tangent vector $dw/dt = \Psi'(t)$, and extend the definition of $d_{k,m}(h)$ given in (2.4) to*

$$d_{k,m}(h) = \begin{bmatrix} h^{-1} \int_0^h P_{k,m}(t_k + \sigma) d\sigma, & h \neq 0 \\ T_k, & h = 0 \end{bmatrix}.$$

Then $d_{k,m}$ is a polynomial in h and

$$d'_{k,m}(h) = \begin{bmatrix} (P_{k,m}(t_k + h) - d_{k,m}(h))/h, & h \neq 0 \\ T[t_k, t_{k-1}]/2 & h = 0 \end{bmatrix}.$$

In the next theorem, the existence and uniqueness of the local parameterization of the curve is stated for completeness. This local versus global parameterization is certainly implied in, for example, the work of Allgower and Georg [1], [2]; Keller [16], [17]; Rheinboldt and Burkardt [30], [33]; and Watson [45].

THEOREM 2. *Let $F : \mathbb{R}^{n+1} \to \mathbb{R}^n$ be at least C^1 in a neighborhood B of w_k, where w_k is a nonsingular point on the path P defined in Theorem 1, and let T_k be a unit vector in the null space of $D_w F(w_k)$. Consider $G : \mathbb{R}^{n+2} \to \mathbb{R}^{n+1}$ as given in* (3.1)–(3.2), *namely,*

$$G(w,h) = \begin{bmatrix} F(w) \\ d_{k,m}(h)^T(w - w_k - h d_{k,m}(h)) \end{bmatrix},$$

where $d_{k,m}$ is defined as in Theorem 1. *Then the equation $G(w,h) = 0$ defines a local parameterization of the path P by h for h in a sufficiently small neighborhood of zero; that is, there exist (open) neighborhoods $I = (-\epsilon, \epsilon)$ and $U \subset B$ of w_k such that for each $h \in I$ there is a unique $w \in U$ for which $G(w,h) = 0$ and dw/dh is continuous and nonzero. Furthermore, the function $w = w(h)$ is as smooth as F on I, and satisfies $dw(0)/dh = T_k$.*

THEOREM 3. *Let F satisfy the assumptions of Theorem* 1, *s denote the arclength along the solution path P with $w_k = w(s_k)$, h be the parameter given in Theorem* 2, *and $T(w)$ be a unit vector in the nullspace of $D_w F(w)$ with orientation corresponding to the parameter s. Then there is a neighborhood $I = (-\epsilon, \epsilon)$ of $h = 0$, and a corresponding neighborhood V of $s = s_k$ such that $s(h) : I \to V$ is a well-defined, one-to-one function which is as smooth as F. Furthermore, $s = s(h)$ satisfies $ds(0)/dh = 1$ and*

$$\frac{ds}{dh}(h) = \frac{d_{k,m}(h)^T d_{k,m}(h) - [w(h) - w_k - 2h d_{k,m}(h)]^T d'_{k,m}(h)}{T(w(h))^T d_{k,m}(h)}. \tag{3.5}$$

We now assume that the path P is parameterized by arclength so that "s" will replace the variable "t" in the theorems above. Unfortunately, the local parameter h is not a sufficiently accurate approximation to the arclength along the curve from w_k to $w(h)$. Given a step h_k, better estimates of arclength can be obtained by approximating the arclength integral in the local parameter $\triangle s_k = \int_0^{h_k} \|dw(h)/dh\|_2 dh$. Note that since dw/ds is a unit vector and $dw/dh = (dw/ds)(ds/dh)$, the arclength increment $\triangle s_k$ may be computed as

$$\triangle s_k = \int_0^{h_k} \left| ds(h)/dh \right| dh. \tag{3.6}$$

Now $ds(h_k)/dh$ can be obtained cheaply from the formulas given in the above theorems since the quantities $h_k, d_{k,m}(h_k)$, and $P_{k,m}(s_k + h_k)$ have been computed for the predictor $wp(k+1, m, h_k)$, $w(h_k)$ is the new corrected point, and $T_{k+1} = T(w(h_k))$ must in any case be computed for the next predictor step. Thus, in the normal course of taking steps along the path, $ds(h)/dh$ is known at $h = 0$ and $h = h_k$. Simple integration rules using this information include the rectangle rule ($s_{k+1} - s_k = h_k(ds(0)/dh) + \mathcal{O}(h_k^2) = h_k + \mathcal{O}(h_k^2)$) and the trapezoidal rule ($s_{k+1} - s_k = .5h_k(ds(0)/dh + ds(h_k)/dh) + \mathcal{O}(h_k^3)$). These rules are, however, not sufficiently accurate for the higher-order predictors, so that we must turn to Gaussian or some other higher-order quadrature rules. The difficulty encountered in this direction is the prohibitively expensive cost of evaluating $ds(h)/dh = \|dw(h)/dh\|_2$ at values of h between zero and h_k: we must first predict

and then correct back to the curve to evaluate $ds(h)/dh$. One possibility is to approximate $\Delta s_k = \int_0^{h_k} \|dw(h)/dh\|_2 dh$ by extrapolating $dw(h)/dh$ from the polynomial $P_{k,m}(s_k + h)$ which interpolates $dw(h)/dh$ at $h = 0, -\Delta s_{k-1}, \cdots, -(s_k - s_{k-m})$. Since we must compute the $(k+1)$st point and unit tangent anyway, we prefer to use an updated polynomial and thus polynomial interpolation. After h_k and $dw(h_k)/dh$ are known, a new interpolating polynomial $\tilde{P}_{k+1,m+1}$ of degree $m+1$ can be constructed which is an update of $P_{k,m}$, incorporates the stepsize h_k and renormalized tangent $dw(h_k)/dh$, and agrees with $P_{k,m}$ at s_i for $i = k, \cdots, k-m$. Then we may write (assuming $s_i - s_{i-1} = \mathcal{O}(h_k)$ as $h_k \to 0$ for $i = k, \cdots, k-m+1$)

$$s_{k+1} - s_k = \int_0^{h_k} \left\| \frac{dw}{dh}(\xi) \right\|_2 d\xi = \int_0^{h_k} \left\| \tilde{P}_{k+1,m+1}(s_k + \xi) \right\|_2 d\xi + \mathcal{O}(h_k^{m+3}). \tag{3.7}$$

The latter integral may be estimated using a higher-order quadrature rule as discussed above. The order of the integration scheme used should be commensurate with the error in approximation (3.7). The Gaussian–Lobatto integration rules [39] are a good choice since they make explicit use of the exact information at the endpoints. Since this rule with q points has an asymptotic error which is $\mathcal{O}(h_k^{2q-1})$ [38], q is chosen to satisfy $2q - 1 > m + 3$ so that the error from numerical quadrature does not contribute (asymptotically) to the dominant error term in (3.7). The details of these computations are summarized in the following high-level pseudocode.

Arclength Correction Algorithm. Given the predicted tangent $T = Tp_{k+1,m}(h_k)$, new tangent T_{k+1}, point w_{k+1} on the path P, the predictor degree $m := m_k$ together with $\varphi_{k,i}$ and $\beta_{k,i}$ for $i = 0, \cdots, m$, the point w_k, the prediction direction $d = d_{k,m}(h_k)$, and the predictor stepsize h_k from the kth predictor step, the following algorithm produces an improved estimate of the stepsize Δs_k in arclength between w_k and w_{k+1} on the path P:

1. $d' := (Tp - d)/h_k := (P_{k,m}(s_k + h_k) - d)/h_k$ { Theorem 1, §3, (2.8) }.
2. $ds(0)/dh := 1$ { Theorem 3, §3 }
3. $ds(h_k)/dh := [d^T d - (w_{k+1} - w_k - 2h_k d)^T d']/T_{k+1}^T d$ {Theorem 3, §3 }.

Compute modified divided differences for polynomial $\tilde{P}_{k+1,m+1}$:

4. $\tilde{\varphi}_{k+1,0} := (ds(h_k)/dh)T_{k+1}$.
5. Set $\tilde{\varphi}_{k+1,j} := \tilde{\varphi}_{k+1,j-1} - \beta_{k,j-1}\varphi_{k,j-1}$ for $j = 1, \cdots, m+1$ { Use (2.9)}
6. Choose the number of Lobatto points: $q :=$ integer part $(m+5)/2$ and let π_i and ω_i for $i = 0, \cdots, q-1$ denote the Lobatto points and corresponding weights on the interval [0,1] [39].
7. Perform the Lobatto quadrature on (3.7):

$$\Delta s_k := \omega_o \cdot \frac{ds}{dh}(0) + \sum_{i=1}^{q-2} \omega_i \|\tilde{P}_{k+1,m+1}(s_{k+1} + (1 - \pi_i)h_k)\|_2 + \omega_{q-1} \cdot \frac{ds}{dh}(h_k).$$

{ Here, the $\tilde{P}_{k+1,m+1}(s_k + (1-\pi_i)h_k) = Tp_{k+2,m+1}((1-\pi_i)h_k)$ are computed using formula (2.8) with $\Delta t := (1 - \pi_i)h_k$, the $\tilde{\varphi}$'s in place of the φ's, and m and k replaced by $m+1$ and $k+1$, respectively.}

4. Error-stepsize control and order selection. Given error tolerances for the predicted point, the error-stepsize control developed here yields an estimate of the stepsize that satisfies these error tolerances and then adjusts the order to produce the largest such stepsize. Although this methodology is motivated by similar techniques

in ordinary differential equations, there are significant differences that arise out of the objective to track a solution of the underdetermined system of equations $F(w) = 0$ rather than to solve the initial value problem (1.1). This is most evident in the correction process which involves a Newton or Newton-like correction (solve) back to the solution of the system $F(w) = 0$. Two fundamental implications of this correction are that the global rather than local error is controlled at each step and, due to the expense of the correction process, (it involves the solution of several linear systems), the error-stepsize control generally (but not always) avoids the corrector in the decision of the stepsize. Thus we use consecutive order Adams–Bashforth predictors to estimate the stepsize that should be taken in the present step rather than using the Adams–Bashforth–Moulton pair as in ordinary differential equations [37]. Also, in ordinary differential equations we frequently use as the current stepsize an optimal stepsize that should have been taken on the previous step, but this strategy is too expensive here since it involves the corrector process. Although this procedure is not fail-safe, the correction process and the underlying equation $F = 0$ provide a verification of the accuracy of the predictor. Thus we can be more aggressive in the choice of stepsize—if the stepsize is too large or the prediction is too inaccurate, we can detect this, modify the predictor by cutting the stepsize and order, and restart the corrector. Corrector failures are an inevitable consequence of this strategy but are acceptable in that we can deal with them effectively; however, too many such failures can cause the methodology to be inefficient. (We hasten to emphasize that corrector failures are generally due to sharp turns in the solution path which may or may not be associated with singularities on the path.) In the remainder of this section, we present the error-stepsize control and order selection scheme.

To estimate a stepsize $h_k := \gamma h_{k-1}$ which achieves a combined absolute (Eabs) and relative (Erel) error requirement in the predictor, we use consecutive order Adams–Bashforth formulas to estimate the error in the lower order formula. Requiring that the estimate of the error in each component meet the given tolerances gives the formula

$$\gamma_m = \min_i \left[\frac{\text{Eabs} + \text{Erel}|[wp(k+1, m+1, h_{k-1})](i)|}{2|[wp(k+1, m+1, h_{k-1}) - wp(k+1, m, h_{k-1})](i)|} \right]^{1/(m+2)}, \tag{4.1}$$

where the customary 2 has been added in the denominator inside the radical to account for the asymptotic nature of this formula and the (i) refers to the ith component of the vector contained in brackets. Our algorithm makes use of this estimate to select the order and stepsize at each step. In general, estimates γ_{m-1}, γ_m and γ_{m+1}, corresponding to three consecutive orders, are computed where m is the order of the predictor used on the previous step. The order giving the largest value of γ is selected, along with the corresponding stepsize γh_{k-1} subject to the constraints that $\gamma \in [0.1, 10.0]$, and $\gamma h_{k-1} \in [h_{\min}, h_{\max}] = [10^{-7}, 100.0]$. Then, assuming the stepsize has not been decreased, local extrapolation is used, that is, the predictor of order one higher than the selected order is used. (The use of local extrapolation followed by a Newton-like correction generally produces between a half and a full digit of additional accuracy in the corrected point on the curve.) Finally, we impose a maximum order $Maxm$, which we typically choose to be four when Eabs = Erel = 0.01 is used. For smaller predictor tolerances, larger maximum orders are suggested.

Asymptotic estimates similar to (4.1) with associated error-stepsize and order controls have been discussed in Shampine and Gordon [37] and Krogh [20], so we will not derive our version here. Krogh [20], however, points out several advantages to allowing even small changes in the stepsize at every step, among which is greater

efficiency in terms of function evaluations. In our cases the overhead involved in recomputing the integration coefficients at each step is insignificant in comparison to the cost of the linear algebra at each step, so we allow the stepsize to change on each step. Timing studies on a Cyber 205, using problems one and five in §7 demonstrated that the entire prediction phase of the algorithm typically consumes less than 10 percent of the CPU time for problems of dimension 20 and less than 1 percent on problems of dimension 100.

An examination of the error term (2.4) suggests several appropriate safeguards in the use of (4.1) [37]. The derivation of this formula assumes that $E_{k,m}(h_k) = \mathcal{O}(h_k^{m+2})$ as $h_k \to 0$. If $h_{k-i} = \mathcal{O}(h_k)$ for $i = 1, \cdots, m$ and w is sufficiently smooth, this is indeed the case. On the other hand, if $h_k << h_{k-i}$ for $i = 1, \cdots, m$ then $E_{k,m}(h_k) = \mathcal{O}(h_k^2)$, so that the predictor behaves like Euler's method. We can conclude that when the stepsize is being drastically reduced, the higher-order method may be less accurate than Euler's method. Thus, in the error-stepsize control we drop to Euler's method when drastic stepsize reductions are required, as often occurs in the case of a corrector failure. This also suggests that local extrapolation be abandoned and that the order not be raised when the stepsize is being decreased. The following pseudocode summarizes these computations.

Order and Stepsize Selection Algorithm. Given the step index k ($k = 0$ corresponds to the first step), the previous degree m_{k-1} and stepsize h_{k-1} (for $k \geq 1$), the maximum degree, $Maxm$, the current point w_k, the predictor information for computing $P_{k,i}$, for $0 \leq i \leq M := \min\{k, m_{k-1} + 2, Maxm + 1\}$ (see (2.6)–(2.8)), the following algorithm selects the degree m_k and the stepsize h_k for computing the predicted point $wp(k+1, m_k, h_k)$.

Input Variables and Parameters:

k:	index denoting that $(k+1)$ points $\{w_i\}_{i=0}^k$ on the path have been computed.
h_{k-1}:	the predictor stepsize from the previous step if $k \geq 1$.
w_k:	the current point on the curve.
$\phi_{k,i}$ $(i = 0, \cdots, M)$, $\psi_{k-1,i}$, and $\beta_{k-1,i}$ $(i = 0, \cdots, m_{k-1}+1)$:	predictor information from previous steps.
$fail$:	a logical variable indicating corrector failure at stepsize $hfail$.
$hfail$:	attempted stepsize if $fail$ = true.
m_{k-1}:	the predictor degree selected on the previous step without local extrapolation.
Erel, Eabs:	relative and absolute predictor error tolerances, as appearing in the step control formula (4.1), e.g., Erel = Eabs = 0.01.
$Maxm$:	the maximum degree allowed for the predictor. This corresponds to a $Maxm + 1$ step predictor which relies on a $Maxm + 2$ step predictor for error control and local extrapolation, e.g., $Maxm := 4$.
$hmin$, $hmax$:	minimum and maximum steplengths allowed, e.g., $hmin = 0.000001$ and $hmax = 100.0$.

Output Variables:

h_k:	predictor stepsize selected.
m_k:	predictor order selected.

Algorithm:

1. If $k = 0$, do 1a-1c:
   1a. $h_0 = (\sqrt{\text{Eabs}} + \|w_0\|_2\sqrt{\text{Erel}})/2$
   1b. $m_0 = 0$
   1c. Return
2. If $k \geq 1$ do 2a-2d:
   2a. If fail = true, then go to 2c.
   2b. Select the predictor order using (4.1) and comparing consecutive orders:
      2b.1. Set $m := m_{k-1}$.
      2b.2. If $m < Maxm$ and $k \geq 2$, then compute γ_{m+1} and γ_m, and set
      $m_k := \text{argmax}\{\gamma_{m+1}, \gamma_m\}$
      $\{m_k$ is either m or $m+1\}$.
      2b.3. If $\gamma_{m_k} < 1$, then $m_k := m$ { Do not raise order if stepsize is decreased }.
      2b.4. If $m \geq 1$, then compute γ_{m_k} and γ_{m-1}, and set
      $m_k := \text{argmax}\{\gamma_{m_k}, \gamma_{m-1}\}$
      { Choose between orders m_k and $m-1$}.
      2b.5. If $m = Maxm \geq 2$, then compute γ_{m-2} and γ_{m_k}, and set
      $m_k := \text{argmax}\{\gamma_{m-2}, \gamma_{m_k}\}$
      { Choose between orders m_k and $m-2$}.
   2c. Select stepsize using (4.1) to compute γ_{m_k}:
      2c.1. $h_k := \text{Min}\{\text{Max}\{0.1, \gamma_{m_k}\}, 10.0\}h_{k-1}$.
      2c.2. If $fail$ = true, then $h_k := \min\{hfail, h_k\}$
      { If corrector failed, do not increase h}.
      2c.3. $h_k := \min\{hmax, \max\{h_k, hmin\}\}$ { Limit the stepsize }.
   2d. Return

The current choice of the initial stepsize h_0 is a simple and conservative one, and we would conjecture that more sophisticated starting schemes, such as those presented by Watts [46]–[48] for ordinary differential equation solvers, would yield considerable improvement. No doubt, predictor-corrector continuation codes could, in general, benefit from improved starting schemes. The issues, however, are somewhat different from those in ordinary differential equations. The main requirement for the starting stepsize is that the first step should yield a predicted point within the domain of convergence of the Newton-like method used in the correction process. If this requirement is not satisfied, then the stepsize is reduced until it is, resulting in many Newton-like corrections and considerable inefficiency. In the current methodology, an initial step which is a few orders of magnitude too small is adjusted within a few steps since we allow a large (10 times) change in the stepsize at each step. The issues are thus clouded by the fact that each step is equivalent to a function evaluation and most robust starting schemes for ordinary differential equations require several function evaluations (Watts's scheme requires four). Nevertheless, this is a topic that needs further investigation.

5. The corrector and the computation of the tangent. Having completed the discussion of the prediction phase and error-stepsize control, we now turn to a discussion of the corrector phase and the computation of the tangent to the curve. The corrected point on the curve is the solution of the augmented system of equations (3.1) and (3.2), i.e., $F(w) = 0$ and $N_1(w) = 0$, where $F : \mathbb{R}^{n+1} \to \mathbb{R}^n$ and $N_1 : \mathbb{R}^{n+1} \to \mathbb{R}$. The predicted point is usually used as an initial approximation to a solution of these equations. Methods ranging from Newton, Newton-like, or quasi-Newton methods (which may or may not employ a trust region or line search as a globalizer) to conjugate gradient methods may be used to solve this system of nonlinear equations. Some of

these methods are discussed in the reviews of Watson [43] and Allgower and Georg [1] as well as the work of Rheinboldt [33] and Keller [16]–[19]. Also, specific linear algebra methods have been developed to take advantage of the special structure of this system, but it is not our intent to defend or recommend one class of methods over another. For completeness, we describe the method used here, which is the bordering algorithm of Keller [18]. This method preserves the underlying matrix structure that may be present in problems with special structure, but theoretically it can fail at or near a singularity. In practice it performs reliably without modification; however, many additional refinements to this general procedure have been suggested in the literature [7] to overcome these deficiencies. We have also experimented with damped Newton's method and the use of the augmenting equation (3.3). Due to the philosophy of closely following the curve, we have not found these latter two procedures to give any significant improvement. For example, we have yet to find a problem for which damped Newton's method improves efficiency or reduces the number of corrector failures. The reason is that in the current methodology failures are almost always associated with the lack of a solution to (3.1), which most often occurs near sharp turns in the path.

The computation of the tangent to the curve is based on the fact that the tangent lies in the nullspace of the Jacobian matrix $D_w F(w)$, which we assume to be of full rank. This tangent is determined only to within a multiplicative constant, and thus the scaling and orientation of this vector must be specified to remove this nonuniqueness. Again, there are several methods for deciding on magnitude and orientation as discussed in the review of Allgower and Georg [1], as well as the work of Watson [42], Keller [16], and Rheinboldt [33]. Although any of these methods can be used with the current predictors, we have used Keller's bordering algorithm [16]–[19], which is briefly described next.

If $w = \binom{u}{\lambda}$, where λ is the parameter in the problem and $d_{k,m}(h_k) = \binom{\theta_k}{\rho_k}$, then (3.1) can be written as

$$\begin{bmatrix} F(w) \\ N(w) \end{bmatrix} = \begin{bmatrix} F(u,\lambda) \\ (u - up_{k+1})^T \theta_k + (\lambda - \lambda p_{k+1})\rho_k \end{bmatrix} = 0. \tag{5.1}$$

Furthermore, the tangent T satisfies

$$[D_u F(u_{k+1}, \lambda_{k+1}) : D_\lambda F(u_{k+1}, \lambda_{k+1})] T(u_{k+1}, \lambda_{k+1}) = 0, \tag{5.2}$$

where (u_{k+1}, λ_{k+1}) is the corrected point on the curve. Before explaining Keller's bordering algorithm, it is insightful to first recall Newton's method for the system (5.1) wherein we start from an initial approximation

$$\begin{pmatrix} u^0 \\ \lambda^0 \end{pmatrix} = \begin{pmatrix} up_{k+1} \\ \lambda p_{k+1} \end{pmatrix} = wp_{k+1}$$

and generate a sequence of points $\left\{\binom{u^i}{\lambda^i}\right\}$ where $\binom{u^{i+1}}{\lambda^{i+1}}$ is obtained from $\binom{u^i}{\lambda^i}$ by first solving the system

$$\begin{bmatrix} D_u F & D_\lambda F \\ \theta_k^T & \rho_k \end{bmatrix} \begin{bmatrix} \triangle u^i \\ \triangle \lambda^i \end{bmatrix} = - \begin{bmatrix} F(u^i, \lambda^i) \\ N(u^i, \lambda^i) \end{bmatrix} \tag{5.3a}$$

and then updating

$$\begin{pmatrix} u^{i+1} \\ \lambda^{i+1} \end{pmatrix} = \begin{pmatrix} u^i \\ \lambda^i \end{pmatrix} + \begin{pmatrix} \triangle u^i \\ \triangle \lambda^i \end{pmatrix}. \tag{5.3b}$$

On each iteration the matrix $D_u F$ and vector $D_\lambda F$ are evaluated at the point $\binom{u^i}{\lambda^i}$ for Newton's method but are left fixed at $\binom{u^0}{\lambda^0}$ for the chord method. In Keller's bordering algorithm, one first solves for z^i and v^i from the systems

$$[D_u F]z^i = -F(u^i, \lambda^i), \tag{5.4a}$$

$$[D_u F]v^i = -D_\lambda F, \tag{5.4b}$$

respectively. Then any solution to (5.3a) has the form

$$\triangle u^i = z^i + \triangle\lambda^i v^i \tag{5.4c}$$

where

$$\triangle\lambda^i = -(N(u^i, \lambda^i) + \theta_k^T v^i)/(\theta_k^T z^i + \rho_k). \tag{5.4d}$$

The iterate can then be updated according to (5.3b). Note that (5.4a–d) enable us to obtain $\binom{\triangle u^i}{\triangle\lambda^i}$ while exploiting any special structure in $D_u F$. For this reason (5.4a–d) can be less expensive than the direct factorization of the matrix in (5.3a).

Another advantage of the bordering algorithm is that the tangent to the path is easily obtained at little extra expense. Suppose

$$\begin{pmatrix} u^{i+1} \\ \lambda^{i+1} \end{pmatrix}$$

is accepted as the corrected point on the curve so that

$$w_{k+1} = \begin{pmatrix} u_{k+1} \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} u^{i+1} \\ \lambda^{i+1} \end{pmatrix},$$

and the factorization of $D_u F$ is still available from the previous step. Then a good approximation to the unit vector T which solves $[D_u F : D_\lambda F]T = 0$ is

$$T = \pm \begin{pmatrix} v^i \\ 1 \end{pmatrix} \Big/ \sqrt{1 + \|v^i\|_2^2}. \tag{5.5}$$

The sign determines the orientation of the continuation and is subject to change only when $\det(D_u F)$ changes sign. If this determinant changes sign, but that of the Jacobian of the augmented system (5.1) does not, then we declare that a fold point singularity has been passed and change the orientation of T. If both determinants change sign, then a bifurcation point is declared [16]. (These determinants can be computed cheaply from the factorization of $D_u F$ if the elimination and pivoting steps are also performed on $D_\lambda F$. A simple procedure used by Watson [44] to maintain orientation is based on maintaining a positive $T_k^T T_{k+1}$ and this works well when no bifurcation points are present.) Next, if w_{k+1} is a fold point singularity, then $D_u F$ is singular and $D_\lambda F$ is not in the range of $D_u F$, so that v^i must be defined alternately as a null vector of $D_u F$, and each "1" in formula (5.5) must be changed to "0." Keller [18] has shown, however, that (5.4a–d) and (5.5) remain effective computationally even when $D_u F$ is ill-conditioned as long as the pivoting strategy used gives a small last pivot. In this case, (5.4b) is essentially a step of inverse iteration for a null vector, and relative to the size of the computed v^i, the 1's in (5.5) become negligible.

Since the factorization of the matrix $D_u F$ is the major expense in many continuation problems, the simplified Newton iteration in which the Jacobian is evaluated and factored only at the predicted point (the chord method) gives increased efficiency [16], [25], [33], [36]. In this procedure one exchanges the quadratic rate for a linear rate of convergence, but retains the same Kantorovich domain of convergence [27]. However, the contraction factor in the linear rate of convergence decreases with increased accuracy in the predictor.

When using the above Newton iteration, a termination criterion must be provided to determine when an iterate w_{k+1}^i is to be accepted as the corrected approximation to $w_{k+1} = w(s_{k+1})$ or when to declare that the iteration is to be stopped due to the lack of convergence. Our algorithm employs a user-specified error corrector tolerance, Ecor, and maximum iteration limit, $Maxit$ (e.g., $Maxit = 7$). The corrector tolerance is usually chosen such that $[(\text{Eabs} + \text{Erel})/2]^2 \leq \text{Ecor}$, and Ecor > Macheps, where Macheps denotes machine precision or unit roundoff. The idea is to require roughly the error reduction of one iteration of a quadratically convergent process, so that the minimal Ecor will require one or two iterations of the corrector, assuming the predictor has satisfied its given tolerance. For our termination criterion we also demand that corrector iterate w^{i+1} satisfy

$$\text{(5.6a)} \quad \|F(w^{i+1})\|_\infty \leq \text{Ecor} \cdot \max\{\|F(w^0)\|, 1\},$$

$$\text{(5.6b)} \quad |[w^{i+1} - w^i](j)| \leq \text{Ecor} \cdot \max\{|typw|, |[w^{i+1}](j)|\} \quad \text{for } j = 1, \cdots, n+1,$$

where $w^{(0)} = wp_{k+1}$ is the predicted point and $typw$ is a pre-chosen "typical value" of $[w^{i+1}](j)$, which is usually set to 1.0 [10]. This requires (roughly) that the change in each component of w^i relative to w^{i+1} be less than Ecor. If these criteria are not satisfied for some $i + 1 \leq Maxit$, the iteration is aborted and termed a corrector failure. We also stop the iteration and declare a corrector failure if either

$$\text{(5.6c)} \qquad \|w^{i+1} - w^i\|_\infty \geq \epsilon_1 \|w^i - w^{i-1}\|_\infty, \qquad i \geq 0$$

or

$$\text{(5.6d)} \qquad \|F(w^{i+1})\|_\infty \geq \epsilon_2 \|F(w^i)\|_\infty, \qquad i \geq 0,$$

where ϵ_1 and ϵ_2 are typically chosen to be 1.5. As can happen with any stepsize control, the predicted point may not lie within the domain of attraction of Newton's method for (3.1) and thus a method for dealing with a corrector failure is needed.

In case of a failed corrector we immediately drop to Euler's method and recompute the stepsize h_k using the error-stepsize control, with the additional constraint that the new stepsize not be greater than the failed stepsize. Then we use an Euler predictor with this stepsize and try the corrector again from this new point. In case of subsequent failures or if the first failure involved an Euler predictor, we continue halving the stepsize (thus circumventing the error control procedure) until corrector convergence is obtained or some minimum stepsize $h_{\min}$ is reached, in which case the entire algorithm is stopped. Another modification to the stepsize control occurs when we want to compute a special point on the curve. For higher-order predictors we must solve a nonlinear equation to find the stepsize $\bar{h}$ for which the predicted point has the same parameter value as the special point; however, Newton's method can be used effectively to find this $\bar{h}$. Then we perform a vertical correction that involves the use of the augmenting equation (3.4). A potentially more robust procedure using inverse interpolation can be found in the work of Watson, Billups, and Morgan [44].

6. A high level pseudocode. We summarize the discussion of the previous three sections by providing in this section an outline of a continuation algorithm which incorporates our predictor strategy.

It should be emphasized that many different linear algebra methods may be used in the corrector and the tangent computation, but we have used those described in the previous section to obtain the numerical results reported for our code ABCON in §7.

Algorithm: Predictor-Corrector Continuation Using Variable Order Adams–Bashforth Predictors. Given $F : \mathbb{R}^{n+1} \to \mathbb{R}^n, w_0 = \binom{u_0}{\lambda_0}$ such that $F(w_0) = 0$ and parameter bounds λ_L and λ_R with $\lambda_0 \in (\lambda_L, \lambda_R)$, the following algorithm obtains a computational trace $\{w_k\}_{i=0}^{K}$ of a segment of the solution path of the equation $F(w) = 0$, terminating with $[w_K]_{n+1} = \lambda_R$ (or λ_L) or $K = Kmax$.

Input Parameters (with default values):

Orient:	determines the orientation of the path and has a value of ± 1 as appears in (5.5).
λ_L, λ_R :	lower and upper bounds on the continuation parameter λ.
$Kmax$:	the maximum number of points computed (5000).
Erel, Eabs:	relative and absolute predictor error tolerances, as appearing the step control formula (4.1) (0.01,0.01).
Ecor:	error tolerance for the computed points, as appears in the acceptance criterion (5.6a) and (5.6b) (.0001).
Efin:	error tolerance for final point (at $\lambda = \lambda_L$ or λ_R); Ecor is reset to this value for the final correction (0.0001)
w_0:	the initial point on the path satisfying $\lambda_0 \in (\lambda_L, \lambda_R)$ and $\|F(w_0)\| \leq$Ecor.
$Maxm$:	the maximum degree allowed for the predictor. This corresponds to a $Maxm + 1$ step predictor which relies on a $Maxm + 2$ step predictor for error control (4).
$hmin$, $hmax$:	minimum and maximum steplengths allowed (0.000001,100.0).
$Maxit$:	maximum number of Newton-like iterations in one correction attempt (7).

Output Parameters:

$\{w_k\}_{k=1}^{K}$:	Computed points on the path with $K = Kmax$ or $[w_K]_{n+1} = \lambda_L$ or λ_R.
$\{\triangle s_k\}_{k=0}^{K-1}$:	Computed arclength stepsizes between w_k and w_{k+1}.
$\{T_k\}_{k=0}^{K-1}$:	Computed tangents to the path at w_k.

Algorithm:

1. Set $fail$:=false and EC := Ecor.
2. Compute tangent $T_0 = T(w_0)$ via (5.4b) and (5.5) (or other methods) with the sign of Orient; set $\varphi_{0,0} := T_0$.
3. Compute the determinant of $S_0 = D_u F(w_0)$ and of the matrix

$$L_0 = \begin{bmatrix} D_w F(w_0) \\ D_w N(w_0) \end{bmatrix}$$

   using the factorization of $D_u F(w_0)$ from step 2.
4. Set $k = 0$
5. While $k \leq Kmax$ do steps 5–18

{ Prediction phase occurs in steps 6–9 }

6. If $k \geq 1$, then update predictor information:
   6a. Reset $h_{k-1} := \Delta s_{k-1}$.
   6b. Compute the $\varphi_{k,i}$ for forming $P_{k,i}(s)$ for $i = 0, \cdots, \text{Min}\{m_{k-1} + 2, k, maxm + 1\}$, using T_k, Δs_{k-1}, the $\varphi_{k-1,i}$, and the formulas in §2.
7. Select the predictor order m_k and stepsize h_k using the algorithm in §4.
8. If $k \geq 1$ and $m_k \geq m_{k-1}$, then $m := m_k + 1$ { use local extrapolation }, else $m := m_k$.
9. Compute the predictor:
   9a. Compute the predictor using (2.5)–(2.7) and (2.9):
      9a.1. $wp := wp(k + 1, h_k, m)$.
      9a.2. $d := d_{k,m}(h_k)$.
   9b. If $\lambda p \ (= [wp]_{n+1}) \leq \lambda_L$ (respectively, $\lambda p \geq \lambda_R$), then compute target point predictor:
      9b.1. Compute $\bar{h}$ with $\lambda p = [wp(k + 1, \bar{h}, m)]_{n+1} = \lambda_L$ (respectively, λ_R) using Newton's method.
      9b.2. $h_k := \bar{h}$.
      9b.3. $wp := wp(k + 1, h_k, m)$ (using 2.5–2.7,2.9).
      9b.4. $d := e_{n+1}$ (direction for correction with λ fixed).
      9b.5. $m := 0$, EC := Efin.

{ Correction, tangent, and arclength computation occur in steps 10–17 }

10. Perform and monitor chord, Newton, or quasi-Newton iterations on system (5.1).
    10a. Set $i := 0$ and $w^0 := wp$
    10b. While $i \leq Maxit$ Do 10b.1–10b.5:
       10b.1. Compute z^i, v^i, and $\Delta w^i := \binom{\Delta u^i}{\Delta \lambda^i}$ using equations (5.4a–d)
       10b.2. Update $w^{i+1} := w^i + \Delta w^i$
       10b.3. If w^{i+1} satisfies (5.6a) and (5.6b), set $w_{k+1} := w^{i+1}$ and Go To 12.
       10b.4. If w^{i+1} satisfies (5.6c) (for $i \geq 1$) or (5.6d), declare a corrector failure and Go To 11.
       10b.5 $i := i + 1$ and Go To 10b.
11. Cut order and/or steplength and prepare for another corrector attempt.
    11a. $Fail$:=true, $hfail := h_k$, EC := Ecor.
    11b. If $m_k > 0$, then reset $m_k := 0$ and Go To 7.
    11c. $h := h/2$.
    11d. If $2h \leq hmin$, then STOP since the run failed.
    11e. Go To 9.
12. If $\lambda_{n+1} = [w_{k+1}]_{n+1} = \lambda_R$ (or λ_L), then target has been reached, OUTPUT (w_{k+1}), and STOP.
13. Compute tangent T_{k+1} via (5.5) using the v^i from the above corrector iteration, and with the sign of Orient.
14. Compute the determinant of $S_{k+1} = D_u F(w_{k+1})$ and of the matrix

$$L_{k+1} = \begin{bmatrix} D_w F(w_{k+1}) \\ D_w N(w_{k+1}) \end{bmatrix}$$

    from the last PLU factorization of $D_w F$ in step 9.
15. If $\det(S_k)\det(S_{k+1}) < 0$ and $\det(L_k)\det(L_{k+1}) > 0$, then declare that a fold point has been passed, set $T_{k+1} := -T_{k+1}$ and Orient := − Orient.
    If $\det(S_k)\det(S_{k+1}) < 0$ and $\det(L_k)\det(L_{k+1}) < 0$,

then declare that a bifurcation point has been passed.
16. Compute the corrected arclength $\triangle s_k$ by using h_k in the arclength correction algorithm in §3.
17. OUTPUT w_{k+1}, T_{k+1}, and $\triangle s_k$.
18. Set $k := k + 1$ and Go To 5.
19. STOP and OUTPUT ("Maximum Number of Steps Exceeded").

It should be observed that a Newton, quasi-Newton, or chord iteration could be used in step 10 above. Also, additional accuracy in the tangent may be obtained by solving (5.4b) with the derivatives evaluated at w_{k+1} and using the resulting solution in (5.5). Each of these entails additional computation.

7. Numerical examples. To examine the efficiency and robustness of the above methodology, we have developed a research code called ABCON for *A*dams–*B*ashforth *con*tinuation which employs two methods for solving the corrector equations (3.1) or (5.1). The first employs a full Newton method with tangents computed at the next-to-last corrected point and is called ABCON Full; the second uses a chord method with tangents computed at the predicted rather than corrected point and is called ABCON Chord, somewhat in analogy to Rheinboldt's PITCON Full and Chord [30], [31], [33]. ABCON was "tuned" by numerically computing the penalty paths for constrained optimization [23], [28] and the path defined by the ellipse $au^2 + b\lambda^2 = 1$ for the ratio a/b ranging from 10^{-10} to 10^{10}. Eleven difficult small test problems were then chosen from the continuation literature [9], [36], [45], [49] to provide a basis for testing the robustness and efficiency of ABCON as well as a numerical comparison with the state-of-the-art software packages HOMPACK by Watson, Billups, and Morgan [44] and PITCON by Rheinboldt and Burkardt [30], [31]. (HOMPACK has three methodologies in the codes QF (augmented Jacobian), NF (normal flow), and DF (ordinary differential equations method) as well as sparse versions of these (QS, NS, and DS, respectively).) Since the test problems were small dense ones, the conclusions in this section are necessarily confined to this class of problems; however, to give some indication of potential performances for other problem classes, we report the number of computed points on the curve, the number of matrix factorizations, the number of function evaluations, the number of linear systems solved, the arclength of the curve, tolerances used, and the number of runs required to successfully solve a problem. Before continuing with a discussion of the test problems and the numerical results, we first discuss the design objectives in these three codes and some of the implications.

A philosophy behind the use of these Adams–Bashforth predictors and with the developed error-stepsize control has been to follow the curve "closely" for robustness and to take shorter steps, so that only two or three corrections back to the curve are generally required. The second correction is essentially used only to check the accuracy of the first full Newton correction, and thus the power of Newton's method in the correction process is not utilized. ABCON is a research code tuned on small problems, while HOMPACK and PITCON are production software packages intended and tuned for the solution of a wide variety of problems. HOMPACK focuses on nonlinear equations via homotopy methods, particularly the regularizing homotopy, and PITCON, on large sparse problems. Since the design objectives of these three codes differ, we should not make any general conclusions about their relative efficiency and robustness; the test set is much too small for such a comparison. Instead, the objective is to investigate the viability of the use of the Adams–Bashforth predictors with the developed error-stepsize control in continuation codes.

As stated earlier the focus in this work has been the development of Adams–

Bashforth predictors and a corresponding error-stepsize control. Thus we have not tried to develop any of the special linear algebra methods [43] for different problem classes. The linear algebra methods in the development of ABCON were based on the bordering algorithm of Keller [17], which is generally applicable, and Gaussian elimination with scaled partial pivoting to solve the reduced problem in this bordering algorithm. The test problems reflect this development in that they are restricted to small dense ones with dimensions ranging from 2–50; however, they are difficult ones used in the continuation literature [9], [36], [44], [45] to test continuation methodologies. Ten of the 11 problems are homotopy problems; eight of these were taken from the papers of Watson, Billups, and Morgan [44], [45] and were used in testing HOMPACK. With regard to the remaining three test problems, number 11 in the following list was taken from the paper of Heijer and Rheinboldt [9] in which the error-stepsize control for PITCON was developed. Problems 3 and 4 can be found in the work of Schwetlick and Cleve [36] with the former being a parameter continuation problem and the latter a homotopy problem. The 11 test problems are as follows:

1. The Watson curve with $n = 10$ [45]: $F_i(u,\lambda) = u_i - \lambda \exp[\cos(i \cdot \Sigma_{k=1}^{n} u_k)]$ for $i = 1, \cdots, n$ and $n = 10$. The path defined by $F = 0$ is traced from $\lambda = 0$ with $u = 0$ to $\lambda = 1$. Forty-six fold point singularities are encountered along the path.

2. The Watson curve with $n = 12$ [45]: This is the same as problem 1, except the number of equations is twelve (12) and 56 fold point singularities are encountered along the path.

3. A homotopy problem [36]. $F(u,\lambda) = f(u) - (1-\lambda)f(u_0)$, where $f(u) = [D_u G(u)]^T G(u)$, $G(u)^T = (10(u_2 - u_1^2), 1 - u_1, 3\sqrt{10}\ (u_4 - u_3^2), 1 - u_3, \sqrt{10}\ (u_2 + u_4 - 2), (u_2 - u_4)/\sqrt{10}\ )$, and $u_0^T = (-3,-1,-3,-1)$. The solution path is traced from $(u_0^T, 0)$ to $(1,1,1,1)^T$. Four fold points were detected for λ between 0.999 and 1.0.

4. $F_1(u,\lambda) = (u_1 - u_3)/10^4 + (u_1 - u_2)/39 + (u_1 + \lambda)/51$
   $F_2(u,\lambda) = (u_2 - u_6)/10 + (u_2 - u_1)/39 + I(u_2)$
   $F_3(u,\lambda) = (u_3 - u_1)/10^4 + (u_3 - u_4)/25.5$
   $F_4(u,\lambda) = (u_4 - u_3)/25.5 + u_4/0.62 + u_4 - u_5$
   $F_5(u,\lambda) = (u_5 - u_6)/13 + u_5 - u_4 + I(u_5)$
   $F_6(u,\lambda) = (u_6 - u_2)/10 + (u_6 - u_5)/13 + u_6 - U(u_3 - u_1)/0.201$
   $I(x) = 5.6 \cdot 10^{-8}[\exp(25x) - 1]$, $U(x) = 7.65 \arctan(1962x)$. The path defined by $F = 0$ is traced from $\lambda = 0$ to the first point with $\lambda = 1$. Two fold points were encountered [36].

5. $F(u,\lambda) = u - \lambda f(u)$ where $f_i(u) = (\Sigma u_k^3 + i)/2n$ for $i = 1, \cdots, n$ and $n = 10$. The path is traced from $\lambda = 0$ with $u = 0$ to $\lambda = 1$ [45].

6. $F(u,\lambda) = u - \lambda f(u)$ where $f_1(u) = 0.01(u_1 + u_2 + 1)^3$, $f_i(u) = 0.01(u_{i-1} + u_i + u_{i+1} + 1)^3$ for $i = 2, \cdots, n-1$, $f_n(u) = 0.01(u_{n-1} + u_n + 1)^3$, and $n = 10$. The path is traced from $\lambda = 0$ with $u = 0$ to $\lambda = 1$ [45].

7. $F(u,\lambda) = u - \lambda f(u)$ where $f_1(u) = u_1 - \prod_{j=1}^{n} u_j + 1$ and $f_i(u) = -\Sigma_{j=1}^{n} u_j + n + 1$ for $i = 2, \cdots, n$, and $n = 10$. The path is traced from $\lambda = 0$ with $u = 0$ to $\lambda = 1$ [45].

8. Same as problem 7 except that the dimension $n = 25$.

9. Same as problem 8 except that the dimension $n = 50$.

10. The Freudenstein–Roth problem [9], [49] with a regularizing homotopy:

$$f(u_1, u_2) = \begin{bmatrix} u_1 + 5u_2^2 - u_2^3 - 2u_2 - 13 \\ u_1 + u_2^2 + u_2^3 - 14u_2 - 29 \end{bmatrix} = 0,$$

is solved using the regularizing homotopy $F(u, \lambda) = \lambda f(u) + (1 - \lambda)(u - u_0)$ with $u_0 = (15, -2)^T$. The solution is $u^* = (5, 4)^T$ and two fold points are passed.

11. The Freudenstein–Roth problem [9], [49] with a Newton global homotopy:

$$f(u_1, u_2) = \begin{bmatrix} u_1 + 5u_2^2 - u_2^3 - 2u_2 - 13 \\ u_1 + u_2^2 + u_2^3 - 14u_2 - 29 \end{bmatrix} = 0,$$

is solved using the Newton global homotopy $F(u, \lambda) = f(u) - (1 - \lambda)f(u_0)$ and with $u_0 = (15, -2)^T$. In each case the solution $u^* = (5, 4)^T$ is computed and two fold points are passed.

The next objective is to describe the test conditions and strategies. All computations were performed on a CDC Cyber 180/840 computer in single precision which has a unit roundoff (machine epsilon) of approximately 7×10^{-15}. The minimum and maximum stepsize limits in all codes was set to $hmin = 10^{-7}$ and $hmax = 100$, respectively. All problems were attempted with the error tolerance for the computed solution set to 10^{-4} for each code (abserr = relerr = 10^{-4} for PITCON, arcae = arcre = ansre = 10^{-4} for HOMPACK, and Ecor = Efin = 10^{-4} for ABCON). For ABCON, predictor error tolerances were set to Eabs = Erel = 10^{-2}, the maximum number of corrector iterations to 7, and the maximum order, $Maxm$, used in the Adams–Bashforth predictors was set to 4. If a run at this error tolerance was unsuccessful, the error tolerance was divided by 10 and the problem was rerun. This was repeated until a successful run was obtained or an error tolerance of 10^{-12} led to failure. In the case of ABCON, which is the only code that allows explicit control of the error in the predictors, the predictor error tolerances were also divided by 10 after each failure. A run was considered successful if the correct final point was computed to at least three significant digits, no error flag was returned by the code, arclength was reported with less than five percent error (PITCON does not report arclength), and the proper number of fold (limit, turning) points were detected (HOMPACK does not report fold points). PITCON and HOMPACK were not penalized for numbers which they did not report and the extra work required by PITCON to compute the number of fold points was not counted in the tables below. ABCON was, however, penalized for not reporting each of these correctly and the extra work required to compute the number of fold points is included in the reported numbers. Reliable and accurate values of these quantities for each problem were obtained by running all codes on the problems using smaller and smaller error tolerances.

Since the test problems are small dense ones ranging in dimension from 2–50, several different performance measurements are presented in the table below to give some indication of the potential performance in other problem classes. These include the number of computed points (PTS) on the curve, the number of Jacobian evaluations and factorizations (NJF), the number of linear systems solved using these factorizations (NLS), the number of function evaluations (NFE), and the computed arclength (ARC) for each run. Notes are also given which indicate the tolerances used, the number of runs required to obtain success, and the reason for failure. The average counts are summarized at the end of the table; however, the homotopy problem 11 is omitted from these averages for HOMPACK, since it is not designed to solve problems where $\lambda < 0$ on a segment of the homotopy path.

TABLE
Numerical comparisons.

CODE	*PTS*	*NJF*	*NFE*	*NLS*	*ARC*	*TOL*	*#Runs*	*Notes*
Example 1 (n=10)								
ABCON Full	572	1280	1869	1869	87.38	10^{-5}	2	d
ABCON Chord	354	385	1406	1406	87.26	10^{-4}	1	
PITCON Full	282	1478	1534	1478	*	10^{-10}	7	d
PITCON Chord	218	541	1581	1475	*	10^{-5}	2	d
HOMPACK QF	333	417	3133	2672	87.05	10^{-6}	3	b, g
HOMPACK NF	258	1062	1062	2124	85.9	10^{-10}	7	b, g
HOMPACK DF	2603	5307	5311	5307	87.5	10^{-8}	5	b, g
Example 2 (n=12)								
ABCON Full	716	1593	2335	2335	108.47	10^{-5}	2	d
ABCON Chord	438	473	1696	1696	108.3	10^{-4}	1	
PITCON Full	254	1272	1387	1272	*	10^{-5}	2	c, d, g
PITCON Chord	280	677	2625	2508	*	10^{-8}	5	c, d, g
HOMPACK QF	902	949	7206	6217	108.10	10^{-9}	6	b, g
HOMPACK NF	323	1275	1275	2550	108.73	10^{-9}	6	b, g
HOMPACK DF	3172	6466	6471	6466	108.21	10^{-8}	5	b, g
Example 3 (n=4)								
ABCON Full	168	342	509	509	16.73	10^{-7}	4	d
ABCON Chord	193	195	593	593	16.73	10^{-7}	4	d
PITCON Full	FAILED AT ALL TOLERANCES						9	c, d, g
PITCON Chord	39	96	303	205	*		1	
HOMPACK QF	120	124	851	851	16.72	10^{-11}	8	d, e, g
HOMPACK NF	52	236	236	472	16.66	10^{-11}	8	d, e, g
HOMPACK DF	173	361	361	361	16.73	10^{-5}	2	b, g
Example 4 (n=6)								
ABCON Full	41	104	148	148	51.69	10^{-5}	2	d
ABCON Chord	55	60	224	224	51.71	10^{-5}	2	d
PITCON Full	16	76	80	76	*	10^{-4}	1	
PITCON Chord	28	67	305	293	*	10^{-4}	1	
HOMPACK QF	61	65	284	284	51.67	10^{-8}	5	b, f
HOMPACK NF	64	91	91	182	51.67	10^{-4}	1	
HOMPACK DF	95	203	203	203	51.83	10^{-4}	1	
Example 5 (n=10)								
ABCON Full	3	7	9	9	1.41	10^{-4}	1	
ABCON Chord	3	4	9	9	1.41	10^{-4}	1	
PITCON Full	3	11	12	11	*	10^{-4}	1	
PITCON Chord	3	6	12	11	*	10^{-4}	1	
HOMPACK QF	3	4	16	12	1.45	10^{-4}	1	
HOMPACK NF	4	12	12	24	1.45	10^{-4}	1	
HOMPACK DF	13	30	31	30	1.45	10^{-4}	1	
Example 6 (n=10)								
ABCON Full	3	6	8	8	1.0005	10^{-4}	1	
ABCON Chord	3	4	8	8	1.0005	10^{-4}	1	
PITCON Full	3	9	10	9	*	10^{-4}	1	
PITCON Chord	3	7	10	9	*	10^{-4}	1	
HOMPACK QF	3	4	14	10	1.001	10^{-4}	1	
HOMPACK NF	6	13	13	26	1.001	10^{-4}	1	
HOMPACK DF	12	28	29	28	1.001	10^{-4}	1	
Example 7 (n=10)								
ABCON Full	7	20	26	26	3.71	10^{-4}	1	
ABCON Chord	7	8	26	26	3.71	10^{-4}	1	
PITCON Full	11	50	55	50	*	10^{-4}	1	
PITCON Chord	6	15	54	51	*	10^{-4}	1	

TABLE (*cont.*)

HOMPACK QF	7	8	58	50	3.71	10^{-5}	2	c, g
HOMPACK NF	7	25	25	50	3.67	10^{-4}	1	
HOMPACK DF	26	56	57	56	3.72	10^{-4}	1	
Example 8 (n=25)								
ABCON Full	10	35	46	46	5.71	10^{-4}	1	
ABCON Chord	10	12	51	51	5.67	10^{-4}	1	
PITCON Full	15	65	70	65	*	10^{-4}	1	
PITCON Chord	12	29	67	62	*	10^{-4}	1	
HOMPACK QF	10	11	79	68	5.67	10^{-6}	3	c, g
HOMPACK NF	10	26	26	52	5.66	10^{-4}	1	
HOMPACK DF	27	61	62	61	5.69	10^{-4}	1	
Example 9 (n=50)								
ABCON Full	11	32	44	44	7.86	10^{-4}	1	
ABCON Chord	11	14	51	51	7.82	10^{-4}	1	
PITCON Full	17	71	77	71	*	10^{-4}	1	
PITCON Chord	19	46	88	80	*	10^{-4}	1	
HOMPACK QF	11	13	67	53	7.81	10^{-4}	1	
HOMPACK NF	12	27	27	54	7.76	10^{-4}	1	
HOMPACK DF	32	73	74	73	7.86	10^{-4}	1	
Example 10 (n=2)								
ABCON Full	33	95	130	130	33.94	10^{-4}	1	
ABCON Chord	44	50	186	186	32.75	10^{-4}	1	
PITCON Full	17	95	100	95	*	10^{-8}	5	d
PITCON Chord	18	43	116	109	*	10^{-4}	1	
HOMPACK QF	42	50	249	194	32.69	10^{-4}	1	
HOMPACK NF	43	78	78	156	32.67	10^{-4}	1	
HOMPACK DF	202	420	423	420	32.75	10^{-6}	3	b, g
Example 11 (n=2)								
ABCON Full	51	126	182	182	105.39	10^{-4}	1	
ABCON Chord	50	58	203	203	104.95	10^{-4}	1	
PITCON Full	13	60	64	60	*	10^{-6}	3	d
PITCON Chord	14	33	131	126	*	10^{-10}	7	c, g
HOMPACK QF								b, g, h
HOMPACK NF								b, g, h
HOMPACK DF								b, g, h

Average counts

	PTS	NJF	NFE	NLS	RUNS
ABCON Full	146.8	330.9	482.4	482.4	1.5
ABCON Chord	106.2	114.8	404.8	404.8	1.4
PITCON Full	63.1	318.7	338.9	318.7	2.9
PITCON Chord	58.2	141.8	481.1	448.1	2.0
HOMPACK QF	149.2	164.5	1195.7	1041.1	2.7
HOMPACK NF	77.9	284.5	284.5	569	2.8
HOMPACK DF	635.5	1300.5	1302.2	1300.5	2.1

* - results not reported by this code.
a - over five percent error in the computed arclength.
b - lost path, did not track all the way to $\lambda = 1$.
c - failed in final point correction.
d - did not detect proper number of limit points.
e - final solution wrong.
f - execution terminated due to overflow or an infinite result.
g - code stopped and returned error flag.
h - code is not intended for this type of problem where $\lambda < 0$ on a segment of the homotopy path.

The performance measurements presented in this table illustrate the overall efficiency of these Adams–Bashforth predictors. First, the average number of runs

required to "solve" the problem to the given tolerances may in itself be a most significant number. ABCON Chord was the most efficient with an average number of runs per problem of 1.4 followed by ABCON Full (1.5), but with PITCON Chord (2.0) and HOMPACK DF (2.1) close behind. Of the two problems on which additional runs of ABCON Chord were required, the answer was found correctly on the first run, but the code detected too many fold points. For example, in problem 3 the solution was computed efficiently with NJF = 62 and NLS = 219 on the first run as opposed to the final count of NJF = 195 and NLS = 593 on the fourth run. The version of ABCON that uses a *full* Newton method appears to be less efficient and no more robust than the *chord* version. Note that PITCON also shows this phenomenon with the difference in failure rate being even more pronounced. In ABCON the predicted point is close to the curve, resulting in only two or three corrections per step, but the second correction is generally used only to verify the accuracy of the first Newton correction in both the full and chord methods, so that the full power of Newton's method in the correction process is generally not utilized. The apparent lower efficiency of ABCON Full in comparison to ABCON Chord is also, curiously, due to larger stepsizes resulting from the more accurate tangents. These larger stepsizes caused the code to jump curves and to miss the fold points more often, resulting in additional reruns. If one considers only the examples on which ABCON Full and ABCON Chord used the same error tolerances, the full method does tend to compute fewer points and uses fewer solves. The tentative conclusion that we would like to draw from these numbers is that the Adams–Bashforth predictors, wherein the predictor error is controlled, may lead to a more robust and less sensitive method for continuation procedures.

Next, ABCON Chord appears to be more efficient in its use of Jacobian evaluations and factorizations (NJF) than the other codes. For problems in which NJF is a good measure of the efficiency, ABCON Chord is superior, using over 19 percent fewer NJF on problems 1–11 than the leading competitor, PITCON Chord. This conclusion is even more significant given that ABCON Chord computed nearly twice as many points and was more robust (1.4 versus 2.0 average runs required for success) than PITCON Chord. Thus in parameter studies of nonlinear systems wherein one uses matrix factorizations to detect singularities along the path, ABCON Chord is a method of choice. In terms of the number of function evaluations, the codes HOMPACK NF and PITCON Full required 33 percent and 13 percent fewer, respectively, but required at least twice as many runs as ABCON Chord to achieve a successful run.

We might argue that the number of linear systems solved is a good measure of the efficiency of these methods on larger-scale problems wherein one might use iterative methods. Before making this comparison, it is necessary to describe differences in matrix algebra in the various codes and to explain how we have counted NLS for each code. Both PITCON and HOMPACK QF solve an $(n+1) \times (n+1)$ system similar to (5.3) each time a corrector iterate or tangent is computed—the former by use of a PLU factorization and the latter by using a more stable but expensive QR factorization with column pivoting. In PITCON Chord and HOMPACK QF a factorization may be used in several back solves. HOMPACK NF and DF use a QR factorization on the $(n+1) \times (n)$ matrix $D_w F$ and back substitute using an $n \times n$ upper triangular block of R. To be consistent with the varying methods of computing tangents and/or iterates, we report in NLS the number of corrector iterates computed plus the number of tangents computed by each code, even though, when solving (5.3), ABCON uses the bordering algorithm which involves a PLU factorization of the $n \times n$ matrix $D_u F$ and which involves one or two back solves with this factorization. In the

case of ABCON Chord this gives the same number as the number of solves of (5.4a) plus the number of solves of (5.4b), but for ABCON Full, the two methods of counting give different results. Also, the cost of the quasi-Newton updates in HOMPACK QF, which for small to medium sized systems is comparable to the cost of a backsolve, is not counted.

For the number of linear systems solved, ABCON Chord is exceeded in efficiency by PITCON Full (21.3 percent fewer); however, this difference is slight in comparison to the large corresponding increase (278 percent) in the number of Jacobian factorizations (NJF) with double the failure rate (2.9 versus 1.4). If we restrict the comparison to those methods where a full Newton method is used in the correction process, then the winners are HOMPACK for NJF (14 percent fewer) and PITCON for NLS (33.9 percent fewer). (Such a full Newton method may be needed in the correction process for certain problem classes or specific applications.) Note that these comparisons of runs obtained at near optimal settings do not reflect the fact that the code PITCON Full (2.9) and HOMPACK NF (2.8) have nearly twice the failure rate as ABCON Full (1.5) and ABCON Chord (1.4). Thus the conclusions in this paragraph are somewhat clouded by the failure rates, by the fact that more experienced users of HOMPACK and PITCON may be able to solve a given problem with these packages without so many reruns, and the potential use of ABCON Chord even on large scale problems. (Recall that the second correction is generally used only to verify the accuracy of the first Newton correction in both ABCON Full and ABCON Chord.) However, the general trend of these failure rates support the obvious fact that following the solution path "closely" does in general produce a more robust code, which is especially important in parameter as opposed to homotopy continuation.

With regard to the number of points computed, ABCON Chord produced nearly double the number of points on the path than did the competing codes PITCON Full, PITCON Chord, and HOMPACK NF. While this may not be a direct advantage (additional points may always be computed by interpolation), it does provide an indirect advantage due to the added robustness obtained and the increased confidence we have in the details of the computed path. Although this may not be of interest in homotopy problems, the focus of HOMPACK, it is of fundamental interest in parametric studies of systems of nonlinear equations. Furthermore, we have found that the competing codes would more often jump over a portion of the curve, compute the wrong solution and arclength, or detect the wrong number of fold points without returning an error message. Thus following the path with an accurate predictor has its benefits, particularly with respect to natural parameter continuation.

Next, we note that Schwetlick and Cleve [36] report results for their code which uses Padé predictors on problems 1, 3, and 4, which give a slightly lower average NJF than those for ABCON; however, these results were not reproduced here since they were run at tolerances unknown to us and on a different machine. We were unable to determine from the published data whether the path was being properly traced by the criteria we imposed. Also, Ypma [49] has used a Hermite predictor and reported results on problem 11, but these are less efficient than those reported here for ABCON Chord.

Finally, we should comment on the vectorization of ABCON. A version of our code has been written to take advantage of the special vector hardware and FORTRAN 200 extensions resident on the CDC Cyber 205 supercomputer. A vectorized corrector which employs a PLU decomposition with scaled partial pivoting based on an extension of the ideas found in Dongarra, Gustavson, and Karp [13] was developed. The speedup obtained from these modifications of our original (compiler vectorized) code was over

40 times, and 99 percent of the speedup resulted from the explicit vectorization of the decomposition and solve routines (on problems with $n > 100$).

8. Summary and concluding remarks. Predictor-corrector methods for tracing solution paths of an underdetermined system of nonlinear equations $F(w) = 0$, where $F : \mathbb{R}^{n+1} \to \mathbb{R}^n$ have proved to be effective numerical procedures for various problems ranging from homotopy methods for nonlinear equations and fixed-point problems to parametric investigations of nonlinear equations arising in continuum mechanics, optimization, and dynamical systems. Robustness of these procedures is generally enhanced by following the solution path "closely," which can penalize efficiency. Although there has been much developed for these continuation algorithms, the use of Adams–Bashforth predictors has been recommended but not explored in a continuation procedure that employs a Newton-like correction back to the path. Thus variable order Adams–Bashforth predictors with an adaptive error-stepsize control for an arclength parameterization of the Davidenko equation or, more generally, equation (1.1) have been developed and tested in this work with the objective being to achieve both robustness and efficiency. A philosophy of this methodology, which aids considerably in the robustness, has been to follow the solution path closely wherein generally only two corrections back to the path are required. To achieve efficiency, the developed research code (ABCON) employs consecutive order Adams–Bashforth predictors to determine the stepsize by estimating the local truncation error in the lower-order formula, a comparison of three consecutive order methods to choose the order that yields the largest stepsize, and local extrapolation to improve accuracy. A key to the efficiency and robustness of these predictors, especially when combined with a Newton-like correction back to the solution path, is the approximation of arclength along the path at an accuracy commensurate with that of the tangents to the path.

The focus in this work has been on the predictor phase with an error stepsize control and not the development of a general predictor corrector-continuation algorithm. The method used in the computation of the tangent and the correction process is based on the bordering algorithm of Keller [17], which is quite general; however, the linear algebra used to solve the reduced problem is Gaussian elimination with scaled partial pivoting. Thus the test problems used for testing and comparison were small, dense but difficult ones taken from the papers of Watson, Billups, and Morgan [44], [45]; Heijer and Rheinboldt [9]; and Schwetlick and Cleve [36]. To compensate for this small class of problems and to give an indication of the potential performance in other problem classes, we have given performance measurements for several categories including the number of runs required to achieve success, the number of function evaluations, the number of linear systems solved, the number of matrix factorizations, and number of computed points on the path. ABCON has been tested and compared numerically with the state-of-the-art software packages HOMPACK (QF, NF, and DF) by Watson, Billups, and Morgan [45] and PITCON (Full and Chord) by Rheinboldt and Burkardt [31], [32]. Before summarizing these comparisons and conclusions, several caveats are in order. HOMPACK was designed for the solution of nonlinear equations by homotopy methods, particularly the regularizing homotopy, and the focus of PITCON was large sparse problems arising, for example, from practical engineering applications. Thus the performance measurements and test problems given in §7 are appropriate but not necessarily consistent with the design objectives of these latter two packages. Our objective has not been to show superiority of ABCON over HOMPACK and PITCON or to make any claim about their relative efficiency and robustness on their intended problem classes. The test set is much too small to extract

any such information. Instead, the objective has been to demonstrate the potential efficiency and robustness of the Adams–Bashforth predictors for predictor-corrector continuation methods. In the testing, HOMPACK and PITCON were not penalized for numbers they do not report; the extra work required by PITCON to compute the number of fold points was not counted (HOMPACK does not report these numbers); however, ABCON was penalized for not reporting all numbers correctly and the extra work required to determine the correct number of fold points was counted in the numbers for ABCON.

From the comparisons given in §7 and our further extensive experimentation with these variable order Adams–Bashforth predictors in the optimization setting [23], we can begin to make some general conclusions about the potential performances of these predictors. The successful use of a multistep method using more than two steps requires the stepsize in arclength along the path to be approximated with an accuracy commensurate with the accuracy required of the predictors. These higher-order Adams–Bashforth predictors, wherein a Newton-like correction back to the path is employed, require that one take relatively small and more accurate prediction steps, but this strategy can produce a more robust and efficient procedure, giving a more reliable trace of the path. As observed by Watson [45], an issue which is closely related to the latter point is that the general testing procedures used in evaluating the effectiveness of continuation procedures should account somehow for the total work done in solving a given problem, including the work involved in failed runs or runs used to determine optimal parameter settings. Indeed, ABCON performs exceedingly well in this regard, principally because the solution path is followed closely. Since the second correction is generally used only to verify the accuracy of the first full Newton correction in both ABCON Full and Chord, the full power of Newton's method is generally not useful in the correction process. Thus, in this methodology the chord method or a regular or sparse quasi-Newton method for the correction phase may be just as effective as a full Newton method on both small- and large-scale problems.

As discussed in the introduction and as reported in §7, the measurement of efficiency in these procedures is problem class dependent. If the number of matrix factorizations is the major concern, as it would be in the parametric study of nonlinear systems wherein one would encounter bifurcation and singularities, then ABCON Chord with its chord iteration appears to be more efficient than the remaining algorithms on these test problems. PITCON Chord was a close competitor and had a comparable success or failure rate. If the number of linear systems or the number of function evaluations is the gauge of efficiency, as might be appropriate for large scale systems wherein iterative methods are used for the linear systems, then one algorithm performed better than ABCON Chord, namely, PITCON Full, but it had the highest failure rate of all codes, nearly twice that of ABCON Chord and Full. Furthermore, PITCON Full used almost three times the number of matrix factorizations as ABCON Chord. PITCON Chord was, on the other hand, almost as efficient but its failure rate was slightly higher (2.0 versus 1.4) and it computed about half as many points. If we compare only those methods in which a full Newton method is used in the correction process, then the winners are ABCON Full for the number of points, HOMPACK NF for the number of Jacobian factorizations and the number of function evaluations, and PITCON Full for the number of linear systems solved; however, again ABCON Full (1.5) had nearly half the failure rate of HOMPACK NF (2.8) and PITCON Full (2.9). The only conclusion that we wish to extract from these numbers is that Adams–Bashforth predictors with the error-stepsize control developed in this work are potentially capable of adding efficiency to the robust procedure of following

the path closely, particularly for parameter continuation, and should be considered for general predictor-continuation procedures.

Another advantage of the Adams–Bashforth predictors is that, along with the predicted point on the curve as given by formula (2.5), there is the expression (2.8) for the predicted tangent, which has not been utilized in the current work since direct methods have been used for the solution of the linear systems. For problems using iterative methods, we could possibly utilize this predicted tangent to reduce the expense of the linear systems solved. Several other suggestions can be made for possible improvements in the efficiency of ABCON. One possibility is to factor the matrix only on the curve and to use the previous factored matrix in the correction process. This would give a more accurate tangent and, based on our experience, up to approximately 10 percent fewer steps along a given path. The recent theoretical work of Bourji and Walker [6] and Walker and Watson [41] and the numerical experience of Watson [43] suggest that the computation of both the corrected point and the tangent might be accomplished, at least for a limited number of steps, by using quasi-Newton updates instead of factoring the Jacobian at each step. This would certainly reduce the number of matrix factorizations, but may increase the number of linear systems solved.

Finally, Hermite predictors appear to be an attractive alternative to the Adams–Bashforth predictors used here since the orders increase by two as opposed to one per step. While two step Hermite predictors have been used effectively by various authors [36], [45], [49], our preliminary testing with variable order Hermite predictors has not been promising, possibly due to their greater sensitivity to errors in the tangents and computed arclength. However, if the use of quasi-Newton methods, as mentioned above, allows one to obtain a more accurate tangent at little extra expense, then the use of three-step (and higher) Hermite predictors may be efficient.

Acknowledgment. We wish to thank E. L. Allgower, K. Georg, L. T. Watson, and two referees for their constructive comments on an earlier version of this work. Time on a CDC Cyber 205 Supercomputer was provided by Colorado State University.

REFERENCES

[1] E. L. Allgower and K. Georg, *Predictor-corrector and simplicial methods for approximating fixed points and zero points of nonlinear mappings*, in Mathematical Programming: The State of the Art, A. Bachem, M. Grötschel, and B. Korte, eds., Springer-Verlag, New York, 1983.

[2] ———, *An Introduction to Numerical Continuation Methods*, Springer-Verlag, New York, 1990.

[3] J. H. Avila, *The feasibility of continuation methods for nonlinear equations*, SIAM J. Numer. Anal., 11 (1974), pp. 102–122.

[4] M. Berger and B. Gostiaux, *Differential Geometry: Manifolds, Curves and Surfaces*, Springer-Verlag, New York, 1988.

[5] W. E. Bosarge, *Iterative continuation and the solution of nonlinear two-point boundary value problems*, Numer. Math., 17 (1971), pp. 268–283.

[6] S. K. Bourji and H. F. Walker, *Least-change secant updates of nonsquare matrices*, Res. Report 38, Department of Mathematics, Utah State University, Logan, UT, 1987.

[7] T. Chan, *Techniques for large sparse systems arising from continuation methods*, in Numerical Methods for Bifurcation Problems, T. Kupper, H. Mittelmann, and H. Weber, eds., Birkhäuser, Boston, 1984, pp. 116–128.

[8] S.-N. Chow and J. C. Hale, *Methods of Bifurcation Theory*, Springer-Verlag, New York, 1982.

[9] C. Den Heijer and W. C. Rheinboldt, *On steplength algorithms for a class of continuation methods*, SIAM J. Numer. Anal., 18 (1981), pp. 925–948.

[10] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations,* Prentice-Hall, Englewood Cliffs, NJ, 1983.

[11] P. DEUFLHARD, B. FIEDLER, AND P. KUNKEL, *Efficient numerical path following beyond critical points,* SIAM J. Numer. Anal., 24 (1987), pp. 912–927.

[12] P. DEUFLHARD, *A stepsize control for continuation methods and its special application to multiple shooting techniques,* Numer. Math., 33 (1979), pp. 115–146.

[13] J. J. DONGARRA, F. G. GUSTAVSON AND A. KARP, *Implementing linear algebra algorithms for dense matrices on a vector pipeline machine,* SIAM Rev., 26 (1984), pp. 91–112.

[14] K. GEORG, *Numerical integration of the Davidenko equation,* Lecture Notes in Mathematics 878, Springer-Verlag, New York, 1981, pp. 17–27.

[15] ———, *A note on stepsize control for numerical curve following,* in Homotopy Methods and Global Convergence, B. C. Eaves, F. J. Gould, H.-D. Peitgen, and M. J. Todd, eds., Plenum Press, New York, 1983, pp. 145–154.

[16] H. B. KELLER, *Numerical solution of bifurcation and nonlinear eigenvalue problems,* in Applications of Bifurcation Theory, P. H. Rabinowitz, ed., Academic Press, New York, 1977, pp. 359–384.

[17] ———, *Global homotopies and Newton methods,* in Recent Advances in Numerical Analysis, C. de Boor and G. H. Golub, eds., Academic Press, New York, 1978, pp. 73–94.

[18] ———, *The Bordering algorithm and path following near singular points of higher nullity,* SIAM J. Sci. Statist. Comput., 4 (1983), pp. 573–582.

[19] ———, *Numerical Methods in Bifurcation Problems,* Springer-Verlag, Berlin, 1987.

[20] F. T. KROGH, *Algorithms for changing the step size,* SIAM J. Numer. Anal., 10 (1973), pp. 949–965.

[21] T. KUPPER, H. D. MITTELMANN, AND H. WEBER, *Numerical Methods for Bifurcation Problems,* Birkhäuser-Verlag, Boston, 1984.

[22] T. KUPPER, R. SEYDEL, AND H. TROGER, EDS., *Bifurcation: Analysis, Algorithms, Applications,* Birkhäuser-Verlag, Boston, 1987.

[23] B. N. LUNDBERG, A. B. POORE AND B. YANG, *Smooth penalty functions and continuation methods for constrained optimization,* E. L. Allgower and K. Georg, eds., Lectures in Applied Mathematics, American Mathematical Society, Providence, RI, 1990.

[24] R. MEJIA, CONKUB: *A conversational path-follower for systems of nonlinear equations,* J. Comput. Phys., 63 (1986), pp. 67–84.

[25] R. MENZEL AND H. SCHWETLICK, *Parameterization via secant length and application to path following methods,* Numer. Math., 47 (1985), pp. 401–412.

[26] H. MITTELMANN, *A pseudo-arclength continuation method for nonlinear eigenvalue problems,* SIAM J. Numer. Anal., 23 (1986), pp. 1007–1016.

[27] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables,* Academic Press, New York, 1970.

[28] A. B. POORE AND K. AL-HASSAN, *The expanded Lagrangian system for constrained optimization problems,* SIAM J. Control Optim., 26 (1988), pp. 416–427.

[29] W. C. RHEINBOLDT, *Numerical analysis of continuation methods for nonlinear structural problems,* Comput. & Structures, 13 (1981), pp. 103–113.

[30] W. C. RHEINBOLDT AND J.V. BURKARDT, *A locally parameterized continuation process,* ACM Trans. Math. Software, 9 (1983), pp. 215–235.

[31] ———, *Algorithm* 596: *A program for a locally parameterized continuation process,* ACM Trans. Math. Software, 9 (1987), pp. 236–241.

[32] W. C. RHEINBOLDT, *On the solution of some nonlinear equations arising in the application of finite element methods,* in The Mathematics of Finite Elements and Applications II: MAFELAP 1975, J. R. Whiteman, ed., Academic Press, New York, 1976, pp. 465–482.

[33] ———, *Numerical Analysis of Parameterized Nonlinear Equations,* John Wiley, New York, 1986.

[34] R. D. RUSSELL AND J. CHRISTIANSEN, *Adaptive mesh selection strategies for solving boundary value problems,* SIAM J. Numer. Anal., 15 (1978), pp. 59–80.

[35] H. SCHWETLICK, *On the choice of steplength in path following methods,* Z. Angew. Math. Mech., 64 (1984), pp. 391–396.

[36] H. SCHWETLICK AND J. CLEVE, *Higher order predictors and adaptive steplength control in path following algorithms,* SIAM J. Numer. Anal., 24 (1987), pp. 1382–1393.

[37] L. F. SHAMPINE AND J. K. GORDON, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem,* W. H. Freeman, San Francisco, CA, 1975.

[38] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis,* Springer-Verlag, New York, 1980.

[39] A. H. STROUD AND D. SECREST, *Gaussian Quadrature Formula,* Prentice-Hall, NJ, 1966.

[40] H. WACKER, E. ZARZER, AND W. ZULEHNER, *Optimal stepsize control for globalized Newton method,* in Continuation Methods, H. Wacker, ed., Academic Press, New York, 1978.

[41] H. F. WALKER AND L. T. WATSON, *Least-change secant update methods for underdetermined systems,* Res. Report 41, Department of Mathematics, Utah State University, Logan, UT, 1988.

[42] L. T. WATSON, *An algorithm that is globally convergent with probability one for a class of nonlinear two-point boundary value problems,* SIAM J. Numer. Anal., 16 (1979), pp. 394–401.

[43] ———, *Numerical linear algebra aspects of globally convergent homotopy methods,* SIAM Rev., 28 (1986), pp. 529–545.

[44] L. T. WATSON, S. C. BILLUPS, AND A. P. MORGAN, *Algorithm* 652, HOMPACK: *A suite of codes for globally convergent homotopy algorithms,* ACM Trans. Math. Software, 13 (1987), pp. 281–310.

[45] L. T. WATSON, *A globally convergent algorithm for computing fixed points of C^2 maps,* Appl. Math. Comput., 5 (1979), pp. 297–311.

[46] H. A. WATTS, *The starting stepsize for an* O.D.E. *solver,* J. Comput. Appl. Math., 7 (1983), pp. 177–191.

[47] ———, *Step size control in ordinary differential equation solvers,* Trans. Soc. Comput. Simulation, 1 (1984), pp. 15–25.

[48] ———, HSTART—*An improved initial step size routine for* ODE *Codes,* Sandia Report SAND86-2633, Sandia National Laboratories, Albuquerque, NM, 1986.

[49] T. J. YPMA, *Following paths through turning points,* BIT, 22 (1982), pp. 368–383.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 725–742, July 1991
© 1991 Society for Industrial and Applied Mathematics

FAST ALGORITHMS FOR COMPACT FIXED POINT PROBLEMS WITH INEXACT FUNCTION EVALUATIONS*

C.T. KELLEY† AND E.W. SACHS‡

Abstract. A class of fast algorithms for computation of fixed points of completely continuous maps on Banach spaces is described and analyzed. These algorithms are motivated by parabolic boundary control problems where the time integration is done by a high order backward difference formula, a variable stepsize, variable order method, or a combination of such methods. In these cases, the nonlinear maps do not have the smoothness or collective compactness properties required by known fast algorithms. This paper shows how a multilevel technique of Atkinson can be modified to attack such problems, discusses how quasi-Newton methods can improve performance, and finally, shows how this approach can be applied to parabolic boundary control problems.

Key words. collective compactness, parabolic control problems, method of lines, quasi-Newton methods

AMS(MOS) subject classifications. 45G10, 49B22, 49D15, 65H10

1. Introduction.

The purpose of this paper is to extend a multilevel method of Atkinson and Brakhage, [2], [3], for computation of fixed points of completely continuous nonlinear maps on Banach spaces to a more general class of problems in which the nonlinear map and its approximations cannot be evaluated exactly. Atkinson's method is a form of the chord, or modified Newton method, and hence its convergence can sometimes be improved by application of an appropriate quasi-Newton method. We show how algorithms developed in [19] can be used to advantage in such situations. Our work is motivated by the application of high order time integration, with stepsize and order control, or adaptive methods such as those discussed in [20], to parabolic boundary control problems. When such integration methods are used, the errors in approximations of the necessary conditions for optimality have no special smoothness or compactness properties and the fast multilevel algorithms that require such smoothness may not be applicable. As a specific example, we consider use of the code DASSL and the method of lines, [4], to such problems.

In this section we begin with a description of an abstract form of the fast algorithm for integral equations described in [2] and give a simple example. We then give enough detail of the boundary control problem to make the issues clear and describe the general structure of the types of approximate function evaluations we will consider. In § 2 we state and prove the convergence estimates and show how the cost of our new algorithm can be less than that in [2] in situations where both are applicable. Finally, in § 3 we show that a boundary control problem in one space dimension satisfies our hypotheses and report on some computations.

We now give an abstract description of a form of the method in [2] that was discussed in [17]. Consider the fixed point problem,

$$\mathcal{F}(u) = u - \mathcal{K}(u) = 0, \tag{1.1}$$

* Received by the editors August 1, 1989; accepted for publication (in revised form) April 9, 1990.
† North Carolina State University, Department of Mathematics, Box 8205, Raleigh, North Carolina 27695-8205. The research of this author was supported by National Science Foundation grants DMS-8601139, INT-8800560, and DMS-8900410, and Air Force Office of Scientific Research grants AFOSR-ISSA-890044, AFOSR-ISSA-870060, and AFOSR-89-0124.
‡ Universität Trier, FB IV—Mathematik, Postfach 3825, 5500 Trier, Federal Republic of Germany.

in a Banach space X where $\mathcal{K} : X \to X$ is completely continuous and Lipschitz continuously Fréchet differentiable. We assume that (1.1) has a solution, $u^* \in X$, and that $I - \mathcal{K}'(u^*)$ is nonsingular.

The method requires a sequence of approximate maps $\{\mathcal{K}_N\}$ on X that converges strongly to $\mathcal{K}$ in some neighborhood of u^*. In this paper we denote strong convergence of $\mathcal{K}_N$ to $\mathcal{K}$ by

$$\mathcal{K}_N \xrightarrow{s} \mathcal{K}.$$

It is also necessary that in some neighborhood $\mathcal{N}$ of u^* the sequence of Fréchet derivatives,

$$\{\mathcal{K}_N'(u)\}_{N>0,u\in\mathcal{N}},$$

be uniformly Lipschitz continuous and collectively compact. Recall, [1], that a family of operators, $\{K_\lambda\}_\lambda$, is collectively compact if there is a compact set G in X such that the image of the unit ball in X under K_λ is a subset of G for all λ.

A consequence of these assumptions, [1], is that for all $u \in \mathcal{N}$

$$(I - \mathcal{K}_N'(u))^{-1} \xrightarrow{s} (I - \mathcal{K}'(u))^{-1}. \tag{1.2}$$

Therefore, for all $u \in \mathcal{N}$,

$$(I - \mathcal{K}_N'(u))^{-1}\mathcal{K}'(u) \to (I - \mathcal{K}'(u))^{-1}\mathcal{K}'(u) = (I - \mathcal{K}'(u))^{-1} - I$$

in the operator norm. Hence, $\mathcal{F}'(u^*)$ can be accurately approximated by $\mathcal{A}$, which we define through its inverse by

$$\mathcal{A}^{-1} = I + (I - \mathcal{K}_N'(y_1))^{-1}\mathcal{K}'(y_2), \tag{1.3}$$

for any y_j, $j = 1, 2$, sufficiently near u^*. For example, y_1 could be a root of $\mathcal{F}_N = I - \mathcal{K}_N$. The underlying idea is that $(I - \mathcal{K}_N'(u))^{-1}$ can be evaluated very cheaply, say, by factoring a low-dimensional matrix and using the factorization.

In this paper we describe a variety of approximations to $\mathcal{F}'(u^*)$, all of which will be defined through their inverses and denoted by $\mathcal{A}$ with various superscripts or subscripts. Having such an approximation, we define an approximate Newton iterate by (1.1),

$$u_{n+1} = u_n - \mathcal{A}^{-1}\mathcal{F}(u_n). \tag{1.4}$$

The convergence result is that if the value of N used in (1.3) is sufficiently large and if y_1, y_2, and u_0 are sufficiently near u^*, the iterates given by (1.4) will converge q-linearly to u^*. In fact, letting $e = u - u^*$, we have

$$\|e_{n+1}\|_X \le \frac{\gamma\|\mathcal{F}'(u^*)^{-1}\|}{2}\|e_n\|_X^2 + M_F\|\mathcal{A}^{-1} - \mathcal{F}'(u^*)^{-1}\|\|e_n\|_X,$$

where γ is the Lipschitz constant for the maps

$$\{\mathcal{K}_N'(u)\}_{N>0,u\in\mathcal{N}},$$

and

$$M_F = \sup_{u\in\mathcal{N}} \|\mathcal{F}'(u)\|_X.$$

In [18] it was pointed out that quasi-Newton methods, in particular, Broyden's method, [5], could be used to obtain superlinear convergence by replacing $\mathcal{F}$ by $\mathcal{A}^{-1}\mathcal{F}$ and using the identity as the initial Fréchet derivative approximation.

We illustrate these ideas with a simple example. Consider the linear problem in $X = C[0,1]$,

$$u(x) - \int_0^1 k(x,y)u(y)\,dy - g(x) = 0. \tag{1.5}$$

In this example,

$$\mathcal{K}(u)(x) = \int_0^1 k(x,y)u(y)\,dy + g(x).$$

Note that $\mathcal{K}'$, which is independent of u, is the linear integral operator given by

$$\mathcal{K}'(u)(v)(x) = \int_0^1 k(x,y)v(y)\,dy.$$

We assume that $g \in C[0,1]$ and that $k \in C^2([0,1]\times[0,1])$. In [2] the trapezoid rule was used to discretize the integrals in this problem. Here we use a different approximation scheme that will be important in the following sections.

We let Q_N be the map that takes a continuous function u to the piecewise linear function that agrees with u at the points $\{i/N\}_{i=0}^N$; we express this as

$$(Q_N u)(x) = \sum_{i=0}^N u(i/N)\phi_i(x),$$

where ϕ_i is the piecewise linear function that has the value 1 at i/N and 0 at $\{j/N\}_{j=0,j\neq i}^N$. Define

$$\mathcal{K}_N(u) = Q_N\mathcal{K}(Q_N u).$$

Note that if $u_N - \mathcal{K}_N(u_N) = 0$, then u_N is piecewise linear with nodes at the points $\{i/N\}_{i=0}^N$. If we set $u_N^i = u_N(i/N)$, then the vector $\{u_N^i\}_{i=0}^N$ satisfies

$$u_N^i - \sum_{j=0}^N \kappa_{ij}u_N^j - g(i/N) = 0 \quad \text{for } i = 0,\cdots,N, \tag{1.6}$$

where

$$\kappa_{ij} = \int_0^1 k(i/N,y)\phi_j(y)\,dy \quad \text{for } i,j = 0,\cdots,N.$$

If $I - \mathcal{K}'$ is nonsingular, $I - \mathcal{K}_N'$ will also be nonsingular for N sufficiently large. Moreover, the operators $\mathcal{K}_N'$ satisfy the hypotheses mentioned above for $\mathcal{A}^{-1}$ to be a good approximation to $(I - \mathcal{K}')^{-1}$.

To compute $\mathcal{A}^{-1}v$ for $v \in X$, we evaluate $w = Q_N\mathcal{K}'v$ and then $z = (I - \mathcal{K}_N')^{-1}w$. To compute z from w, let B be the $(N+1)\times(N+1)$ matrix with entries

$$B_{ij} = \delta_{ij} - \kappa_{ij}, \quad i,j = 0,\cdots,N.$$

Let P_N be the map that takes a function in $v \in C[0,1]$ to the vector with components $v(i/N)$, and let Π_N be the map that takes a vector with components $\bar{v}_i$ into the piecewise linear function v that satisfies $v(i/N) = \bar{v}_i$ for $i = 0, \cdots, N$. Then, since $w \in \mathrm{Ran}(Q_N)$,

$$z = \Pi_N B^{-1} P_N w,$$

and, if $\mathcal{A}^{-1}$ is given by (1.3),

$$\mathcal{A}^{-1} = I + \Pi_N B^{-1} P_N \mathcal{K}' + (I - Q_N)\mathcal{K}'.$$

However, as $\mathcal{K}'$ is compact and $Q_N \xrightarrow{s} I$, the second term on the right side of the above equation goes to 0 in the operator norm. We may therefore use the simpler approximation,

$$\mathcal{A}^{-1} = I + \Pi_N B^{-1} P_N \mathcal{K}'. \tag{1.7}$$

The key points in the above specification of $\mathcal{A}^{-1}$ are the facts that $\{\mathcal{K}'_N\}$ is a collectively compact sequence of operators, that $(I - \mathcal{K}'_N)^{-1}$ exists near u^*, and that its matrix representation can be easily computed. In order to verify the hypotheses of collective compactness and strong convergence, the explicit representation of the kernel, k, and its smoothness are important. For such problems, similar assumptions are used in the convergence theorems for second kind multigrid methods, [15]. Though in some cases these assumptions can be verified without explicit evaluation of the kernel, [15], [13], such verification is quite technical and can be done only in special cases.

This paper considers problems where $\{\mathcal{K}_N\}$ can be evaluated only inexactly, with errors that have no special compactness or smoothness properties. Our motivating example is a boundary control problem for a parabolic partial differential equation.

For given functions $z, y_0 \in C[0,1]$ and $g \in L^\infty(R)$, we seek a $u \in L^\infty[0,T]$ that minimizes

$$f(u) = \frac{1}{2}\int_0^1 (y(u;T,x) - z(x))^2\,dx + \frac{\alpha}{2}\int_0^T u^2(t)\,dt, \tag{1.8}$$

where $\alpha > 0$ is given and $y(t,x) = y(u;t,x)$ is the solution to the nonlinear parabolic problem

$$\begin{aligned} y_t(t,x) &= y_{xx}(t,x), \quad 0 < x < 1, \quad 0 < t < T; \\ y(0,x) &= y_0(x), \quad 0 < x < 1; \\ y_x(t,0) &= 0, \quad y_x(t,1) = g(y(t,1)) + u(t), \quad 0 < t < T. \end{aligned} \tag{1.9}$$

The nonlinear function g is assumed to satisfy

$$g \in C^2(R), \qquad g', g'' \in L^\infty(R). \tag{1.10}$$

The gradient of f, in the L^2 sense, is given by

$$(\nabla f(u))(t) = \alpha u(t) + d(t,1), \tag{1.11}$$

where $d(t,x)$ is the solution of the backwards problem

$$\begin{aligned} -d_t(t,x) &= d_{xx}(t,x), \quad 0 < x < 1, \quad 0 < t < T; \\ d(T,x) &= y(T,x) - z(x), \quad 0 < x < 1; \\ d_x(t,0) &= 0, \quad d_x(t,1) = g'(y(t,1))d(t,1), \quad 0 < t < T. \end{aligned} \tag{1.12}$$

We may therefore write the necessary conditions for optimality as

$$u - \mathcal{K}(u) = 0, \tag{1.13}$$

where $\mathcal{K}$ is the map that takes u to $-d(t,1)/\alpha$. It is known, [21], that $\mathcal{K}$ is a compact map on $C[0,T]$. In [14] it was shown that for the standard Galerkin finite-element discretization and continuous time or certain time discretizations, such as backward Euler or Crank–Nicolson, the sequence $\{\mathcal{K}_N\}$ could be constructed to satisfy the hypotheses needed for either a multigrid approach or the approach in [2]. In the computations reported in § 3 of this paper, however, we approximate $\mathcal{K}u$ by applying a code for differential algebraic equations, DASSL, [4], for the time integration, together with a standard finite-element discretization in space. Such an approach is typically more efficient than fixed-step/fixed-order schemes, such as the Crank–Nicolson or backward Euler methods. The disadvantage from a theoretical perspective is that such approximations to $\mathcal{K}$ have errors with no special smoothness or compactness properties, and hence the theory for algorithms like that discussed above for (1.5) is not directly applicable. In this paper we develop a theory to allow for these more efficient time discretizations.

Our solution of (1.1) is via a sequence of approximate maps $\mathcal{K}_N$, which will be constructed using a sequence of finite-rank projections $\{Q_N\}$ on X. For notational consistency we will set $\mathcal{K} = \mathcal{K}_\infty$ and $I = Q_\infty$. We assume that the projections satisfy

$$\begin{aligned} Q_N^2 &= Q_N, \quad Q_N \xrightarrow{s} I, \quad \mathrm{Ran}(Q_N) = X_N, \\ \mathrm{Dim}(X_N) &= D_N < \infty, \quad \text{for } N < \infty. \end{aligned} \tag{1.14}$$

We assume that there is a neighborhood $\mathcal{N}$ of u^* such that

(1.15)
(a) $\mathcal{K}_N(u) = Q_N\mathcal{K}(Q_N u) \xrightarrow{s} \mathcal{K}(u)$ for all $u \in \mathcal{N}$,
(b) the family $\{\mathcal{K}'_N\}_{N\le\infty}$ is uniformly Lipschitz continuous in $\mathcal{N}$, with Lipschitz constant γ, and
(c) the family $\{\mathcal{K}'_N(u)\}_{N\le\infty,u\in\mathcal{N}}$ is collectively compact.

Note that (1.15) (a) and (1.15) (b) imply

$$\mathcal{K}'_N(u) \xrightarrow{s} \mathcal{K}'(u) \quad \text{for all } u \in \mathcal{N}. \tag{1.16}$$

Another important consequence of (1.15) that we will need, which might require that the radius of $\mathcal{N}$ be reduced, is that there is M_I such that

$$\|\mathcal{F}'_N(u)^{-1}\| \le M_I, \tag{1.17}$$

for all $u \in \mathcal{N}$ and N sufficiently large. Assumption (1.15) implies (1.17) if the radius of $\mathcal{N}$ is sufficiently small.

We use the notation $\mathcal{K}_\infty = \mathcal{K}$. The sequence $\{\mathcal{K}_N\}_{N\le\infty}$ is itself approximated by the family $\{\mathcal{K}_{N,\epsilon}\}$, which satisfies

$$\begin{aligned} \mathcal{K}_{N,\epsilon}(u) &= Q_N\mathcal{K}_{N,\epsilon}(Q_N u), \\ \|\mathcal{K}_{N,\epsilon}(u) - \mathcal{K}_N(u)\|_X &\le \epsilon, \quad \text{for all } u \in X. \end{aligned} \tag{1.18}$$

The absolute error condition in (1.18) is typical in ode packages such as DASSL. DASSL allows a combination of relative and absolute errors to be controlled. In this

paper the focus is on a root-finding problem and our termination criterion will be a nonlinear residual with a small norm. Hence, we use an absolute error condition in DASSL. The motivating idea behind (1.18) is to admit situations in which $\mathcal{K}$ is approximated by an algorithm that allows the accuracy to be controlled, but in a more adaptive way than a simple fixed stepsize approximation of the type discussed for (1.5). Aside from the method of lines technique of the examples in this paper, adaptive methods such as those discussed in [20] or schemes based on fast evaluation of heat potentials, [10], could be used. We want to use these more general and efficient ways to approximate $\mathcal{K}$ but still use the fast methods that need the collective compactness and smoothness assumptions.

In § 2 we discuss our algorithm and prove an error estimate. We briefly discuss quasi-Newton acceleration and show that the ideas in [19] can be useful. In § 3 we present a numerical example.

As a final remark in this section, we discuss our choice of Broyden's method. In the context of the parabolic control problem, the map $I - \mathcal{K}'(u^*)$ is selfadjoint and positive definite in $L^2(0,T)$, [12]. The BFGS method, [6], [8], [9], [24], which has similar convergence properties to Broyden's method for the types of problems we consider, preserves this structure. However, the approximation $\mathcal{A}^{-1}$, given in (1.7), is not necessarily selfadjoint and positive definite in $L^2(0,T)$, even if the matrix B is. An approximation that would provide the selfadjoint and positive definite structure is

$$\mathcal{A}^{-1} = I + \mathcal{K}' + \mathcal{K}'\Pi_N B^{-1} P_N \mathcal{K}'. \tag{1.19}$$

In [17] we discuss the cost of this type of approximation, which is essentially that of an extra function evaluation at the fine grid level. We cannot replace the last term on the right-hand side of (1.19) by

$$\mathcal{K}'_N \Pi_N B^{-1} P_N \mathcal{K}'$$

without violating symmetry, and we cannot use

$$\mathcal{K}'_N \Pi_N B^{-1} P_N \mathcal{K}'_N$$

and preserve norm convergence. It is for this reason that we prefer Broyden's method to the BFGS method.

2. Local convergence. In this section we describe and analyze an algorithm based on (1.7) that allows for inaccuracy in the function evaluations. We begin with the definitions of the approximations to the derivatives that we use. Then we prove two lemmas that relate the inaccuracy in the function evaluation to that in the approximate derivatives. This leads to the main theorem on the convergence properties of the approximations to $\mathcal{F}'(u^*)^{-1}$ that we use in the approximate form of (1.7). The final theorem discusses how termination criteria are related to the inaccuracy in the function and derivative evaluation.

Our first task in this section is to obtain a good approximation of $\mathcal{K}'(u)$ from $\mathcal{K}_{N,\epsilon}(u)$. To this end, let $\delta(\epsilon)$ be such that

$$\lim_{\epsilon \to 0} \delta(\epsilon) = \lim_{\epsilon \to 0} \frac{\epsilon}{\delta(\epsilon)} = 0.$$

Let $\epsilon_N \to 0$ as $N \to \infty$ and define, for $v \neq 0$,

$$\tilde{\mathcal{K}}'_N(u)v = \|v\|_X \frac{\mathcal{K}_{N,\epsilon_N}(u + \delta(\epsilon_N)v/\|v\|_X) - \mathcal{K}_{N,\epsilon_N}(u)}{\delta(\epsilon_N)}. \tag{2.1}$$

Define $\tilde{\mathcal{K}}'_N(u)(0) = 0$. Then (1.15) and (1.18) imply that for all $u \in \mathcal{N}, v \in X$, and ϵ_N sufficiently small,

$$\|\tilde{\mathcal{K}}'_N(u)v - \mathcal{K}'_N(u)v\|_X \leq \sigma(\epsilon_N)\|v\|_X, \tag{2.2}$$

where

$$\sigma(\epsilon) = \max\{2, \gamma/2\}(\delta(\epsilon) + \epsilon/\delta(\epsilon)).$$

Recall that in (2.2) γ is the Lipschitz constant for all elements of the sequence $\{\mathcal{K}'_N\}$, including, of course, $\mathcal{K}'_\infty = \mathcal{K}'$, as specified in (1.15).

We will also have need to approximate $\mathcal{K}'$ itself. To do this, we define

$$\hat{\mathcal{K}}'_M(u)v = \|v\|_X \frac{\mathcal{K}_{\infty,\epsilon_M}(u + \delta(\epsilon_M)v/\|v\|_X) - \mathcal{K}_{\infty,\epsilon_M}(u)}{\delta(\epsilon_M)} \tag{2.3}$$

for $v \neq 0$ and set $\hat{\mathcal{K}}'_M(u)(0) = 0$. As was the case with $\tilde{\mathcal{K}}'_N$ for $N < \infty$, we have

$$\|\hat{\mathcal{K}}'_M(u)v - \mathcal{K}'(u)v\|_X \leq \sigma(\epsilon_M)\|v\|_X. \tag{2.4}$$

As we have chosen to define $\tilde{\mathcal{K}}'_N$ with forward differences, an obvious choice of $\delta(\epsilon)$ is

$$\delta(\epsilon) = O(\sqrt{\epsilon}), \tag{2.5}$$

as $\epsilon \to 0$. Centered differences, which would cost more in terms of function evaluations, would require that the second derivatives of $\mathcal{K}_N$ be uniformly Lipschitz and have the effect of replacing the first $\delta(\epsilon)$ term in (2.2) by $\delta(\epsilon)^2$. This would make the obvious choice $\delta(\epsilon) = O(\epsilon^{1/3})$.

Let Q_N satisfy (1.14). In our algorithm $I - \tilde{\mathcal{K}}'_N$ will be used to construct a linear map that approximates $I - \mathcal{K}'_N$ on X_N. Then this approximation will be used to approximate $\mathcal{F}'(u^*)$ on finer grids via a formula like (1.3), as in the example discussed in § 1. The derivative approximation will be done by evaluation of the nonlinear operator $I - \tilde{\mathcal{K}}'_N$ on a basis for X_N to construct a linear map on the finite-dimensional space X_N. Let $\{\xi_i^N\}_{i=1}^{D_N}$ be a basis of unit vectors for X_N and define a linear map $B_N(u)$ on X_N by its action on the basis elements

$$B_N(u)(\xi_i^N) = \xi_i^N - \tilde{\mathcal{K}}'_N(u)\xi_i^N. \tag{2.6}$$

We let $\beta_N \geq 1$ be such that

$$\sum_{j=1}^{D_N} |x_j| \leq \beta_N \| \sum_{j=1}^{D_N} x_j \xi_j^N \|_X \tag{2.7}$$

for all sequences $\{x_j\}_{j=1}^{D_N}$. In the example in § 1, $\beta_N = D_N$.

For linear maps defined on X_N we denote the operator norm induced by $\|\cdot\|_{X_N} = \|\cdot\|_X$ by $\|\cdot\|_{X_N}$ as well. This notation is intended to enable us to compare $B_N(u)$, which is defined only on X_N, to $\mathcal{F}'_N(u)$, which is defined on all of X. It also allows us to consider their inverses. Note that if $A \in \mathcal{L}(X)$ is such that $AX_N \subset X_N$, then $\|A\|_{X_N} \leq \|A\|_X$.

LEMMA 2.1. *Let* (1.15), (1.14), *and* (1.18) *hold. Assume that* (2.2) *holds for all* ϵ_N. *Then for all* N *and* $u \in \mathcal{N}$,

$$\|B_N(u) - \mathcal{F}'_N(u)\|_{X_N} \leq \beta_N \sigma(\epsilon_N). \tag{2.8}$$

Moreover, if

$$M_I \beta_N \sigma(\epsilon_N) \leq \tfrac{1}{2}, \tag{2.9}$$

and (1.17) *holds, then* $B_N(u)^{-1}$ *exists and there is* C_I*, independent of* N *such that for all* $u \in \mathcal{N}$

$$\|B_N(u)^{-1} - \mathcal{F}'_N(u)^{-1}\|_{X_N} \leq C_I \beta_N \sigma(\epsilon_N). \tag{2.10}$$

Proof. The first assertion, (2.8), is an immediate consequence of (2.2), (2.7), and the definition of $B_N(u)$. Let $x = \sum_{j=1}^{D_N} x_i \xi_i^N \in X_N$. Note that

$$\begin{aligned} \|B_N(u)x - \mathcal{F}'_N(u)x\|_{X_N} &\leq \textstyle\sum_{j=1}^{D_N} \|(\tilde{\mathcal{K}}'_N(u) - \mathcal{K}'_N(u))\,\xi_j^N\|_X |x_j| \\ &\leq \sigma(\epsilon_N) \textstyle\sum_{j=1}^{D_N} |x_j| \leq \beta_N \sigma(\epsilon_N) \|x\|_X. \end{aligned}$$

This proves (2.8).

By (1.17) and (2.9),

$$\begin{aligned} \|B_N(u)^{-1} - \mathcal{F}'_N(u)^{-1}\|_{X_N} &\leq \frac{\|\mathcal{F}'_N(u)^{-1}\|_{X_N}^2 \beta_N \sigma(\epsilon_N)}{(1 - \|\mathcal{F}'_N(u)^{-1}\|_{X_N} \beta_N \sigma(\epsilon_N))} \\ &\leq \frac{M_I^2 \beta_N \sigma(\epsilon_N)}{1 - M_I \beta_N \sigma(\epsilon_N)} \leq 2 M_I^2 \beta_N \sigma(\epsilon_N). \end{aligned} \tag{2.11}$$

To complete the proof, we let $C_I = 2M_I^2$. □

If T is any linear map on X_N, we extend T to all of X by defining $\mathrm{Ran}(T) = TX_N \subset X_N$ and

$$Tx = TQ_N x.$$

With this understanding, our assumptions on $\mathcal{K}_N$ immediately imply the following result.

LEMMA 2.2. *Let the assumptions of Lemma* 2.1 *hold and assume that*

$$\lim_{N \to \infty} \beta_N \sigma(\epsilon_N) = 0. \tag{2.12}$$

Let $B_N(u)$ *be given by* (2.6)*. Then for all* $u \in \mathcal{N}$ $B_N^{-1}(u) \xrightarrow{s} (I - \mathcal{K}'(u))^{-1}$.

A natural approximation to $\mathcal{F}'(u)^{-1}$ is

$$\mathcal{A}_N^{-1} = I + B_N^{-1}(y_1)\mathcal{K}'(y_2). \tag{2.13}$$

However, in a practical situation we would use

$$\mathcal{A}_{M,N}^{-1} = I + B_N^{-1}(y_1)\hat{\mathcal{K}}'_M(y_2), \tag{2.14}$$

M may be different from N. Note that $\mathcal{A}_{M,N}^{-1}$ is a nonlinear map. Our theory, therefore, can not make assertions about convergence of $\mathcal{A}_{M,N}$, as did the work in [2], [18], [17]. We obtain estimates only on $\mathcal{A}_{M,N}^{-1}$. These estimates will show that $\mathcal{A}_{M,N}^{-1}$ is a small perturbation of $\mathcal{F}'(u^*)^{-1}$ and from that fact the type of estimates in [2], [18],

[17] would follow. We need only the results on $\mathcal{A}_{M,N}^{-1}$ in this work. Equation (2.13) is the special case $M = \infty$ and $\epsilon_M = 0$ of (2.14).

The main result of this section is presented in the following theorem.

THEOREM 2.3. *Let the hypotheses of Lemma* 2.2 *hold and assume that* (2.9) *holds for all* N. *Let* $\mathcal{A}_{M,N}^{-1}$ *be given by* (2.14) *with* $y_1, y_2 \in \mathcal{N}$. *Then there is* $C_D > 0$ *and a sequence* $\{\mu_{M,N}\}$ *with*

$$\lim_{M,N\to\infty} \mu_{M,N} = 0,$$

such that for all $v \in X$, *and* N *such that* (1.17) *holds*

$$\|(\mathcal{A}_{M,N}^{-1} - \mathcal{F}'(u^*)^{-1})v\|_X \le (\mu_{M,N} + C_D \sum_{i=1}^{2} \|y_i - u^*\|_X)\|v\|_X.$$

Proof. Let $\epsilon_N \le \bar{\epsilon}$ for all N. This implies that there is $\bar{\sigma}$ such that $\sigma(\epsilon_N) \le \bar{\sigma}$. Let N be large enough so that (1.17) holds. By Lemma 2.1

$$\|B_N(u)^{-1} - \mathcal{F}_N'(u)^{-1}\|_{X_N} \le C_I \beta_N \sigma(\epsilon_N),$$

for all $u \in \mathcal{N}$. Assumption (1.15) implies that $\mathcal{F}_N'(u)^{-1}$ is Lipschitz continuous in $\mathcal{N}$, with Lipschitz constant γ_Z independent of N. In fact, $\gamma_Z \le M_I^2 \gamma$ by (1.15) and (1.17). We have $\|\mathcal{F}_N'(u)^{-1} - \mathcal{F}_N'(u^*)^{-1}\|_X \le \gamma_Z \|u - u^*\|$ and so,

$$\|B_N(u)^{-1} - \mathcal{F}_N'(u^*)^{-1}\|_{X_N} \le C_I \beta_N \sigma(\epsilon_N) + \gamma_Z \|u - u^*\|. \tag{2.15}$$

For all $v \in X$

$$\|\widehat{\mathcal{K}}_M'(u)v - \mathcal{K}'(u)v\|_X \le \sigma(\epsilon_M)\|v\|_X$$

and therefore,

$$\|\widehat{\mathcal{K}}_M'(u)v - \mathcal{K}'(u^*)v\|_X \le (\sigma(\epsilon_M) + \gamma\|u - u^*\|_X)\|v\|_X. \tag{2.16}$$

Letting

$$\zeta = \bar{\sigma} + \|\mathcal{K}'(u^*)\|_X + \gamma \operatorname{diam}(\mathcal{N}),$$

we have

$$\|\widehat{\mathcal{K}}_M'(u)v\|_X \le \zeta \|v\|_X. \tag{2.17}$$

At this point we may use a combination of (2.15), (1.17), (2.16), and (2.17) to obtain

$$\|\mathcal{A}_{M,N}^{-1}v - \mathcal{B}_N^{-1}v\| \le (M_I\sigma(\epsilon_M) + \zeta C_I \beta_N \sigma(\epsilon_N) + C_D \sum_{i=1}^{2} \|y_i - u^*\|_X)\|v\|_X, \tag{2.18}$$

where

$$\mathcal{B}_N^{-1} = I + \mathcal{F}_N'(u^*)^{-1}\mathcal{K}'(u^*), \tag{2.19}$$

and $C_D = M_I\gamma + \zeta\gamma_Z$.

Since the sequence $\{\mathcal{K}'_N(u^*)\}$ is collectively compact and $\mathcal{K}'_N(u^*) \xrightarrow{s} \mathcal{K}'(u^*)$, we have, [1],

$$\mathcal{F}'_N(u^*)^{-1} \xrightarrow{s} \mathcal{F}'(u^*)^{-1}.$$

Therefore, by compactness of $\mathcal{K}'(u^*)$,

$$\mathcal{F}'_N(u^*)^{-1}\mathcal{K}'(u^*) \to \mathcal{F}'(u^*)^{-1} - I$$

in the norm topology of $\mathcal{L}(X)$ as $N \to \infty$. So,

$$\lim_{N\to\infty} \mathcal{B}_N^{-1} = \mathcal{F}'(u^*)^{-1}$$

in the operator norm sense. Hence if we set

$$\mu_{M,N} = \|\mathcal{B}_N^{-1} - \mathcal{F}'(u^*)^{-1}\|_X + M_I\sigma(\epsilon_M) + \zeta C_I\beta_N\sigma(\epsilon_N)$$

the proof will be complete. □

Condition (2.9) is important to allow us to make any estimate of the accuracy of the approximation $\mathcal{A}_{M,N}v$ to $\mathcal{F}'(u^*)v$. In the case of the example in § 1, where $\beta_N = D_N$, $D_N = N+1$ and $\sigma(\epsilon_N) = O(\sqrt{\epsilon_N})$, (2.12) indicates one would have to use $\epsilon_N = o(h_N^2)$, where $h_N = 1/N$. This is necessary to guarantee that $\mathcal{A}_{M,N}$ is a good approximation to $\mathcal{F}'(u^*)$.

The special case where $\mathcal{A}_N$ is given by (2.13), i. e. , $\epsilon_M = 0$ and $M = \infty$ in (2.14), is important in its own right. In this case the conclusion becomes a norm estimate because $\mathcal{A}_N$ is a linear map.

COROLLARY 2.4. *Let the assumptions of Theorem* 2.3 *and* $\mathcal{A}_N^{-1}$ *be given by* (2.13). *Then there is* $C_D > 0$ *and a sequence* $\{\mu_N\}$ *with* $\lim_{N\to\infty} \mu_N = 0$ *such that for all* N *such that* (1.17) *holds*

$$\|\mathcal{A}_N^{-1} - \mathcal{F}'(u^*)^{-1}\|_X \le \mu_N + C_D \sum_{i=1}^{2} \|y_i - u^*\|_X.$$

Proof. The proof is the same as that of Theorem 2.3 with

$$\mu_N = \|\mathcal{B}_N^{-1} - \mathcal{F}(u^*)^{-1}\|_X + \zeta C_I\beta_N\sigma(\epsilon_N)$$

and

$$C_D = M_I\gamma + \zeta\gamma_Z,$$

as one would have in the proof of Theorem 2.3 with $M = \infty$. In this special case, however, we can take $\zeta = \|\mathcal{K}'(u^*)\|_X + \gamma\,\mathrm{diam}(\mathcal{N})$. □

As the function $\mathcal{F}$ can only be evaluated approximately, our iteration scheme and the criterion for its termination will be approximated as well. Ideally we would terminate the iteration when

$$\|\mathcal{F}(u)\|_X < \tau$$

for some small τ. We give results for the criterion

$$\|\mathcal{F}_\epsilon(u)\|_X < \tau, \tag{2.20}$$

where

$$\mathcal{F}_\epsilon(u) = u - \mathcal{K}_{\infty,\epsilon}(u),$$

and ϵ depends on τ. We point out that in practice, of course, $\mathcal{F}_\epsilon$ would be given by

$$\mathcal{F}_\epsilon(u) = u - \mathcal{K}_{N_f,\epsilon}(u),$$

for some large N_f, which would be the highest level of approximation considered. The termination level τ would depend on N_f, as we describe below. In the examples in § 3 we explicitly describe the choices of ϵ and N_f in the examples. The theorems are stated in terms of the limiting situation, $N_f = \infty$, for simplicity.

Given a current iterate u_c the new iterate u_+ for the method in this paper is

$$u_+ = u_c - \mathcal{A}_{M,N}^{-1}\mathcal{F}_\epsilon(u_c), \tag{2.21}$$

where ϵ would depend on the termination tolerance τ and the value of τ would be a constant multiple of an approximation for $\tau_f = \|\mathcal{F}_{N_f}(u^*)\|_X$ or some other estimate of truncation error. At this point we remark that any differences introduced in the theory by setting $N_f = \infty$ are of the order of τ_f and hence negligible. Our final results in this section will be on the interpretation of (2.20) and the relation of the sequence generated by (2.21) to the Newton sequence for $\mathcal{F}$.

Note that the nondifferentiable error E makes it impossible to apply the Kantorovich theorem, [16], to conclude that $\mathcal{F}_\epsilon$ has a root in $\mathcal{N}$. To see what the issue is, note that

$$\|\mathcal{F}_\epsilon(u^*)\| \le \tau_f + \epsilon.$$

To make (2.20) hold with $u = u^*$, we would need ϵ sufficiently small and

$$\tau_f + \epsilon < \tau.$$

However, the criterion (2.20) is a good approximation to

$$\|\mathcal{F}(u)\|_X \le \tau \tag{2.22}$$

since

$$\left|\|\mathcal{F}_\epsilon(u)\|_X - \|\mathcal{F}(u)\|_X\right| \le \epsilon.$$

The final theorem in this section states that $\|u_i - u^*\|_X$, and hence $\|\mathcal{F}(u_i)\|_X$ can be driven to $O(\tau)$ if M and N are sufficiently large and y_1 and y_2 sufficiently near u^*.

The final result in this section uses the estimates on accuracy of function and derivative evaluations to describe the convergence behavior of the iterates. This behavior was called "local improvement" rather than convergence in [7], as the sequence $\{u_n\}$ can only be guaranteed to converge to u^* if arithmetic is exact and sufficient relative accuracy in the function evaluations is maintained throughout the iteration. Results like Theorem 2.5 can be found in [25] as well as in [7], where effects of inaccuracy in function and Jacobian evaluation for problems in finite dimension have been considered. The proofs in those papers, together with the previous results in this section, lead to results in our situation that assert, like Theorem 2.5, that the sequence $\{u_n\}$ eventually lies in a neighborhood of u^* whose size is determined by the

accuracy in the function evaluation. Our result expresses this behavior in terms of the estimates in our assumptions. We express the convergence in terms of $e = u - u^*$.

THEOREM 2.5. *Let the hypotheses of Lemma 2.2 hold. Let $\omega < 1$ be given. If M and N are sufficiently large, u_c, y_1, and y_2 sufficiently near u^*, and u_+ is given by (2.21), then there is C_C such that*

$$\|e_+\|_X \le \omega\|e_c\|_X + C_C\epsilon. \tag{2.23}$$

Proof. Clearly

$$\begin{aligned}\|e_+\|_X &= \|e_c - \mathcal{A}_{M,N}^{-1}\mathcal{F}_\epsilon(u_c)\|_X \\ &\le \|e_c - \mathcal{F}'(u^*)^{-1}\mathcal{F}(u_c)\|_X + \|(\mathcal{A}_{M,N}^{-1} - \mathcal{F}'(u^*)^{-1})\mathcal{F}_\epsilon(u_c)\|_X \\ &\quad +\|\mathcal{F}'(u^*)^{-1}(\mathcal{K}_{\infty,\epsilon}(u_c) - \mathcal{K}(u_c))\|_X.\end{aligned}$$

Hence, using standard results for Newton's method, Theorem 2.3, and (1.18), we find

$$\begin{aligned}\|e_+\|_X &\le \gamma\|\mathcal{F}'(u^*)^{-1}\|\,\|e_c\|_X^2/2 \\ &\quad +(\mu_{M,N} + C_D\textstyle\sum_{i=1}^2\|y_i - u^*\|_X)(\|\mathcal{F}(u_c)\|_X + \epsilon) + \|\mathcal{F}'(u^*)^{-1}\|\epsilon.\end{aligned}$$

Recall that we let

$$M_F = \sup_{u\in\mathcal{N}}\|\mathcal{F}'(u)\|_X,$$

and hence

$$\|\mathcal{F}(u_c)\|_X \le M_F\|e_c\|_X.$$

So, if we let e_c, y_1, and y_2 be near enough to u^* and M and N be sufficiently large so that

$$M_F(\mu_{M,N} + C_D\sum_{i=1}^2\|y_i - u^*\|_X) + \frac{\gamma\|\mathcal{F}'(u^*)^{-1}\|\,\|e_c\|_X}{2} < \omega,$$

and let

$$C_C = \|\mathcal{F}'(u^*)^{-1}\|_X + \mu_{M,N} + C_D\sum_{i=1}^2\|y_i - u^*\|_X,$$

the proof is complete. □

We remark that the results in this section remain valid if we replace $\widehat{\mathcal{K}}'_M(u)v$ with

$$\|v\|_X\frac{\mathcal{K}_{\infty,\epsilon_M}(u + \delta(\epsilon_M)v/\|v\|_X) - \mathcal{K}_{\infty,\epsilon}(u)}{\delta(\epsilon_M)}.$$

In view of the fact that in the iteration the evaluation of $\mathcal{K}(u)$ must be done anyway, this would provide a savings of a single function evaluation at accuracy level M, a small benefit. In any case, the cost of a single evaluation of $\mathcal{A}_{M,N}^{-1}\mathcal{F}$ is little more than that of an evaluation of $\mathcal{F}$ alone. The performance of the iteration can be improved by use of the quasi-Newton and the nested iteration approach suggested in [18]. We discuss these implementation issues in § 3. Also note that even if $\mathcal{K}_N$ can be evaluated exactly, the algorithm described in this paper has an advantage over that given in [2] and [3] if $\mathcal{K}_{N,\epsilon}$ becomes more expensive to evaluate as $\epsilon \to 0$.

3. Example. As a test problem, we consider maps $\mathcal{K}$ defined by

$$\mathcal{K}(u) = -d(t,1)/\alpha,$$

where for $x \in (0,1)$ and $t \in (0,T)$,

$$\begin{array}{rclcrcl} -d_{tt} & = & d_{xx} + f_2(t,x), & & d(T,x) & = & y(T,x) - z(x), \\ d_x(t,0) & = & h_l(t), & & d_x(t,1) & = & h_r(t), \\ y_{tt} & = & y_{xx} + f_1(t,x), & & y(0,x) & = & y_0(x), \\ y_x(t,0) & = & g_l(t), & & y_x(t,1) & = & g_r(y(t,1)) + q(t) + u(t). \end{array} \tag{3.1}$$

We adjusted the right-hand sides and boundary values so that a particular solution, u^*, could be specified. Discretization in space was by piecewise linear finite elements. The functions $\mathcal{K}_{N,\epsilon_N}$ were evaluated by using DASSL to solve the systems of ordinary differential equations obtained by discretizing (3.1) in space and setting the absolute error to ϵ_N.

In the particular example reported here, $g_r(y) = \tan^{-1}(y)$, $T = 1$, $\alpha = .5$,

$$y(t,x) = \sin(xt - x^2)e^{xt} \quad \text{and}$$

$$d(t,x) = 1 + x + \sin(xt) + e^{x-t}.$$

In the computations that follow we let $X = C[0,1]$ and used piecewise linear approximations. The maps $\{Q_N\}$ were computed as in § 1 and therefore $D_N = N+1$. We let $\delta(\epsilon_N) = 2\sqrt{\epsilon_N}$. In our runs, $\epsilon_N = N^{-3}$. The reasoning behind this choice is to make the derivative approximation on the coarse grid accurate without too much cost. The choice $\epsilon_N = N^{-4}$ is consistent with the overall accuracy of the spatial discretization, but we found that with this choice the evaluation of $\mathcal{A}_{M,N}^{-1}\mathcal{F}$ was too costly. Larger values of ϵ_N produced slower convergence. Hence (2.12) and the assumptions of Lemma 2.1 hold for N sufficiently large. We expect our accuracy to be second order in $h = 1/N$. To reflect this we set $\tau = 20h^2$ in (2.20). We made the arbitrary choice, $\epsilon = .02\tau$. Larger choices of ϵ sometimes made the termination criterion difficult to satisfy. For the computations reported in Tables 3.1, 3.2, and 3.3, we set $M = N = 20$ in the computation of $\mathcal{A}_{M,N}^{-1}$. Solution at the coarsest level was done by a variation of the Shamanskii method [23]. Here the Jacobian was recomputed if the reduction in the norm of the function was larger than .5 or more than eight iterates had been taken with the same Jacobian. That iteration was terminated when the norm of $\mathcal{F}_{N,\epsilon_N}$ was below $\tau_N = 20h_N^2$. The result of that iteration was used to compute the matrix $B_N(u)$. The initial iterate on the coarse level was $Q_2 u_0$, where

$$u_0(t) = u^*(t) + .1t\sin(t).$$

There are two levels in our computation. The low level, corresponding to $M = N = 20$ and the fine grid level, corresponding to $\mathcal{F}$; we let N_f denote the latter level. Under various variations on the method described by (2.21), and for each level and iterate u_i, we tabulate $\|\mathcal{F}_{N_f}(u_i)\|_X$, $\|Q_{N_f}u^* - u_i\|_X$, and the ratios

$$R_i = \|Q_{N_f}u^* - u_i\|_X / \|Q_{N_f}u^* - u_{i-1}\|_X$$

for $i > 0$. We also give the time in seconds for the iteration at each level. The computations were done on an Alliant FX/40 with two processors using Alliant Concentrix 5.0.0 and Alliant Fortran 4.2.40. The timings indicate that use of DASSL for the time integration is more efficient than the second order Crank–Nicolson method, as

we would expect. To add some perspective to the timings, we remark that 52 seconds were spent on computations on the coarsest level; these computations were dominated by the single Jacobian evaluation required by the Shamanskii method to satisfy the termination criterion on that grid and a second Jacobian evaluation to compute $B_N(u)$ for use in the following grids. We also observed that a single computation of B_N for $N = 40$ required 138 seconds.

For Crank–Nicolson, one would expect the cost for an evaluation of $\mathcal{K}_{N_f}$ to increase at a rate proportional to N_f^2 as N_f gets large; here we see much slower growth. We let y_1 be the solution we accepted at the coarsest grid and let $y_2 = Q_N u_c$ at each iterate. Note that the theory allows y_1 and y_2 to vary with the iterate and the mesh level.

In Table 3.1 we use the iteration scheme in (2.21) for levels $N_f = 40$, 80, 160, 320, and 640. For each level we use as the initial iterate the solution at $N = 20$. At each level, the timings include the final evaluation of $\mathcal{A}^{-1}\mathcal{F}$, which is used only to check the function evaluation. The ratios R_i indicate a rapid rate of linear convergence.

TABLE 3.1
Basic iteration, $T = 1$, $\alpha = .5$.

N_f	i	$\|\mathcal{F}_{N_f}(u_i)\|_X$	$\|Q_{N_f}u^* - u_i\|_X$	R_i	Time
40	0	0.12D-01	0.80D-02		
	1	0.16D-02	0.14D-02	0.1748	6.1
80	0	0.13D-01	0.80D-02		
	1	0.16D-02	0.14D-02	0.1711	13.0
160	0	0.14D-01	0.80D-02		
	1	0.15D-02	0.22D-02	0.2715	
	2	0.58D-03	0.50D-03	0.2322	41.9
320	0	0.14D-01	0.80D-02		
	1	0.18D-02	0.26D-02	0.3271	
	2	0.65D-03	0.71D-03	0.2704	
	3	0.25D-03	0.20D-03	0.2765	
	4	0.81D-04	0.75D-04	0.3833	159.4
640	0	0.14D-01	0.80D-02		
	1	0.18D-02	0.26D-02	0.3286	
	2	0.63D-03	0.72D-03	0.2747	
	3	0.22D-03	0.24D-03	0.3382	
	4	0.77D-04	0.73D-04	0.2972	
	5	0.30D-04	0.23D-04	0.3174	466.5

In Table 3.2 we apply Broyden's method to the function $\mathcal{G} = \mathcal{A}_{M,N}^{-1}\mathcal{F}$. The initial iterate was the solution from the coarse mesh $N = 20$ and the identity was used as the initial approximation to the Fréchet derivative. Recall that Broyden's method updates a current approximation B_c to

$$\mathcal{G}'(u^*) = \mathcal{A}_{M,N}^{-1}\mathcal{F}'(u^*)$$

by the formula

$$B_+ = B_c + \frac{(y - B_c s) \otimes s}{\|s\|_2^2},$$

where $\otimes$ denotes the outer product in $L^2(0,T)$ and $y = \mathcal{G}(u_+) - \mathcal{G}(u_c)$. We implement this update by a pair of vectors for each Broyden iterate and using the Sherman–Morrison–Woodbury formula to recover $s_+ = -B_+^{-1}\mathcal{G}(u_+)$ by storing an

approximation,

$$H_n = B_n^{-1}$$

as a product

$$H_n = (I + u_n v_n^T) H_{n-1},$$

where $H_0 = I$. Our approach is that of [19], where the history of the iteration is stored in the piecewise linear functions $\{u_n\}$ and $\{v_n\}$. Convergence theory, [11], [22], for Broyden's method in infinite dimension indicates that superlinear convergence in $L^2(0,T)$ should be expected as $\mathcal{G}'$ differs from the identity by a small, compact error.

TABLE 3.2
Broyden's method, $T = 1$, $\alpha = .5$.

N_f	i	$\|\mathcal{F}_{N_f}(u_i)\|_X$	$\|Q_{N_f}u^* - u_i\|_X$	R_i	Time
40	0	0.12D-01	0.80D-02		
	1	0.16D-02	0.14D-02	0.1748	6.1
80	0	0.13D-01	0.80D-02		
	1	0.16D-02	0.14D-02	0.1711	13.0
160	0	0.14D-01	0.80D-02		
	1	0.15D-02	0.22D-02	0.2715	
	2	0.58D-03	0.49D-03	0.2274	42.0
320	0	0.14D-01	0.80D-02		
	1	0.18D-02	0.26D-02	0.3271	
	2	0.65D-03	0.71D-03	0.2706	
	3	0.11D-03	0.12D-03	0.1717	132.0
640	0	0.14D-01	0.80D-02		
	1	0.18D-02	0.26D-02	0.3286	
	2	0.64D-03	0.72D-03	0.2736	
	3	0.36D-04	0.39D-04	0.0544	333.3

Local q-superlinear convergence of Broyden's method in the Hilbert space setting would lead us to expect that the ratios R_i would tend to zero, even though they measure convergence in the L^∞ sense. In fact, the ratios become smaller as i increases, although not in a monotone way, while those for the basic method do not seem to be tending to zero. However, the improvement in performance caused by use of Broyden's method is not as dramatic as that reported in [18] and, in fact, is only important at all when N_f is large. The reason for this is that in this example the solution is smooth and therefore the initial iterates are better approximations to the solution and Fréchet derivative than was the case for the problem considered in [18].

In Table 3.3 we use the iteration scheme in (2.21) for values $N_f = 10 * 2^k$, $k = 2, \cdots, 6$. At each level the initial iterate is a linear interpolation of the accepted solution at the previous level. The timings are roughly double that actually needed for the final result. Note that the timings are not cumulative. Two function evaluations only are needed, the last being used only to check the termination criterion. For this particular example, therefore, a quasi-Newton method is not needed as $\mathcal{A}_{M,N}^{-1}$ is such a good approximation to $\mathcal{F}'(u^*)^{-1}$.

A more interesting use of Broyden's method is as a part of a nested iteration for a more difficult problem. In Tables 3.4 and 3.5 we report results for the problem given by (3.1) with $M = N = 5$. The idea is to examine situations where a larger value of N would make B_N too expensive to compute or impossible to store in core. Here

TABLE 3.3
Nested iteration, $T = 1$, $\alpha = .5$.

N_f	i	$\|\mathcal{F}_{N_f}(u_i)\|_X$	$\|Q_{N_f}u^* - u_i\|_X$	R_i	Time
40	0	0.12D-01	0.80D-02		
	1	0.16D-02	0.14D-02	0.1748	6.1
80	0	0.21D-02	0.14D-02		
	1	0.39D-03	0.50D-03	0.3564	11.7
160	0	0.80D-03	0.58D-03		
	1	0.26D-03	0.21D-03	0.3578	25.6
320	0	0.28D-03	0.21D-03		
	1	0.92D-04	0.89D-04	0.4324	56.4
640	0	0.83D-04	0.89D-04		
	1	0.31D-04	0.23D-04	0.2627	138.5

we consider the nested iteration scheme as used to generate Table 3.3. The choices $M = N < 5$ either failed to converge or converged in an erratic fashion. For this example, more than one iterate is often required at each mesh level. Table 3.4 shows the progress of the iteration without Broyden's method.

TABLE 3.4
Nested iteration, $M = N = 5$, $T = 1$, $\alpha = .5$

N_f	i	$\|\mathcal{F}_{N_f}(u_i)\|_X$	$\|Q_{N_f}u^* - u_i\|_X$	R_i	Time
10	0	0.34D+00	0.29D+00		
	1	0.64D-01	0.34D-01	0.1167	1.7
20	0	0.64D-01	0.34D-01		
	1	0.22D-01	0.16D-01	0.4715	2.9
40	0	0.27D-01	0.16D-01		
	1	0.13D-01	0.65D-02	0.4077	
	2	0.67D-02	0.35D-02	0.5408	8.0
80	0	0.70D-02	0.38D-02		
	1	0.30D-02	0.18D-02	0.4853	11.0
160	0	0.35D-02	0.19D-02		
	1	0.17D-02	0.10D-02	0.5328	
	2	0.98D-03	0.52D-03	0.5097	
	3	0.49D-03	0.27D-03	0.5208	47.4
320	0	0.54D-03	0.29D-03		
	1	0.22D-03	0.12D-03	0.3982	
	2	0.11D-03	0.54D-04	0.4614	79.1
640	0	0.96D-04	0.54D-04		
	1	0.49D-04	0.24D-04	0.4436	
	2	0.24D-04	0.12D-04	0.5196	191.4

Table 3.5 uses Broyden's method as in Table 3.2, but we do not reinitialize H when the grid is refined. For example, at the iterate at level 40, we have stored two pairs of vectors from the iterates at levels 10 and 20. In this way the history of the iteration is preserved by storage of a few vectors. Note that the number of iterates is reduced at all the finer levels, giving a significant savings. In this way use of Broyden's method gives improved performance in those cases where, for storage reasons, M and N cannot be taken large enough to allow for convergence like that in Table 3.3. Savings also come from the cost of computing B_N, which is reduced from 52 seconds with $M = N = 20$ to 5 seconds with $M = N = 5$.

TABLE 3.5
Nested iteration with Broyden's method, $M = N = 5$, $T = 1$, $\alpha = .5$.

N_f	i	$\|\mathcal{F}_{N_f}(u_i)\|_X$	$\|Q_{N_f}u^* - u_i\|_X$	R_i	Time
10	0	0.34D+00	0.29D+00		
	1	0.64D-01	0.34D-01	0.1167	1.8
20	0	0.64D-01	0.34D-01		
	1	0.22D-01	0.15D-01	0.4532	2.9
40	0	0.25D-01	0.15D-01		
	1	0.50D-02	0.34D-02	0.2195	5.4
80	0	0.52D-02	0.34D-02		
	1	0.32D-03	0.47D-03	0.1391	11.2
160	0	0.70D-03	0.56D-03		
	1	0.96D-04	0.73D-04	0.1321	24.0
320	0	0.11D-03	0.79D-04		
	1	0.11D-03	0.77D-04	0.9652	53.1
640	0	0.12D-03	0.77D-04		
	1	0.32D-04	0.21D-04	0.2798	127.9

Acknowledgments. The authors would like to thank Steve Campbell, Bill McKinney, Linda Petzold, Clement Ullrich, and Dong-Ming Hwang for many conversations and much help on the method of lines and use of DASSL. The authors also appreciate the careful efforts of the referees.

REFERENCES

[1] P. M. ANSELONE, *Collectively Compact Operator Approximation Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1971.
[2] K. E. ATKINSON, *Iterative variants of the Nyström method for the numerical solution of integral equations*, Numer. Math., 22 (1973), pp. 17–31.
[3] H. BRAKHAGE, *Über die numerische Behandlung von Integralgleichungen nach der Quadraturformelmethode*, Numer. Math., 2 (1960), pp. 183–196.
[4] K. E. BRENAN, S. L. CAMPBELL, AND L. R. PETZOLD, *The Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, Elsevier, New York, 1989.
[5] C. G. BROYDEN, *A class of methods for solving simultaneous equations*, Math. Comp., 19 (1965), pp. 577–593.
[6] ———, *A new double-rank minimization algorithm*, AMS Notices, 16 (1969), p. 670.
[7] J. E. DENNIS AND H. F. WALKER, *Inaccuracy in quasi-Newton methods: Local improvement theorems*, in Mathematical Programming Study 22: Mathematical Programming at Oberwolfach II, North–Holland, Amsterdam, 1984, pp. 70–85.
[8] R. FLETCHER, *A new approach to variable metric methods*, Comput. J., 13 (1970), pp. 317–322.
[9] D. GOLDFARB, *A family of variable metric methods derived by variational means*, Math. Comp., 24 (1970), pp. 23–26.
[10] L. GREENGARD AND J. STRAIN, *A fast algorithm for the evaluation of heat potentials*, Tech. Report YALEU/DCS/RR-700, Dept. of Computer Science, Yale University, New Haven, CT, 1989.
[11] A. GRIEWANK, *The local convergence of Broyden-like methods on Lipschitzian problems in Hilbert space*, SIAM J. Numer. Anal., 24 (1987), pp. 684–705.
[12] W. A. GRUVER AND E. SACHS, *Algorithmic Methods In Optimal Control*, Pitman, London, 1980.
[13] W. HACKBUSCH, *On the fast solving of parabolic boundary control problems*, SIAM J. Control Optim., 17 (1979), pp. 231–244.
[14] ———, *Optimal $H^{p,p/2}$ error estimates for a parabolic Galerkin method*, SIAM J. Numer. Anal., 18 (1981), pp. 681–692.
[15] ———, *Multigrid methods of the second kind*, in Multigrid Methods for Integral and Differential Equations, Oxford University Press, Oxford, 1985.
[16] L. KANTOROVICH AND G. AKILOV, *Functional Analysis in Normed Spaces*, Pergamon Press, New York, 1964.

[17] C. T. KELLEY, *A fast two-grid method for matrix H-equations*, Transport. Theory Statist. Phys., 18 (1989), pp. 185–204.
[18] ———, *Operator prolongation methods for nonlinear equations*, in Computational Solution of Nonlinear Systems of Equations, E. L. Allgower and K. Georg, eds., Lectures in Applied Mathematics Vol. 26, American Mathematical Society, Providence, RI, 1990, pp. 359–388.
[19] C. T. KELLEY AND E. W. SACHS, *Approximate quasi-Newton methods*, Math. Programming, Ser. B, 48 (1990), pp. 41–70.
[20] P. K. MOORE AND J. E. FLAHERTY, *A local refinement finite element method for one-dimensional parabolic systems*, Tech. Report 88-13, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, 1988.
[21] E. SACHS, *A parabolic control problem with a boundary condition of the Stefan–Boltzmann type*, ZAMM, 58 (1978), pp. 443–449.
[22] ———, *Broyden's method in Hilbert space*, Math. Programming, 35 (1986), pp. 71–82.
[23] V. E. SHAMANSKII, *A modification of Newton's method*, Ukran. Mat. Zh., 19 (1967), pp. 133–138.
[24] D. F. SHANNO, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp., 24 (1970), pp. 647–657.
[25] T. J. YPMA, *The effect of rounding errors on Newton-like methods*, IMA J. Numer. Anal., 3 (1983), pp. 109–118.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 743–754, July 1991
© 1991 Society for Industrial and Applied Mathematics
002

COMPUTING EIGENVALUES OF BANDED SYMMETRIC TOEPLITZ MATRICES*

PETER ARBENZ†

Abstract. A recently proposed algorithm for the computation of the eigenvalues of symmetric banded Toeplitz matrices is investigated.

The basic idea of the algorithm is to embed the Toeplitz matrix in a symmetric circulant matrix of higher order. After having computed the spectral decomposition of the circulant matrix—which is trivial since the eigenvectors of circulants are known a priori and the eigenvalues can be obtained by Fourier transform—the Toeplitz eigenvalue problem is treated as a restricted eigenvalue problem. In doing this, use can be made of the theory for intermediate problems of Weinstein and Aronszajn and its recent refinements.

Since the main part of the proposed algorithm consists of independent searches for zeros in disjoint real intervals, the algorithm is well suited for parallel computers. Implementations of the algorithm are discussed and some numerical results are given. The (sequential) complexity of the computation of s eigenvalues of the Toeplitz matrix T is $O(sr(n+r^2))$, where n is the order and r the bandwidth of T.

Key words. Toeplitz matrix, circulant matrix, restricted eigenvalue problem, parallel computer

AMS(MOS) subject classifications. 15A18, 65F15, 65W05

1. Introduction.

In this note we pursue an idea that was presented by Golub and Arbenz [3] to compute eigenvalues and eigenvectors of a *symmetric banded* Toeplitz matrix. A symmetric banded Toeplitz matrix with bandwidth r is a matrix of the form

$$T = \begin{pmatrix} t_0 & \cdots & t_r & & \\ \vdots & \ddots & & \ddots & \\ t_r & & \ddots & & t_r \\ & \ddots & & \ddots & \vdots \\ & & t_r & \cdots & t_0 \end{pmatrix} \in \mathbb{R}^{n\times n}, \tag{1.1}$$

i.e., a matrix with elements $t_{ij} = t_{|i-j|}$, where $t = (t_0, \ldots, t_{n-1})^*$ is a vector with vanishing components $t_{r+1}, \ldots, t_{n-1}$. The asterisk * denotes Hermitian transposition. $\lambda \in \mathbb{R}$ is called an eigenvalue of T if there is a nonzero vector $x \in \mathbb{R}^n$ so that

$$Tx = \lambda x. \tag{1.2}$$

Such a vector is called an eigenvector of T corresponding to λ.

Considerable effort has been invested in finding fast $O(n^2)$ [20, §5.7] and superfast $O(n \log^2 n)$ [1] algorithms to solve linear systems of equations $Tx = y$ with symmetric positive definite Toeplitz matrices T. Such systems play an important role in such areas as signal processing, linear filtering, linear prediction, and time series analysis. Iterative methods to solve $Tx = y$ have also been investigated [12], [13].

In contrast to the linear systems of equations, the symmetric Toeplitz eigenvalue problem has not found as much attention. Problem (1.2), however with in general full matrices, plays an important role in Pisarenko's approach to model stationary

* Received by the editors June 26, 1989; accepted for publication (in revised form) May 14, 1990.
† Eidgenössiche Technische Hochschule Zentrum, Institut für Wissenschaftliches Rechnen, 8092 Zürich, Switzerland (arbenz@inf.ethz.ch).

stochastic processes. In this method, the components of the eigenvector corresponding to the smallest eigenvalue of the Toeplitz autocorrelation matrix of the process is used to determine the frequencies of the sinusoidal waves which make up the incoming signal. The problem of determining the "smallest" eigenpair of T has been investigated by Cybenko and van Loan [14] and Hu and Kung [21]. To its solution these authors propose algorithms which combine inverse vector iteration and Rayleigh quotient iteration, respectively, with fast methods for solving $(T - \lambda I)x = y$. In the first-mentioned paper this equation is even a Yule–Walker equation. Trench [28] extended the algorithm of Cybenko and van Loan by an eigenvalue counting formula in order to be able to safely compute interior eigenvalues as well.

Banded symmetric Toeplitz eigenvalue problems seem to have attracted only numerical analysts so far. They occur, e.g., in eigenvalue problems for finite difference matrices originating from one-dimensional differential equations with constant coefficients, or when the optimal polynomial is to be determined for the solution of an *indefinite* banded Toeplitz system of equations [13] by the polynomial preconditioned conjugate gradient method [4]. The tridiagonal Toeplitz eigenvalue problem as the simplest nontrivial example is easy to solve as then T is a linear polynomial in

$$H = \begin{pmatrix} 0 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 0 \end{pmatrix}.$$

Five-diagonal (nonsymmetric) Toeplitz matrices have been investigated by Marr and Vineyard [23]. Trench [27], as well as Bini and Pan [9], developed algorithms for the computation of eigenvalues of Toeplitz matrices with wider band. Their methods are based on algorithms to compute the determinant $f(\lambda) := \det(T - \lambda I)$ and its derivative $f'(\lambda)$ in $O(r^2 \log n + r^3)$ flops or in $O(r \log r \log n + r^3)$ flops if the Fast Fourier Transform is applicable.

Bini and Capovani [8] noted that a banded Toeplitz matrix T can be embedded in a polynomial in H, where H generally has larger order than T. This observation led them to a further fast algorithm for computing $\det(T - \lambda I)$.

Here we propose to embed T in a circulant matrix. To that end let

$$P := 0_r \oplus I_n = \operatorname{diag}(0_r, I_n) \in \mathbb{R}^{m \times m}, \qquad m = n + r, \tag{1.3}$$

where I_n and 0_r are the identity and zero matrices of order n and r, respectively.

In [3] it was noted that the eigenvalue problem (1.2) is equivalent to

$$PCPy = \lambda y, \qquad Py = y, \tag{1.4}$$

where

$$C = \begin{pmatrix} t_0 & \cdots & t_r & & & t_r & \cdots & t_1 \\ \vdots & \ddots & & \ddots & & & \ddots & \vdots \\ t_r & & \ddots & & \ddots & & & t_r \\ & \ddots & & \ddots & & \ddots & & \\ & & \ddots & & \ddots & & \ddots & \\ t_r & & & \ddots & & \ddots & & t_r \\ \vdots & \ddots & & & \ddots & & \ddots & \vdots \\ t_1 & \cdots & t_r & & & t_r & \cdots & t_0 \end{pmatrix} \in \mathbb{R}^{m\times m} \tag{1.5}$$

is a *symmetric circulant* matrix. Note that $PCP = 0_r \oplus T$. Problem (1.4) is a *restricted* eigenvalue problem, as considered in detail in [2] and [6]. The extension of Toeplitz matrices to circulant ones is classic. Its use for banded matrices has been proposed by Jain [22] to solve the linear equation $Tx = y$.

In §2 we deal with the eigenvalue problem of circulant matrices which can be solved very quickly by the Fourier transform. In §3 we review the theory of restricted eigenvalue problems of the form (1.4). We will show how the eigenvalues and eigenvectors of a restricted eigenvalue problem can be computed from those of the unconstrained eigenvalue problem $Cx = \lambda x$. In §4 we apply the results obtained in §§2 and 3 to problem (1.4)–(1.5) and in the final section, §5, we report on numerical experiments.

2. Circulant matrices. Let

$$C = \operatorname{circ}(c) := \begin{pmatrix} c_0 & c_1 & \cdots & c_{m-2} & c_{m-1} \\ c_{m-1} & c_0 & \cdots & c_{m-3} & c_{m-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_2 & c_3 & \cdots & c_0 & c_1 \\ c_1 & c_2 & \cdots & c_{m-1} & c_0 \end{pmatrix} \in \mathbb{R}^{m\times m}, \tag{2.1}$$

be a circulant matrix. $c \in \mathbb{R}^m$ is the column vector with elements $c_0, \ldots, c_{m-1}$. Then

$$C = \sum_{j=0}^{m-1} c_j \Pi^j, \tag{2.2}$$

where Π is the circulant matrix

$$\Pi = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 1 & & & 0 \end{pmatrix}.$$

The spectral decomposition of Π is known to be [15, p. 72]

$$\Pi = F^* \Omega F, \tag{2.3}$$

where

$$\Omega = \operatorname{diag}(1, \omega, \omega^2, \ldots, \omega^{m-1}), \quad \omega = e^{2\pi i/m}, \quad i = \sqrt{-1},$$

and

$$F = (f_{kj}) := \frac{1}{\sqrt{m}}(\bar{\omega}^{kj}), \qquad 0 \le k, j \le m-1.$$

$\bar{\omega}$ denotes the complex conjugate of ω. The symmetric unitary matrix F is the so-called *Fourier matrix* [15, p. 31]. From (2.2) and (2.3) we get

$$C = F^*\Lambda F, \quad \Lambda = \operatorname{diag}(\lambda_0, \ldots, \lambda_{m-1}) = \sum_{j=0}^{m-1} c_j \Omega^j, \tag{2.4}$$

i.e., the eigenvalues λ_k of C are the values of the polynomial $p(z) := \sum_{j=0}^{m-1} c_j z^j$ in the points $\omega^k, k = 0, \ldots, m-1$. The vector l with elements $\lambda_0, \ldots, \lambda_{m-1}$ is thus given by

$$l = p(\Omega) = \sqrt{m}\, F^* c. \tag{2.5}$$

Therefore, computing the eigenvalues of a circulant matrix merely amounts to performing an inverse Fourier transform (harmonic synthesis), which can, if m is a power of two, be done by the Fast Fourier Transform in $O(m \log m)$ flops. A flop is the computational cost for a floating point add *and* a floating point multiply [20, p. 32].

Let us now assume that C is *real symmetric*, i.e., that $c \in \mathbb{R}^m$ and $c_k = c_{m-k}$ for $k = 1, \ldots, m-1$. Then the eigenvalues of C are real and it is easy to verify that

$$\lambda_k = \lambda_{m-k}, \qquad k = 1, \ldots, m-1. \tag{2.6}$$

At most λ_0 and $\lambda_{m/2}$ for even m can be simple eigenvalues! The eigenvectors F^*e_k and F^*e_{m-k} corresponding to λ_k are elementwise complex conjugate such that their real and imaginary parts form real orthogonal eigenvectors for λ_k. From (2.5) we easily deduce a real representation for λ_k and even m :

$$\lambda_k = c_0 + 2 \sum_{j=1}^{m/2-1} c_j \cos 2\pi kj/m + (-1)^k c_{m/2}. \tag{2.7}$$

A similar expression holds for odd m.

3. Restricted eigenvalue problems. In this section we recall the theory of restricted eigenvalue problems of the form (1.4)

$$PAPx = \lambda x, \qquad Px = x, \tag{3.1}$$

as it was treated in [2]. Here $A \in \mathbb{R}^{n\times n}$ is a symmetric matrix and $P \in \mathbb{R}^{n\times n}$ is an orthogonal projector. Let $V \in \mathbb{R}^{n\times r}$ be a matrix with orthonormal columns spanning $\mathcal{R}(P)^\perp$. Then $P = I - VV^*$ and (3.1) is equivalent to

$$(A - \lambda I)x = Va, \qquad V^*x = 0, \tag{3.2}$$

for some $a \in \mathbb{R}^r$. So λ is an eigenvalue of (3.1) if and only if the matrix

$$\begin{pmatrix} \lambda I - A & V \\ V^* & 0 \end{pmatrix} \in \mathbb{R}^{m\times m}, \qquad m = n + r, \tag{3.3}$$

is singular.

Provided that λ is not an eigenvalue of A, the block LDL^T decomposition [20, p. 84] of the 2×2 block matrix in (3.3) yields [2]

$$\det Z(\lambda) := \det V^*(A - \lambda I)^{-1}V = \det \begin{pmatrix} \lambda I - A & V \\ V^* & 0 \end{pmatrix} / \det(A - \lambda I). \tag{3.4}$$

Therefore, the eigenvalues of the restricted eigenvalue problem (3.1) which are *not* at the same time eigenvalues of A can be determined by finding the roots of the equation

$$\det Z(\lambda) = 0. \tag{3.5}$$

Often, (3.5) is referred to as the *secular equation* [19], [20]. Mainly, in the literature on the methods of intermediate problems, $\det V^*(A - \lambda I)^{-1}V$ is called the *Weinstein determinant* [29], [30]. In structural engineering this determinant is known as *Kron's determinant* [25].

If $\det Z(\lambda)$ vanishes, the eigenspace of (3.1) corresponding to λ can be constructed from the nullspace of $Z(\lambda)$. Furthermore, not only the determinant but also the *inertia* of $Z(\lambda)$ gives information about the location of the eigenvalues of (3.1). We resume the results in the following theorem, which is valid for any $\lambda \in \mathbb{R}$.

THEOREM 3.1. *Let $\lambda \in \mathbb{R}$. Let $\mu \geq 0$ be the multiplicity of λ as an eigenvalue of A and let $W \in \mathbb{R}^{n \times \mu}$ be a matrix, the columns of which form an orthonormal basis of the eigenspace $\mathcal{N}(A - \lambda I)$ corresponding to λ. Let $\nu(\lambda)$ be the number of negative eigenvalues of the extended Weinstein matrix*

$$Z_+(\lambda) := \begin{pmatrix} V^*(A - \lambda I)^+V & V^*W \\ W^*V & 0 \end{pmatrix} \in \mathbb{R}^{(r+\mu)\times(r+\mu)}. \tag{3.6}$$

If $N(\lambda)$ and $\tilde{N}(\lambda)$ denote the numbers of the eigenvalues of A and of the restricted eigenvalue problem (3.1)*, respectively, which are less than or equal to λ, then*

$$\tilde{N}(\lambda) + \nu(\lambda) = N(\lambda). \tag{3.7}$$

The mapping defined by

$$\begin{pmatrix} a \\ z \end{pmatrix} \longmapsto x := (A - \lambda I)^+Va + Wz \tag{3.8}$$

bijectively maps the nullspace of $Z_+(\lambda)$ onto the eigenspace of the restricted eigenvalue problem corresponding to λ. □

$Z_+(\lambda)$ is a generalization of $Z(\lambda)$: If λ is in the resolvent set of A, we have $\mu = 0$ and the Moore–Penrose generalized inverse $(A - \lambda I)^+$ of $A - \lambda I$ coincides with $(A - \lambda I)^{-1}$. In this case (3.8) takes on the simple form

$$a \longmapsto x := (A - \lambda I)^{-1}Va.$$

The extended Weinstein matrix is the limit of the matrix

$$\begin{pmatrix} V^*Q^\perp(A - \xi I)^{-1}Q^\perp V & V^*W \\ W^*V & (\xi - \lambda)I_\mu \end{pmatrix}, \qquad \xi \notin \sigma(A), \tag{3.9}$$

for ξ tending to λ. Here $Q^\perp = I - WW^*$ is the orthogonal projector on $\mathcal{R}(W)^\perp = \mathcal{N}(A - \lambda I)^\perp$. The matrix in (3.9) is constructed in such a way that only the components of the columns of V in $\mathcal{R}(W)^\perp$ are multiplied by $(A - \xi I)^{-1}$ while the components in $\mathcal{R}(W)$ remain essentially unchanged. In the unmodified Weinstein matrix

these components are multiplied by $(\xi-\lambda)^{-1}$, which causes a pole in the Weinstein determinant at λ. So it is not surprising that the matrix (3.9) is congruent with

$$V^*(A-\xi I)^{-1}V \oplus (\xi-\lambda)I_\mu, \qquad \xi \notin \sigma(A), \tag{3.10}$$

with respect to a unit triangular matrix [2]. Its determinant is therefore *bounded* in a neighbourhood of the eigenvalue λ and the relation

$$\lim_{\xi\to\lambda}(\xi-\lambda)^\mu \det Z(\xi) = \det Z_+(\lambda) \tag{3.11}$$

holds. Similarly, more than one pole can be smoothed out from the Weinstein determinant [2].

Denoting the nondecreasingly ordered eigenvalues of A and of (3.1), respectively, by $\lambda_0,\dots,\lambda_{m-1}$ and $\tilde\lambda_0,\dots,\tilde\lambda_{n-1}$, we can easily get from (3.7) the interlacing property

$$\lambda_k \le \tilde\lambda_k \le \lambda_{k+r}, \qquad 0 \le k < n. \tag{3.12}$$

4. The banded symmetric Toeplitz eigenvalue problem. In this section we apply the theory of the two previous sections on the restricted eigenvalue problem (1.4)–(1.5) and formulate an algorithm for the computation of a few of its smallest eigenvalues.

The orthogonal projector P in (1.3) can be written as $P = I - VV^*$ with

$$V := \begin{pmatrix} I_r \\ 0 \end{pmatrix} \in \mathbb{R}^{m\times r}.$$

With the easily obtainable spectral decomposition (2.4) of C, the Weinstein matrix in (3.6) becomes

$$\begin{pmatrix} V^*(C-\lambda I)^+V & V^*W \\ W^*V & 0 \end{pmatrix} = \begin{pmatrix} U^*(\Lambda-\lambda I)^+U & V^*W \\ W^*V & 0 \end{pmatrix}, \qquad U = FV. \tag{4.1}$$

The matrix $U^* = (FV)^* \in \mathbb{R}^{r\times m}$ consists of the first r components of the eigenvectors of C while $V^*W \in \mathbb{R}^{r\times\mu}$ consists of the first r components of the eigenvectors corresponding to λ. The elements of the submatrices of the Weinstein matrix are given by

$$(U^*(\Lambda-\lambda I)^+U)_{ij} = \frac{1}{m}\sum_{\substack{k=0\\ \lambda_k\neq\lambda}}^{m-1} \frac{\omega^{k(i-j)}}{\lambda_k-\lambda}, \qquad 0 \le i,j \le r-1, \tag{4.2}$$

and by

$$(V^*W)_{ih} = g_{\eta i}, \qquad 0 \le i \le r-1, \qquad 1 \le h \le \mu. \tag{4.3}$$

Here, $h \mapsto \eta$ is a bijective map from $\{1,\dots,\mu\}$ on $\{k|\lambda_k = \lambda\}$. The $g_{\eta i}$ are elements of the real eigenvector matrix mentioned in the last paragraph of §2. As in (2.7) we may write, for even m and $\lambda \notin \sigma(C)$,

$$(U^*(\Lambda-\lambda I)^{-1}U)_{ij} = \frac{1}{m}\left[\frac{1}{\lambda_0-\lambda} + 2\sum_{k=1}^{\frac{m}{2}-1}\frac{1}{\lambda_k-\lambda}\cos\frac{2\pi k(i-j)}{m} + \frac{(-1)^{i-j}}{\lambda_{\frac{m}{2}}-\lambda}\right]. \tag{4.4}$$

A similar equation holds for odd m. If $\lambda \in \sigma(C)$ the corresponding expression for $(U^*(\Lambda - \lambda I)^+ U)_{ij}$ is obtained from (4.4) by omitting the terms with vanishing denominator. Thus $U^*(\Lambda - \lambda I)^+ U$ is a real symmetric Toeplitz matrix! If we assume that the values $(2/m)\cos 2\pi kl/m, 0 \le k < m, 0 \le l < r$, are precomputed, the cost of the calculation of $Z(\lambda) \in \mathbb{R}^{r\times r}$ is only $rm/2 + O(m+r)$ flops. (The assumption is reasonable as these values are already needed when computing the eigenvalues λ_k by (2.7).)

Let $\hat{\lambda}_0 \le \cdots \le \hat{\lambda}_{m-1}$ be the eigenvalues of C arranged in nondecreasing order. Then, because of (3.12), to compute the s smallest eigenvalues $\tilde{\lambda}_0, \ldots, \tilde{\lambda}_{s-1}$ of T we can restrict ourselves to the investigation of the Weinstein matrix in the interval $[\hat{\lambda}_0, \hat{\lambda}_{s+r-1}]$. By the inertia of the generalized Weinstein matrix evaluated for the eigenvalues $\hat{\lambda}_0, \ldots, \hat{\lambda}_{s+r-1}$ we can determine which of these eigenvalues persist, i.e., which are eigenvalues of T as well as of C. (Note that at most two of these eigenvalues are simple.) Corresponding eigenvectors are obtained by (3.8). At the same time, using (3.7), we can determine the number of eigenvalues of T in each open subinterval $(\hat{\lambda}_k, \hat{\lambda}_{k+1})$. Then in each subinterval which contains at least one eigenvalue of T, the zeros of the secular equation (3.5) can be found by some rootfinder.

These considerations immediately lead to the following algorithm.

ALGORITHM 4.1. Let $T \in \mathbb{R}^{n\times n}$ be a symmetric banded Toeplitz matrix with bandwidth r. This algorithm computes the s smallest eigenvalues of T.

1. (a) Compute all eigenvalues $\lambda_i, 0 \le i \le m-1, m = n+r$, of the circulant matrix C as defined in (1.5) and the first r components of the corresponding eigenvectors.
   (b) Sort the eigenvalues λ_i into $\hat{\lambda}_i$ such that $\hat{\lambda}_i \le \hat{\lambda}_{i+1}$. Determine $m' \in \mathbb{Z}$ and the map $\sigma : \mathbb{Z} \supset [0, m'-1] \longrightarrow [0, m-1] \subset \mathbb{Z}$, $m' \le m$, such that

$$\bigcup_{0\le j\le m'-1} \{\hat{\lambda}_{\sigma(i)}\} = \bigcup_{0\le j\le m-1} \{\hat{\lambda}_j\}$$

   and

$$\hat{\lambda}_{\sigma(i)} < \hat{\lambda}_{\sigma(i+1)}, \qquad 0 \le i < m'-1.$$

   (c) Determine s' such that $\hat{\lambda}_{\sigma(s')} = \lambda_{r+s-1}$.
2. Compute the extended Weinstein matrix

$$Z_+(\lambda) = \begin{pmatrix} U^*(\Lambda - \lambda I)^+ U & V^*W \\ W^*V & 0 \end{pmatrix}$$

   for $\lambda = \hat{\lambda}_{\sigma(1)}, \ldots, \hat{\lambda}_{\sigma(s')}$ and determine its inertia. The zero inertia of $Z_+(\hat{\lambda}_{\sigma(j)})$ equals the multiplicity of $\hat{\lambda}_{\sigma(j)}$ as eigenvalue of T.
3. Determine the number of eigenvalues in the intervals $(\hat{\lambda}_{\sigma(j)}, \hat{\lambda}_{\sigma(j+1)})$, $0 \le j < s'$, and compute them (if any) by finding the zeros of the secular equation.

Clearly, the $s'+1$ tasks in step 2 are mutually independent. They can be performed in parallel as soon as the preparatory step 1 has been completed.

That task in step 3 computing the zeros of the Weinstein determinant in the interval $(\hat{\lambda}_{\sigma(j)}, \hat{\lambda}_{\sigma(j+1)})$ can be started as soon as the two tasks of step 2 corresponding to $\hat{\lambda}_{\sigma(j)}$ and $\hat{\lambda}_{\sigma(j+1)}$ have been finished. All s' tasks in step 3 are mutually independent. Thus we get a data dependency graph for the algorithm above for the case $s' = 4$ as

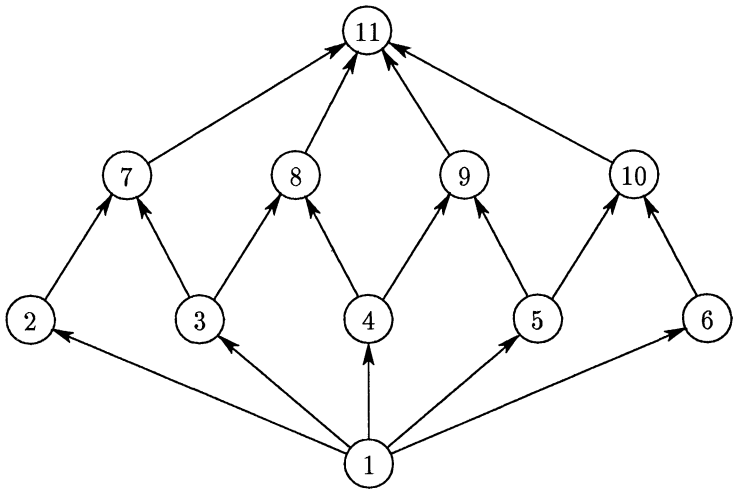


FIG. 4.1. *Data dependency graph with* $s' = 4$.

depicted in Fig. 4.1. In this figure, node 1 corresponds to step 1 of the algorithm. Nodes 2–6 correspond to the $s' + 1$ tasks of step 2 while nodes 7–10 correspond to step 3. The sole purpose of node 11 is to check if all tasks of step 3 have been completed. This node is needed by SCHEDULE [18], which was used to implement our algorithm on an Alliant FX-8.

The amount of work in node 1 is $7(r+1)m/2 + O(r)$ flops. This includes the computation of the eigenvalues of C and the constants $(2/m)\cos 2\pi kl/m$ but not the sorting of the eigenvalues. The work to be done in each of the nodes of step 2 is $rm/2 + (r+\mu)^3/6 + O(r+\mu+m)$ flops. μ will usually be two. The amount of work in each of the nodes corresponding to step 3 is essentially the number of Weinstein matrix evaluations times $rm/2 + r^3/6$. If the zeros of the Weinstein determinant were to be computed by bisection this number would be fairly high. The number of bisection steps to shrink a bracketing interval of length l to one of length less than or equal to ϵl is $\lceil -\log_2 \epsilon \rceil$. For $\epsilon = 10^{-15}$ this number is 50. So the amount of work needed to compute one eigenvalue of T to machine precision is approximately $50(rm/2 + r^3/6)$. The sequential complexity of the computation of s eigenvalues of T is thus $O(sr(m + r^2))$.

Remark 1. If r is not small and if m is a power of two, then the Weinstein matrix can be computed by the Fast Fourier Transform in $O(m \log_2 m)$ flops.

Remark 2. An eigenvector corresponding to an eigenvalue $\tilde{\lambda} \notin \sigma(C)$ is obtained by

$$\tilde{x} = (C - \tilde{\lambda}I)^{-1}Va = F^*(\Lambda - \tilde{\lambda}I)^{-1}FVa, \qquad a \in \mathcal{N}(Z(\tilde{\lambda})) \subset \mathbb{R}^r.$$

The computation of $\tilde{x}$ costs $3m \log_2 m + rm/2 + O(m)$ flops if the Fast Fourier Transform can be applied. This is only a fraction of the average complexity of the computation of an eigenvalue.

Remark 3. The modification of the theory above to the generalized eigenvalue problem $Tx = \lambda Sx$, T, S Toeplitz and S positive definite, is straightforward if r is defined to be the maximum of the bandwidths of T and S and if the circulant matrix corresponding to S is still positive definite.

5. Numerical experiments. In this section we discuss three implementations of Algorithm 4.1. They differ from each other only in the way the zeros of the Weinstein determinant are computed in step 3.

In the first implementation each task of step 3 is executed on a single processor. The zeros of the Weinstein determinant are determined by a bisection algorithm analogous to the one proposed by Barth, Martin, and Wilkinson [5]. This implementation works solely with the inertia of the Weinstein matrix. The inertia are computed by the Bunch–Kaufman algorithm [11] as implemented in the LINPACK routines `dsifa` and `dsidi` [16].

The second implementation is a refinement of the first one. If a point in an interval is found dividing the set of eigenvalues of the interval in two, a process is spawned dynamically to compute the eigenvalues below the separation point. The spawning process continues work on the eigenvalues above this point. The idea to parallelize bisection this way has been proposed by Bernstein and Goldstein [7].

The third implementation is a further refinement of the second one: As soon as an interval containing only one zero is detected, bisection is replaced by the zerofinder of Brent [10], which is available as FORTRAN procedure `zeroin` from netlib [17]. `zeroin` is applied on the bounded function $f(\xi) = (\xi - \hat{\lambda}_{\sigma(k)})^{\mu_{\sigma(k)}}(\xi - \hat{\lambda}_{\sigma(k+1)})^{\mu_{\sigma(k+1)}} \det Z(\xi)$ (cf. (3.11)). Brent's zerofinder is a combination of bisection, secant method, and quadratic inverse interpolation. Its convergence rate is ≈ 1.618. As only one function evaluation per iteration step is required the method has an efficiency index ≈ 1.618, which is higher than the efficiency index $\sqrt{2} \approx 1.414$ of Newton's method.

These algorithms are compared with the combination of EISPACK routines which computes a few of the smallest eigenvalues of (general) symmetric banded matrices [26]. Subroutine `bandr` reduces a banded matrix to tridiagonal form while subroutine `tridib` computes selected eigenvalues of the tridiagonal matrix by the bisection algorithm mentioned above.

TABLE 5.1
Timings for the different implementations in seconds.

n	r	s	s'	p	Static bisection	Dynamic bisection	Dynamic bisection/ `zeroin`	`bandr`/ `tridib`
100	3	10	7	1	.304	.335	.108	.557
				2	.189	.180	.067	
				3	.160	.145	.055	
				4	.111	.120	.050	
100	3	25	14	1	.685	.771	.243	.900
				2	.398	.410	.140	
				3	.294	.298	.105	
				4	.245	.238	.086	
300	9	25	17	1	3.66	3.96	1.18	7.90
				2	2.05	2.08	.683	
				3	1.42	1.49	.520	
				4	1.41	1.21	.447	
300	9	50	30	1	7.02	7.62	2.13	9.30
				2	3.70	3.96	1.16	
				3	2.59	2.75	.854	
				4	2.18	2.15	.692	

In Table 5.1 the cpu times are listed that we obtained when the four algorithms were applied on four eigenvalue problems. We call the three implementations in the order of their introduction static bisection, dynamic bisection, and dynamic bi-

section/**zeroin**. The elements of the test Toeplitz matrices (1.1) were defined by $t_i := \max\{r+1-i, 0\}$. s' is the number defined in part 1(c) of Algorithm 4.1. p is the number of processors used. Note that $\hat{\lambda}_{\sigma(s')}$ is not necessarily the smallest eigenvalue of C greater or equal to $\tilde{\lambda}_{s-1}$. In the first example, for instance, the sixth interval $(\hat{\lambda}_5, \hat{\lambda}_6)$ contains the single eigenvalue $\tilde{\lambda}_{10}$ of T. This is known after the computation of the inertia of $Z_+(\hat{\lambda}_5)$. In order not to destroy the parallelism of the algorithm all s' tasks of step 1 have to be executed. Intervals containing no eigenvalues of interest are then not investigated any further, i.e., the corresponding task is empty.

It is surprising that dynamic bisection is slower than static bisection in most cases. Of course dynamic bisection must be slower than static bisection due to the increased overhead if only one processor is used—if dynamic spawning occurs at all. But this is indeed the case, as can be seen in Table 5.2, where the number of intervals $(\hat{\lambda}_{\sigma(k)}, \hat{\lambda}_{\sigma(k+1)})$ is listed containing a certain number of eigenvalues $\tilde{\lambda}_j$ of T.

TABLE 5.2
Number of intervals containing a certain number of eigenvalues of T.

Example	0	1	2	3	4	5	6
1	–	2	1	2	–	–	–
2	–	5	4	4	–	–	–
3	4	2	5	–	1	2	–
4	6	6	8	2	2	2	1

Dynamic bisection refines the granularity of the tasks and in this way increases the potential to balance the work load of the different processors. Except for the first example it seems that s' is sufficiently big and p is sufficiently small to yield fairly well-balanced work loads. Dynamic bisection does not bring essential advantages if the spawned process can be executed on an idle processor. The finer granularity may be the reason that the speedups from the one to the multiprocessor version are the highest for dynamic bisection. If we assume that the computation of each eigenvalue needs the same amount of time and is performed in one independent task, the speedup to be expected is $s/\lceil s/p \rceil$. The speedups observed with dynamic bisection are above 90 percent of these theoretical numbers.

Dynamic bisection combined with **zeroin** is evidently superior to the pure bisection methods by a factor of about three or more. The speedups observed are however not so high as for dynamic bisection. This is a consequence of Amdahl's law: the slow parts of the program, i.e., the initial bisection steps do not parallelize as well as **zeroin**.

The overall speedups of dynamic bisection/**zeroin** on four processors compared with **bandr**/**tridib** are much higher than four. They range from 10.5 in Example 2 to 17.7 in Example 3.

By the timings of Examples 3 and 4 the different behavior of the newly developed parallel algorithms and the combination of **bandr** and **tridib** becomes evident. The parallel algorithms are built up of components with $O(rm)$ complexity. Thus the overall work is proportional to the number s of desired eigenvalues.

The EISPACK routines behave completely differently. An initial amount of work of $O(rn^2)$ flops is spent by routine **bandr** [24]. (For the examples with $n = 300$, **bandr** takes about 6.5 sec.) Then work proportional to s is done in **tridib**, the complexity of which is $O(n)$! So the superiority of the parallel algorithms decreases with increasing s. Big r's are not disadvantageous as long as the $O(r^3)$ parts of the algorithm, i.e., the computation of inertia/determinant of $Z(\lambda)$ does not outweigh the computation

of $Z(\lambda)$ itself. Big values of r would have a much higher influence if the Weinstein matrix were not Toeplitz, as would be the case if we considered the restricted eigenvalue problem of an arbitrary symmetric matrix A whose spectral decomposition is known (cf. §3). In this more general situation, numerical results would certainly not be so favorable for the parallel algorithms.

REFERENCES

[1] G. S. AMMAR AND W. B. GRAGG, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 61–76.

[2] P. ARBENZ, W. GANDER, AND G. H. GOLUB, *Restricted rank modification of the symmetric eigenvalue problem: Theoretical considerations*, Linear Algebra Appl., 104 (1988), pp. 75–95.

[3] P. ARBENZ AND G. H. GOLUB, *On the spectral decomposition of Hermitian matrices modified by low rank perturbations with applications*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 40–58.

[4] S. F. ASHBY, T. A. MANTEUFFEL, AND P. E. SAYLOR, *Adaptive polynomial preconditioning for Hermitian indefinite linear systems*, BIT, 29 (1989), pp. 583–609.

[5] W. BARTH, R. S. MARTIN, AND J. H. WILKINSON, *Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection*, Numer. Math., 9 (1967), pp. 386–393. Reprinted in Linear Algebra, J.H. Wilkinson and C. Reinsch, eds., Springer-Verlag, Berlin, 1971, pp. 249–256.

[6] C. BEATTIE AND D. FOX, *Schur complements and the Weinstein–Aronszajn theory for modified matrix eigenvalue problems*, Linear Algebra Appl., 108 (1988), pp. 37–61.

[7] H. J. BERNSTEIN AND M. GOLDSTEIN, *Parallel implementation of bisection for the calculation of eigenvalues of tridiagonal symmetric matrices*, Computing, 37 (1986), pp. 85–91.

[8] D. BINI AND M. CAPOVANI, *Spectral and computational properties of band symmetric Toeplitz matrices*, Linear Algebra Appl., 53 (1983), pp. 99–126.

[9] D. BINI AND V. PAN, *Efficient algorithms for the evaluation of the eigenvalues of (block) banded Toeplitz matrices*, Math. Comp., 50 (1988), pp. 431–448.

[10] R. P. BRENT, *Algorithms for Minimization without Derivatives*, Series in Automatic Computation, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[11] J. R. BUNCH AND L. KAUFMAN, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comp., 31 (1977), pp. 163–179.

[12] R. CHAN, *Circulant preconditioners for Hermitian Toeplitz systems*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 542–550.

[13] P. CONCUS AND P. SAYLOR, *Preconditioned iterative methods for indefinite symmetric Toeplitz systems*, in Iterative Methods for Large Linear Systems, D. R. Kincaid and L. J. Hayes, eds., Academic Press, Boston, 1990, pp. 65–79.

[14] G. CYBENKO AND C. VAN LOAN, *Computing the minimum eigenvalue of a symmetric positive definite Toeplitz matrix*, SIAM J. Sci. Statist. Comput., 7 (1981), pp. 123–131.

[15] P. J. DAVIS, *Circulant Matrices*, John Wiley, New York, 1979.

[16] J. J. DONGARRA, J. R. BUNCH, C. B. MOLER, AND G. W. STEWART, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1978.

[17] J. J. DONGARRA AND E. GROSSE, *Distribution of mathematical software via electronic mail*, Comm. Assoc. Comput. Mach., 30 (1987), pp. 403–407.

[18] J. J. DONGARRA AND D. C. SORENSEN, *A portable environment for developing parallel* FORTRAN *programs*, Parallel Comput., 5 (1987), pp. 175–186.

[19] G. H. GOLUB, *Some modified matrix eigenvalue problems*, SIAM Rev., 15 (1973), pp. 318–334.

[20] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[21] Y. H. HU AND S. Y. KUNG, *Toeplitz eigensystem solver*, IEEE Trans. Acoustics Speech Signal Process., 33 (1985), pp. 1264–1271.

[22] A. K. JAIN, *Fast inversion of banded Toeplitz matrices by circular decompositions*, IEEE Trans. Acoustics Speech Signal Process., 26 (1978), pp. 121–126.

[23] R. B. MARR AND G. H. VINEYARD, *Five-diagonal Toeplitz determinants and their relation to Chebyshev polynomials*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 579–586.

[24] H. R. SCHWARZ, *Tridiagonalization of a symmetric band matrix*, Numer. Math., 12 (1968), pp. 231–241. Reprinted in Linear Algebra, J.H. Wilkinson and C. Reinsch, eds., Springer-Verlag, Berlin, 1971, pp. 273–283.

[25] A. SIMPSON, *The Kron methodology and practical algorithms for eigenvalue, sensitivity and*

response analyses of large scale structural systems, Aeronaut. J., 84 (1980), pp. 417–433.

[26] B. T. SMITH, J. M. BOYLE, J. J. DONGARRA, B. S. GARBOW, Y. IKEBE, V. C. KLEMA, AND C. B. MOLER, *Matrix Eigensystem Routines–EISPACK Guide*, Second Edition, Lecture Notes in Computer Science 6, Springer-Verlag, Berlin, 1976.

[27] W. F. TRENCH, *On the eigenvalue problem for Toeplitz band matrices*, Linear Algebra Appl., 64 (1985), pp. 199–214.

[28] ———, *Numerical solution of the eigenvalue problem for Hermitian Toeplitz matrices*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 135–146.

[29] H. F. WEINBERGER, *Variational Methods for Eigenvalue Approximation*, CBMS-NSF Regional Conference Series in Applied Mathematics 15, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1974.

[30] A. WEINSTEIN AND W. STENGER, *Methods of Intermediate Problems for Eigenvalues*, Academic Press, New York, 1972.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 755–786, July 1991

OPTIMAL TRIANGULAR MESH GENERATION BY COORDINATE TRANSFORMATION*

E. F. D'AZEVEDO†

Abstract. This paper presents the motivation for and construction of coordinate transformations that generate optimally efficient meshes for linear interpolation. The coordinate transformations are derived from a result in differential geometry characterizing a "flat" space. The optimality results are demonstrated for some numerical examples. Adaptive meshes produced by PLTMG [R. E. Bank, PLTMG: *A Software Package for Solving Elliptic Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990] are included for comparison. The paper concludes that coordinate transformation is a promising strategy for investigation into more complex optimal meshing problems in finite element analysis.

Key words. optimal triangular mesh, function interpolation, coordinate transformation

AMS(MOS) subject classifications. 65D05, 65L05

1. Introduction.

The novelty of this paper is in the construction and demonstration of coordinate transformations that generate optimally efficient meshes for linear interpolation.

Consider the problem of interpolating a function with piecewise linear patches over a domain to within a given error tolerance. A triangular mesh that achieves this error tolerance with the *fewest* triangles is defined to be optimally efficient. The goal of this paper is the computation of an optimally efficient mesh, where we assume the error tolerance is small and the influence of domain shape on triangulation is insignificant.

This paper is a basic study on optimal meshes with the intention of providing insight into the more complex meshing problems in finite element analysis. The engineering community is interested in finding optimal meshes under the criterion of minimizing the energy norm error of variation problems in finite element analysis [15]. Some attempts applied mathematical programming to find an optimal placement of mesh nodes, but such schemes are computationally expensive and typically encounter problems in mesh entanglement [5], [7], [8]. Others have applied effective heuristics in the generation of near optimal meshes [6], [9]–[11], [14], [16]. This paper presents the approach of coordinate transformation of a regular mesh as a promising strategy for optimal mesh generation.

Coordinate transformation, especially conformal mappings obtained by the solution of Laplace's equation, is a technique used to generate boundary-conforming curvilinear coordinate systems in finite difference solution of fluid dynamics. A rectangular grid is mapped to fit the computational domain, such as an aircraft wing, with some prescribed boundary conditions. The works of Thompson [19]–[21] contain an extensive treatment of this subject. The coordinate transformation described

* Received by the editors April 24, 1989; accepted for publication (in revised form) May 24, 1990. This work was supported by the Natural Sciences and Engineering Research Council of Canada; by the Information Technology Research Centre, which is funded by the Province of Ontario; and by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc., through an appointment to the U.S. Department of Energy Postgraduate Research Program administered by Oak Ridge Associated Universities.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. Present address, Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831-8083 (efdazedo@msr.epm.ornl.gov).

in this paper is *not* determined by geometric considerations but derived from error properties in interpolation.

Nadler [12] and D'Azevedo and Simpson [3] have studied local coordinate transformations for the generation of optimally efficient meshes. Péraire et al. [13] applied such local coordinate transformations in mesh generation for dynamic remeshing in solving compressible flow problems. In these works, piecewise linear approximation of a quadratic function is used as the model for local analysis. This paper extends the local results on optimally efficient meshes to apply *globally* for a class of functions, which includes harmonic and quadratic functions. In the following discussion, we focus on the approximation of harmonic functions.

An outline of the paper follows. In §2, we derive the linear transformation that maps a regular mesh into an optimal mesh for the interpolation of a quadratic function. The transformation is suggested by simplification of the exact error expression. In §3, this simplification of the local error expression motivates the extension of the coordinate transformation for optimal mesh generation to a wider class of functions. To state the condition for such transformations, we invoke a basic result in differential geometry characterizing a "flat" space. The condition can be demonstrated by direct consideration of the solvability of some differential equations. We show that these transformations can be computed as the solution to a system of ordinary differential equations posed as an initial value problem. We derive some important properties of these coordinate transformations in §4. Section 5 contains discussions and error profiles from numerical experiments in the interpolation of harmonic functions. The results show remarkable agreement with the theoretical predictions. Experimental results for the meshes generated by PLTMG [1], a PDE solver with adaptive mesh refinement, in solving Laplace's equation with Dirichlet boundary conditions are included for comparison. The error profiles show the optimality of the triangulation, and also provide a measure of the efficiency of the adaptively generated mesh. Finally, in §6 we state our conclusion and suggest directions for further research.

2. Local analysis by quadratic model. In this section, we show how a linear transformation of a regular mesh of optimal-shape triangles yields an optimally efficient mesh for interpolating a quadratic function.

Let $E(x,y)$ be the error function obtained in the linear interpolation of a quadratic function,

$$f(x,y) = \tfrac{1}{2}(x,y)H(x,y)^t + d_1x + d_2y + d_3 \ , \tag{1}$$

over a triangle T. By definition, the error function $E(x,y)$ is a quadratic function and its level curves form a family of conics with a common center. It is a family of ellipses if $\det(H) > 0$, and a family of hyperbolas if $\det(H) < 0$. Let the common center of the family of conics be at (x_c, y_c). The error at a displacement from the center is given by

$$E(x_c + dx, y_c + dy) = \mathcal{E} - \tfrac{1}{2}(dx,dy)H(dx,dy)^t \ , \tag{2}$$

where $\mathcal{E} = E(x_c, y_c)$. The details of the derivation of (2) are contained in Appendix A.

We see that if the symmetric matrix H is diagonalizable as

$$H = Q^t \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} Q = R^t \begin{bmatrix} 1 & 0 \\ 0 & \epsilon \end{bmatrix} R \ , \tag{3}$$

where

$$R = \begin{bmatrix} \sqrt{|\lambda_1|} & 0 \\ 0 & \sqrt{|\lambda_2|} \end{bmatrix} Q , \qquad \epsilon = \operatorname{sign}(\det(H)) , \tag{4}$$

then with $(d\tilde{x}, d\tilde{y})^t = R(dx, dy)^t$, the expression $(dx, dy)H(dx, dy)^t$ reduces to $(d\tilde{x})^2 + \epsilon(d\tilde{y})^2$. The error function can be rewritten as

$$\begin{aligned} E(x_c + dx, y_c + dy) &= \mathcal{E} - \tfrac{1}{2}(dx, dy)H(dx, dy)^t \\ &= \mathcal{E} - \tfrac{1}{2}((d\tilde{x})^2 + \epsilon(d\tilde{y})^2) \\ &= \tilde{E}(\tilde{x}_c + d\tilde{x}, \tilde{y}_c + d\tilde{y}) , \end{aligned} \tag{5}$$

where $\tilde{E}(\tilde{x}, \tilde{y})$ denotes the corresponding error function under transformation R.

In the following, we shall determine the best triangle shape that minimizes the error in the approximation. Consider first the case where $f(x, y)$ is convex, then the level curves or contours of $\tilde{E}(\tilde{x}, \tilde{y})$ are concentric circles given by

$$\tilde{E}(\tilde{x}_c + d\tilde{x}, \tilde{y}_c + d\tilde{y}) = \mathcal{E} - \tfrac{1}{2}((d\tilde{x})^2 + (d\tilde{y})^2) . \tag{6}$$

Let $\tilde{T}$ be the transformed image of triangle T with vertices at $(\tilde{x}_1, \tilde{y}_1)$, $(\tilde{x}_2, \tilde{y}_2)$ and $(\tilde{x}_3, \tilde{y}_3)$. The circumcircle of $\tilde{T}$ corresponds to the level curve of value zero. Hence the radius of this circumcircle is sqrt$(2|\mathcal{E}|)$ and relates directly to the maximum error attainable. We can easily see that an equilateral triangle covers the most area for a fixed circumcircle; therefore an equilateral triangle is of optimal-shape.

Consider next the case where $f(x, y)$ is not convex but with a saddle-shaped graph, then we have

$$\begin{aligned} \tilde{E}(\tilde{x}, \tilde{y}) &= \tilde{E}(\tilde{x}_c + d\tilde{x}, \tilde{y}_c + d\tilde{y}) \\ &= \mathcal{E} - \tfrac{1}{2}((d\tilde{x})^2 - (d\tilde{y})^2) \\ &= \mathcal{E} - \tfrac{1}{2}((\tilde{x} - \tilde{x}_c)^2 - (\tilde{y} - \tilde{y}_c)^2) . \end{aligned} \tag{7}$$

We note that the error function $\tilde{E}(\tilde{x}, \tilde{y})$ is a harmonic function and attains its extrema on the boundary of $\tilde{T}$. By calculus, we can show that the local extrema along edge $(\tilde{x}_i, \tilde{y}_i)$, $(\tilde{x}_j, \tilde{y}_j)$ is attained at the midpoint with value

$$\tilde{E}\left(\frac{\tilde{x}_i + \tilde{x}_j}{2}, \frac{\tilde{y}_i + \tilde{y}_j}{2}\right) = \frac{1}{8}\left|(\tilde{x}_i - \tilde{x}_j)^2 - (\tilde{y}_i - \tilde{y}_j)^2\right| . \tag{8}$$

Thus the maximum error attained is

$$\frac{1}{8}\max\left|(\tilde{x}_i - \tilde{x}_j)^2 - (\tilde{y}_i - \tilde{y}_j)^2\right| . \tag{9}$$

The details in finding the optimal-shape triangle are found in Appendix B. The optimal-shape triangle geometry in this case is not unique and attains the same error on all three edges. Fig. 1 shows an optimal-shape triangle geometry chosen for the saddle shape functions.

Applying the results on optimal-shape triangles, we see that a regular mesh of optimal-shape triangles over the transformed plane corresponds to an optimally efficient mesh. Every triangle attains the same maximum error; moreover, these triangles cover the most area for the error attained and so are optimally efficient. Since the transformation R in (4) is basically a rotation followed by a rescaling of coordinate axes, we find the areas of triangles are scaled accordingly. Hence the inverse transformation R^{-1}, maps this regular mesh to produce an optimally efficient mesh in the original plane.

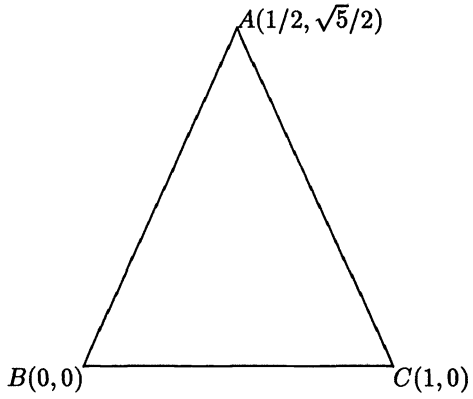


FIG. 1. *Optimal triangle geometry for saddle shape function.*

3. Coordinate transformation. Locally, a function behaves as a quadratic given by its Taylor expansion

$$f(x+dx, y+dy) \approx \tfrac{1}{2}(dx,dy)H(dx,dy)^t + (dx,dy)\nabla f(x,y) + f(x,y) \ , \tag{10}$$

where H is the Hessian matrix at (x,y). Hence, we can define local coordinate transformations for generation of optimally efficient meshes. The main goal of this section is to investigate the extension of these local results to a wider class of functions.

3.1. Optimal meshes in one dimension. Mesh generation by equidistribution in one-dimensional problems can be viewed as optimal mesh generation by coordinate transformation. As an example, consider de Boor's suggestion [4] on the optimal placement of knots for linear interpolation by equidistribution of the density function

$$\Phi(x) = \sqrt{|f''(x)|} \ . \tag{11}$$

Let the error function be represented as

$$E(x) = \tfrac{1}{2}(x-a)(x-b)f_2 \ . \tag{12}$$

By calculus, $E(x)$ achieves a local extremum at $x_c = (a+b)/2$:

$$\begin{aligned} E(x_c+dx) &= \tfrac{1}{2}((x_c+dx)-a)((x_c+dx)-b)f_2 \\ &= \tfrac{1}{2}((x_c-a)+dx)((x_c-b)+dx)f_2 \\ &= \tfrac{1}{2}\left[(x_c-a)(x_c-b) + ((x_c-a)+(x_c-b))\,dx + (dx)^2\right] f_2 \\ &= E(x_c) + \tfrac{1}{2}(dx)^2 f_2 \\ &= E(x_c) + \tfrac{1}{2}\epsilon d\tilde{x}^2 \ , \end{aligned} \tag{13}$$

where

$$d\tilde{x} = dx\sqrt{|f_2|} \ , \qquad \epsilon = \operatorname{sign}(f_2) \ . \tag{14}$$

By extension of the local analysis, we require the coordinate transformation $\tilde{x}(x)$ such that

$$d\tilde{x} = \sqrt{|f''(x)|}dx \ . \tag{15}$$

One choice for $\tilde{x}(x)$ is

$$\tilde{x}(x) = \int_{x_0}^{x} \sqrt{|f''(t)|}\ dt + \tilde{x}(x_0) \ . \tag{16}$$

Ideally, a uniform mesh in $\tilde{x}$-coordinate will yield an optimal mesh in the original coordinate space. Then the mesh nodes $\{x_i\}$ are obtained by equidistribution of sqrt($|f''(x)|$), which is suggested by de Boor [4],

$$\begin{aligned} \text{const.} &= \tilde{x}_{i+1} - \tilde{x}_i \\ &= \left(\int_{x_0}^{x_{i+1}} \sqrt{|f''(t)|}\ dt + \tilde{x}(x_0)\right) - \left(\int_{x_0}^{x_i} \sqrt{|f''(t)|}\ dt + \tilde{x}(x_0)\right) \\ &= \int_{x_i}^{x_{i+1}} \sqrt{|f''(t)|}\ dt \ . \end{aligned} \tag{17}$$

Therefore optimal triangular mesh generation by coordinate transformation can be viewed as a generalization of the techniques of equidistribution in one dimension. White [22] has developed a coordinate transformation to map uniform meshes to equidistributing meshes for solving two-point boundary value problems.

The generalization of (15) for triangular mesh generation requires the determination of coordinate transformation $(\tilde{x}(x,y), \tilde{y}(x,y))$ that satisfies

$$(dx, dy)H(dx, dy)^t = (d\tilde{x})^2 + \epsilon(d\tilde{y})^2 \ , \tag{18}$$

where H is the Hessian matrix, and ϵ is the sign of $\det(H)$. We can see that the mesh density function $\sqrt{|f''(x)|}$ applied in one-dimensional equidistribution is generalized to use the Hessian matrix. The constant ϵ appears in the canonical form (18) since the Hessian matrix is neither positive definite nor negative definite for a saddle shape function. Note that by the same strategy of transforming a uniform mesh into an optimal mesh, we see that the error-equilibrating property of optimal meshes in one dimension is extended to two dimensions.

The conditions for finding transformations $\tilde{x}(x,y)$, $\tilde{y}(x,y)$ are given by a classical result in differential geometry characterizing a "flat" space. To be consistent with the notation used in differential geometry, we rename the original space coordinates (x,y) as (x_1, x_2), the transformed coordinates $(\tilde{x}, \tilde{y})$ as (y_1, y_2) and entries in the Hessian matrix as $\{g_{ij}\}$. Let $y_1(x_1, x_2)$, $y_2(x_1, x_2)$ be the required transformation; then we require that

$$\begin{aligned} (dy_1)^2 + \epsilon(dy_2)^2 &= \left(\frac{\partial y_1}{\partial x_1}dx_1 + \frac{\partial y_1}{\partial x_2}dx_2\right)^2 + \epsilon\left(\frac{\partial y_2}{\partial x_1}dx_1 + \frac{\partial y_2}{\partial x_2}dx_2\right)^2 \\ &= \left(\left(\frac{\partial y_1}{\partial x_1}\right)^2 + \epsilon\left(\frac{\partial y_2}{\partial x_1}\right)^2\right)(dx_1)^2 + \left(\left(\frac{\partial y_1}{\partial x_2}\right)^2 + \epsilon\left(\frac{\partial y_2}{\partial x_2}\right)^2\right)(dx_2)^2 \\ &\qquad + 2\left(\frac{\partial y_1}{\partial x_1}\frac{\partial y_1}{\partial x_2} + \epsilon\frac{\partial y_2}{\partial x_1}\frac{\partial y_2}{\partial x_2}\right)dx_1 dx_2 \\ &= g_{11}(dx_1)^2 + 2g_{12}(dx_1)(dx_2) + g_{22}(dx_2)^2 \ . \end{aligned} \tag{19}$$

Therefore the transformation being sought must satisfy

$$
\begin{aligned}
g_{11} &= \left(\frac{\partial y_1}{\partial x_1}\right)^2 + \epsilon\left(\frac{\partial y_2}{\partial x_1}\right)^2 , \\
g_{12} &= \frac{\partial y_1}{\partial x_1}\frac{\partial y_1}{\partial x_2} + \epsilon\frac{\partial y_2}{\partial x_1}\frac{\partial y_2}{\partial x_2} , \\
g_{22} &= \left(\frac{\partial y_1}{\partial x_2}\right)^2 + \epsilon\left(\frac{\partial y_2}{\partial x_2}\right)^2 .
\end{aligned} \tag{20}
$$

3.2. A classical result in differential geometry. A basic question in differential geometry is, "Given a symmetric tensor $g_{ij}(x)$, under what restrictions on g_{ij} can there be a coordinate system defined by

$$y_i = y_i(x_1, \cdots, x_n) , \qquad (i = 1, \cdots, n) \tag{21}$$

in which the transformed tensor $g_{ij}(x)$ has constant components h_{ij} everywhere?" A classical result in differential geometry gives the necessary and sufficient conditions on $g_{ij}(x)$. Details can be found in textbooks such as [17] and [18].

THEOREM 3.1 ([17, p. 96]). *A necessary and sufficient condition that a symmetric tensor $g_{ij}(x)$, with $|g_{ij}| \neq 0$, reduce under a suitable transformation of coordinates to a tensor h_{ij}, where the h_{ij}'s are constants, is that the Riemann–Christoffel tensor formed from the g_{ij}'s be a zero tensor.*

Synge and Schild [18] have shown that the quadratic form

$$(ds)^2 = h_{11}{dy_1}^2 + h_{22}{dy_2}^2 + 2h_{12}dy_1dy_2 \tag{22}$$

can be further reduced by a nonsingular linear transformation to

$$(ds)^2 = {d\tilde{y}_1}^2 + \epsilon{d\tilde{y}_2}^2 , \tag{23}$$

where ϵ is sign of $\det(h_{ij})$.

For the special case where the metric matrix $\{g_{ij}\}$ is a Hessian matrix, the Christoffel symbols simplify in the following way.

Let the partial derivatives of the entries in the Hessian matrix (metric tensor) be denoted as

$$\frac{\partial g_{ij}}{\partial x_k} = g_{ijk} , \qquad g = g_{11}g_{22} - {g_{12}}^2 ; \tag{24}$$

then the Christoffel symbols of the second kind are

$$\left\{ \begin{matrix} 1 \\ ij \end{matrix} \right\} = \frac{g_{22}g_{1ij} - g_{12}g_{ij2}}{2g} , \qquad \left\{ \begin{matrix} 2 \\ ij \end{matrix} \right\} = \frac{g_{11}g_{ij2} - g_{12}g_{1ij}}{2g} , \tag{25}$$

and the independent components of the Riemann–Christoffel symbols can be simplified to

$$
\begin{aligned}
R^1_{212} &= \frac{g_{22}}{4g^2}\Gamma , \qquad R^1_{112} = \frac{g_{12}}{4g^2}\Gamma , \\
R^2_{212} &= -\frac{g_{12}}{4g^2}\Gamma , \qquad R^2_{112} = -\frac{g_{11}}{4g^2}\Gamma ,
\end{aligned} \tag{26}
$$

where

$$\Gamma = g_{11}({g_{122}}^2 - g_{222}g_{112}) + g_{12}(g_{111}g_{222} - g_{112}g_{122}) + g_{22}({g_{112}}^2 - g_{122}g_{111}) \ . \tag{27}$$

We can easily verify that if for some constants K_1, K_2, K_3, the entries in the metric matrix satisfy

$$K_1 g_{11} + K_2 g_{12} + K_3 g_{22} = 0 \ , \tag{28}$$

then the Riemann–Christoffel tensor given by (27) vanishes identically. In particular, harmonic and quadratic functions satisfy the above condition.

Sokolnikoff [17, p. 93] shows that $y_1(x_1, x_2)$, $y_2(x_1, x_2)$ satisfy the system of equations

$$\frac{\partial^2 y_m}{\partial x_i \partial y_j} = \left\{ \begin{matrix} 1 \\ ij \end{matrix} \right\} \frac{\partial y_m}{\partial x_1} + \left\{ \begin{matrix} 2 \\ ij \end{matrix} \right\} \frac{\partial y_m}{\partial x_2} \quad \text{for} \quad m = 1, 2 \ . \tag{29}$$

Taking u_1,u_2 as representing the first partial derivatives of $y_m(x_1, x_2)$,

$$u_1 = \frac{\partial y_m}{\partial x_1} \ , \qquad u_2 = \frac{\partial y_m}{\partial x_2} \ , \tag{30}$$

the system of first-order partial differential equations (31)

$$\begin{aligned} \frac{\partial u_1}{\partial x_1} &= \left\{ \begin{matrix} 1 \\ 11 \end{matrix} \right\} u_1 + \left\{ \begin{matrix} 2 \\ 11 \end{matrix} \right\} u_2 \ , \\ \frac{\partial u_2}{\partial x_2} &= \left\{ \begin{matrix} 1 \\ 22 \end{matrix} \right\} u_1 + \left\{ \begin{matrix} 2 \\ 22 \end{matrix} \right\} u_2 \ . \end{aligned} \tag{31}$$

follows immediately from (30).

3.3. Initial value problem. Due to the special structure of the exact differential equation in (31), $y_1(x_1, x_2)$ can be integrated as the solution to an initial value problem.

From (31) and by a basic property of line integrals

$$\begin{aligned} y_1(x_1, x_2) - y_1(a, b) &= \int_P^Q dy_1 \\ &= \int_P^Q \frac{\partial y_1}{\partial x_1} dx_1 + \frac{\partial y_1}{\partial x_2} dx_2 \\ &= \int_P^Q u_1 dx_1 + u_2 dx_2 \ , \end{aligned} \tag{32}$$

where the line integral from $P(a, b)$ to $Q(x_1, x_2)$ is path independent. Let $(x_1(t), x_2(t))$ be a parametric smooth continuous path of integration from (a, b) to (x_1, x_2). Then (31) can be formulated as a system of ordinary differential equations

$$x'_1 = \frac{d}{dt} x_1(t) \ , \qquad x'_2 = \frac{d}{dt} x_2(t) \ , \tag{33}$$

$$\frac{d}{dt}y_1 = u_1 x'_1 + u_2 x'_2 ,$$
$$\frac{d}{dt}u_1 = \left(\left\{ \begin{matrix} 1 \\ 11 \end{matrix} \right\} x'_1 + \left\{ \begin{matrix} 1 \\ 12 \end{matrix} \right\} x'_2 \right) u_1 + \left(\left\{ \begin{matrix} 2 \\ 11 \end{matrix} \right\} x'_1 + \left\{ \begin{matrix} 2 \\ 12 \end{matrix} \right\} x'_2 \right) u_2 ,$$
$$\frac{d}{dt}u_2 = \left(\left\{ \begin{matrix} 1 \\ 12 \end{matrix} \right\} x'_1 + \left\{ \begin{matrix} 1 \\ 22 \end{matrix} \right\} x'_2 \right) u_1 + \left(\left\{ \begin{matrix} 2 \\ 12 \end{matrix} \right\} x'_1 + \left\{ \begin{matrix} 2 \\ 22 \end{matrix} \right\} x'_2 \right) u_2 .$$

If the function to be approximated is a harmonic function, then $g_{22} = -g_{11}$ and equation (33) can be further simplified to

$$\begin{aligned} \frac{d}{dt}y_1 &= u_1 x'_1 + u_2 x'_2 , \\ \frac{d}{dt}u_1 &= (\alpha x'_1 + \beta x'_2)u_1 - (\beta x'_1 - \alpha x'_2)u_2 , \\ \frac{d}{dt}u_2 &= (\beta x'_1 - \alpha x'_2)u_1 + (\alpha x'_1 + \beta x'_2)u_2 , \\ \alpha &= \frac{1}{4}\frac{\partial}{\partial x_1}\ln|\det(H)| , \qquad \beta = \frac{1}{4}\frac{\partial}{\partial x_2}\ln|\det(H)| . \end{aligned} \tag{34}$$

Hence with initial conditions for $y_1(a,b)$, $u_1(a,b)$, and $u_2(a,b)$, equation (33) can be solved as an initial value problem. In particular, the path of integration can be chosen to be taken along only horizontal and vertical directions. First integrate horizontally from (a,b) to (x_1,b) by

$$\begin{aligned} x_1(t) &= t , \qquad x'_1(t) = 1 , \qquad a \le t \le x_1 , \\ x_2(t) &= b , \qquad x'_2(t) = 0 , \end{aligned} \tag{35}$$

then vertically from (x_1,b) to (x_1,x_2) by

$$\begin{aligned} x_1(t) &= x_1 , \qquad x'_1(t) = 0 , \\ x_2(t) &= t , \qquad x'_2(t) = 1 , \qquad b \le t \le x_2 . \end{aligned} \tag{36}$$

Since $y_2(x_1,x_2)$ also satisfies (31) and therefore (33), $y_2(x_1,x_2)$ can be integrated similarly to $y_1(x_1,x_2)$. The coefficients in the transformed metric tensor are constants and obviously independent of (y_1,y_2), therefore the tensor can be determined by consideration of any point. To achieve (23) and to be consistent with the transformation used in the local quadratic model, the initial values of (u_1,u_2), for $y_1(x_1,x_2)$, $y_2(x_1,x_2)$, are chosen to be the orthogonal scaled eigenvectors of the Hessian matrix at (a,b),

$$(u_1,u_2) = \sqrt{|\lambda_1|}(q_{11},q_{12}) \tag{37}$$

for $y_1(x_1,x_2)$ and for $y_2(x_1,x_2)$,

$$(u_1,u_2) = \sqrt{|\lambda_2|}(q_{21},q_{22}) , \tag{38}$$

where

$$Q = \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix}, \qquad H = Q^t \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} Q \,. \tag{39}$$

The metric tensor then satisfies (23). Therefore, $y_1(x_1, x_2)$ and $y_2(x_1, x_2)$ of the coordinate transformation can be numerically computed as solutions to initial value problems.

3.4. Algorithm for optimal mesh generation. The steps for generating an optimally efficient mesh for a given data function over a domain Ω can be summarized as follows:

Step 1.

Determine $y_1(x_1, x_2)$ of the coordinate transformation through the solution of the initial value problem (35). Repeat for $y_2(x_1, x_2)$. Use the transformation $(y_1(x_1, x_2), y_2(x_1, x_2))$ to compute the corresponding region $\tilde{\Omega}$ over the "flat" (y_1, y_2)-space.

Step 2.

Generate a regular triangulation $\mathcal{T}$ of optimal shape triangles over domain $\tilde{\Omega}$. The size of triangle is determined by the prescribed error tolerance.

Step 3.

Compute the images of triangulation $\mathcal{T}$ in the original (x_1, x_2)-space.

If the domain of computation Ω is the unit square, the procedures for Step 1 and Step 3 can be further simplified as follows:

Step 1a.

Integrate the initial value problem (35) along the x_1 axis with starting values at the origin and save values for

$$y_m(i/n, 0), \quad u_1(i/n, 0), \quad u_2(i/n, 0), \qquad i = 1 \cdots n \quad \text{for} \quad m = 1, 2 \,. \tag{40}$$

Step 1b.

For each i from zero to n, integrate vertically along the coordinate line $x_1 = i/n$ with initial conditions given at $(x_1, x_2) = (i/n, 0)$ to compute

$$y_m(i/n, j/n) \,, \qquad j = 0 \cdots n \quad \text{for} \quad m = 1, 2 \,. \tag{41}$$

Simplified Step 3.

For n sufficiently large, we can assume that over each rectangular cell with vertices,

$$\begin{aligned} (x_1, x_2) = {} & (i/n, j/n), \ (i/n, (j+1)/n), \\ & ((i+1)/n, (j+1)/n), \ ((i+1)/n, j/n), \end{aligned} \tag{42}$$

the coordinate transformation $y_m(x_1, x_2)$ is well approximated by a bilinear function,

$$y_m(x_1, x_2) = c_1 x_1 + c_2 x_2 + c_3 x_1 x_2 + c_4 \quad \text{for} \quad m = 1, 2 \,, \tag{43}$$

which interpolates $y_m(x_1, x_2)$ at the four corner vertices. For each vertex $(\hat{y}_1, \hat{y}_2)$ in the regular triangulation $\mathcal{T}$, the inverse image $(\hat{x}_1, \hat{x}_2)$ is then determined by solving (43).

4. Properties of coordinate transformations.

4.1. Eigentrajectories. The development of the transformation to the (y_1, y_2)-space, where the error metric is uniform, may have obscured a rather deep connection with the eigenvalues and eigendirections of H, that the transformation generated by solving the initial value problem is indeed a generalization of the transformation in §2. Eigendirections (u, v) of the Hessian matrix H satisfy

$$g_{11}u + g_{12}v = \lambda u \ , \tag{44}$$

and hence setting $u = g_{12}$, we can see that

$$(g_{12}, \lambda_1 - g_{11}) \quad \text{and} \quad (g_{12}, \lambda_2 - g_{11}) \tag{45}$$

are eigendirections,

$$\lambda_1, \lambda_2 = \tfrac{1}{2}\left((g_{11} + g_{22}) \pm ((g_{11} - g_{22})^2 + 4g_{12})^{1/2}\right) \ . \tag{46}$$

Furthermore, since H is symmetric, these vectors are orthogonal. If we follow an eigendirection, we trace a trajectory curve that satisfies the differential equation

$$\frac{dx_1}{dx_2} = \frac{g_{12}}{\lambda - g_{11}} \ . \tag{47}$$

In general, (47) prescribes a family of orthogonal curves with tangent vectors along the eigendirection for the Hessian matrix. This family of curves is the eigentrajectory of the Hessian matrix. In [2] D'Azevedo shows that such a family of eigentrajectory curves is mapped by $y_1(x_1, x_2)$ and $y_2(x_1, x_2)$ into coordinate lines over the transformed plane.

4.2. Generalized equidistribution. We have seen in §3.1 that in one dimension, an optimal mesh for the linear interpolation problem equidistributes the density function

$$\Phi(x) = \sqrt{|f''(x)|} \ . \tag{48}$$

Here we can show that a regular triangulation over the transformed (y_1, y_2)-space equidistributes the density function

$$\Phi(x_1, x_2) = \sqrt{|\det(H)|} \ . \tag{49}$$

Now

$$\begin{aligned} \det(H) &= g_{11}g_{22} - g_{12}^2 \\ &= \left(\left(\frac{\partial y_1}{\partial x_1}\right)^2 + \epsilon\left(\frac{\partial y_2}{\partial x_1}\right)^2\right)\left(\left(\frac{\partial y_1}{\partial x_2}\right)^2 + \epsilon\left(\frac{\partial y_2}{\partial x_2}\right)^2\right) \\ &\qquad - \left(\frac{\partial y_1}{\partial x_1}\frac{\partial y_1}{\partial x_2} + \epsilon\frac{\partial y_2}{\partial x_1}\frac{\partial y_2}{\partial x_2}\right) \\ &= \epsilon\left(\frac{\partial y_1}{\partial x_2}\frac{\partial y_2}{\partial x_1} - \frac{\partial y_1}{\partial x_1}\frac{\partial y_2}{\partial x_2}\right)^2 \ . \end{aligned} \tag{50}$$

The integral of the density function $\Phi(x_1, x_2)$ over a region Ω is

$$\int_\Omega \sqrt{|\det(H)|}\, dx_1 dx_2 = \int_\Omega \sqrt{|g_{11}g_{22} - {g_{12}}^2|}\, dx_1 dx_2 = \int_\Omega \left| \frac{\partial y_1}{\partial x_2}\frac{\partial y_2}{\partial x_1} - \frac{\partial y_1}{\partial x_1}\frac{\partial y_2}{\partial x_2} \right| \mathcal{J}\, dy_1 dy_2 , \tag{51}$$

where the Jacobian for the transformation to (y_1, y_2)-space is

$$\mathcal{J} = \left| \frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} \right| = \left| \frac{\partial(y_1, y_2)}{\partial(x_1, x_2)} \right|^{-1} = \left| \frac{\partial y_1}{\partial x_2}\frac{\partial y_2}{\partial x_1} - \frac{\partial y_1}{\partial x_1}\frac{\partial y_2}{\partial x_2} \right|^{-1} . \tag{52}$$

Hence the integral

$$\int_\Omega \sqrt{|\det(H)|}\, dx_1 dx_2 = \int_{\tilde{\Omega}} dy_1 dy_2 , \tag{53}$$

is proportional to the area of the transformed region $\tilde{\Omega}$ over the (y_1, y_2)-space.

4.3. Conformal mapping. If the function to be approximated is a harmonic function, the coordinate transformations $(y_1(x_1, x_2), y_2(x_1, x_2))$ determined by initial conditions (37), (38) can be shown to be a conformal mapping. Now the transformed metric tensor is diagonal, hence the gradient vectors for $y_1(x_1, x_2)$, $y_2(x_1, x_2)$ are orthogonal:

$$\frac{\partial y_1}{\partial x_1}\frac{\partial y_2}{\partial x_1} + \frac{\partial y_1}{\partial x_2}\frac{\partial y_2}{\partial x_2} = 0 . \tag{54}$$

By (20) the transformations $(y_1(x_1, x_2), y_2(x_1, x_2))$ satisfy

$$g_{11} = \left(\frac{\partial y_1}{\partial x_1}\right)^2 - \left(\frac{\partial y_2}{\partial x_1}\right)^2 , \qquad g_{22} = -g_{11} = \left(\frac{\partial y_1}{\partial x_2}\right)^2 - \left(\frac{\partial y_2}{\partial x_2}\right)^2 , \tag{55}$$

which implies

$$\frac{\partial y_1}{\partial x_1}^2 + \frac{\partial y_1}{\partial x_2}^2 = \frac{\partial y_2}{\partial x_1}^2 + \frac{\partial y_2}{\partial x_2}^2 . \tag{56}$$

From (54) and (56), we can show either

$$\frac{\partial y_1}{\partial x_1} = \frac{\partial y_2}{\partial x_2} , \qquad \frac{\partial y_1}{\partial x_2} = -\frac{\partial y_2}{\partial x_1} , \tag{57}$$

or

$$\frac{\partial y_1}{\partial x_2} = \frac{\partial y_2}{\partial x_1} , \qquad \frac{\partial y_1}{\partial x_1} = -\frac{\partial y_2}{\partial x_2} . \tag{58}$$

We recognize that (57) and (58) are related by exchange of coordinates x_1, x_2 and (57) as the Cauchy–Riemann equations for a complex function. Therefore $(y_1(x_1, x_2), y_2(x_1, x_2))$ is a conformal mapping. Note in contrast to the classical applications of conformal mappings, this mapping is determined by consideration of error properties and *not* by geometric considerations of transforming the region of computation to some canonical domain.

5. Numerical experiments. In this section, we shall examine some results from numerical experiments of optimal mesh-generation for interpolating a harmonic function over the unit square. Three mesh-generation strategies are used for each example. One strategy is to generate a mesh by mapping a regular mesh of optimal-shape triangles into the *interior* of the unit square. To cover the unit square exactly, boundary triangles must be distorted from the optimal-shape to fit the boundary edges, thus incurring a change in error properties. This phenomenon is exhibited by a mesh obtained by tiling the transformed unit square with equilateral triangles. The equilateral triangles are generated by D03MAF of the NAG library. Last, we included a mesh obtained by PLTMG. PLTMG [1] is a multigrid package with adaptive triangular mesh refinement for solving second-order partial differential equations. Since a triangular mesh in PLTMG is stored in a complicated data structure of nested trees of triangles, the mesh node positions from PLTMG are simply extracted to form a Delaunay triangulation to be used in this comparison. The maximum interpolation error over each triangle is plotted in ascending order to form an error profile. Ideally, by the equilibrating property of an optimal mesh, the error profile should be almost level.

D02CBF, a variable step Adams code in the NAG library, integrates the initial value problem (33) as described in (35) and (36), to yield values of $y_1(x_1, x_2)$, $y_2(x_1, x_2)$ over a regular 40×40 grid. Note that the unit square is distorted over the transformed plane by the coordinate transformation. D03MAF of the NAG library triangulates this distorted region by equilateral triangles. The corresponding triangulation in the original coordinates is determined by the inverse mapping of this regular mesh. The inverse mapping is computed by assuming that each rectangular cell on the square grid determines a bilinear function. More sophisticated bicubic splines approximation and numerical techniques for inverse mapping yields no significant improvements over this simple technique.

The triangular meshes produced by D03MAF are compared with meshes generated by PLTMG in solving Laplace's problem with Dirichlet boundary conditions. A regular triangulation by optimal-shape triangles over the *interior* of the domain is used to demonstrate the equilibrating property and act as a indicator of closeness of PLTMG to an optimal mesh. The comparison is not precise since the number of triangles used cannot be tightly controlled.

Four interpolation examples are chosen to demonstrate the effectiveness of optimal mesh generation by coordinate transformation. Since these examples exhibit sharp increases, we expect triangle sizes to vary accordingly.

Example 1. Exponential increase along x_1-axis,

$$f(x_1, x_2) = \exp(5x_1)\sin(5x_2)\ . \tag{59}$$

The coordinate transformations are

$$\begin{aligned} y_1(x_1, x_2) &= \frac{2}{\sqrt{2}}\left(\exp(5x_1/2)\left(\cos(5x_2/2) + \sin(5x_2/2)\right) - 1\right)\ , \\ y_2(x_1, x_2) &= \frac{2}{\sqrt{2}}\left(\exp(5x_1/2)\left(\cos(5x_2/2) - \sin(5x_2/2)\right) - 1\right)\ . \end{aligned} \tag{60}$$

Example 2. A near singularity with $z = x + iy$ and $z_0 = x_0 + iy_0 = (1/2) - (1/5)i$,

$$f(x_1, x_2) = \Re e\ 1/(z - z_0)^2\ , \tag{61}$$

$$= \frac{(x_1 - x_0)^2 - (x_2 - y_0)^2}{((x_1 - x_0)^2 + (x_2 - y_0)^2)^2} .$$

The coordinate transformations are

$$y_1(x_1, x_2) = \sqrt{6}\left(1 - \frac{x_1 - x_0}{(x_1 - x_0)^2 + (x_2 - y_0)^2}\right) ,$$

(62)

$$y_2(x_1, x_2) = \sqrt{6}\frac{x_2 - y_0}{(x_1 - x_0)^2 + (x_2 - y_0)^2} .$$

Example 3. A more severe near singularity,

(63)
$$f(x_1, x_2) = \Re e\ 1/(z - z_0)^4$$
$$= \frac{((x_1 - x_0)^2 + (x_2 - y_0)^2)^2 - 8(x_1 - x_0)^2(x_2 - y_0)^2}{((x_1 - x_0)^2 + (x_2 - y_0)^2)^4} .$$

The coordinate transformations are

$$y_1(x_1, x_2) = \sqrt{5}\left(1 + \frac{(x_2 - y_0)^2 - (x_1 - x_0)^2}{((x_1 - x_0)^2 + (x_2 - y_0)^2)^2}\right) ,$$

(64)

$$y_2(x_1, x_2) = 2\sqrt{5}\frac{(x_1 - x_0)(x_2 - y_0)}{((x_1 - x_0)^2 + (x_2 - y_0)^2)^2} .$$

Example 4. Example 4 is Example 2 modified by a rescaling of x_2-variable to satisfy

(65)
$$10\frac{\partial^2 f}{\partial {x_1}^2} + \frac{\partial^2 f}{\partial {x_2}^2} = 0 .$$

Then the data function is given by

(66)
$$f(x_1, x_2) = \Re e\ 1/(\tilde{z} - z_0)^2$$
$$= \frac{(x_1 - x_0)^2 - (\sqrt{10}x_2 - y_0)^2}{((x_1 - x_0)^2 + (\sqrt{10}x_2 - y_0)^2)^2} ,$$

where

$$\tilde{z} = x_1 + i\sqrt{10}x_2 .$$

The results of the numerical experiments are summarized in Tables 1–4. Error profiles for the four problems are contained in Figs. 2–5. Meshes obtained by transformation of optimal-shape triangles show an almost level error profile and remarkable equilibrating properties in agreement with theoretical prediction (see Figs. 6–9). Meshes produced by `D03MAF` by tiling the transformed domain by equilateral triangles are shown in Figs. 10–13. The corresponding meshes produced by `PLTMG` are shown in Figs. 14–17. The error profiles for the equilateral meshes show a level curve with an abrupt increase in error. The anomalous increases are caused by distortion of triangle geometry along the boundary (see Figs. 18–23). Note that by the maximum principle, extremal values of harmonic functions also occur along the boundary.

Although PLTMG produces meshes without the equilibrating property and incurs larger maximum errors, for the purpose of finite element computation, such meshes can be considered to be "near" optimal since their errors are within a small multiple of the theoretical optimum. From another perspective, the error tolerance for meshes produced by PLTMG can be similarly obtained by meshes produced through coordinate transformation with only a fraction of the number of triangles.

6. Conclusion. A basic study on optimal meshes has been undertaken with the intention of providing insight into more complex optimal mesh problems in finite element computation. The motivation for and development of coordinate transformation for generating optimally efficient meshes have been presented. The effectiveness of such a strategy is demonstrated by the numerical examples. We conclude that mesh generation by coordinate transformation is a promising strategy in the investigation of optimal meshes.

The existence of a global coordinate transformation for optimally efficient mesh generation relies on $f(x_1, x_2)$ satisfying

$$(67)\quad 0 = g_{11}({g_{122}}^2 - g_{222}g_{112}) + g_{12}(g_{111}g_{222} - g_{112}g_{122}) + g_{22}({g_{112}}^2 - g_{122}g_{111}) \ .$$

An interesting question is whether the class of functions in $\mathcal{C}^3$ that satisfy (67) is dense in the L_∞ norm. Note that harmonic functions are in $\mathcal{C}^3$ but not dense, and piecewise quadratic functions are dense but not in $\mathcal{C}^3$.

TABLE 1
Summary of results for Example 1.

$f(x_1, x_2) = \exp(5x_1)\sin(5x_2)$					
	Minimum error	Median error	90 percentile	Maximum error	Number of triangles
Optimal	5.273d-2	5.390d-2	5.497d-2	5.740d-2	2923
Equilateral	1.943d-2	6.644d-2	6.793d-2	14.88d-2	3190
PLTMG	1.912d-2	7.600d-2	14.10d-2	21.78d-2	3080

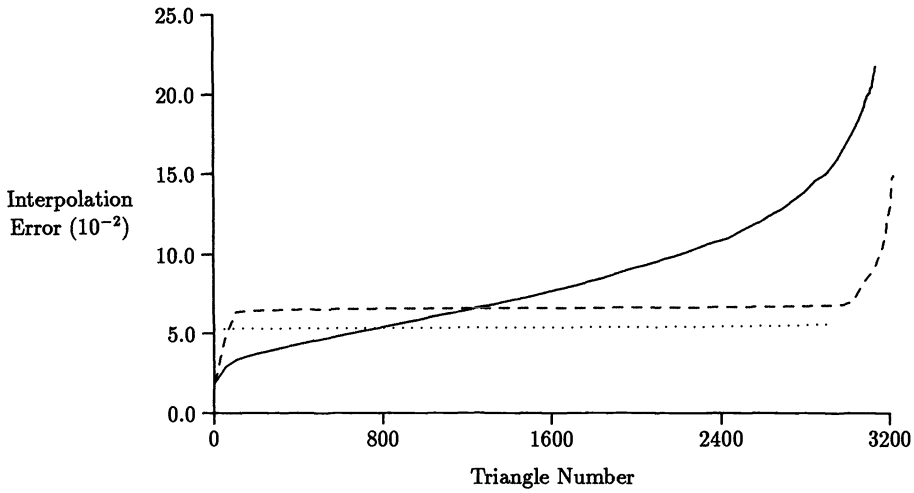


FIG. 2. *Error profiles for Example* 1. — PLTMG; - - *equilateral;* ··· *optimal shape.*

TABLE 2
Summary of results for Example 2.

$f(x_1, x_2) = \Re e\ 1/(z - z_0)^2$					
	Minimum error	Median error	90 percentile	Maximum error	Number of triangles
Optimal	1.874d-2	2.009d-2	2.155d-2	2.566d-2	1072
Equilateral	1.327d-2	2.833d-2	3.420d-2	6.693d-2	1035
PLTMG	0.849d-2	2.890d-2	5.213d-2	8.426d-2	1168

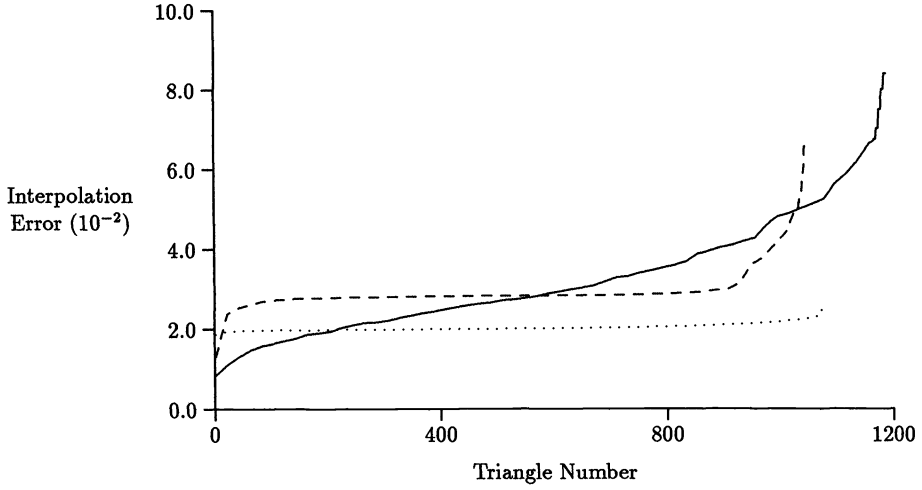


FIG. 3. *Error profiles for Example 2.* — PLTMG; - - *equilateral;* ··· *optimal shape.*

TABLE 3
Summary of results for Example 3.

$f(x_1, x_2) = \Re e\ 1/(z - z_0)^4$					
	Minimum error	Median error	90 percentile	Maximum error	Number of triangles
Optimal	1.021	1.160	1.322	1.696	650
Equilateral	0.535	1.737	2.136	3.566	595
PLTMG	0.145	1.689	3.055	5.184	720

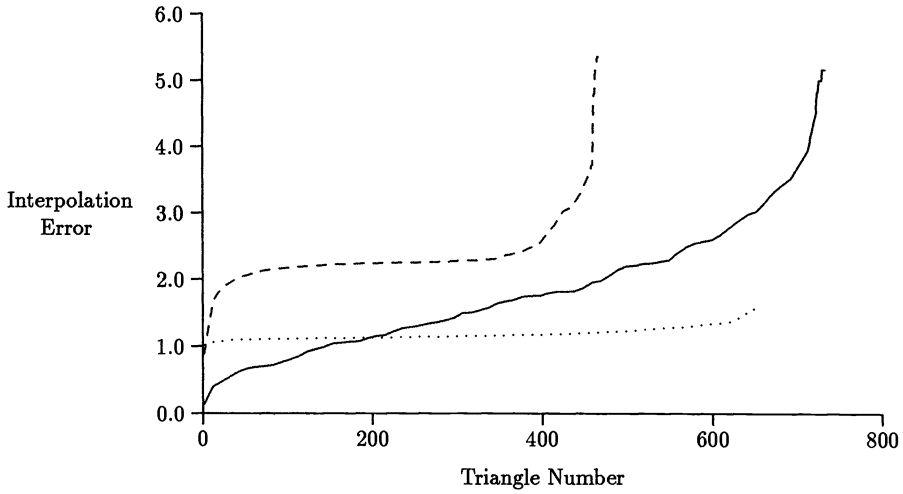


FIG. 4. *Error profiles for Example* 3. — PLTMG; - - *equilateral;* ··· *optimal shape.*

TABLE 4
Summary of results for Example 4.

$f(x_1,x_2) = \Re e\ 1/(\tilde{z}-z_0)^2\,, \quad \tilde{z} = x_1 + i\sqrt{10}x_2$					
	Minimum error	Median error	90 percentile	Maximum error	Number of triangles
Optimal	2.906d-2	3.681d-2	4.605d-2	6.462d-2	608
Equilateral	2.593d-2	6.283d-2	9.187d-2	16.71d-2	469
PLTMG	1.683d-2	8.571d-2	19.51d-2	40.42d-2	686

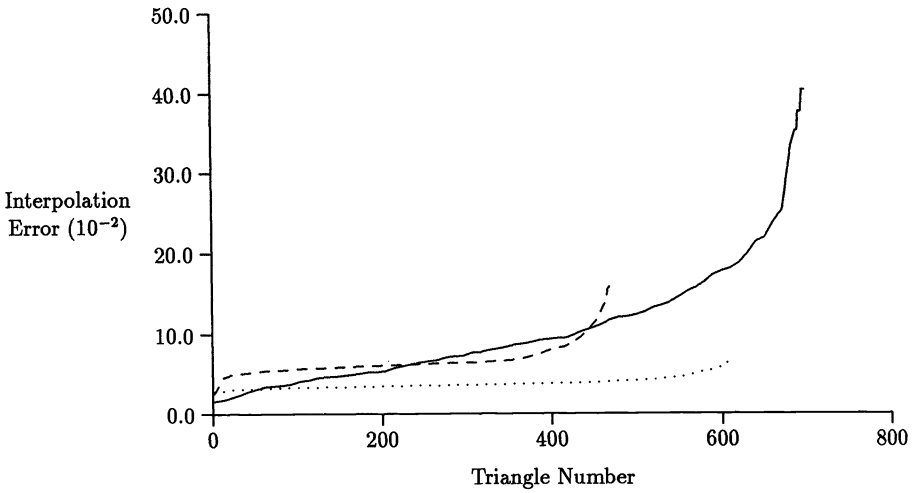


FIG. 5. *Error profiles for Example* 4. — PLTMG; - - *equilateral;* ··· *optimal shape.*

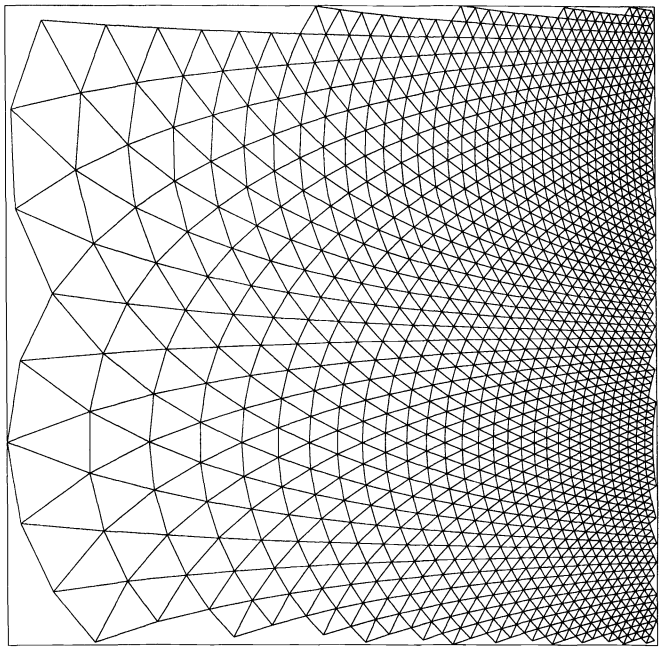


FIG. 6. *Optimally efficient triangulation for Example* 1.

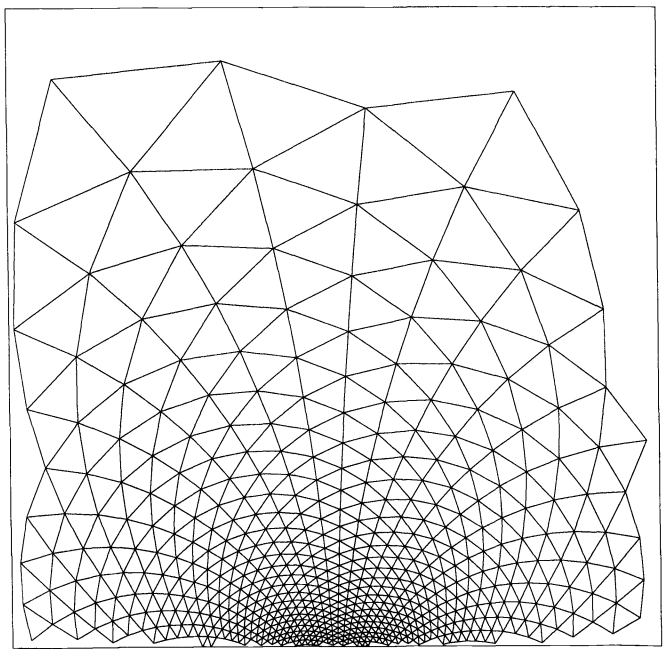


FIG. 7. *Optimally efficient triangulation for Example* 2.

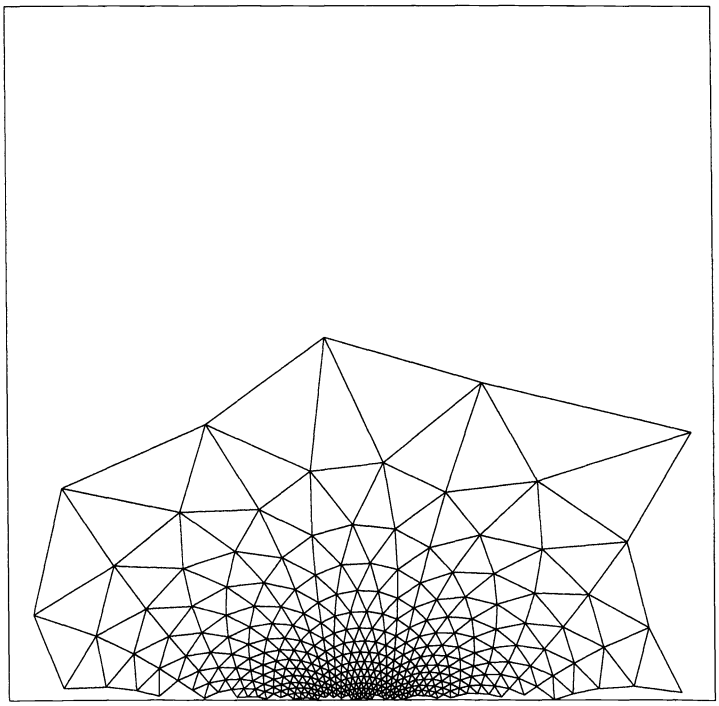


FIG. 8. *Optimally efficient triangulation for Example* 3.

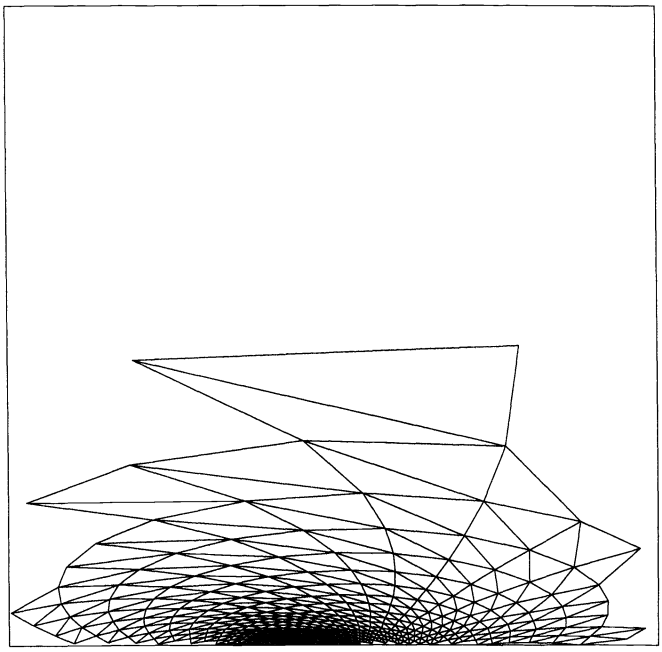


FIG. 9. *Optimally efficient triangulation for Example* 4.

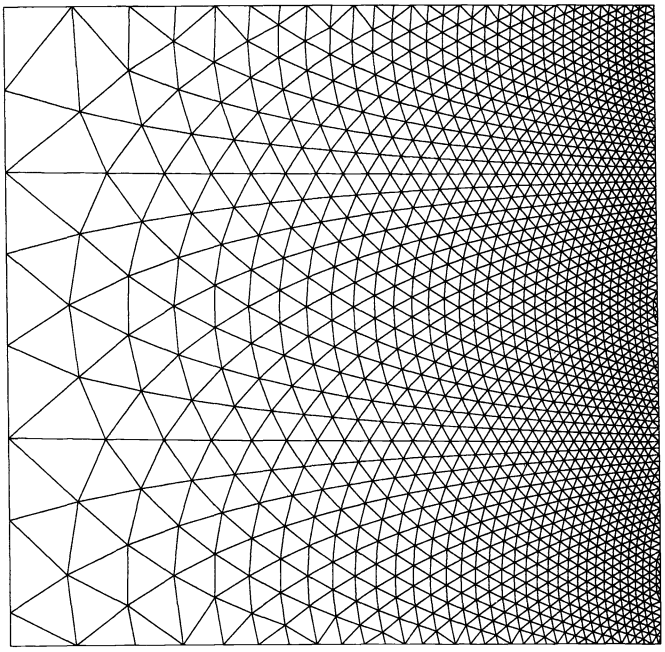


FIG. 10. *Mesh produced by equilateral triangles for Example* 1.

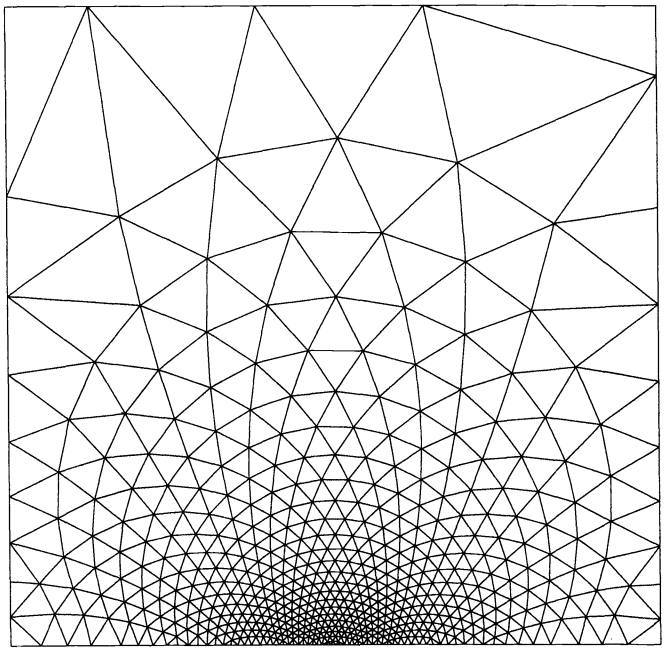


FIG. 11. *Mesh produced by equilateral triangles for Example* 2.

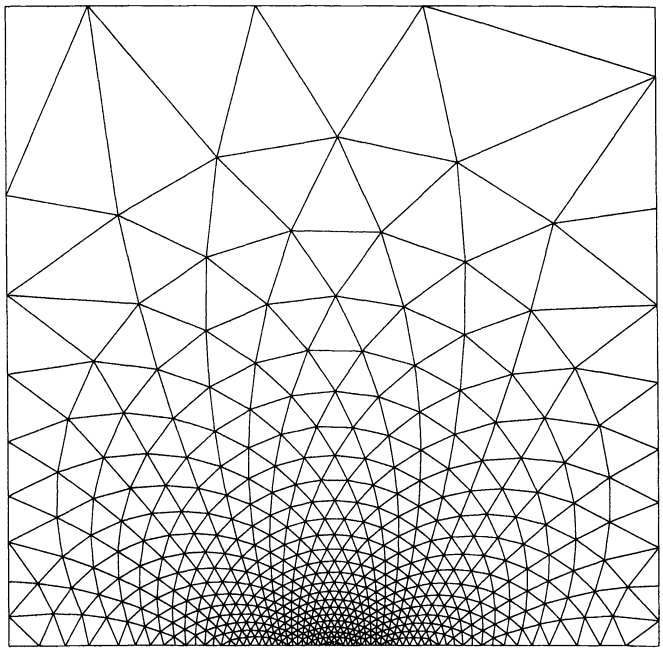


FIG. 12. *Mesh produced by equilateral triangles for Example* 3.

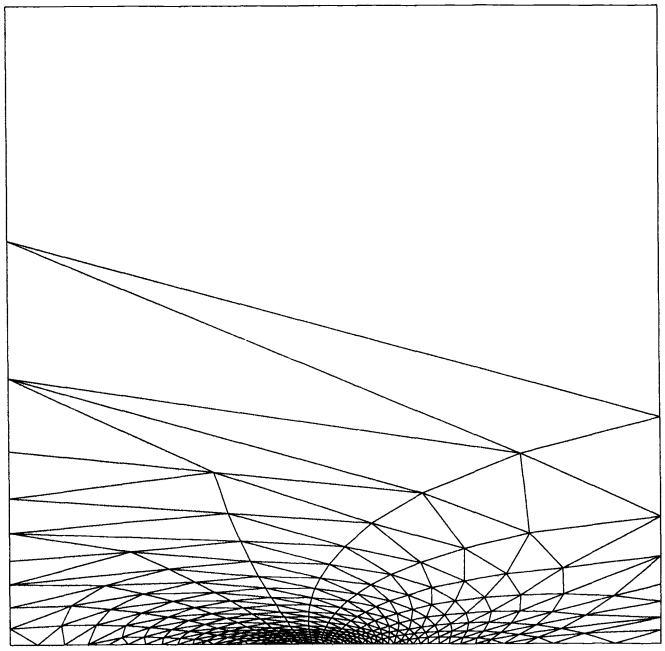


FIG. 13. *Mesh produced by equilateral triangles for Example* 4.

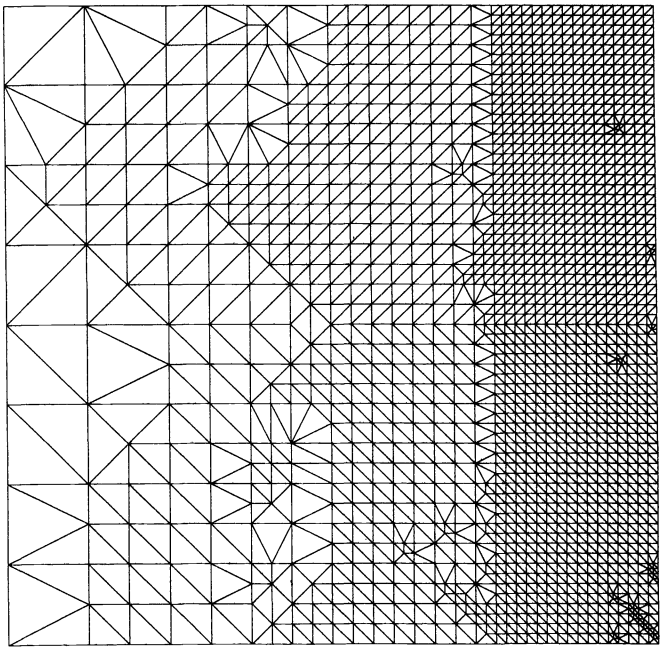


FIG. 14. *Mesh produced by* PLTMG *for Example* 1.

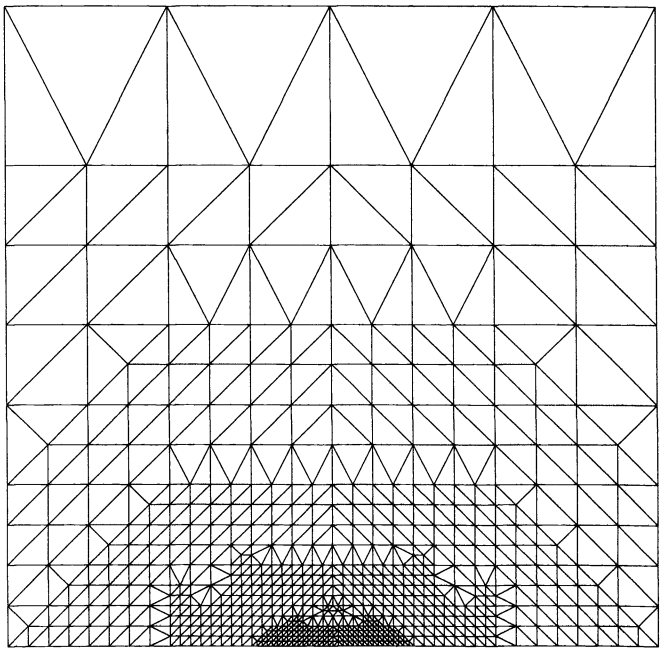


FIG. 15. *Mesh produced by* PLTMG *for Example* 2.

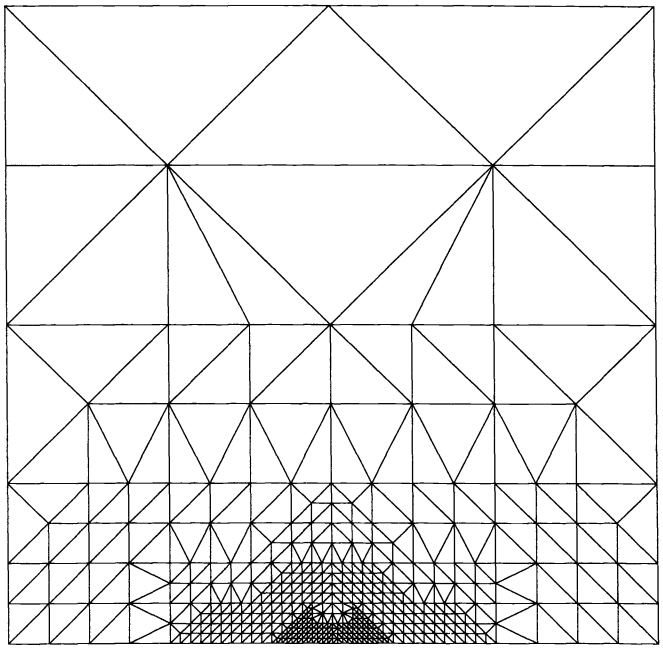


FIG. 16. *Mesh produced by* PLTMG *for Example* 3.

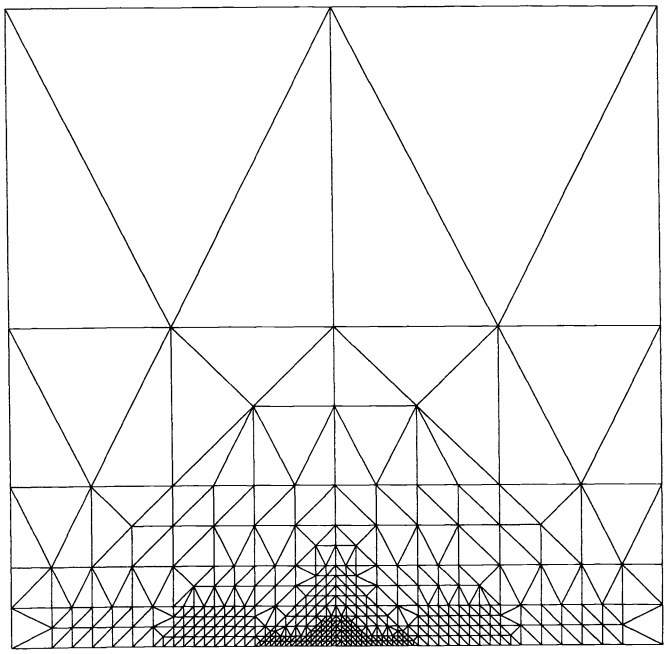


FIG. 17. *Mesh produced by* PLTMG *for Example* 4.

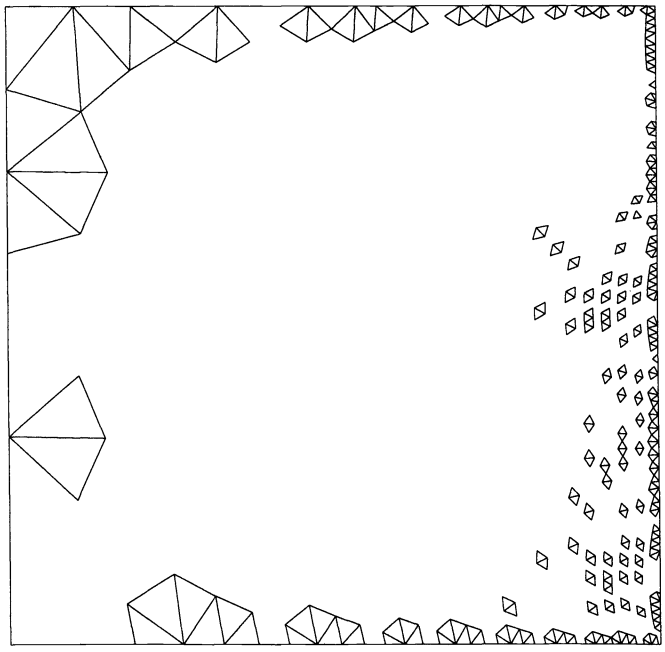


FIG. 18. *Top* 10 *percent error of equilateral triangles for Example* 1.

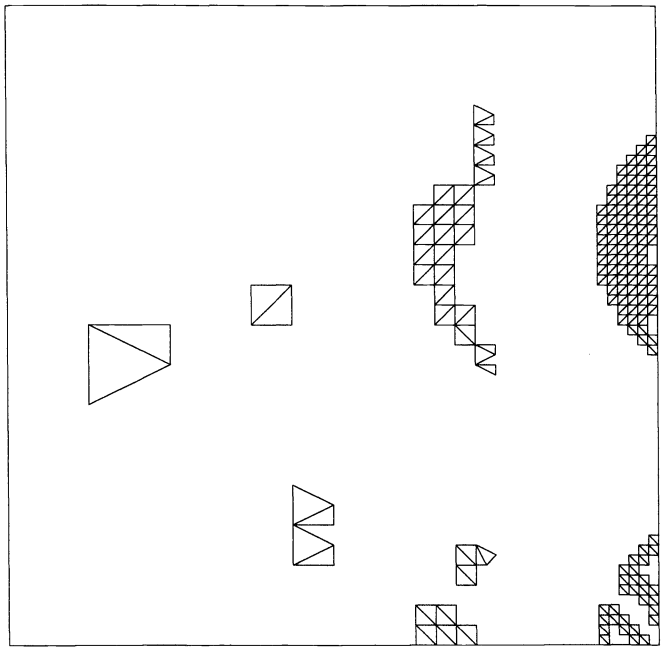


FIG. 19. *Top* 10 *percent error of* PLTMG *for Example* 1.

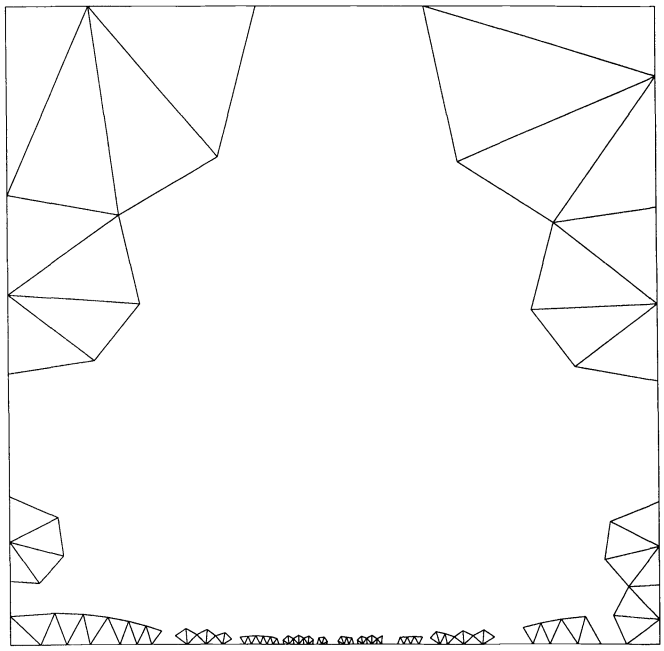


FIG. 20. *Top* 10 *percent error of equilateral triangles for Example* 2.

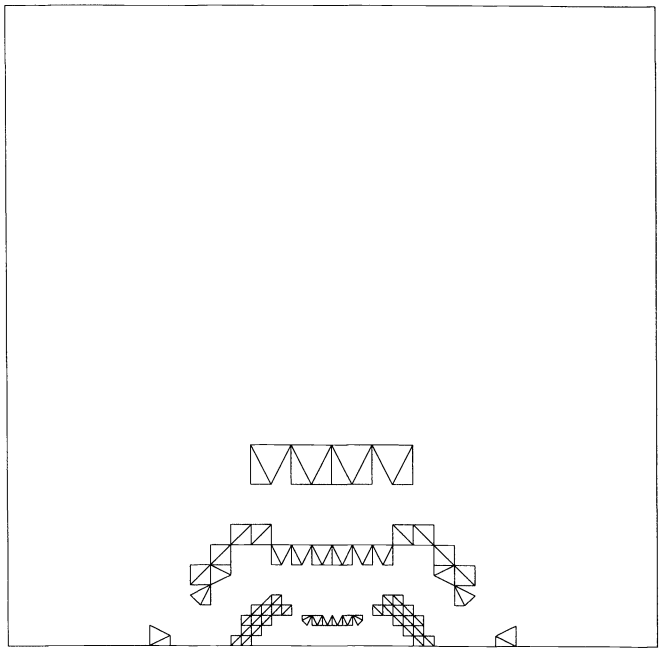


FIG. 21. *Top* 10 *percent error of* PLTMG *for Example* 2.

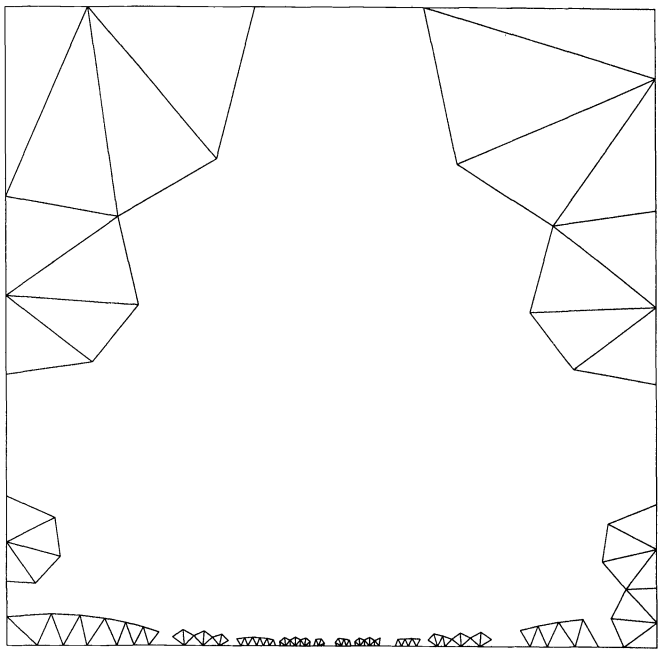


FIG. 22. *Top* 10 *percent error of equilateral triangles for Example* 3.

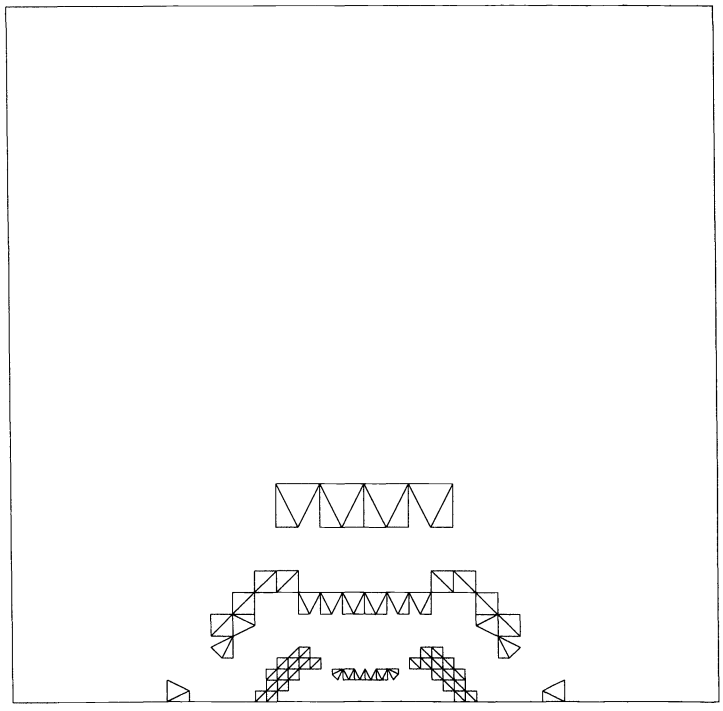


FIG. 23. *Top* 10 *percent error of* PLTMG *for Example* 3.

Appendix A. In this section, we derive the error expression $E(x,y)$ obtained in the interpolation of a quadratic function

$$f(x,y) = \tfrac{1}{2}(x,y)H(x,y)^t + d_1x + d_2y + d_3 \ , \tag{68}$$

where matrix H is symmetric and the triangle vertices are at (x_1,y_1), (x_2,y_2), (x_3,y_3). Let $p_l(x,y)$ be the linear interpolating function and the error function be given as

$$\begin{aligned} E(x,y) &= p_l(x,y) - f(x,y) \\ &= -\tfrac{1}{2}(x,y)H(x,y)^t + b_1x + b_2y + b_3 \ . \end{aligned} \tag{69}$$

The errors at the triangle vertices are given by

$$E(x_1,y_1) = E(x_2,y_2) = E(x_3,y_3) = 0 \ . \tag{70}$$

The unknowns b_1, b_2, b_3 can be obtained easily from (70) by solving the following:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix} \ , \qquad r_i = \tfrac{1}{2}(x_i,y_i)H(x_i,y_i)^t \ . \tag{71}$$

The solution to the above system of linear equations is

$$\begin{aligned} 2Ab_1 &= r_1(y_2-y_3) + r_2(y_3-y_1) + r_3(y_1-y_2) \ , \\ 2Ab_2 &= r_1(x_3-x_2) + r_2(x_1-x_3) + r_3(x_2-x_1) \ , \\ 2Ab_3 &= r_1(x_2y_3-x_3y_2) + r_2(x_3y_1-x_1y_3) + r_3(x_1y_2-x_2y_1) \ , \end{aligned} \tag{72}$$

where A is the area of the triangle

$$A = \tfrac{1}{2}\left(x_1(y_2-y_3) + x_2(y_3-y_1) + x_3(y_1-y_2)\right) \ . \tag{73}$$

THEOREM 6.1. *The curves of constant error of* (69) *form a family of conics with a common center at* (x_c,y_c), *where* $H(x_c,y_c)^t = (b_1,b_2)^t$. *It is a family of ellipses if* $\det(H) > 0$, *and a family of hyperbolas if* $\det(H) < 0$. *Moreover, the error at a displacement* (dx,dy) *from the center* (x_c,y_c) *is given by*

$$E(x_c+dx, y_c+dy) = E(x_c,y_c) - \tfrac{1}{2}(dx,dy)H(dx,dy)^t \ . \tag{74}$$

Proof. Let matrix H be diagonalized as

$$H = Q^t \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} Q \ , \tag{75}$$

with orthogonal matrix Q. Consider the error curve of value c,

$$\begin{aligned} c &= E(x,y) \\ &= -\tfrac{1}{2}(x,y)H(x,y)^t + b_1x + b_2y + b_3 \\ &= -\tfrac{1}{2}(x,y)Q^t \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} Q(x,y)^t + b_1x + b_2y + b_3 \ . \end{aligned} \tag{76}$$

To reduce (76) to canonical form, we perform a rotation through Q to align eigendirections of H along the coordinate axes. Let $(\bar{x},\bar{y})$ be the transformed coordinates,

$$\begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix} = Q \begin{bmatrix} x \\ y \end{bmatrix} \ , \qquad \begin{bmatrix} \bar{b}_1 \\ \bar{b}_2 \end{bmatrix} = Q \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \ , \tag{77}$$

then from (76),

$$
\begin{aligned}
& c = -\tfrac{1}{2}(\lambda_1\bar{x}^2 + \lambda_2\bar{y}^2) + \bar{b}_1\bar{x} + \bar{b}_2\bar{y} + b_3 \ , \\
(78) \qquad & 2c = -\left(\lambda_1(\bar{x} - \bar{b}_1/\lambda_1)^2 + \lambda_2(\bar{y} - \bar{b}_2/\lambda_2)^2\right) + 2b_3 + \bar{b}_1^2/\lambda_1 + \bar{b}_2^2/\lambda_2 \ , \\
& 2c = -\left(\lambda_1(\bar{x} - \bar{b}_1/\lambda_1)^2 + \lambda_2(\bar{y} - \bar{b}_2/\lambda_2)^2\right) + 2\mathcal{E} \ ,
\end{aligned}
$$

where

$$
\mathcal{E} = b_3 + \tfrac{1}{2}\left(\bar{b}_1^2/\lambda_1 + \bar{b}_2^2/\lambda_2\right) \ . \tag{79}
$$

Then (78) can be rewritten as

$$
\lambda_1(\bar{x} - \bar{x}_c)^2 + \lambda_2(\bar{y} - \bar{y}_c)^2 = 2(\mathcal{E} - c) \ , \tag{80}
$$

or

$$
\frac{(\bar{x} - \bar{x}_c)^2}{2(\mathcal{E} - c)/\lambda_1} + \frac{(\bar{y} - \bar{y}_c)^2}{2(\mathcal{E} - c)/\lambda_2} = 1 \ , \tag{81}
$$

where

$$
\bar{x}_c = \bar{b}_1/\lambda_1 \ , \qquad \bar{y}_c = \bar{b}_2/\lambda_2 \ . \tag{82}
$$

Equation (81) describes an ellipse in canonical form with center at $(\bar{x}_c, \bar{y}_c)$ if $\lambda_1\lambda_2 = \det(H) > 0$ and a hyperbola if $\det(H) < 0$.

Let (x_c, y_c) be the coordinates of $(\bar{x}_c, \bar{y}_c)$ before the rotation; then

$$
Q\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} \bar{x}_c \\ \bar{y}_c \end{bmatrix} = \begin{bmatrix} \bar{b}_1/\lambda_1 \\ \bar{b}_2/\lambda_2 \end{bmatrix} = \begin{bmatrix} 1/\lambda_1 & 0 \\ 0 & 1/\lambda_2 \end{bmatrix} Q\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \ , \tag{83}
$$

and thus,

$$
Q^t\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} Q\begin{bmatrix} x_c \\ y_c \end{bmatrix} = H\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} . \tag{84}
$$

Immediately, we have $\mathcal{E} = E(x_c, y_c)$ from (78).

The error at a displacement (dx, dy) from the center (x_c, y_c) is given by

$$
\begin{aligned}
E(x, y) &= E(x_c + dx, y_c + dy) \\
&= -\tfrac{1}{2}(x_c + dx, y_c + dy)H(x_c + dx, y_c + dy)^t \\
&\qquad + b_1(x_c + dx) + b_2(y_c + dy) + b_3 \\
&= -\tfrac{1}{2}(x_c, y_c)H(x_c, y_c)^t - \tfrac{1}{2}(dx, dy)H(dx, dy)^t \\
&\qquad - (dx, dy)H(x_c, y_c)^t + b_1(x_c + dx) + b_2(y_c + dy) + b_3 \\
&= E(x_c, y_c) - \tfrac{1}{2}(dx, dy)H(dx, dy)^t \\
&= \mathcal{E} - \tfrac{1}{2}(dx, dy)H(dx, dy)^t \ .
\end{aligned} \tag{85}
$$

Hence the theorem follows. □

Appendix B. In this section, we find the geometry of an optimally efficient triangle for approximation of a saddle shape function. We recall from §2 that for a triangle over the transformed plane with vertices at $(\tilde{x}_1, \tilde{y}_1)$, $(\tilde{x}_2, \tilde{y}_2)$, $(\tilde{x}_3, \tilde{y}_3)$, the local maximum error along edge $(\tilde{x}_i, \tilde{y}_i)$, $(\tilde{x}_j, \tilde{y}_j)$ is

$$E_{ij} = \frac{1}{8} \left| (\tilde{x}_i - \tilde{x}_j)^2 - (\tilde{y}_i - \tilde{y}_j)^2 \right| . \tag{86}$$

The triangle geometry shown in Fig. 1 attains the same error on all three sides:

$$E_{12} = E_{23} = E_{31} . \tag{87}$$

We can show that this equality of error on all sides is a necessary condition for an optimally efficient triangle. Assume the contrary that the error on one edge is strictly smaller,

$$E_{12} < E_{23} , \qquad E_{12} < E_{31} . \tag{88}$$

Then $(\tilde{x}_1, \tilde{y}_1)$ can be perturbed along the hyperbola given by

$$(x - \tilde{x}_3)^2 - (y - \tilde{y}_3)^2 = (\tilde{x}_1 - \tilde{x}_3)^2 - (\tilde{y}_1 - \tilde{y}_3)^2 , \tag{89}$$

with no changes to E_{23} nor E_{31} (see Fig. 25). We note that if the tangent of the hyperbola at $(\tilde{x}_1, \tilde{y}_1)$ is not parallel to edge $(\tilde{x}_2, \tilde{y}_2)$, $(\tilde{x}_3, \tilde{x}_3)$, the area of this triangle increases as $(\tilde{x}_1, \tilde{y}_1)$ is perturbed. The maximum error is unchanged but the area can be increased by such a perturbation, thus it cannot be optimally efficient. The optimal case occurs when the tangent at $(\tilde{x}_1, \tilde{y}_1)$ is parallel to edge $(\tilde{x}_2, \tilde{y}_2)$, $(\tilde{x}_3, \tilde{y}_3)$ (see Fig. 26).

Let us consider the problem of finding a triangle geometry which minimizes the squared ratio of error to area and subject to condition (87) that all edges attain the same error. Let a triangle of arbitrary shape be described as having vertices at $(0, 0)$, $(\cos\theta, \sin\theta)$, (x, y). Then the above conditions can be formulated as

$$\min \left(\frac{\text{Error}}{\text{Area}} \right)^2 = \min \frac{(\cos^2\theta - \sin^2\theta)^2}{4(y\cos\theta - x\sin\theta)^2} , \tag{90}$$

subject to

$$x^2 - y^2 = \epsilon_1 (\cos^2\theta - \sin^2\theta) , \qquad |\epsilon_1| = 1 , \tag{91}$$

and

$$(x - \cos\theta)^2 - (y - \sin\theta)^2 = \epsilon_2 (\cos^2\theta - \sin^2\theta) , \qquad |\epsilon_2| = 1 . \tag{92}$$

Now from (92)

$$\begin{aligned} \epsilon_2(\cos^2\theta - \sin^2\theta) &= (x^2 - 2x\cos\theta + \cos^2\theta) - (y^2 - 2y\sin\theta + \sin^2\theta) \\ &= (x^2 - y^2) + (\cos^2\theta - \sin^2\theta) - 2(x\cos\theta - y\sin\theta) . \end{aligned} \tag{93}$$

Hence (92) can be simplified to

$$2(x\cos\theta - y\sin\theta) = (\cos^2\theta - \sin^2\theta)(1 + \epsilon_1 - \epsilon_2) . \tag{94}$$

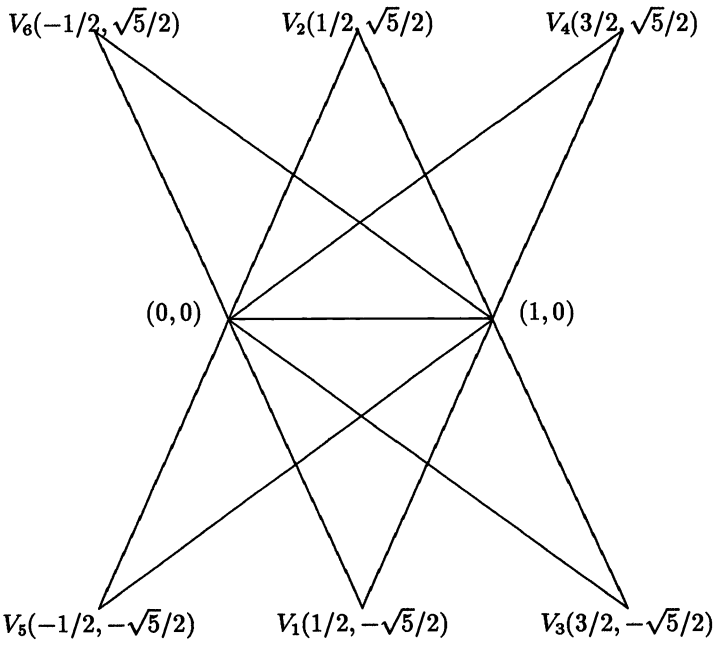


FIG. 24. *Optimal shape triangles for* $\theta = 0$.

Solving (94) and (91) yields two solutions for (x, y) given as

$$(95) \qquad \begin{aligned} (x,y) &= (K/2\cos\theta - \sqrt{R}/2\sin\theta, K/2\sin\theta - \sqrt{R}/2\cos\theta)\ , \\ (x,y) &= (K/2\cos\theta + \sqrt{R}/2\sin\theta, K/2\sin\theta + \sqrt{R}/2\cos\theta)\ , \end{aligned}$$

where

$$(96) \qquad K = (1 + \epsilon_1 - \epsilon_2)\ , \qquad R = 1 - 2(\epsilon_1 + \epsilon_2) + (\epsilon_1 - \epsilon_2)^2\ .$$

For both values of (x, y), formula (90) reduces to

$$(97) \qquad \frac{(\cos^2\theta - \sin^2\theta)^2}{4(y\cos\theta - x\sin\theta)^2} = \frac{16}{1 - 2(\epsilon_1 + \epsilon_2) + (\epsilon_1 - \epsilon_2)^2} = \frac{16}{R}\ .$$

Expression R equals 5 except for $(\epsilon_1, \epsilon_2) = (1, 1)$ where R is -3, which is impossible.

Thus for each orientation of angle θ, there are six configurations for an optimal-shape triangle. The results are summarized in Table 5. In particular, for zero rotation ($\theta = 0$), the triangle shapes are displayed in Fig. 24. Note the triangle geometry shown in Fig. 1 corresponds to triangle with vertex V_2.

TABLE 5
Summary of solutions for vertex (x, y).

$\epsilon_1 = -1$	$V_1 : \frac{1}{2}(-\sqrt{5}\sin\theta + \cos\theta, -\sqrt{5}\cos\theta + \sin\theta)$
$\epsilon_2 = -1$	$V_2 : \frac{1}{2}(\sqrt{5}\sin\theta + \cos\theta,\ \sqrt{5}\cos\theta + \sin\theta)$
$\epsilon_1 = 1$	$V_3 : \frac{1}{2}(-\sqrt{5}\sin\theta + 3\cos\theta, 3\sin\theta - \sqrt{5}\cos\theta)$
$\epsilon_2 = -1$	$V_4 : \frac{1}{2}(\sqrt{5}\sin\theta + 3\cos\theta, 3\sin\theta + \sqrt{5}\cos\theta)$
$\epsilon_1 = -1$	$V_5 : \frac{1}{2}(-\sqrt{5}\sin\theta - \cos\theta, -\sin\theta - \sqrt{5}\cos\theta)$
$\epsilon_2 = 1$	$V_6 : \frac{1}{2}(\sqrt{5}\sin\theta - \cos\theta, -\sin\theta + \sqrt{5}\cos\theta)$

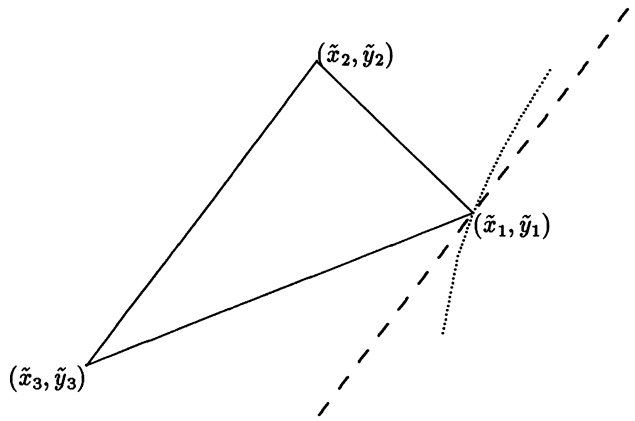


FIG. 25. *Tangent at* $(\tilde{x}_1, \tilde{y}_1)$ *is not parallel to edge* $(\tilde{x}_2, \tilde{y}_2)$, $(\tilde{x}_3, \tilde{y}_3)$.

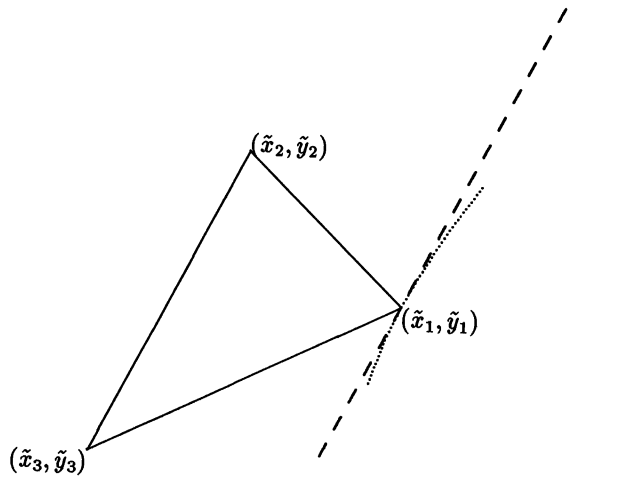


FIG. 26. *Tangent at* $(\tilde{x}_1, \tilde{y}_1)$ *is parallel to edge* $(\tilde{x}_2, \tilde{y}_2)$, $(\tilde{x}_3, \tilde{y}_3)$.

Acknowledgment. The author thanks his thesis supervisor, Professor R. B. Simpson, for his advice and helpful comments in clarifying the concept of optimality in the preparation of this manuscript.

REFERENCES

[1] R. E. BANK, PLTMG: *A Software Package for Solving Elliptic Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[2] E. F. D'AZEVEDO, *On optimal triangulation for piecewise linear approximation*, Ph.D. thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1989.

[3] E. F. D'AZEVEDO AND R. B. SIMPSON, *On optimal interpolation incidences*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1063–1075.

[4] C. DE BOOR, *Good approximation by splines with variable knots*, in Spline Functions and Approximation Theory, A. Meir and A. Sharma, eds., Springer-Verlag, Berlin, New York, 1973, pp. 57–71.

[5] M. DELFOUR, G. PAYRE, AND J.-P. ZOLESIO, *An optimal triangulation for second-order elliptic problems*, Comput. Methods Appl. Mech. Engrg., 50 (1985), pp. 231–261.

[6] A. R. DIAZ, N. KIKUCHI, AND J. E. TAYLOR, *A method of grid optimization for the finite element method*, Comput. Methods Appl. Mech. Engrg., 41 (1983), pp. 29–45.

[7] C. A. FELIPPA, *Optimization of finite element grids by direct energy search*, Appl. Math. Modelling, 1 (1976), pp. 93–96.

[8] ———, *Numerical experiments in finite element grid optimization by direct energy search*, Appl. Math. Modelling, 1 (1977), pp. 239–244.

[9] R. LÖHNER, K. MORGAN, AND O. C. ZIENKIEWICZ, *Adaptive grid refinement for the compressible Euler equations*, in Accuracy Estimates and Adaptive Refinements in Finite Element Computations, I. Babuška, O. C. Zienkiewicz, J. Gago, and E. R. de Arantes e Oliveira, eds., Wiley-Interscience, New York, 1986, pp. 281–297.

[10] M. B. MCGIRR, D. J. H. CORDEROY, A. K. HELLIER, AND P. C. EASTERBROOK, *The performance of an automatic, self-adaptive finite element technique*, in Computational Techniques & Applications: CTAC-83, J. Noye and C. Fletcher, eds., Elsevier Science Publishers B. V., North-Holland, 1984, pp. 236–240.

[11] M. B. MCGIRR AND P. KRAUKLIS, *The automatic generation of arrays of nodes with varying density*, in Computational Techniques & Applications: CTAC-83, J. Noye and C. Fletcher, eds., Elsevier Science Publishers B. V., North-Holland, 1984, pp. 229–235.

[12] E. NADLER, *Piecewise linear best L_2 approximation on triangulations*, in Approximation Theory V, C. K. Chui, L. L. Schumaker, and J. D. Ward, eds., Academic Press, Boston, 1986, pp. 499–502.

[13] J. PÉRAIRE, M. VAHDATI, K. MORGAN, AND O. C. ZIENKIEWICZ, *Adaptive remeshing for compressible flow computations*, J. Comput. Phys., 72 (1987), pp. 449–466.

[14] M. S. SHEPHARD, *Finite element grid optimization—a review*, in Finite Element Grid Optimization, M. S. Shephard and R. H. Gallagher, eds., The American Society of Mechanical Engineers, New York, 1979.

[15] M. S. SHEPHARD AND R. H. GALLAGHER, *Finite Element Grid Optimization*, The American Society of Mechanical Engineers, New York, 1979.

[16] M. S. SHEPHARD, R. H. GALLAGHER, AND J. F. ABEL, *Experience with interactive computer graphics for the synthesis of optimal finite element meshes*, in Finite Element Grid Optimization, M. S. Shephard and R. H. Gallagher, eds., The American Society of Mechanical Engineers, New York, 1979.

[17] I. S. SOKOLNIKOFF, *Tensor Analysis, Theory and Applications to Geometry and Mechanics of Continua*, John Wiley, New York, Second Edition, 1964.

[18] J. L. SYNGE AND A. SCHILD, *Tensor Calculus*, University of Toronto Press, Toronto, Ontario, Canada, 1949.

[19] J. F. THOMPSON, *Numerical Grid Generation*, North-Holland, Amsterdam, 1982.

[20] J. F. THOMPSON, Z. U. A. WARSI, AND C. W. MASTIN, *Boundary-fitted coordinate systems for numerical solution of partial differential equations—a review*, J. Comput. Phys., 47 (1982), pp. 1–108.

[21] ———, *Numerical Grid Generation: Foundations and Applications*, North-Holland, New York, 1985.

[22] A. B. WHITE, JR., *On selection of equidistributing meshes for two-point boundary-value problems*, SIAM J. Numer. Anal., 16 (1979), pp. 472–502.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 787-799, July 1991
© 1991 Society for Industrial and Applied Mathematics
004

PARALLEL TRIANGULAR SYSTEM SOLVING ON A MESH NETWORK OF TRANSPUTERS*

ROB H. BISSELING† AND JOHANNES G. G. VAN DE VORST†

Abstract. A parallel algorithm is presented for triangular system solving on a distributed-memory MIMD computer with a square mesh topology. The algorithm is based on the square grid (scattered) distribution of matrix elements across the processors. The theoretical time complexity is $n^2/p + O(n)$, for p processors and an $n \times n$ matrix. Experimental timings of an implementation in occam 2 on a square mesh of $p = 36$ transputers confirm the theoretical time model. The scaled speedup achieved for $n = 1200$ is 24 on a 36 transputer mesh. This corresponds to a computing rate of 11.7 Mflop/s.

Key words. parallel algorithms, triangular systems, transputers

AMS(MOS) subject classifications. 65F05, 65W05

1. Introduction. This paper presents a parallel algorithm for the solution of the lower triangular system

$$Lx = b \tag{1.1}$$

on a distributed-memory MIMD computer with a square mesh topology. The matrix $L = (L_{ij}, 0 \leqq i, j < n)$ is a dense $n \times n$ unit lower triangular matrix ($L_{ij} = 0$ for $i < j$, and $L_{ii} = 1$), and $x = (x_i, 0 \leqq i < n)$ and $b = (b_i, 0 \leqq i < n)$ are vectors of length n.

The solution of triangular systems is an important part of scientific computing: triangular solvers are often included in linear system solvers, and these in turn lie at the heart of many scientific codes. In certain situations, the portion of overall computing time consumed by the triangular solver may be large. Developing an efficient parallel triangular solver for a distributed-memory MIMD computer is a particularly difficult task, because it is very important to limit the amount of communication between the processors and to spread the work load evenly among them.

Recently, much effort has been devoted to the development of parallel triangular system solvers for distributed-memory MIMD computers [4], [5], [8]–[12], [15], [17]. These solvers can be divided into two classes, on the basis of their data distribution scheme. The first class of solvers [4], [8], [10], [11], [15] assumes a row- or column-oriented distribution of matrix elements across the processors. For a review of these solvers and an experimental comparison on a hypercube, see Heath and Romine [8].

The second class of solvers [5], [9], [12], [17] assumes the *square grid distribution* [16], defined by assigning matrix element L_{ij} to processor (i **mod** Q, j **mod** Q) of a $Q \times Q$ mesh. The algorithm described in the present paper falls into this class. The square grid distribution was introduced by Fox et al. [5] as *scattered square decomposition*, and by Johnsson [9] as *cyclic storage*. Fox et al. [5], [17] present an algorithm that is intended for banded triangular systems with multiple right-hand sides b. Asymptotically (for large problems) it achieves a full speedup of Q^2 on Q^2 processors, provided the number of right-hand sides is a multiple of Q. On the other hand, for a single right-hand side it only achieves an asymptotic speedup of Q. Our algorithm improves on this by achieving an asymptotic speedup of Q^2, for a single right-hand side. Johnsson [9] discusses the complexity of an algorithm which is similar to our algorithm, except for the fact that his algorithm assumes a hypercube topology, and

* Received by the editors October 24, 1988; accepted for publication (in revised form) May 22, 1990.
† Koninklijke/Shell-Laboratorium, Amsterdam, P.O. Box 3003, 1003 AA Amsterdam, the Netherlands.

makes use of spanning trees to broadcast newly computed components of x. A formal derivation of our algorithm for a completely connected network (the "QWERTY" algorithm) has been published elsewhere [12].

In many situations, the data distribution for the triangular solve cannot be chosen freely, because the data are already distributed as a result of previous calculations, and the cost of redistribution would be higher than the cost of the triangular solve itself. Our motivation in developing the algorithm described in this paper was the desire to solve triangular systems with L and U factors in square grid distribution, since this is the optimal distribution for the LU decomposition [1], [16]. Combining the parallel program for the decomposition $A = LU$ with the parallel programs solving the triangular systems $Lx = b$ and $Uy = x$, all with the square grid distribution, yields an efficient parallel program solving the linear system $Ay = b$.

It turns out that our grid-oriented algorithm for triangular system solving has good load balancing properties and a low communication overhead. A theoretical analysis shows that the computation complexity of the algorithm is $n^2/Q^2 + 5n$ floating point operations, and that the communication complexity is $4n$ neighbour-to-neighbour communications.

2. Outline of the algorithm. The purpose of this section is to establish the notation, state the assumptions, and present an outline of the parallel triangular system solving algorithm. The algorithm consists of the parallel composition of $p = Q^2$ processes (s, t), $0 \leqq s,\ t < Q$, each executing on one processor, denoted by (s, t). To simplify the exposition of the algorithm, it is assumed that n **mod** $Q = 0$. Each processor has a local memory. The processors use a square mesh communication network without wrap-around links to pass messages between them. Two processors, (s, t) and (s', t'), $0 \leqq s,\ s',\ t,\ t' < Q$, are therefore able to communicate if and only if $|s - s'| + |t - t'| = 1$. The communication mechanism may be either synchronous or asynchronous. The only assumption made is that message order is preserved between pairs of communicating processors. To simplify the following program text, it is assumed that communications are safeguarded against the crossing of mesh boundaries. For example, a processor (s, t) with $s = 0$ executes the statement "send y to process $(s-1, t)$" as "skip." Similarly, it executes the statement "receive y from process $(s-1, t)$" as "$y := 0$."

The vector $\hat{x}$ denotes the unique solution of (1.1). It satisfies

$$\hat{x}_i = b_i - \sum_{j=0}^{i-1} L_{ij}\hat{x}_j, \qquad 0 \leqq i < n, \tag{2.1}$$

because L is unit lower triangular. The matrix L is distributed across the processes according to the square grid distribution, which assigns element L_{ij}, $0 \leqq i, j < n$ to process $(s, t) = (i \textbf{ mod } Q, j \textbf{ mod } Q)$. Define an $n \times Q$ matrix $\hat{w}$, whose elements are the partial sums

$$\hat{w}_{it} = \sum_{\substack{j=0 \\ j \textbf{ mod } Q = t}}^{i-1} L_{ij}\hat{x}_j, \qquad 0 \leqq i < n, \qquad 0 \leqq t < Q. \tag{2.2}$$

Because of the square grid distribution of L, process (s, t) can compute the partial sums $\hat{w}_{it}$ with $0 \leqq i < n$ and i **mod** $Q = s$ locally without any communication of matrix elements L_{ij}, and hence only with the communication of values $\hat{x}_j$.

A variable vector x is manipulated by the program until it contains the solution $\hat{x}$. The distinction between the variable vector x and the constant vector $\hat{x}$ enables us to make assertions about computed or communicated values, such as "$x_i = \hat{x}_i$." It is

convenient to distribute the vectors x and b in the same way as the diagonal of L, i.e., elements x_i and b_i are assigned to process (i **mod** Q, i **mod** Q). An $n \times Q$ variable matrix w is used as working space to compute $\hat{w}$. This matrix is distributed in the obvious way, by assigning w_{it} to process (i **mod** Q, t). The distribution of L, w, b, and x is illustrated in Fig. 1.

The constants $\hat{x}$ and $\hat{w}$ are used only in assertions; they do not occur in the program text itself. The matrix L and the vector b are constants which are known at the start of the program, so that they can be used both in the program text and in assertions.

The following program consists of the parallel composition of Q^2 processes (s, t), each of which executes an initialisation, followed by a loop of n/Q steps, numbered $k = 0, Q, 2Q, \cdots, n-Q$. Each step consists of four phases, lettered (a)-(d). In step k of the program the values $\hat{x}_k, \hat{x}_{k+1}, \cdots, \hat{x}_{k+Q-1}$ are computed. To achieve this, every process (s, t) keeps two process *invariants* [6] true. The main process invariant is the logical expression $P_1[s, t]$,

$$P_1[s,t] \equiv \forall_i[(0 \leqq i < k \wedge i \textbf{ mod } Q = s \wedge s = t) \Rightarrow x_i = \hat{x}_i] \wedge$$
$$0 \leqq k \leqq n \wedge k \textbf{ mod } Q = 0. \tag{2.3}$$

At the start of the computation, i.e., for $k = 0$, and also for $s \neq t$, the invariant is trivially true. On the other hand, the truth of all process invariants $P_1[s, s]$, $0 \leqq s < Q$, for $k = n$ implies the successful termination of the program. An additional invariant $P_2[s, t]$

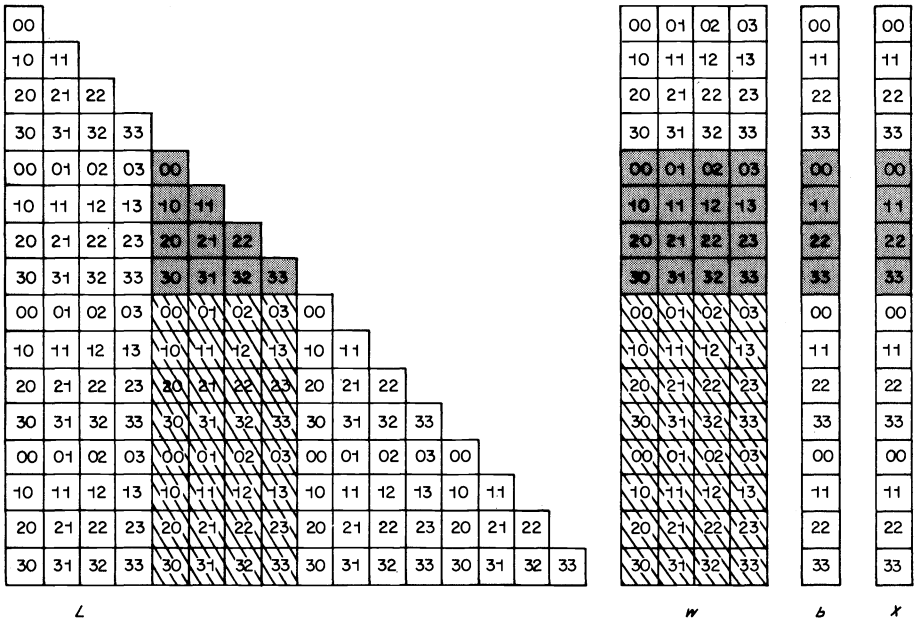


FIG. 1. *Matrix view of the distribution of data elements across the processors. The elements of the lower triangular part of L and the elements of w, b, and x are represented by cells. Each element is assigned to a processor* (s, t), $0 \leqq s, t < Q = 4$. *The order of L is* $n = 16$. *The number of processors is* $p = 16$. *The shaded elements of L, w, and b are used in phases* (a)-(c) *of step* $k = 4$ *to compute the shaded elements of x. These elements in turn, together with the hatched elements of L, are used in phase* (d) *to update the hatched elements of w.*

gives the relevant information about the current status of w,

$$P_2[s,t] \equiv \forall_i \left[(k \leqq i < n \wedge i \bmod Q = s) \Rightarrow w_{it} = \sum_{\substack{j=0 \\ j \bmod Q = t}}^{k-1} L_{ij}\hat{x}_j \right]. \tag{2.4}$$

In step k, variable w_{it} contains the accumulated sum of local values $L_{ij}\hat{x}_j$ for $j < k$. Such sums w_{it} are needed only for $i \geqq k$.

An outline of the algorithm is given by the following program text which records the status change caused by each phase, but which hides the details of the phase itself. For each phase, an annotated program text is given in § 3. For the sake of brevity, only the *changes* of the status are recorded in the comments between the four phases; comments which remain valid are not repeated. For example, in the program text below, $\{P_1[s,t] \wedge P_2[s,t]\}$ is valid prior to phase (a) and it will still be valid prior to phase (b). The program text is:

PROGRAM TEXT OF THE PARALLEL TRIANGULAR SYSTEM SOLVER.

```
S:              for all s, t: 0 ≦ s, t < Q par do process (s, t)
process (s, t): begin
                    for r := 0 to n − Q step Q do w_{r+s,t} := 0;
                    for k := 0 to n − Q step Q do
                    begin
                        {P1[s, t] ∧ P2[s, t] for k}
                        phase (a);
                        { w_right = Σ_{t≦q<Q} ŵ_{k+s,q} ∨ s > t }
                        phase (b);
                        {P1[s, t] for k := k + Q}
                        {y = x̂_{k+t} ∨ s < t}
                        phase (c);
                        {y = x̂_{k+t}}
                        phase (d)
                        {P1[s, t] ∧ P2[s, t] for k := k + Q}
                    end
                end.
```

The main loop of the program achieves the aim of the computation: at the start of the first step, $k = 0$, $P_1[s,t]$ is trivially true, and $P_2[s,t]$ is true after the initialisation $w := 0$. At the end of the last step, $k = n - Q$, the assertion $\{P_1[s,t] \wedge P_2[s,t]$ for $k := k + Q\}$ implies $P_1[s,t]$ for $k = n - Q + Q = n$. Since this holds for all s and t, it follows that all components of $\hat{x}$ have been computed. It remains to be shown (see § 3) that the phases of step k establish $P_1[s,t] \wedge P_2[s,t]$ for $k := k + Q$, starting from $P_1[s,t] \wedge P_2[s,t]$ for k.

In phase (a), the processes in the right upper corner of the mesh ($s \leqq t$) cooperate in computing $w_{\text{right}} = \sum_{t \leqq q < Q} \hat{w}_{k+s,q}$ for any of these processes. Phase (a) consists of Q independent fan-ins from right to left. In phase (b), the processes in the left lower corner ($s > t$) cooperate in computing $w_{\text{left}} = \sum_{0 \leqq q \leqq t} \hat{w}_{k+s,q}$ for any of these processes and $w_{\text{left}} = \sum_{0 \leqq q < t} \hat{w}_{k+s,q}$ for the diagonal processes ($s = t$). Each diagonal process uses its w_{left} and w_{right} to compute one new component of $\hat{x}$. This is expressed by the comment $\{P_1[s,t]$ for $k := k + Q\}$. During this phase, the Q newly computed components of $\hat{x}$ are communicated downwards from the diagonal. This is expressed by

the comment $\{y = \hat{x}_{k+t} \vee s < t\}$. Phase (c) takes care of the communication of the newly computed components of $\hat{x}$ in the upward direction from the diagonal. Finally, in phase (d) the working space w is updated. As each process possesses all the necessary values ($\hat{x}_{k+t}$ and part of the L-values), this phase does not require any communications. The use of data elements in these phases is illustrated in Fig. 1.

3. Phases of the algorithm. This section presents and verifies the four phases of one step of the algorithm. Each step has been divided into phases in such a way that no communication between processes in different phases is necessary. Therefore, each phase can be verified as a separate parallel program, which consists of the parallel composition of Q^2 processes. For each phase a *precondition* and a *postcondition* [6] is stated. The precondition of a phase equals the postcondition of the previous phase. To verify a phase it is necessary to check whether its postcondition follows from its precondition and from the execution of its program.

A phase is verified by using assertions in the process text. Similarly to the Gries–Owicki theory [13], [14], an assertion is made about the value of every communicated item. These values are expressed in global constants (such as $\hat{x}$ and $\hat{w}$) which do not depend upon local process variables. The processes are first checked in isolation, as a sequential program, by assuming that assertions about received values are true. The aim of this check is to show that the values computed and sent by any process are indeed the asserted ones. Since the processes are parametrised by s and t, this check involves only a single process text. (There is no need to check Q^2 different processes.) After this check, it should be verified that send and receive assertions of different processes match, and that no deadlock occurs; this verification is omitted for the sake of brevity. The communication pattern of the phases is illustrated in Fig. 2.

The program text of phase (a) is:

ANNOTATED PROGRAM TEXT OF PHASE (a) OF STEP k.

$\{P_1[s, t] \wedge P_2[s, t]\}$
if $s \leqq t$ **then**
begin

receive w_{right} from process $(s, t+1)$; $\left\{ w_{\text{right}} = \sum_{t<q<Q} \hat{w}_{k+s,q} \right\}$

$\{w_{k+s,t} = \hat{w}_{k+s,t}\} w_{\text{right}} := w_{\text{right}} + w_{k+s,t} \left\{ w_{\text{right}} = \sum_{t \leqq q<Q} \hat{w}_{k+s,q} \right\}$

end;
if $s < t$ **then** send w_{right} to process $(s, t-1)$
$\left\{ w_{\text{right}} = \sum_{t \leqq q<Q} \hat{w}_{k+s,q} \vee s > t \right\}$

The verification of this phase is mainly the verification of the assertion $\{w_{k+s,t} = \hat{w}_{k+s,t}\}$ for processes (s, t) with $s \leqq t$. The precondition of phase (a) implies $P_2[s, t]$. Substituting $i = k + s$ in the definition (2.4) of $P_2[s, t]$ and in (2.2) gives

$$w_{k+s,t} = \sum_{\substack{j=0 \\ j \bmod Q = t}}^{k-1} L_{k+s,j} \hat{x}_j = \sum_{\substack{j=0 \\ j \bmod Q = t}}^{k+s-1} L_{k+s,j} \hat{x}_j = \hat{w}_{k+s,t}, \tag{3.1}$$

since there are no indices j with $k \leqq j < k + s$ and j **mod** $Q = t$, for $s \leqq t$ and k **mod** $Q = 0$.

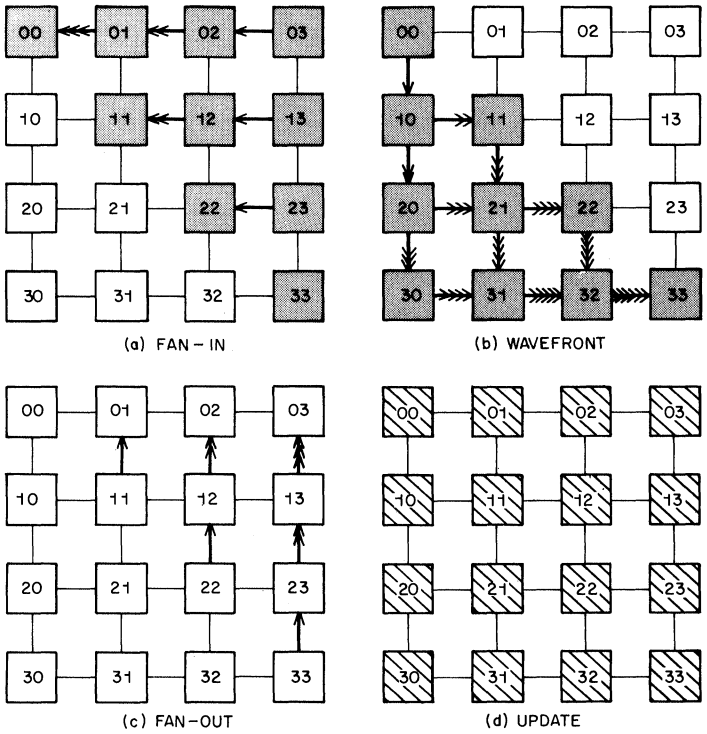


FIG. 2. *Mesh view of the four phases of one step of the triangular system solving algorithm*: (a) *horizontal fan-in of partial sums* $\hat{w}_{it}$; (b) *wavefront computation of* $\hat{x}_i$ *values*; (c) *vertical fan-out of* $\hat{x}_i$ *values*; (d) *update of partial sums* w_{it}. *Each square represents a processor* (s, t), $0 \leqq s, t < Q = 4$. *The processors are connected in a square mesh. Communications are shown by arrows. The communications performed first in a phase are represented by a single arrow; those performed next by a double arrow, and so on. Computations are shown by shading or hatching of processors, in correspondence with Fig.* 1.

The program text of phase (b) is:

ANNOTATED PROGRAM TEXT OF PHASE (b) OF STEP k.

$$\left\{ P_1[s, t] \wedge P_2[s, t] \wedge \left(w_{\text{right}} = \sum_{t \leqq q < Q} \hat{w}_{k+s,q} \vee s > t \right) \right\}$$

if $s > t$ **then**
begin
  par begin
    receive y from process $(s-1, t)$; $\{y = \hat{x}_{k+t}\}$
    receive w_{left} from process $(s, t-1)$ $\left\{ w_{\text{left}} = \sum_{0 \leqq q < t} \hat{w}_{k+s,q} \right\}$
  par end;
  par begin
    $\{y = \hat{x}_{k+t}\}$ send y to process $(s+1, t)$;
    begin
      $\{w_{k+s,t} + L_{k+s,k+t} y = \hat{w}_{k+s,t}\} w_{\text{left}} := w_{\text{left}} + w_{k+s,t} + L_{k+s,k+t} y;$
      $\left\{ w_{\text{left}} = \sum_{0 \leqq q \leqq t} \hat{w}_{k+s,q} \right\}$ send w_{left} to process $(s, t+1)$
    end
  par end
end;

if $s = t$ **then**
begin
$x_{k+s} := b_{k+s} - w_{\text{right}}$;
receive w_{left} from process $(s, t-1)$; $\left\{ w_{\text{left}} = \sum_{0 \leqq q < t} \hat{w}_{k+s,q} \right\}$
$x_{k+s} := x_{k+s} - w_{\text{left}}$; $\{x_{k+s} = \hat{x}_{k+s}\}$
$y := x_{k+s}$;
$\{y = \hat{x}_{k+t}\}$ send y to process $(s+1, t)$
end
$\{P_1[s, t]$ for $k := k+Q\}$
$\{y = \hat{x}_{k+t} \vee s < t\}$

In the algorithmic notation above, statements between **par begin** and **par end** are performed in parallel, and those between **begin** and **end** are executed in sequence. Nesting of **begin-end** pairs is done with the conventional scope rules. For processes (s, t) with $s > t$ the current value of $w_{k+s,t}$ is not necessarily equal to $\hat{w}_{k+s,t}$ (in contrast to the case $s \leqq t$), but

$$w_{k+s,t} = \sum_{\substack{j=0 \\ j \bmod Q = t}}^{k-1} L_{k+s,j}\hat{x}_j = \sum_{\substack{j=0 \\ j \bmod Q = t}}^{k+s-1} L_{k+s,j}\hat{x}_j - L_{k+s,k+t}\hat{x}_{k+t}$$
$$= \hat{w}_{k+s,t} - L_{k+s,k+t}\hat{x}_{k+t}. \tag{3.2}$$

This equation follows from $P_2[s, t]$, because in this case there is exactly one index j, $j = k+t$, such that $k \leqq j < k+s$ and j **mod** $Q = t$. Processes (s, t) with $s = t$ can subtract the values of their local variables w_{left} and w_{right} from b_{k+s}, to obtain $\hat{x}_{k+s}$ and thereby to establish $P_1[s, t]$ for $k := k+Q$. This follows from

$$w_{\text{left}} + w_{\text{right}} = \sum_{0 \leqq q < s} \hat{w}_{k+s,q} + \sum_{s \leqq q < Q} \hat{w}_{k+s,q} = \sum_{0 \leqq q < Q} \hat{w}_{k+s,q}$$
$$= \sum_{0 \leqq q < Q} \sum_{\substack{j=0 \\ j \bmod Q = q}}^{k+s-1} L_{k+s,j}\hat{x}_j = \sum_{j=0}^{k+s-1} L_{k+s,j}\hat{x}_j = b_{k+s} - \hat{x}_{k+s}. \tag{3.3}$$

The program text of phase (c) is:

ANNOTATED PROGRAM TEXT OF PHASE (c) OF STEP k.

$\{y = \hat{x}_{k+t} \vee s < t\}$
if $s < t$ **then** receive y from process $(s+1, t)$; $\{y = \hat{x}_{k+t}\}$
if $s \leqq t$ **then** send y to process $(s-1, t)$
$\{y = \hat{x}_{k+t}\}$

The program text of phase (d) is:

ANNOTATED PROGRAM TEXT OF PHASE (d) OF STEP k.

$\{P_2[s, t] \wedge y = \hat{x}_{k+t}\}$
for $r := k+Q$ **to** $n-Q$ **step** Q **do** $w_{r+s,t} := w_{r+s,t} + L_{r+s,k+t}y$
$\{P_2[s, t]$ for $k := k+Q\}$

Comparing the invariant $P_2[s, t]$ for k and $k+Q$, (2.4), and substituting $i = r+s$ gives

$$w_{r+s,t}(\text{for } k+Q) = \sum_{\substack{j=0 \\ j \bmod Q = t}}^{k+Q-1} L_{r+s,j}\hat{x}_j = \sum_{\substack{j=0 \\ j \bmod Q = t}}^{k-1} L_{r+s,j}\hat{x}_j + L_{r+s,k+t}\hat{x}_{k+t}$$
$$= w_{r+s,t}(\text{for } k) + L_{r+s,k+t}\hat{x}_{k+t}, \tag{3.4}$$

since there is exactly one index $j, j = k+t$, with $k \leqq j < k+Q$ and $j \bmod Q = t$. This shows that the loop of phase (d) indeed establishes the invariant $P_2[s, t]$ for incremented k.

4. Theoretical time analysis. The aim of this section is to derive a simple theoretical model that accurately predicts the performance of the parallel triangular system solving algorithm. To simplify the analysis it is assumed that there are synchronisation barriers between the different steps k of the algorithm, and between the different phases (a)-(d) of each step. This means that processes are assumed to wait until all other processes are ready before proceeding to the next step or phase. However, in the actual program the separate steps or phases may overlap to some extent. The predicted time is therefore an upper bound on the actual time. The time unit is the time needed to perform a *flop*, a floating point operation. The time, measured in flop units, needed to transfer one real value to a neighbouring process is assumed to be a constant α, which is the *communication-to-computation ratio.* It is assumed that each process is able to compute and communicate with all the neighbouring processes in parallel. (This assumption reflects the architecture of the transputer.) In the following analysis we neglect the time needed to evaluate conditionals, because these can be removed from the main loop of the algorithm. We also neglect any other overhead, such as indexing and addressing.

Phase (a) consists of Q simultaneous fan-ins (see Fig. 2(a)). The longest fan-in is from process $(0, Q-1)$ to process $(0, 0)$, and involves $Q-1$ communications and $Q-1$ additions. The time t_a of this phase is

$$t_a = (Q-1)(\alpha+1). \tag{4.1}$$

Phase (b) starts with the simultaneous subtraction of w_{right} from b_{k+s} by all diagonal processes, in one time unit. This is followed by a wavefront-like flow of data from process $(0, 0)$, through all processes (s, t) with $s \geqq t$, to process $(Q-1, Q-1)$ (see Fig. 2(b)). Process $(0, 0)$ initiates the data flow at time 1 by sending $\hat{x}_k$ to process $(1, 0)$; this communication terminates at time $\alpha+1$. Process $(1, 0)$ then sends $\hat{x}_k$ to process $(2, 0)$, finishing at time $2\alpha+1$. Parallel to this, process $(1, 0)$ also performs two flops to compute w_{left} and then sends the result to process $(1, 1)$; this operation is completed at time $2\alpha+3$. Continuing the analysis in the same manner for the other processes (s, t) with $s \geqq t$, it turns out that the last process, $(Q-1, Q-1)$, finishes at time $(Q-1)(2\alpha+4)-1$, for $Q > 1$. The time of phase (b) is therefore

$$t_b = (Q-1)(2\alpha+4)-1. \tag{4.2}$$

Note that processes which have to receive values are always ready to do this. However, processes which have to send values often do this after some delay, because they must perform a number of flops first. For horizontal communication (from (s, t) to $(s, t+1)$) the delay is three flops if $t > 0$, and two flops if $t = 0$. For vertical communication there is only a delay, of one flop, if $s = t > 0$. Phase (c) consists of Q simultaneous fan-outs. The time of phase (c) is

$$t_c = (Q-1)\alpha. \tag{4.3}$$

Phase (d) contains the bulk of the computations, and no communications, with

$$t_d = \frac{2(n-k-Q)}{Q}. \tag{4.4}$$

The total time required by the parallel algorithm for $p = Q^2$ processes is

$$T_{\text{par}} \leqq \sum_{\substack{k=0 \\ k \bmod Q = 0}}^{n-Q} \left\{ (Q-1)(4\alpha+5) - 1 + \frac{2(n-k-Q)}{Q} \right\}$$

$$= \frac{n^2}{p} + (4\alpha+5)n - (4\alpha+7)\frac{n}{\sqrt{p}} \tag{4.5}$$

$$\leqq \frac{n^2}{p} + (4\alpha+5)n.$$

This is a simple upper bound which is sharp for $p \gg 1$. The time of the best sequential algorithm is

$$T_{\text{seq}} = n^2 - n. \tag{4.6}$$

An efficiency of 50 percent or more is achieved if $T_{\text{par}} \leqq 2T_{\text{seq}}/p$, i.e., if $p \leqq p_{1/2} \approx n/(4\alpha+5)$.

The algorithm can be generalised to solve triangular systems with multiple right-hand sides. Exploitation of pipelining increases the efficiency in this case. The algorithm for n_b right-hand sides has n/Q steps k, each of which consists of n_b times phase (a), followed by n_b times phase (b), and so on. The complexity of n_b consecutive executions of phase (a) is $(n_b - 1)\alpha$ higher than the complexity of one execution, assuming that flops are overlapped with subsequent communications. The same holds for phases (b) and (c). The complexity of n_b consecutive executions of phase (d) is simply n_b times the complexity of one execution. A simple upper bound can be obtained as above, giving

$$T_{\text{par}}(n_b) \leqq \frac{n_b n^2}{p} + (4\alpha+5)n + 3(n_b-1)\alpha\frac{n}{\sqrt{p}}. \tag{4.7}$$

In the multiple right-hand side case, the lower-order terms are roughly a factor of $\min(n_b, \sqrt{p})$ less important (relative to the first term) than in the single right-hand side case. This means that efficiency loss is reduced by the same factor.

5. Experimental results. The algorithm has been implemented in the parallel programming language occam 2 [3] and executes on a square mesh of INMOS T800-20 transputers. Timing results were obtained for meshes of $p = 1$, 4, 9, 16, 25, and 36 processors, and for matrices of order up to $n = 1200$. Each transputer possesses a local memory of 256 Kbyte, which allows the storage of a 200×200 matrix per processor. The maximum speed of each communication link is 20 Mbit/s. The time we measured for the communication of a data packet is $t_{\text{comm}}(l) = 2.3 + 2.2l$ μs, where l is the length of the message in 32-bit words. This means that the communication of a single word to a neighbouring processor takes $t_{\text{comm}}(1) = 4.5$ μs. All computations were done in single precision (32 bits). All times were measured by an internal timer calibrated with a wall clock. All results were obtained for single right-hand side systems.

Table 1 shows the time $T_p(n)$ of triangular system solving for an $n \times n$ matrix on a square mesh of p transputers. Table 1 shows that the time $T_p(n)$ is a monotonously decreasing function of p, for a fixed matrix order n. Asymptotically the function reaches a lower bound (~1.8 ms for $n = 50$). The simple theoretical model (4.5) explains this behaviour: the term n^2/p decreases with p, and the term $(4\alpha+5)n$ stays constant. This implies that *all* $p \leqq n^2$ processors available can be used to speed up the computations. There is no need to estimate a possible optimal number of processors.

TABLE 1

Timings (in ms) *of the solution of an* $n \times n$ *unit lower triangular system on a network of* p *transputers.*

n	$p=1$ (seq)	$p=1$ (par)	$p=4$	$p=9$	$p=16$	$p=25$	$p=36$
50	5.3	5.8	2.3	1.9	1.9	1.8	1.8
100	20.6	21.6	7.1	5.2	4.3	4.0	3.9
200	80.9	83.8	24.5	14.8	11.3	9.8	9.3
300			52.1	28.8	20.9	17.4	15.4
400			89.6	47.7	32.9	26.3	23.1
600				97.9	64.5	49.3	41.2
800					106	78.7	64.4
1000						115	91.4
1200							123

Figure 3 is a graph of the timings $T_p(n)$, with each curve representing the timings for various matrix orders n and for a fixed number of processors p. The form of the curves is parabolic. The curves with large p have a visible linear component which dominates at small values of n, but which is overtaken by the quadratic component at larger values. This agrees with the qualitative behaviour predicted by the theoretical model.

A quantitative test on the validity of the theoretical model was done as follows. The simple model (4.5) predicts an overall time of

$$T_p(n) \approx t_{\text{flop}} \frac{n^2}{p} + \beta n, \tag{5.1}$$

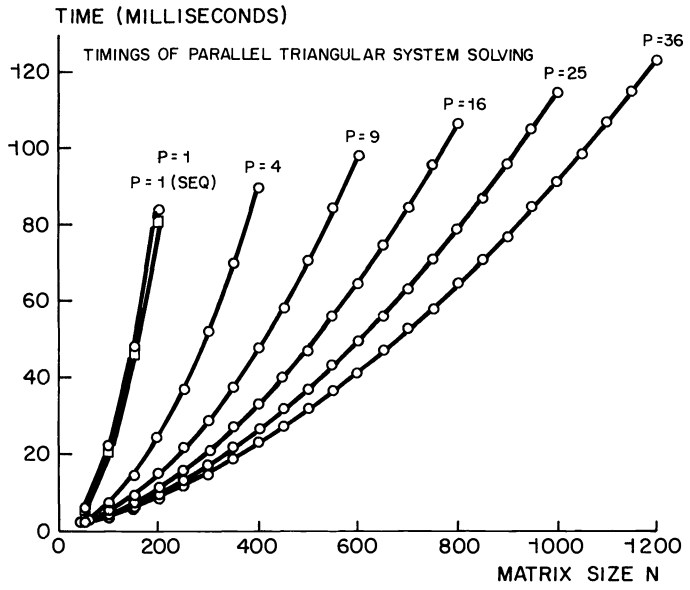


FIG. 3. *Timings of triangular system solving on a square mesh of* T800-20 *transputers. Each curve represents the solution times* $T_p(n)$ *for unit lower triangular matrices of various orders n and for a fixed number of processors p. The timings of the sequential program are marked by a square; those of the parallel program by a circle.*

where t_{flop} is the time needed to perform one flop, and β is, in the first approximation, a constant which does not depend upon p and n. The experimental timing data were used in a least-squares fit to determine $t_{\text{flop}} \approx 1.89$ μs and $\beta \approx 38.0$ μs. These empirical constants were inserted into (5.1), and the resulting times were compared with the experimental times. In most cases the difference between model and experiment was less than 5 percent. The notable exceptions are the cases where both p and n are small, so that $O(n/\sqrt{p})$ terms are significant compared to $O(n)$ terms and also compared to $O(n^2/p)$ terms, and hence cannot be neglected as in the derivation of (4.5). The maximum relative error (for $n = 50$ and $p = 4$) was 32 percent. A least-squares fit to a more refined model which included three terms, n^2/p, n, and $n/\sqrt{p}$, gave an error of typically less than 1 percent, and maximally 9 percent.

Figure 4 shows the efficiency of parallel triangular system solving. Each curve shows the efficiency $E_p(n)$ for various matrix orders n and for a fixed number of processors p. Our measure of efficiency is

$$E_p(n) = \frac{R_p(n)}{pR_{\text{seq, max}}}, \tag{5.2}$$

where $R_p(n) = n^2/T_p(n)$ is the rate in flop/s at which the computation proceeds and $R_{\text{seq, max}}$ is the maximum speed that can be obtained for the problem of triangular system solving by a sequential algorithm on a single transputer with 256 Kbyte memory. This rate $R_{\text{seq, max}} \approx 0.49$ Mflop/s is obtained for $n = 200$. The efficiency of the sequential algorithm is defined as $E_{\text{seq}}(n) = R_{\text{seq}}(n)/R_{\text{seq, max}}$. Note that the sequential efficiency can be less than unity. The maximum rate achieved on $p = 36$ transputers (for $n = 1200$) is 11.7 Mflop/s. The results in Fig. 4 show that for $p \leqq 36$ an efficiency of at least 65 percent can be achieved, if n is large enough. The performance of the algorithm decreases steadily with increasing p, reflecting the fact that the constant communication

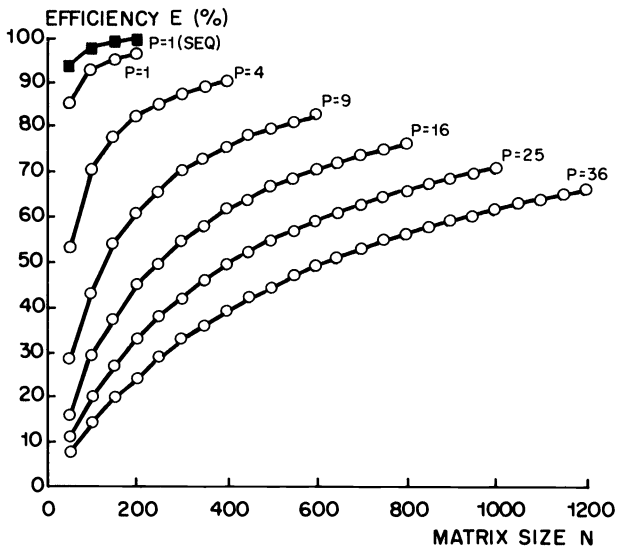


FIG. 4. *Efficiency of triangular system solving on a square mesh of* T800-20 *transputers. Each curve represents the efficiencies* $E_p(n)$ *for unit lower triangular matrices of various orders n and for a fixed number of processors p. The efficiencies of the sequential program are marked by a square; those of the parallel program by a circle. The efficiency* $E_p(n)$ *is defined as the ratio between the solution rate in flop/s of the parallel algorithm on p transputers, and p times the maximum sequential rate that can be obtained in triangular system solving on a single transputer with* 256 Kbyte *memory.*

time becomes more important relative to the decreasing computing time. The figure shows that 36 transputers achieve an efficiency of 50 percent for $n \geqq n_{1/2} \approx 650$, which is close to the value $n_{1/2} = \beta p / t_{\text{flop}} \approx 724$ obtained from (5.1).

An alternative performance measure for parallel algorithms on distributed-memory computers is the *scaled speedup* [7], defined as the ratio between the parallel and the sequential computing rate, for a fixed problem size *per processor.* In our case the scaled speedup is

$$S_p(n) = \frac{R_p(n)}{R_{\text{seq}}(n/\sqrt{p})}. \tag{5.3}$$

The scaled speedup achieved for $n = 1200$ and $p = 36$ is about 24.

6. Conclusions. We have presented an efficient algorithm for parallel triangular system solving on a square mesh of processors. The algorithm is simple to use, since no knowledge of the optimal number of processors is required: using all processors available will solve the problem in the shortest time. The time complexity of the algorithm is $n^2/p + O(n)$, so that full efficiency is achieved asymptotically (for $n \to \infty$). Even for small n an efficiency of more than 50 percent can be achieved, provided the time of a single communication and the time of a single floating point operation are of the same order of magnitude. This condition is met by our transputer network, which has a communication-to-computation ratio $\alpha \approx t_{\text{comm}}(1)/t_{\text{flop}} \approx 4.5/1.89 \approx 2.4$, and for instance by the Caltech/JPL Mark II hypercube, which has a ratio $\alpha = 4.7$ for 64-bit words [5, eq. (20.14)]. (Note that any measured value of α is problem-dependent, because different floating point operations have different computing times, and also because floating point operations may have varying amounts of indexing and addressing operations associated with them.)

The messages of our algorithm are short: each message contains only one real number, in the case of a single right-hand side. Furthermore, the messages are not pipelined, since only one out of $Q-1$ communication links is active at any one time, in each horizontal or vertical chain of processors. Therefore, messages cannot be combined to amortise communication startup time. Unfortunately, architectures such as the present Intel and Ncube hypercubes have high communication startup times, and hence a high communication-to-computation ratio α. For example, the Intel iPSC/2 has a ratio $\alpha = 59$ [2, Table 4]. Our algorithm is inefficient on such machines. In the case of multiple right-hand sides the situation is better, because messages belonging to different right-hand sides can be combined into larger messages, and these in turn can be pipelined.

The present study was performed on a square mesh, since this network has sufficient connectivity for parallel linear algebra algorithms such as LU decomposition and triangular system solving, and since it is trivially embedded in many other topologies such as a square torus, a hypercube of even dimension, and a fully connected network. On networks with a richer topology the algorithm can be executed without modification, or with adjustments to exploit the additional connectivity. The gains that can be obtained from richer connectivity are limited: in the best case, for a fully connected network, the communication complexity is reduced by a factor of two, from $4\alpha n$ to $2\alpha n$ [12]. This is also the communication complexity for a hypercube of even dimension [9].

An interesting extension of the present work is the generalisation of the triangular system solving algorithm for a square $Q \times Q$ mesh to an algorithm for a rectangular $M \times N$ mesh, using the corresponding rectangular grid distribution [1]. Under mild

constraints ($M \geqq N$ and M **mod** $N = 0$) on the mesh dimensions it is easy to derive a generalised algorithm which has n/M steps, each with phases similar to the phases of the original algorithm. The generalised algorithm has the same complexity $n^2/p + O(n)$ as the original algorithm. The particular choice $M = p$ and $N = 1$ leads to a new so-called immediate-update row-wrapped ($ji - r$) algorithm [15], which differs from the $ji - r$ algorithm of [15], for instance, because its communication complexity is independent of the number of processors.

In conclusion, a grid-based parallel algorithm has been presented for triangular system solving that can be used in combination with grid-based LU decomposition algorithms. Experiments on a square mesh of transputers have shown that the algorithm can be implemented efficiently on a distributed-memory MIMD computer with a low communication-to-computation ratio.

Acknowledgments. We are grateful to Daniël Loyens for many valuable suggestions on the exposition of the algorithm. We also thank Netty van Gasteren for proposing a more concise form of invariant P_2.

REFERENCES

[1] R. H. BISSELING AND J. G. G. VAN DE VORST, *Parallel* LU *decomposition on a transputer network*, Lecture Notes in Computer Science 384, Springer-Verlag, Berlin, 1989, pp. 61–77.

[2] L. BOMANS AND D. ROOSE, *Benchmarking the* iPSC/2 *hypercube multiprocessor*, Concurrency: Practice and Experience, 1 (1989), pp. 3–18.

[3] A. BURNS, *Programming in occam* 2, Addison-Wesley, Wokingham, U.K., 1988.

[4] S. C. EISENSTAT, M. T. HEATH, C. S. HENKEL, AND C. H. ROMINE, *Modified cyclic algorithms for solving triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 589–600.

[5] G. C. FOX, M. A. JOHNSON, G. A. LYZENGA, S. W. OTTO, J. K. SALMON, AND D. W. WALKER, *Solving Problems on Concurrent Processors*, Vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[6] D. GRIES, *The Science of Programming*, Springer-Verlag, New York, 1981.

[7] J. L. GUSTAFSON, G. R. MONTRY, AND R. E. BENNER, *Development of parallel methods for a* 1024*-processor hypercube*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 609–638.

[8] M. T. HEATH AND C. H. ROMINE, *Parallel solution of triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 558–588.

[9] S. L. JOHNSSON, *Communication efficient basic linear algebra computations on hypercube architectures*, J. Parallel Distrib. Comput., 4 (1987), pp. 133–172.

[10] G. LI AND T. F. COLEMAN, *A parallel triangular solver for a distributed-memory multiprocessor*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 485–502.

[11] ———, *A new method for solving triangular systems on distributed-memory message-passing multiprocessors*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 382–396.

[12] L. D. J. C. LOYENS AND R. H. BISSELING, *The formal construction of a parallel triangular system solver*, Lecture Notes in Computer Science 375, Springer-Verlag, Berlin, 1989, pp. 325–334.

[13] S. OWICKI AND D. GRIES, *Verifying properties of parallel programs: An axiomatic approach*, Comm. ACM, 19 (1976), pp. 279–285.

[14] ———, *An axiomatic proof technique for parallel programs* I, Acta Inform., 6 (1976), pp. 319–340.

[15] C. H. ROMINE AND J. M. ORTEGA, *Parallel solution of triangular systems of equations*, Parallel Comput., 6 (1988), pp. 109–114.

[16] J. G. G. VAN DE VORST, *The formal development of a parallel program performing* LU*-decomposition*, Acta Inform., 26 (1988), pp. 1–17.

[17] D. W. WALKER, T. ALDCROFT, A. CISNEROS, G. C. FOX, AND W. FURMANSKI, LU *decomposition of banded matrices and the solution of linear systems on hypercubes*, in Proc. Third Conference on Hypercube Concurrent Computers and Applications, Association for Computing Machinery, New York, 1988, pp. 1635–1655.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 800-807, July 1991
© 1991 Society for Industrial and Applied Mathematics
005

A SQUARE ROOT AND DIVISION FREE GIVENS ROTATION FOR SOLVING LEAST SQUARES PROBLEMS ON SYSTOLIC ARRAYS*

J. GÖTZE† AND U. SCHWIEGELSHOHN‡

Abstract. This paper presents a square root and division free Givens rotation (SDFG) to be applied to the QR-decomposition (QRD) for solving linear least squares problems on systolic arrays. The SDFG is based on a special kind of number description of the matrix elements and can be executed by mere application of multiplications and additions. Therefore, it is highly suited for the VLSI-implementation of the QRD on systolic arrays. Roundoff error and stability analyses indicate that the SDFG is numerically as stable as known Givens rotation methods.

Key words. least squares problems, QR-decomposition, Givens rotations, systolic arrays

AMS(MOS) subject classifications. 65F20, 65F25

1. Introduction. Given a full-rank matrix $\mathbf{A}\in\mathbb{R}^{m\times n}$ $(m\geqq n)$ and a vector $\mathbf{b}\in\mathbb{R}^m$, the linear least squares problem is that of finding a vector $\mathbf{x}\in\mathbb{R}^n$ such that

$$\|\mathbf{Ax}-\mathbf{b}\|=\min.$$

Here $\|\cdot\|$ denotes the Euclidean matrix or vector norm depending on the context.

Suppose that an orthogonal matrix $\mathbf{Q}\in\mathbb{R}^{m\times m}$ has been computed such that $\mathbf{QA}=\mathbf{R}=\begin{bmatrix}\mathbf{R}_1\\ \mathbf{0}\end{bmatrix}$ is upper triangular and $\mathbf{Qb}=\begin{bmatrix}\mathbf{c}\\ \mathbf{d}\end{bmatrix}$ $(\mathbf{c}\in\mathbb{R}^n, \mathbf{d}\in\mathbb{R}^{m-n})$. Then the following equation holds for any $\mathbf{x}\in\mathbb{R}^n$:

$$\|\mathbf{Ax}-\mathbf{b}\|^2=\|\mathbf{QAx}-\mathbf{Qb}\|^2=\|\mathbf{R}_1\mathbf{x}-\mathbf{c}\|^2+\|\mathbf{d}\|^2.$$

The least squares solution $\mathbf{x}_s$ can be obtained from the upper triangular system $\mathbf{R}_1\mathbf{x}_s=\mathbf{c}$ by back substitution.

The solution of a least squares problem on a sequential computer requires in general $O(mn^2)$ operations for computing $\mathbf{QA}$ and $O(n^2)$ operations for the back substitution. This time complexity does not comply with many real-time applications and therefore systolic arrays have been developed that solve this problem in $O(m)$ time with $O(n^2)$ processors (see, e.g., [13]). Usually the method of choice for solving least squares problems on systolic arrays is the QR-decomposition (QRD) by Givens rotations (GR).

Several fast algorithms [4] and systolic arrays [1], [3], [7], [12], [13], [15] have been proposed for computing the QRD by GRs. Most of the systolic architectures require the execution of a GR in many processor cells, where therefore the operations $\{+,*,\div,\sqrt{\ }\}$ must be implemented if the standard GR is used. Consequently, these processor cells become complex, which makes the standard GR contradictory to an important criterion for an efficient VLSI-implementation: The parallel algorithms should be based on powerful and primitive operations [1].

Clearly, the efficiency can be improved by avoiding the implementation of the complex operations $\{\div,\sqrt{\ }\}$. This can be partly achieved either by realizing the standard GR with an iterative algorithm, e.g., CORDIC algorithm [1], or by using square root free GRs [5], [10], [11], which only require the operations $\{+,*,\div\}$ [14], [2].

* Received by the editors June 1, 1988; accepted for publication (in revised form) April 23, 1990.

† Department of Computer Science, Yale University, P.O. Box 2158, New Haven, Connecticut 06520. The work of this author was performed while he was on leave from the Institute for Network Theory and Circuit Design, Technical University, Munich, Germany.

‡ IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.

Due to either the number of iterations or the necessity of a division, all these known GRs (standard GR, CORDIC, square root free GRs) require a latency time of $O(w)$ for an elementary rotation, where w is the wordlength of the data. This latency time determines the data pulse frequency of the processor arrays.

In this paper a square root and division free GR (SDFG) is presented. This SDFG can be derived from the standard GR by using a number description of the matrix elements, which is tailored to the standard GR. Although it is similiar to the method used in [2], it only requires the implementation of $\{+, *\}$. Therefore, the application of SDFGs is highly suited for the VLSI-implementation of the QRD by GRs on systolic arrays. The area requirements of a processor cell are decreased since no implementation of $\{\div, \sqrt{\ }\}$ is necessary. The latency time for one rotation can be reduced to $O(\lg w)$, as there are implementations of $\{+, *\}$ needing only $O(\lg w)$ time [16]. The SDFG is numerically as stable as any known GR.

2. QRD by GRs. The QRD by GRs is based on the orthogonal factorization of the system matrix $\mathbf{A} = (a_{ij})$ such that

$$\mathbf{QA} = \mathbf{R},$$

where the orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{m \times m}$ is formed recursively as a product of GR $\mathbf{G}_{ij} \in \mathbb{R}^{m \times m}$.

Without loss of generality and in conformity to other publications (e.g., [2], [5], [11]) we assume that $\mathbf{A}, \bar{\mathbf{A}} \in \mathbb{R}^{2 \times n}$ and only consider the GR $\mathbf{G}$, which annihilates the (2, 1) entry of $\mathbf{A}$, i.e., $\bar{\mathbf{A}} = \mathbf{G} \cdot \mathbf{A}$ yields $\bar{a}_{21} = 0$.

The standard GR $\mathbf{G}$ is defined by

$$\mathbf{G} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix},$$

where

$$c = \frac{a_{11}}{\sqrt{a_{11}^2 + a_{21}^2}}, \tag{2.1a}$$

$$s = \frac{a_{21}}{\sqrt{a_{11}^2 + a_{21}^2}}. \tag{2.1b}$$

Then, the elements of the rotated matrix $\bar{\mathbf{A}} = \mathbf{GA}$ are obtained by

$$\bar{a}_{1k} = ca_{1k} + sa_{2k}, \tag{2.2a}$$

$$\bar{a}_{2k} = -sa_{1k} + ca_{2k} \qquad (\bar{a}_{21} = 0). \tag{2.2b}$$

If $\bar{\mathbf{A}}^*$ is the result of the floating point computation of $\bar{\mathbf{A}} = \mathbf{GA}$ and ε is the largest number such that $1 + \varepsilon$ is computed as 1, the following estimation for the roundoff error of one standard GR holds [17]:

$$\|\bar{\mathbf{A}}^* - \bar{\mathbf{A}}\| \leqq 6.0\varepsilon \|\mathbf{A}\|. \tag{2.3}$$

The error analysis described in [2] is not consistent with the error analysis of the standard GR in [17] as the multiplication of $\mathbf{A}$ with the computed GR is assumed to have a roundoff error of 2ε instead of $2\sqrt{2}\,\varepsilon \approx 3\varepsilon$ as in [17]. As the error analysis of [2] is tailored to square root free GRs, a version where the 2ε bound is changed to 3ε is applied to all GRs throughout this paper, although better upper bounds are often available. However, Wilkinson [18, p. 567] points out that "these bounds itself are usually the least important part, but the main object of such an error analysis is to expose potential instabilities."

Using [2] and [6] the application of the whole QRD to a given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ results in a matrix that will be the exact transformation of $\mathbf{A} + \delta\mathbf{A}$, where

$$\|\delta\mathbf{A}\| \leqq \tau\nu(1+\tau)^{\nu-1}\|\mathbf{A}\| + O(\varepsilon^2).$$

ν is the number of stages in which simultaneously disjoint GRs are applied. The QRD requires ν to be $m+n-3$ [6]. τ is the upper bound, which is obtained for one of the different GRs in the subsequent sections. Due to (2.3), for the standard GR, $\tau = 6.0\varepsilon$ holds.

3. Square root free Givens rotations. Performing a GR directly on the matrix $\mathbf{A}$ involves square root computations. For a square root free GR [5], [10], [11] the matrix $\mathbf{A}$ is factorized into the form

$$\mathbf{A} = \mathbf{D}^{1/2}\mathbf{B} \tag{3.1}$$

with $\mathbf{D} = \operatorname{diag}(d_1, d_2)$ and $\mathbf{B} = (b_{ij}) \in \mathbb{R}^{2 \times m}$. With a further diagonal matrix $\bar{\mathbf{D}} = \operatorname{diag}(\bar{d}_1, \bar{d}_2)$ the square root free GR $\mathbf{F}$ can be obtained from the standard GR as

$$\mathbf{F} = \bar{\mathbf{D}}^{-1/2}\mathbf{G}\mathbf{D}^{1/2}. \tag{3.2}$$

The square root free GR $\mathbf{F}$ is applied to $\mathbf{B}$ instead of $\mathbf{A}$, i.e., $\bar{\mathbf{B}} = \mathbf{F}\mathbf{B}$ is computed. Using (3.1) and (3.2), it is easy to see that the factorization (3.1) is retained by a square root free GR and therefore during the whole QRD as well:

$$\bar{\mathbf{A}} = \mathbf{G} \cdot \mathbf{A} = \bar{\mathbf{D}}^{1/2}\mathbf{F}\mathbf{D}^{-1/2} \cdot \mathbf{D}^{1/2}\mathbf{B} = \bar{\mathbf{D}}^{1/2}\mathbf{F}\mathbf{B} = \bar{\mathbf{D}}^{1/2}\bar{\mathbf{B}}.$$

In Table 1(a) several known square root free GRs are shown in factorized form. For the square root free GR of [2], called scaled GR, the additionally required scaling algorithm [2, Algorithm 3.1] is not included in Table 1.

For the execution of a square root free GR only the rotation matrix $\mathbf{F}$, the rotation $\bar{\mathbf{B}} = \mathbf{F}\mathbf{B}$ itself and the new diagonal matrix $\bar{\mathbf{D}}$ must be computed. Therefore the implementation of $\{\sqrt{\ }\}$ is not required.

The modified error analysis of [2] yields different values of τ for the various square root free GRs (see Table 2).

A deviation factor Δf is introduced in order to determine whether the QRD by square root free GRs runs the risk of additional over-/underflow, that is over-/underflow which will not occur by using the standard GR. The deviation factor describes the maximum deviation of any b_{ij} from the corresponding a_{ij} during the computation of the whole QRD. As $(d_i)^{1/2} = a_{ij}/b_{ij}$ for all j

$$\Delta f = \max_{\text{all stages}} \left(\max_i \left(\max \left(|d_i|^{1/2}, |d_i|^{-1/2} \right) \right) \right). \tag{3.3}$$

As can be seen from Table 2 only for the scaled GR of [2] we must not care for additional over-/underflow as Δf is a small constant. Therefore, the processor arrays for the QRD by scaled GRs require no monitoring process and no local pivoting.

As can be seen from Table 1(a) all square root free GRs require the implementation of a division.

4. Square root and division free Givens rotation (SDFG). The SDFG is based on a number description of the matrix elements, which is tailormade to the standard GR. This number description is defined as follows:

DEFINITION 4.1. Each element a_{ij} of any matrix is described by a pair of numbers (y_{ij}, z_i) with

$$(y_{ij}, z_i) \triangleq a_{ij} = \frac{y_{ij}}{\sqrt{z_i}}. \tag{4.1}$$

TABLE 1
Factorized Givens rotations (1(a) *square root free* GRs; 1(b) SDFG).

1(a)

Gentleman [6]: $\mathbf{A}=\mathbf{D}^{1/2}\mathbf{B}$

$\mathbf{G}=\bar{\mathbf{D}}^{1/2}\mathbf{F}\mathbf{D}^{-1/2}$

$$=\begin{bmatrix} d_1b_{11}^2+d_2b_{21}^2 & 0 \\ 0 & \dfrac{d_2d_1b_{11}^2}{d_1b_{11}^2+d_2b_{21}^2} \end{bmatrix}^{1/2}\begin{bmatrix} \dfrac{d_1b_{11}}{d_1b_{11}^2+d_2b_{21}^2} & \dfrac{d_2b_{21}}{d_1b_{11}^2+d_2b_{21}^2} \\ -\dfrac{b_{21}}{b_{11}} & 1 \end{bmatrix}\begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}^{-1/2}$$

Golub and Van Loan [10]: $\mathbf{A}=\mathbf{D}^{-1/2}\mathbf{B}$

$\mathbf{G}=\bar{\mathbf{D}}^{-1/2}\mathbf{F}\mathbf{D}^{1/2}$

$$=\begin{bmatrix} d_2\left(1+\dfrac{d_2b_{11}^2}{d_1b_{21}^2}\right) & 0 \\ 0 & d_1\left(1+\dfrac{d_2b_{11}^2}{d_1b_{21}^2}\right) \end{bmatrix}^{-1/2}\begin{bmatrix} \dfrac{d_2b_{11}}{d_1b_{21}} & 1 \\ -1 & \dfrac{b_{11}}{b_{21}} \end{bmatrix}\begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}^{1/2} \quad \text{for } \frac{d_1b_{21}^2}{d_2b_{11}^2}>1$$

$$=\begin{bmatrix} d_1\left(1+\dfrac{d_1b_{21}^2}{d_2b_{11}^2}\right) & 0 \\ 0 & d_2\left(1+\dfrac{d_1b_{21}^2}{d_2b_{11}^2}\right) \end{bmatrix}^{-1/2}\begin{bmatrix} 1 & \dfrac{d_1b_{21}}{d_2b_{11}} \\ -\dfrac{b_{21}}{b_{11}} & 1 \end{bmatrix}\begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}^{1/2} \quad \text{for } \frac{d_1b_{21}^2}{d_2b_{11}^2}\leqq 1$$

Barlow and Ipsen [2]: $\mathbf{A}=\mathbf{D}^{1/2}\mathbf{B}$

$\mathbf{G}=\bar{\mathbf{D}}^{1/2}\mathbf{F}\mathbf{D}^{-1/2}$

$$=\begin{bmatrix} \dfrac{1}{d_1b_{11}^2+d_2b_{21}^2} & 0 \\ 0 & \dfrac{d_1d_2}{d_1b_{11}^2+d_2b_{21}^2} \end{bmatrix}^{1/2}\begin{bmatrix} d_1b_{11} & d_2b_{21} \\ -b_{21} & b_{11} \end{bmatrix}\begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}^{-1/2}$$

1(b)

$\mathbf{A}=\mathbf{Z}^{-1/2}\mathbf{Y}$,

$\mathbf{G}=\bar{\mathbf{Z}}^{-1/2}\mathbf{H}\mathbf{Z}^{1/2}$

$$=\begin{bmatrix} z_1z_2(z_2y_{11}^2+z_1y_{21}^2) & 0 \\ 0 & z_2y_{11}^2+z_1y_{21}^2 \end{bmatrix}^{-1/2}\begin{bmatrix} z_2y_{11} & z_1y_{21} \\ -y_{21} & y_{11} \end{bmatrix}\begin{bmatrix} z_1 & 0 \\ 0 & z_2 \end{bmatrix}^{1/2}$$

TABLE 2
Comparison of the different GR *methods.*

GR	Required operations	Roundoff error τ	Deviation factor Δf
standard GR	$+,*,\div,\sqrt{\ }$	6.0ε	-
square root free GR [6]	$+,*,\div$	10.5ε (7.5ε)	unbounded
square root free GR [10]	$+,*,\div$	9.5ε	$\sqrt{2}^{m+n-3}$
scaled GR [2]	$+,*,\div$	6.5ε	2
SDFG	$+,*$	6.5ε	$\sqrt{2}$

Instead of computing the square root and the division for calculating a_{ij} the two numbers y_{ij} and z_i are stored separately. As indicated by its index z_i is assumed to be constant for a whole row. At the beginning $y_{ij}=a_{ij}$ and $z_i=1$ hold.

Now, the standard GR of (2.2) is computed using this number description. By inserting c (2.1a) and s (2.1b) in (2.2) and then replacing a_{ij} with $y_{ij}/\sqrt{z_i}$, we obtain

(4.2a)
$$\bar{a}_{1k}=\frac{a_{11}a_{1k}+a_{21}a_{2k}}{\sqrt{a_{11}^2+a_{21}^2}}=\frac{\frac{y_{11}}{\sqrt{z_1}}\frac{y_{1k}}{\sqrt{z_1}}+\frac{y_{21}}{\sqrt{z_2}}\frac{y_{2k}}{\sqrt{z_2}}}{\sqrt{\frac{y_{11}^2}{z_1}+\frac{y_{21}^2}{z_2}}}$$
$$=\frac{y_{11}z_2y_{1k}+y_{21}z_1y_{2k}}{\sqrt{z_1z_2(y_{11}^2z_2+y_{21}^2z_1)}}=\frac{\bar{y}_{1k}}{\sqrt{\bar{z}_1}}\triangleq(\bar{y}_{1k},\bar{z}_1),$$

(4.2b)
$$\bar{a}_{2k}=\frac{-a_{21}a_{1k}+a_{11}a_{2k}}{\sqrt{a_{11}^2+a_{21}^2}}=\frac{-\frac{y_{21}}{\sqrt{z_2}}\frac{y_{1k}}{\sqrt{z_1}}+\frac{y_{11}}{\sqrt{z_1}}\frac{y_{2k}}{\sqrt{z_2}}}{\sqrt{\frac{y_{11}^2}{z_1}+\frac{y_{21}^2}{z_2}}}$$
$$=\frac{-y_{21}y_{1k}+y_{11}y_{2k}}{\sqrt{y_{11}^2z_2+y_{21}^2z_2}}=\frac{\bar{y}_{2k}}{\sqrt{\bar{z}_2}}\triangleq(\bar{y}_{2k},\bar{z}_2).$$

The rotated matrix elements appear as well in the new number description, i.e., $(\bar{y}_{ij},\bar{z}_i)\triangleq\bar{a}_{ij}=\bar{y}_{ij}/\sqrt{\bar{z}_i}$. Therefore, this number description is retained for one rotation and consequently for the whole QRD as well. Due to (4.2a) and (4.2b) the computation of the rotated pairs $(\bar{y}_{ij},\bar{z}_i)\triangleq\bar{a}_{ij}$ can be described as follows:

(4.3a)
$$\begin{bmatrix}\bar{y}_{1k}\\ \bar{y}_{2k}\end{bmatrix}=\underbrace{\begin{bmatrix}y_{11}z_2 & y_{21}z_1\\ -y_{21} & y_{11}\end{bmatrix}}_{\mathbf{H}}\begin{bmatrix}y_{1k}\\ y_{2k}\end{bmatrix},$$

(4.3b)
$$\bar{z}_1=z_1z_2\bar{y}_{11}\qquad(=z_1z_2(y_{11}^2z_2+y_{21}^2z_1)),$$

(4.3c)
$$\bar{z}_2=\bar{y}_{11}\qquad(=y_{11}^2z_2+y_{21}^2z_1).$$

Clearly, the operations $\{+,*\}$ are sufficient for the computation of all formulae (4.3). These formulae (4.3) together with the following scaling procedure are called SDFG.

As no division is executed during the whole QRD the numbers of (y_{ij},z_i), which are describing the elements of the recursively computed matrices, are not limited, i.e., there is a risk of additional over-/underflow. However, this problem can easily be avoided by the following scaling procedure, which executes the square root and the division of (4.1) for the exponents of (y_{ij},z_i) only. Here exp (r) describes the (integer) exponent of the floating point number r:

IF $z_i>2$ THEN

(4.4a) exp (y_{ij})=exp (y_{ij})−int (exp $(z_i)/2)$

(4.4b) exp (z_i)=exp (z_i) mod 2

IF $z_i\leqq 0.5$ THEN

(4.4c) exp (y_{ij})=exp (y_{ij})+int (|exp (z_i)|/2)

(4.4d) exp (z_i)=−|exp (z_i)| mod 2.

This scaling procedure only requires a single shift (division by two) of $\exp(z_i)$, respectively, $|\exp(z_i)|$ for the square root operation and an exponent addition for the division. The least significant bit of the old exponent $\exp(z_i)$ respectively, $|\exp(z_i)|$ forms the new $\exp(z_i)$.

This scaling procedure is executed after the computation of each new pair $(y_{ij}, z_i) \triangleq a_{ij}$ such that during the whole QRD after scaling the following condition holds for all z_i:

$$z_i \in [0.5; 2].$$

Therefore, the deviation factor of the SDFG is no worse than $\Delta f = \sqrt{2}$ during the whole QRD. With respect to additional over-/underflow the SDFG has the same favourable properties as the scaled GR of [2].

For the roundoff error of one SDFG the modified procedure of [2] results in $\tau = 6.5\varepsilon$ (see Table 2).

Analogous to square root free GRs the SDFG can be described by a similiar factorization of the rotation matrix:

$$\text{square root free GR:} \quad \bar{\mathbf{A}} = \underbrace{\bar{\mathbf{D}}^{1/2}\mathbf{F}\mathbf{D}^{-1/2}} \cdot \underbrace{\mathbf{D}^{1/2}\mathbf{B}} = \bar{\mathbf{D}}^{1/2}\mathbf{F}\mathbf{B} = \bar{\mathbf{D}}^{1/2}\bar{\mathbf{B}}$$

$$\text{standard GR:} \quad \bar{\mathbf{A}} = \mathbf{G} \cdot \mathbf{A}$$

$$\text{SDFG:} \quad \bar{\mathbf{A}} = \overbrace{\bar{\mathbf{Z}}^{-1/2}\mathbf{H}\mathbf{Z}^{1/2}} \cdot \overbrace{\mathbf{Z}^{-1/2}\mathbf{Y}} = \bar{\mathbf{Z}}^{-1/2}\mathbf{H}\mathbf{Y} = \bar{\mathbf{Z}}^{-1/2}\bar{\mathbf{Y}},$$

where $\mathbf{Z} = \operatorname{diag}(z_1, z_2)$ and $\bar{\mathbf{Z}} = \operatorname{diag}(\bar{z}_1, \bar{z}_2)$ are diagonal matrices. In Table 1(b) the factorized form of the SDFG is shown in the same manner as the factorized forms of the various square root free GRs in Table 1(a). The scaling procedure (4.4) is not included. Although this analogy of the SDFG to the square root free GRs can be pointed out, note, that the mere difference between the standard GR and the SDFG is, that in the case of the SDFG the square root and the division is only executed with the exponents. This results in a matrix factorization analogous to the square root free GRs.

The SDFG is highly suited for VLSI-implementation. A systolic implementation of the QRD by SDFGs is possible on the basis of known systolic architectures for computing the QRD by GRs (see, e.g., [3], [7]), where the architecture of [7] is recommended for least squares problems. Only $\{+, *\}$ must be implemented in the processor cells. No pivoting and no monitoring process is necessary since the deviation factor is only a small constant. Therefore, the area requirements of the systolic arrays are significantly decreased. The latency time for one rotation can be reduced to $O(\lg w)$, since the required operations $\{+, *\}$ can be implemented requiring $O(\lg w)$ time [16]. Consequently, the data pulse frequency of the processor arrays can be increased to $1/O(\lg w)$ and almost full (asymptotically 100 percent) utilization of the hardware components $\{+, *\}$ can be achieved [8].

At the end of the QRD by SDFG the result is obtained in the new number description, i.e.,

$$\begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{Z}^{-1/2} \begin{bmatrix} \mathbf{Y}_{R1} \\ \mathbf{0} \end{bmatrix} \qquad (\mathbf{R}_1 = \mathbf{Z}_1^{-1/2}\mathbf{Y}_{R1}),$$

$$\begin{bmatrix} \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \mathbf{Z}^{-1/2} \begin{bmatrix} \mathbf{c}' \\ \mathbf{d}' \end{bmatrix} \qquad (\mathbf{c} = \mathbf{Z}_1^{-1/2}\mathbf{c}';\ \mathbf{d} = \mathbf{Z}_2^{-1/2}\mathbf{d}').$$

Clearly, the denominators $\sqrt{z_i}$, which are constant for all rows of the resulting upper triangular matrix $\mathbf{Y}_{R1}$ and for the corresponding value of the resulting right-hand side $\mathbf{c}'$, can be reduced. Then, the arising system $\mathbf{Y}_{R1}\mathbf{x}=\mathbf{c}'$ can be solved by back substitution and the residual is $\|\mathbf{d}\|=\|\mathbf{Z}_2^{-1/2}\mathbf{d}'\|$.

5. Comparisons of the different Givens rotation methods. In Table 2 comparisons of the different GR methods show the type of the required operations, the bound τ for the roundoff error of one GR and the deviation factor Δf for the whole QRD in the worst case. The latency time of the different GRs is implicitly contained in Table 2, as the time complexity of the required operations determines the latency time of one GR ($O(w)$ for $\{\div, \sqrt{\ }\}$; $O(\lg w)$ for $\{+, *\}$). Furthermore, the QRD has been executed by a sequential program using all these GR methods, whereby the SDFG has yielded the best numerical results.

6. Concluding remarks. In this paper a square root and division free GR has been presented. For solving linear least squares problems on systolic QRD arrays the application of square root and division free GR enables only multiplications and additions to be implemented on the systolic arrays. Therefore, this algorithm is highly suited for VLSI-implementation.

Furthermore, the idea of Definition 4.1 can be modified to

$$(y_{ij}, z_i, w_j) \triangleq a_{ij} = \frac{y_{ij}}{\sqrt{z_i w_j}}, \tag{6.1}$$

where at the beginning $y_{ij}=a_{ij}$, $z_i=1$, and $w_j=1$ hold. The number description of (6.1) can be applied to linear algebra problems, which require two-sided orthogonal transformations, such as, e.g., singular value decomposition, eigenvalue problems, and generalized least squares. By retaining this number description during the whole algorithm, these algorithms can also be executed by mere application of $\{+, *\}$ [9]. The complex operations $\{\div, \sqrt{\ }\}$ are only necessary for translating the resulting matrix back to the usual number description at the end of the computation (e.g., the singular value decomposition requires n square roots and n divisions for translating the resulting singular values back to the usual number description).

REFERENCES

[1] H. M. AHMED, J.-M. DELOSME, AND M. MORF, *Highly concurrent computing structures for matrix arithmetic and signal processing*, Computer, 15 (1982), pp. 65-82.

[2] J. L. BARLOW AND I. C. F. IPSEN, *Scaled Givens rotations for the solution of linear least squares problems on systolic arrays*, SIAM J. Sci. Statist. Comput., 5 (1987), pp. 716-733.

[3] A. BOJANCZYK, R. P. BRENT, AND H. T. KUNG, *Numerically stable solution of dense systems of linear equations using mesh-connected processors*, SIAM J. Sci. Statist. Comput., 1 (1984), pp. 95-104.

[4] J. CHUN, T. KAILATH, AND H. LEV-ARI, *Fast parallel algorithms for* QR *and triangular factorization*, SIAM J. Sci. Statist. Comput., 6 (1987), pp. 899-913.

[5] W. M. GENTLEMAN, *Least squares computations by Givens transformations without square roots*, J. Inst. Maths Appl., 12 (1973), pp. 329-336.

[6] ———, *Error analysis of* QR*-decomposition by Givens transformations*, Linear Algebra Appl., 10 (1975), pp. 189-197.

[7] W. M. GENTLEMAN AND H. T. KUNG, *Matrix triangularization by systolic arrays*, Proc. SPIE, Real-Time Signal Processing IV, 298 (1981), pp. 19-26.

[8] J. GÖTZE AND U. SCHWIEGELSHOHN, VLSI*-suited orthogonal solution of systems of linear equations*, J. Parallel Distributed Comput., to appear.

[9] J. GÖTZE, M. ALI, AND U. SCHWIEGELSHOHN, *Efficient orthogonal matrix decompositions for digital signal processing*, in Proc. Internat. Symposium on Signals, Systems and Electronics, Erlangen, Germany, 1989, pp. 803-806.

[10] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[11] S. HAMMARLING, *A note on modifications to the Givens plane rotation*, J. Inst. Maths Appl., 13 (1974), pp. 215–218.

[12] D. E HELLER AND I. C. F. IPSEN, *Systolic networks for orthogonal decompositions*, SIAM J. Sci. Statist. Comput., 2 (1983), pp. 261–269.

[13] F. T. LUK, *A rotation method for computing the* QR*-decomposition*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 452–459.

[14] T. H.-Y. MENG, E. A. LEE, AND D. G. MESSERSCHMITT, *Least squares computation at arbitrarily high speeds*, in Proc. Internat. Conference on Acoustics, Speech and Signal Processing, Dallas, TX, 1987, pp. 1398–1401.

[15] U. SCHWIEGELSHOHN AND L. THIELE, *One- and two-dimensional arrays for least squares problems*, in Proc. Internat. Conference on Acoustics, Speech and Signal Processing, Dallas, TX, 1987, pp. 791–794.

[16] J. VUILLEMIN, *A very fast multiplication algorithm for* VLSI *implementation*, INTEGRATION, the VLSI J., 1 (1983), pp. 39–52.

[17] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.

[18] ———, *Modern error analysis*, SIAM Rev., 13 (1971), pp. 548–568.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 808-823, July 1991
© 1991 Society for Industrial and Applied Mathematics
006

SELF-SORTING IN-PLACE FAST FOURIER TRANSFORMS*

CLIVE TEMPERTON†

Abstract. It has recently been shown that the familiar radix-2 Fast Fourier Transform (FFT) algorithm can be made both self-sorting and in-place—two useful properties which were previously thought to be mutually exclusive. In this paper the procedure is demonstrated and it is shown how it can be extended to radix-3, radix-4, radix-5, and finally to mixed-radix FFTs. These new versions of the FFT algorithm require neither an unscrambling step nor work space. Implementation on vector computers (for the case of multiple transforms) is discussed. Timing experiments on the Cray X-MP demonstrate that these new variants of the FFT run just as fast as older self-sorting routines which required work space.

Key words. Fast Fourier Transform (FFT), self-sorting FFT, in-place FFT, vector computers

AMS(MOS) subject classification. 65T05

1. Introduction. Since it was introduced in the famous paper by Cooley and Tukey [4], the Fast Fourier Transform (FFT) algorithm has undergone a number of developments, and its evolution continues to produce some surprises. Consider, for example, the familiar radix-2 version of the algorithm, for transform lengths of the form $N = 2^p$. All direct descendants of the Cooley–Tukey algorithm suffer from one of the following disadvantages: either the computation is done in-place, but a scrambling or unscrambling step is required before or after the transform; or the transform is "self-sorting," but a work space of length N is required. In the first case, the reordering phase can significantly increase the computation time. In the second case, the work space required can be a considerable nuisance, particularly in the case of large or multiple transforms; if M transforms are being performed simultaneously, as would be typical on a vector computer [12], then a work array of length MN is needed. Swarztrauber [11] provides a good discussion of these issues.

All this inconvenience would of course be avoided if the FFT algorithm were both self-sorting and in-place—but these two desirable properties seemed until recently to be mutually exclusive.

One FFT algorithm which does succeed in being both self-sorting and in-place [2], [9], [14] is a variant of the Prime Factor Algorithm (PFA), descended from the algorithm of Good [6]. The PFA has the additional advantage of a lower operation count than the Cooley–Tukey algorithm, but unfortunately it is only applicable for a very restricted set of values of the transform length N.

In 1984, Johnson and Burrus [7] showed that the radix-2 Cooley–Tukey algorithm can be restructured so that it too is both self-sorting and in-place. More recently, a similar principle was applied to the radix-2 Fast Hartley Transform by Buneman [1], who pointed out that the basic idea goes back at least to 1975 [8, p. 578]. Johnson and Burrus start from the "decimation-in-time" form of the FFT; the present paper develops the corresponding "decimation-in-frequency" form of the radix-2 algorithm, from a somewhat different viewpoint. We also show that the same idea can be extended to radix-3, radix-4, and radix-5 algorithms, and finally to the mixed-radix case.

* Received by the editors October 11, 1989; accepted for publication (in revised form) February 25, 1990. Much of this work was done at Recherche en Prévision Numérique, Atmospheric Environment Service, Dorval, Québec, Canada.

† European Centre for Medium-Range Weather Forecasts, Shinfield Park, Reading, Berkshire RG2 9AX, United Kingdom.

The paper is organized as follows. Section 2 reviews the decimation-in-frequency form of the conventional radix-2 algorithm, and then derives the new self-sorting in-place version. In § 3, the extension to other radices is demonstrated. Section 4 presents timing results for the new algorithm as implemented for multiple transforms on a Cray X-MP, and shows that self-sorting in-place transforms can be achieved in practice with no penalty in computation time. The generalization to the mixed-radix case is discussed in § 5, while § 6 includes both a summary of this paper and a "trailer" for further developments to be presented in a sequel.

2. Derivation of radix-2 algorithms. In deriving various forms of the FFT algorithm and demonstrating the relationships between them, a useful approach is to represent the Discrete Fourier Transform (DFT) as a matrix-vector multiplication. We define the DFT for complex x_j, z_k as

$$x_j = \sum_{k=0}^{N-1} z_k \omega_N^{jk}, \qquad 0 \leqq j \leqq N-1, \tag{2.1}$$

where $\omega_N = \exp(2i\pi/N)$. (Everything goes through with only minor changes if we define $\omega_N = \exp(-2i\pi/N)$ instead.) Equation (2.1) can be expressed as

$$\mathbf{x} = W_N \mathbf{z} \tag{2.2}$$

where element (j, k) of W_N is ω_N^{jk} (rows and columns of W_N are indexed from 0 to $N-1$). We call W_N the DFT matrix of order N. FFT algorithms are then defined by factorizing W_N into the product of a sequence of sparse matrices. This approach was taken in [13], and the present paper will use the same notation. In this section we will first present the decimation-in-frequency or Gentleman–Sande [5] radix-2 algorithm, which is in-place but yields the results in scrambled order. We will then show how the device of Johnson and Burrus [7] eliminates the need for unscrambling, while maintaining the computation in-place. To make the development less abstract—and to reinforce the links between different ways of looking at FFT algorithms—we will also make some use of "signal flow" diagrams, simple Fortran code, and index permutations.

2.1. The Gentleman–Sande algorithm. The fundamental result needed for building up FFT algorithms, as presented in [13], is as follows. Suppose $N = pq$. Let W_p and W_q be the DFT matrices of order p, q, respectively, and let I_p, I_q be the corresponding identity matrices. Define a permutation matrix P_q^p of order $N = pq$ by

$$P_q^p(j, k) = 1 \quad \text{if } j = rp + s \text{ and } k = sq + r,$$
$$= 0 \quad \text{otherwise.}$$

Finally, define a diagonal matrix D_q^p of order $N = pq$ by

$$D_q^p(j, k) = \omega_N^{sm} \quad \text{if } j = k = sq + m,$$
$$= 0 \quad \text{otherwise.}$$

Then

$$W_{pq} = (W_q \times I_p) P_q^p D_q^p (W_p \times I_q) \tag{2.3}$$

where $\times$ denotes the Kronecker (tensor) product. Equation (2.3) factorizes the DFT of length pq into q DFTs of length p, multiplication by a diagonal matrix of "twiddle factors," a permutation, and finally p DFTs of length q. The Gentleman–Sande algorithm for $N = pq$ follows by using the lemma

$$P_q^p (A_p \times B_q) = (B_q \times A_p) P_q^p \tag{2.4}$$

for any square matrices A_p, B_q of order p, q, respectively. Thus, (2.3) becomes

$$W_{pq} = P_q^p (I_p \times W_q) D_q^p (W_p \times I_q), \tag{2.5}$$

and the permutation matrix P_q^p represents the final "unscrambling" step.

The extension to more factors can be derived by induction. Suppose

$$N = n_1 n_2 \cdots n_{k-1} n_k.$$

Define $l_1 = 1$, $l_{i+1} = n_i l_i$ for $1 \leqq i \leqq k-1$, and $m_i = N/l_{i+1}$. Then the general case is given by

$$W_N = P_1 P_2 \cdots P_{k-1} P_k T_k T_{k-1} \cdots T_2 T_1 \tag{2.6}$$

where

$$T_i = I_{l_i} \times [D_{m_i}^{n_i} (W_{n_i} \times I_{m_i})], \qquad P_i = (I_{l_i} \times P_{m_i}^{n_i}).$$

To make this development more concrete, let us consider the specific case $N = 16 = 2^4$. Then (2.6) becomes

$$W_{16} = P_1 P_2 P_3 P_4 T_4 T_3 T_2 T_1 \tag{2.7}$$

where

$$T_1 = D_8^2 (W_2 \times I_8), \qquad T_2 = I_2 \times [D_4^2 (W_2 \times I_4)],$$
$$T_3 = I_4 \times [D_2^2 (W_2 \times I_2)], \qquad T_4 = I_8 \times W_2, \tag{2.8}$$

$$P_4 = I_8 \times P_1^2 = I_{16}, \qquad P_3 = I_4 \times P_2^2,$$
$$P_2 = I_2 \times P_4^2, \qquad P_1 = P_8^2. \tag{2.9}$$

The sequence of permutation matrices $P_1 P_2 P_3 P_4$ represents the unscrambling, which may in practice be done as a single step. For the time being we will concentrate on the four stages T_1 through T_4, in which the actual floating-point arithmetic is done.

A popular way of visualizing these stages is through a signal flow diagram, based on the FFT "butterfly" as depicted in Fig. 1. In the box, the inputs a, b are added and subtracted; the difference is multiplied by the twiddle factor ω_N^r (the box is labelled with the appropriate exponent of ω_N), and the outputs are written back to the input locations (hence the computation is in-place). A signal flow diagram for the case $N = 16$ is presented in Fig. 2 (T_1, T_2) and Fig. 3 (T_3, T_4). The two halves of the algorithm have been separated in this way both for clarity and in preparation for later modifications.

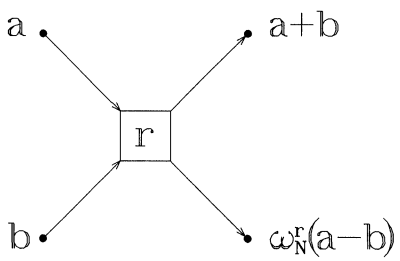


FIG 1. *The* FFT "*butterfly.*"

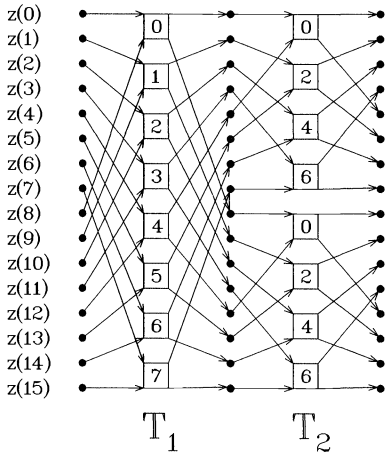


FIG 2. *Signal flow diagram for* $N = 16$: *first and second stages.*

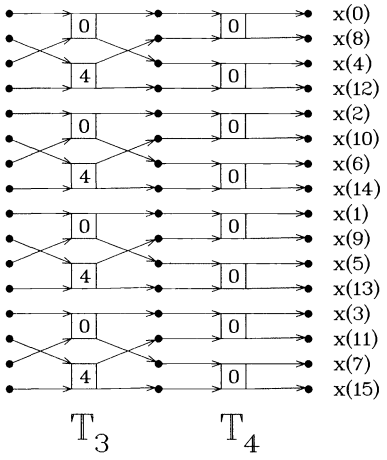


FIG 3. *Signal flow diagram for* $N = 16$: *third and fourth stages.*

Readers who are more familiar with the signal flow view of the FFT than with the matrix factorization approach (or, indeed, vice versa) may find it illuminating to compare Figs. 2 and 3 with equations (2.8). For example, the representation of T_2 as $I_2 \times$ (matrix of order $N/2$) indicates that the corresponding stage of the signal flow diagram consists of identical "top" and "bottom" halves. Similarly, $T_3 = I_4 \times$ (matrix of order $N/4$) implies four identical blocks in the signal flow diagram.

To conclude this section, we present a short Fortran routine which implements the radix-2 algorithm (minus the unscrambling stage) for $N = 2^M$. Assume that the data are in an array X, and that a table

$$\text{TRIGS (K)} = \omega_N^{\text{K}-1}, \qquad 1 \leqq \text{K} \leqq N/2$$

has been precomputed. The following code will then perform the required computation.

```
COMPLEX X(N), TRIGS (N), W, Z
NH = N/2
DO 30 L = 1, M
LA = 2**(L-1)
IA = 0
```

```
      IB = NH/LA
      KK = 1
      DO 20 K = 0, IB-1
      W = TRIGS (KK)
      DO 10 I = K+1, N, N/LA
      Z = W*(X(IA+I) - X(IB+I))
      X(IA+I) = X(IA+I) + X(IB+I)
      X(IB+I) = Z
10    CONTINUE
      KK = KK+LA
20    CONTINUE
30    CONTINUE
```

The base address IA is redundant, but is retained for symmetry (and in readiness for future developments). In practice, a significant saving is obtained by providing a special version of the innermost loop for the case K = 0 (then W = 1). Smaller savings accrue from providing for the special cases W = i, W = exp $(i\pi/4)$, and W = exp $(3i\pi/4)$ (see [3, Chap. 2.2.11], for example).

2.2. Unscrambling the results. As indicated at the right-hand end of Fig. 3, the results of the in-place radix-2 FFT algorithm are in scrambled order. They may be restored to natural order by applying the sequence of permutation matrices defined in (2.9). A faster method would be to combine the permutations into a single stage, using a precomputed index list. It is well known that the required permutation is given by "bit reversal," for example, in the present case $X(1) = X(0001)$ has to be exchanged with $X(1000) = X(8)$. Since it is not immediately obvious that applying the permutation matrices of (2.9) would give the same result, it is instructive to demonstrate the equivalence. To do this, we set up an *index mapping*

$$(k_3, k_2, k_1, k_0) \leftrightarrow k_3 \cdot 2^3 + k_2 \cdot 2^2 + k_1 \cdot 2^1 + k_0 \tag{2.10}$$

where each $k_i (0 \leqq i \leqq 3)$ can take the value zero or one. Equation (2.10) establishes a mapping between the set of integers $0 \leqq k \leqq 15$ and their corresponding representations in binary (radix-2) notation.

Each of the permutation matrices of (2.9) is of the form

$$P_i = I_{2^{i-1}} \times P^2_{16/2^i}. \tag{2.11}$$

In general, premultiplying a vector $\mathbf{x}$ by a permutation matrix P_i sends $x(k)$ to location $\hat{k}$, and we can examine its effect by comparing the binary representation of k and $\hat{k}$. For P_i defined by (2.11) we find that the first $i-1$ bits are unchanged, while the remaining bits undergo a cyclic leftward shift. P_4 is just the identity matrix. The effect of successively applying P_3, P_2, P_1 is shown in Fig. 4, which demonstrates that this sequence is indeed equivalent to bit-reversal.

Note that bit-reversal using a precomputed index list can easily be done in-place, since it consists only of pairwise exchanges. On the other hand, using the three-stage permutation procedure of (2.9) (in general, for $N = 2^M$ there are $(M-1)$ stages) would require a work space of length N to accommodate the necessary data shuffling.

2.3. The Johnson–Burrus scheme. For the case $N = 16$, we have just shown that the bit-reversal permutation is given by the product $P_1P_2P_3$. It is obvious that the same permutation is given by the product $Q^0_3Q^1_2$, where the permutation matrix Q^i_j is such that multiplying $\mathbf{x}$ by Q^i_j interchanges $x(k)$ and $x(\hat{k})$, $\hat{k}$ being obtained from k by interchanging bits i and j of its binary representation. This two-stage version of the

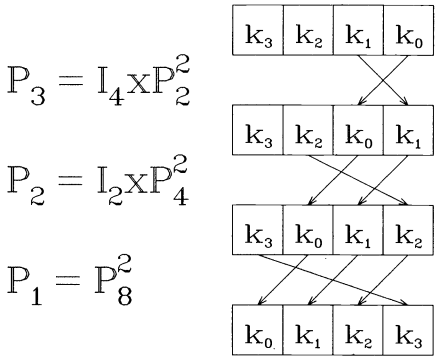


FIG. 4. *Demonstrating bit-reversal by the permutation matrices of* (2.9), *for* $N = 16$.

bit-reversal procedure is demonstrated in Fig. 5. How are the Q-matrices related to the original P-matrices? It is quite easy to show that

$$Q_2^1 = I_2 \times P_2^2 \times I_2, \tag{2.12}$$

but Q_3^0 cannot easily be expressed in terms of the P-matrices.

To look ahead for a moment, suppose $N = 2^M$ where $M = 2B$. In terms of Q-matrices, the bit-reversal permutation can be written

$$Q_{M-1}^0 Q_{M-2}^1 \cdots Q_B^{B-1}.$$

If M is odd, say, $M = 2B+1$, then the permutation is

$$Q_{M-1}^0 Q_{M-2}^1 \cdots Q_{B+1}^{B-1}.$$

In either case,

$$Q_{M-i-1}^i = I_{2^i} \times \Omega_i \times I_{2^i} \tag{2.13}$$

for some permutation matrix Ω_i of order $N/4^i$.

Returning to the case $N = 16$, we can now rewrite (2.7) as

$$W_{16} = Q_3^0 Q_2^1 T_4 T_3 T_2 T_1. \tag{2.14}$$

Using (2.8) and (2.12), and the algebra of Kronecker products,

$$\begin{aligned} Q_2^1 T_4 &= (I_2 \times P_2^2 \times I_2)(I_8 \times W_2) \\ &= (I_2 \times P_2^2 \times I_2)(I_2 \times I_4 \times W_2) \\ &= (I_2 \times P_2^2 \times W_2) \\ &= (I_2 \times I_4 \times W_2)(I_2 \times P_2^2 \times I_2) \\ &= (I_8 \times W_2)(I_2 \times P_2^2 \times I_2) \\ &= T_4 Q_2^1. \end{aligned}$$

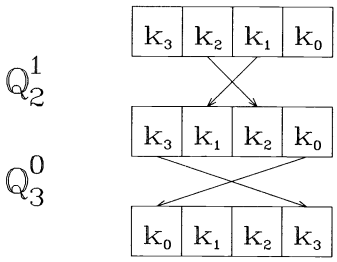


FIG. 5. *Demonstrating bit-reversal by the permutation matrices* Q_2^1, Q_3^0.

Thus Q_2^1 and T_4 commute, and (2.14) becomes

$$W_{16} = (Q_3^0 T_4)(Q_2^1 T_3) T_2 T_1 . \tag{2.15}$$

Now consider

$$W_{64} = Q_5^0 Q_4^1 Q_3^2 T_6 T_5 T_4 T_3 T_2 T_1 . \tag{2.16}$$

Making use of (2.13) and following a similar argument, Q_3^2 commutes with both T_6 and T_5, while Q_4^1 commutes with T_6. Thus, (2.16) becomes

$$W_{64} = (Q_5^0 T_6)(Q_4^1 T_5)(Q_3^2 T_4) T_3 T_2 T_1 . \tag{2.17}$$

Similarly, for the intermediate transform size $N = 32$, we have

$$W_{32} = (Q_4^0 T_5)(Q_3^1 T_4) T_3 T_2 T_1 . \tag{2.18}$$

Equations (2.15), (2.17), and (2.18) demonstrate that the Q-matrices can be "spread back" into the transform stages; each stage during the second half of the algorithm is coupled with a permutation matrix (if N is an odd power of two, then the central stage is of the regular form). The important point here is that the Q-matrices represent pairwise exchanges, so these permutations can be done in-place.

The signal flow graph for the second half of the algorithm is based on "coupled" FFT butterflies, as shown in Fig. 6. Here two butterflies are considered together, and two of the four outputs are interchanged. Nevertheless, the four output values are written back to the four input locations, and the computation can still be done in-place. Conveniently, the two coupled butterflies will always have the same twiddle factor ω_N^r. The signal flow graph for the second half of the revised $N = 16$ algorithm is presented in Fig. 7, which can be compared with Fig. 3; we see that all the output values are now naturally ordered, the permutations having been combined with stages T_3 and T_4 simply by appropriately coupling the butterflies.

The code given earlier is modified to perform only the first half of the algorithm by reducing the limit on the outer DO 30 loop to (M+1)/2. The second half of the algorithm is given by:

```
DO 70 L = (M+3)/2, M
LA = 2**(L-1)
IA = 0
IB = NH/LA
IC = LA
ID = IB+LA
```

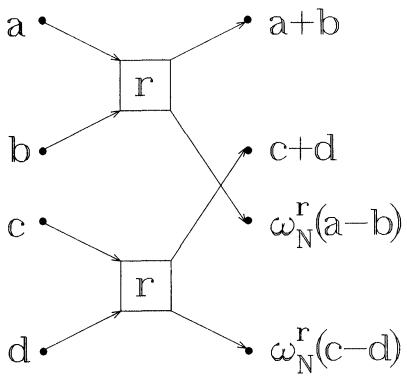


FIG. 6. *Coupled* FFT *butterflies.*

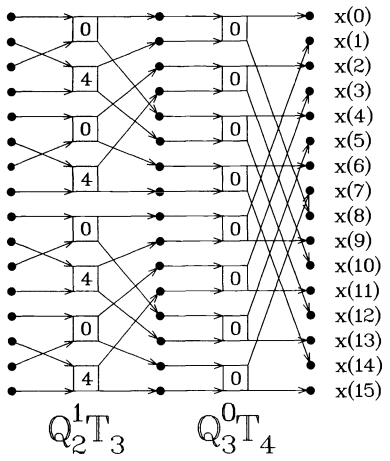


FIG. 7. *Signal flow diagram for* $N = 16$: *third and fourth stages of the self-sorting in-place algorithm.*

```
      KK = 1
      DO 60 K = 0, IB = 1
      W = TRIGS (KK)
      DO 50 J = K, LA - 1, N/LA
      DO 40 I = J + 1, N, 2*LA
      Z = W*(X(IA + I) - X(IB + I))
      X(IA + I) = X(IA + I) + X(IB + I)
      X(IB + I) = X(IC + I) + X(ID + I)
      X(ID + I) = W*(X(IC + I) - X(ID + I))
      X(IC + I) = Z
40    CONTINUE
50    CONTINUE
      KK = KK + LA
60    CONTINUE
70    CONTINUE
```

The innermost loop of this code corresponds to the coupled butterflies of Fig. 6. As before, redundant multiplications by one can be avoided by providing a special inner loop for the case $K = 0$. Johnson and Burrus [7] point out (for the equivalent decimation-in-time formulation) that if we compare the new innermost loop with the original, there is twice as much computation and twice as much data flow between the memory and the arithmetic unit. As before, a single temporary variable (Z) has to be computed and stored (for example, in a register). Thus, coupling the butterflies introduces no additional demands on the hardware.

Similar considerations apply if we implement this algorithm on a vector computer with register-to-register arithmetic, in the case of multiple simultaneous transforms. This introduces a further innermost level of looping into the code; the innermost loop now "steps across" the simultaneous transforms, with constant stride, and the vector length is just the number of transforms. All the indexing complications are then transferred to the outer DO-loops, and become irrelevant from the point of view of vectorization.

We now look at the number of vector loads, stores, real additions (or subtractions), and real multiplications in the innermost loops. There are four cases to consider—single

and coupled butterflies, with ($K > 0$) or without ($K = 0$) nontrivial twiddle factors. The information is summarized in Table 1. If there is a single path between the memory and the registers, as on the Cray-1, then in any of these four cases the time taken for the loads and stores will dominate (the algorithm will be memory-bound). If on the other hand there are two paths, one for loading and one for storing, then only the number of *additions* is relevant—all the memory traffic can be overlapped with them. This situation holds on the Cray X-MP, for example. (In fact, two paths are provided for loading, but only one of them is needed here.) Again, the coupled butterflies place no extra demands on the hardware.

We conclude this section by sketching the derivation of the corresponding decimation-in-time algorithm, in the form proposed by Johnson and Burrus [7]. Equation (2.15) will serve as an example. Since the matrix W_{16} is symmetric, we can transpose both sides of the equation. Using the identities $(AB)^T = B^T A^T$, $(A \times B)^T = (A^T \times B^T)$ and the fact that the W, D, I, and Q matrices are all symmetric, and finally relabelling the transform stages, (2.15) becomes

$$W_{16} = T_4 T_3 (T_2 Q_2^1)(T_1 Q_3^0) \tag{2.19}$$

where

$$T_i = I_{m_i} \times [(W_{n_i} \times I_{l_i}) D_{l_i}^{n_i}]$$

and for the radix-2 algorithm $l_i = 2^{i-1}$, $m_i = N/2^i$, $n_i = 2$. Equation (2.19) shows that the permutations (coupled butterflies) are now done during the *first* half of the algorithm, as described in [7].

Since the operators in (2.19) are read from right to left, the permutation Q_3^0 is done before the butterfly stage T_1, and so on—i.e., the *inputs* to the butterflies are exchanged, as in Fig. 8(a). (The computation in the butterfly is now slightly different from that in the decimation-in-frequency case.) Nevertheless, Johnson and Burrus [7] describe their algorithm as exchanging the *outputs* from the butterflies. The paradox is easily resolved. Since nothing has been said about the relative positions of b and c within the array, we could equally well have drawn the diagram as in Fig. 8(b), which does indeed appear to exchange the outputs as in Fig. 6. By the same token, we could have redrawn Fig. 6 to make it look like Fig. 8(a), and described the algorithm as exchanging the inputs. In principle there is no difference. However, in practice, it is Fig. 6 (for decimation-in-frequency) and Fig. 8(a) (for decimation-in-time) which reflect the true geometry of the situation.

3. Generalization to other bases. In § 2, we showed that a radix-2 FFT could be made both self-sorting and in-place, using the device suggested by Johnson and Burrus [7]. In fact, it turns out that there is nothing special about radix-2 in this regard, and the result generalizes to other bases. By way of example, consider a radix-3 algorithm

TABLE 1
Inner loops of the radix-2 algorithm.

Case	Loads	Stores	Adds	Mults
Single butterfly:				
$K = 0$	4	4	4	0
$K > 0$	4	4	6	4
Coupled butterfly:				
$K = 0$	8	8	8	0
$K > 0$	8	8	12	8

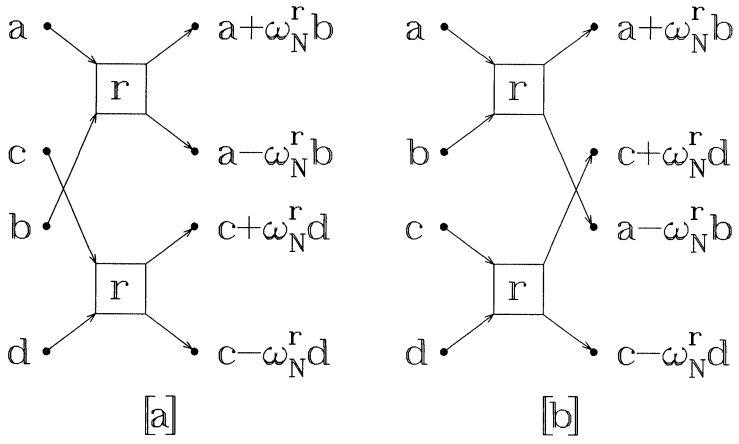


FIG. 8. *Coupled butterflies for decimation-in-time* FFT.

for $N = 81 = 3^4$. The analogue of (2.7) is

$$W_{81} = P_1P_2P_3P_4T_4T_3T_2T_1 \tag{3.1}$$

where

$$T_1 = D_{27}^3(W_3 \times I_{27}), \qquad T_2 = I_3 \times [D_9^3(W_3 \times I_9)],$$
$$T_3 = I_9 \times [D_3^3(W_3 \times I_3)], \qquad T_4 = I_{27} \times W_3.$$

The sequence of permutation matrices $P_1P_2P_3P_4$ is equivalent to "trit-reversal," i.e., exchanging $x(k)$ and $x(\hat{k})$, where

$$k = k_3 \cdot 3^3 + k_2 \cdot 3^2 + k_1 \cdot 3^1 + k_0, \qquad \hat{k} = k_0 \cdot 3^3 + k_1 \cdot 3^2 + k_2 \cdot 3^1 + k_3.$$

Again we can rewrite this permutation as $Q_3^0Q_2^1$, where Q_j^i corresponds to exchanging "trits" i and j of the ternary representation. The analogue of (2.12) is

$$Q_2^1 = I_3 \times P_3^3 \times I_3,$$

and again it is easily shown that Q_2^1 commutes with T_4, so that we have

$$W_{81} = (Q_3^0T_4)(Q_2^1T_3)T_2T_1. \tag{3.2}$$

The extension to more factors (higher powers of three) proceeds exactly as in the radix-2 case.

During the second half of the algorithm, e.g., the stages $(Q_2^1T_3)$ and $(Q_3^0T_4)$ of (3.2), we have to couple the radix-3 "butterflies" in groups of *three*, and interchange the outputs appropriately. The key to visualizing this procedure is to reconsider for a moment the coupled butterflies of the radix-2 algorithm. Another way of looking at the operations performed in Fig. 6 is shown in Fig. 9. The input pairs (a, b) and (c, d) are now regarded as *rows* of a 2×2 array (Fig. 9(a)). We do a length-2 transform along each row (and apply the twiddle factors), yielding the output pairs $(\hat{a}, \hat{b})$ and $(\hat{c}, \hat{d})$ in the corresponding locations (Fig. 9(b)). Finally, we *transpose* the 2×2 array (Fig. 9(c)).

The generalization to the radix-3 case is now fairly obvious, and is depicted in Fig. 10. We do a length-3 transform along each row of a 3×3 array, apply the twiddle factors, and transpose the array of results.

We now consider the practical implementation of this procedure. One way would be simply to do the three transforms one at a time, returning the results to the original locations, and finally to do an explicit transpose of the 3×3 array. However, the

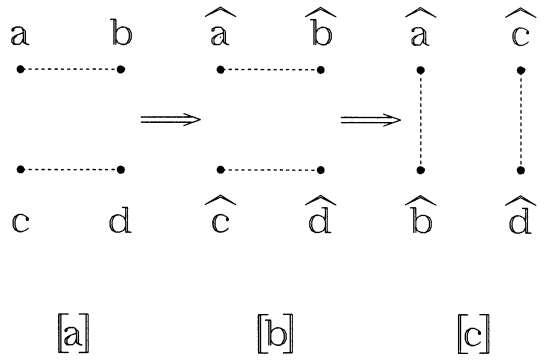


FIG. 9. *Another way of looking at coupled radix-2 butterflies.*

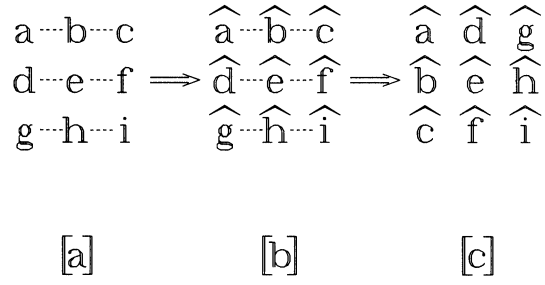


FIG. 10. *Three coupled radix-3 "butterflies."*

transpose would require time which would not be overlapped with floating-point computation.

If we had enough registers in which temporary results could be stored, then a direct analogue of the radix-2 algorithm would be possible. This would proceed as follows:

(1) Do the transform on the first row of Fig. 10(a). Store the result $\hat{a}$ in its correct location, but hold $\hat{b}$ and $\hat{c}$ in temporary registers.

(2) Do the transform on the second row of Fig. 10(a). Store the result $\hat{b}$ from the first transform in its final location (Fig. 10(c)). Results $\hat{d}$ and $\hat{e}$ from the second transform can also be stored in their final locations, but $\hat{f}$ is held in a temporary register along with the result $\hat{c}$ from the first transform.

(3) Do the transform on the third row of Fig. 10(a). The three results $\hat{g}$, $\hat{h}$, $\hat{i}$ can be stored in their final locations, as can the results $\hat{c}$ and $\hat{f}$ which were held over from the previous transforms.

Extending this strategy to radix-p FFTs requires $p^2/4$ temporary results for p even, and $(p^2-1)/4$ for p odd. In fact, each of these temporary results really requires two registers (one for the real part, one for the imaginary part), and the problem is that on machines such as the Cray-1, X-MP, and Y-MP there are only eight vector registers (per processor), which is already only just enough for the FFT computation to proceed smoothly.

The strategy finally adopted for implementation on the Cray X-MP, shown in Fig. 11, makes use of the spare memory bandwidth. It proceeds as follows:

(1) Do the transform on the first row of Fig. 11(a). Once the input value b has been loaded into registers ready for use in the computation, transfer the value d (via

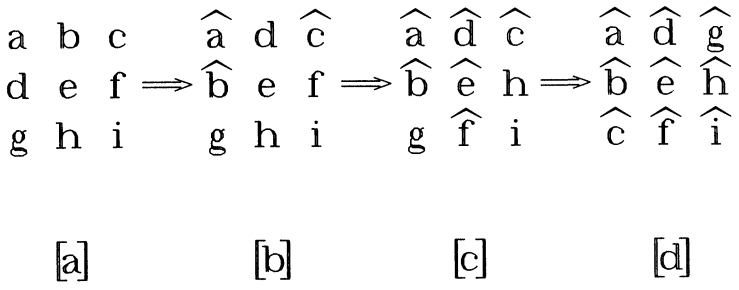


FIG. 11. *Transpose strategy for radix-3.*

a register) to the location formerly occupied by b. Store $\hat{a}$ and $\hat{b}$ in their final locations, and store $\hat{c}$ in the old c-location. The position is now as in Fig. 11(b).

(2) Do the transform on the input values d, e, f (the original second row). Once f has been loaded, transfer h to the location previously occupied by f. The output values $\hat{d}, \hat{e}, \hat{f}$ can now all be stored in their final locations, as shown in Fig. 11(c).

(3) Do the transform on the input values g, h, i (the original third row). Once g has been loaded, transfer $\hat{c}$ to the location previously occupied by g. The output values $\hat{g}, \hat{h}, \hat{i}$ can be stored in their final locations, and the original 3×3 array has now been transformed along each row and the resulting array transposed, as in Fig. 11(d).

During each transform of length three, there are two extra loads (one for the real part of the value being transferred, and one for its imaginary part) and two extra stores. One vector register is rather briefly occupied during each transfer, otherwise there are no additional demands on the allocation of registers. The order in which the transfers have to be done is not unique. The strategy generalizes readily to radix-4 and radix-5; during each radix-p transform, $(p-1)$ real numbers have to be transferred, so there are $(p-1)$ extra loads and $(p-1)$ extra stores.

Table 2 compares the cost of the simple and "coupled" radix-p transforms, using the above strategy, for $p=3$, 4, and 5, in each case considering separately $K=0$ (no twiddle factors) and $K>0$ (twiddle factors present). The information has been compressed by noting that the floating-point computation is the same in the simple and coupled cases; and in either case, half the memory references are loads, and half are stores.

Since there are more additions than multiplications, the important question is whether the memory bandwidth is sufficient to keep the addition unit occupied all the time (or as nearly so as possible). Table 2 shows that for the "simple" cases (as in a more conventional FFT algorithm) this is true even if there is only a single path between

TABLE 2
Cost of a radix-p transform.

Radix	Adds	Mults	Memory references	
			Simple	Coupled
$p=3, K=0$	12	4	12	16
$p=3, K>0$	16	12	12	16
$p=4, K=0$	16	0	16	22
$p=4, K>0$	22	12	16	22
$p=5, K=0$	32	12	20	28
$p=5, K>0$	40	28	20	28

the memory and the registers (as on the Cray-1). For the "coupled" cases, a single path is insufficient for $p=3$ or $p=4$ without twiddle factors, but two paths (one for loading, one for storing) are always easily sufficient.

This analysis ignores the fact that, if only eight vector registers are available, then during the more involved loops there are always some intermediate quantities computed which have to be temporarily stored and later reloaded. However, the timing results to be presented in the next section will demonstrate that the analysis remains valid in practice, and that two paths between memory and vector registers are indeed sufficient to permit maximum utilization of the addition unit during self-sorting, in-place transforms.

4. Implementation on a Cray X-MP. To demonstrate the feasibility of these ideas, three routines have been written and implemented on a Cray X-MP for radix-3, radix-4, and radix-5 transforms. The routines perform M simultaneous complex transforms. The strides "along" and "between" the transforms are input parameters, so that, for example, a two-dimensional transform could be performed without having to transpose the data between the transforms in the x-direction and those in the y-direction. Real and imaginary parts are stored in separate arrays (but since the strides are specified by the user, the two arrays could be interleaved to give the conventional Fortran way of storing complex numbers).

These new routines have been compared against older routines which implement self-sorting transforms by alternating between the original data array and a work array of the same size [13]. Since these older routines were written in CAL (Cray Assembler Language), the new routines were also CAL-coded in order to remain competitive. To reiterate, the new routines compute in-place, in-order transforms *without using a single word of work space.* Whenever intermediate results had to be temporarily stored in order to free registers for further computation, it was always possible to store them in the original data array. It is also worth mentioning that, although free use was made of bidirectional memory transfers (simultaneous loads and stores), the facility to load two vectors simultaneously was almost never used—thus the code would have run almost as well had there been only a single port for vector loads.

For the timing experiments, 64 transforms were performed simultaneously. The experiments were run using a single processor of an X-MP with clock cycle 9.5 ns. The results are shown in Table 3, for various values of N. The operation count for a transform of length N is also shown. It is clear from Table 3 that the new self-sorting in-place transform routines consistently run 10–15 percent *faster* than the old routines which required work space. Since the operation count is the same in both cases, this simply reflects the fact that the old routines had been coded for the Cray-1. The new routines were coded for the X-MP, and were thus able to benefit from new features (notably the facility to use a single vector register to hold both an operand and the result of a vector addition or multiplication; this feature eases considerably the task of keeping everything within eight vector registers). If the old routines were recoded for the X-MP, then they would be as fast as the new routines—but no faster, as we shall see in a moment.

The last column of Table 3 gives a measure of the *efficiency* of the new routines. Since the arithmetic complexity is dominated by the number of additions, the minimum possible time per transform is given by

$$(\text{Number of additions per transform}) \times (68/64) \times (\text{clock cycle time}).$$

The factor (68/64) arises because vector addition instructions with vector length $n \leqq 64$ (in this case $n=64$) can be issued every $(n+4)$ clock cycles—i.e., there is an *effective*

TABLE 3
Timing comparisons for self-sorting transforms of length N.

N	Real operations per transform		Time per transform (μs)		Efficiency of new routine (%)
	+	*	Old routine	New routine	
$125 = 5^3$	2,752	1,604	32.3	28.0	99.3
$243 = 3^5$	5,996	3,892	68.7	61.6	98.2
$256 = 4^4$	5,122	2,052	58.2	52.0	99.5
$625 = 5^4$	18,752	11,504	219.5	189.5	99.9
$729 = 3^6$	21,872	14,584	249.1	224.9	98.2
$1{,}024 = 4^5$	26,114	11,268	294.9	264.5	99.7
$2{,}187 = 3^7$	77,276	52,492	877.2	791.3	98.6
$3{,}125 = 5^5$	118,752	75,004	1,396	1,201	99.8
$4{,}096 = 4^6$	126,978	57,348	1,426	1,284	99.8

vector startup time of four clock cycles. The efficiency is then defined as follows:

$$\frac{\text{Minimum possible time}}{\text{Measured time}} \times 100\%.$$

Since the efficiency is almost 100 percent, this means that the transforms cannot be made to run faster unless the number of additions can be reduced. In particular, it confirms that the self-sorting in-place FFT algorithm achieves its saving in work space without any penalty in computation time.

5. The mixed-radix case. Suppose now that $N = n_1 n_2 \cdots n_{p-1} n_p$. Equation (2.6) for the factorization of W_N remains true even when the n_i's are different; this is the mixed-radix case. Again we can replace the sequence of permutation matrices by $Q_{p-1}^0 Q_{p-2}^1 \cdots$, where Q_j^i is such that premultiplying $\mathbf{x}$ by Q_j^i sends $x(k)$ to location $\hat{k}$ where $\hat{k}$ is obtained from k by interchanging digits i and j of its mixed-radix representation. However, we must now be more careful, since if the factors are used successively in the order $n_1, n_2, \cdots, n_p$, then k is of the form

$$k = k_{p-1} n_{p-1} n_{p-2} \cdots n_1 + k_{p-2} n_{p-2} \cdots n_1 + \cdots + k_1 n_1 + k_0 \tag{5.1}$$

while $\hat{k}$ is of the form

$$\hat{k} = \hat{k}_{p-1} n_p n_{p-1} \cdots n_2 + \hat{k}_{p-2} n_p n_{p-1} \cdots n_3 + \cdots + \hat{k}_1 n_p + k_0. \tag{5.2}$$

The essential feature in § 2.3, which made it possible to combine the permutations with the transform stages by coupling the "butterflies," was that digit reversal (k with $\hat{k}$ in the above example) implied *pairwise exchanges.* Thus, if Q_j^i sent $x(k)$ to location $\hat{k}$, then it also sent $x(\hat{k})$ to location k. Comparing (5.1) and (5.2), this feature carries over to the mixed-radix case provided that $n_{p+1-i} = n_i$ for all i, $1 \leqq i \leqq p$. In other words, everything will still work as before provided that the factorization of N is a *palindrome.* Thus, the algorithm will work for $N = 144 = 3^2 \cdot 4^2$ if the factors are used in the order 3, 4, 4, 3 or 4, 3, 3, 4, but it will *not* work if they are used in the order 3, 3, 4, 4 or 4, 4, 3, 3.

This mixed-radix feature was used to generalize the radix-4 routine (described in the previous section) to perform self-sorting in-place transforms for any length $N = 2^p$, by adding "central" radix-2 and radix-8 sections. Thus, $N = 32$ is factorized as $4 \times 2 \times 4$, $N = 128$ as $4 \times 8 \times 4$, $N = 512$ as $4 \times 4 \times 2 \times 4 \times 4$, and so on. Note that (fortunately in the radix-8 case!) this central section is of the simple rather than the "coupled" type. Some timing results are shown in Table 4, in the same format as those of Table 3. The operation counts are for the new algorithm; since the old algorithm always factorizes

TABLE 4
Timing comparisons for self-sorting transforms of length N.

N	Real operations per transform		Time per transform (μs)		Efficiency of new routine (%)
	+	*	Old routine	New routine	
$128 = 4 \cdot 8 \cdot 4$	2,210	772	26.5	22.5	99.1
$512 = 4^2 \cdot 2 \cdot 4^2$	11,746	5,060	135.9	119.1	99.8
$2{,}048 = 4^2 \cdot 8 \cdot 4^2$	57,858	24,580	666.6	584.5	99.9

odd powers of two in the form $N = 2 \cdot 4^r$, there are now slight differences in the operation counts between the two routines.

For many useful transform lengths of the form $N = 2^p 3^q 5^r$, it is not possible to arrange the factors in the symmetrical manner required for the self-sorting in-place FFT algorithm. One final trick permits the extension of this algorithm to *any* $N = 2^p 3^q 5^r$.

Consider, for example, $N = 6{,}000 = 3 \cdot 4^2 \cdot 5^3$. We split this factorization into a *squared* part and a *square-free* part, $N = (4 \times 5)^2 \times (3 \times 5)$, and arrange the factors as

$$N = 4 \times 5 \times (3 \times 5) \times 5 \times 4.$$

The algorithm will now work provided that we regard the central section as a single radix-15 pass. (Again this central pass is of the "simple" type.) A self-sorting in-place transform of length 15 can be performed using the variant of the prime factor FFT algorithm (PFA) described in [14], and the whole transform of length $N = 6{,}000$ will then be both self-sorting and in-place. It is easily seen that for any $N = 2^p 3^q 5^r$, the radix for the "square-free" central pass (if it is necessary at all) must be a member of the set $\{6, 10, 15, 30\}$, all of which can be handled by the PFA algorithm of [14]. We have thus been able to extend the self-sorting in-place algorithm to any transform length of the form $N = 2^p 3^q 5^r$. In passing, it is no accident that the symmetrical arrangement of the factors about a square-free central portion corresponds exactly to the procedure suggested by Singleton [10] to minimize the amount of work space required during the unscrambling phase of an in-place (but not self-sorting) FFT algorithm.

6. Summary and discussion. It has long been generally thought that FFT algorithms (apart from the special case of the PFA) cannot simultaneously be "self-sorting" and "in-place." Following the demonstration by Johnson and Burrus [7] that these two desirable properties can in fact be combined in a radix-2 algorithm, we have shown in this paper that the same principle can be extended to radix-3, radix-4, and radix-5 algorithms. Timing comparisons on a Cray X-MP show that these algorithms eliminate both the work space of the usual self-sorting algorithm and the permutation step of the usual in-place algorithm, with no penalty in computation time.

We have also shown that the same principle can be used in the general case $N = 2^p 3^q 5^r$; this extension makes some use of the prime factor FFT algorithm (PFA) [14].

However, it turns out that this is not the most efficient way of using the PFA. In a sequel to this paper, it will be shown that through a different combination of ideas from the self-sorting, in-place algorithm described here, and from the PFA, we can put together a new algorithm with the following nice properties:

(1) It works for any transform length of the form $N = 2^p 3^q 5^r$;

(2) It is always self-sorting and in-place;

(3) For values of N suitable for the PFA, the new algorithm reduces to the PFA and is just as fast;

(4) For $N = 2^p$, 3^q, or 5^r, the new algorithm reduces to the algorithm in this paper and is just as fast;

(5) If N contains a mixture of factors but is unsuitable for the PFA, the new algorithm is faster than the algorithm described in this paper.

REFERENCES

[1] O. BUNEMAN, *In-situ bit-reversed ordering for Hartley transforms*, IEEE Trans. Acoust. Speech Signal Process., 37 (1989), pp. 577-580.

[2] C. S. BURRUS AND P. W. ESCHENBACHER, *An in-place, in-order prime factor* FFT *algorithm*, IEEE Trans. Acoust. Speech Signal Process., 29 (1981), pp. 806-817.

[3] C. S. BURRUS AND T. W. PARKS, DFT/FFT *and Convolution Algorithms*, John Wiley & Sons, New York, 1985.

[4] J. W. COOLEY AND J. W. TUKEY, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297-301.

[5] W. M. GENTLEMAN AND G. SANDE, *Fast Fourier Transforms—for fun and profit*, in Proc. AFIPS Joint Computer Conference, Spartan, Washington, D.C., 29 (1966), pp. 563-578.

[6] I. J. GOOD, *The interaction algorithm and practical Fourier analysis*, J. Roy. Statist. Soc. Ser. B, 20 (1958), pp. 361-372.

[7] H. W. JOHNSON AND C. S. BURRUS, *An in-place in-order radix-2* FFT, in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing, 1984, pp. 28A.2.1-4.

[8] L. R. RABINER AND B. GOLD, *Theory and Application of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[9] J. H. ROTHWEILER, *Implementation of the in-order prime factor transform for various sizes*, IEEE Trans. Acoust. Speech Signal Process., 30 (1982), pp. 105-107.

[10] R. C. SINGLETON, *On computing the Fast Fourier Transform*, J. Assoc. Comput. Mach., 10 (1967), pp. 647-654.

[11] P. N. SWARZTRAUBER, *Vectorizing the* FFT*'s*, *Parallel Computations*, G. Rodrigue, ed., Academic Press, New York, 1982, pp. 490-501.

[12] C. TEMPERTON, *Fast Fourier Transforms and Poisson-Solvers on Cray-1*, in Supercomputers, Infotech State of the Art Report, C. R. Jesshope and R. W. Hockney, eds., Infotech International Ltd., Maidenhead, U.K., 1979, pp. 359-379.

[13] ———, *Self-sorting mixed-radix Fast Fourier Transforms*, J. Comput. Phys., 52 (1983), pp. 1-23.

[14] ———, *Implementation of a self-sorting in-place prime factor* FFT *algorithm*, J. Comput. Phys., 58 (1985), pp. 283-299.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 824-842, July 1991

PARALLEL ALGORITHMS FOR BANDED LINEAR SYSTEMS*

STEPHEN J. WRIGHT†

Abstract. A partitioned Gaussian elimination algorithm with partial pivoting which is suitable for multiprocessors with small to moderate numbers of processing elements is described. It is only assumed that the system is nonsingular; hence the submatrices in the chosen partitioning may be rank-deficient and this makes the algorithm more complex than those which have been proposed for diagonally dominant and symmetric positive-definite systems. Operation counts and stability are examined. Some numerical results obtained on Alliant FX/8 and Sequent Balance multiprocessors are presented.

Key words. banded matrices, parallel processors

AMS(MOS) subject classifications. 65F05, 65W05

1. Introduction. We discuss parallel algorithms for solving linear systems of the form

$$Ax = b, \quad A \in R^{n \times n}, \quad \text{nonsingular}, \quad A_{ij} = 0 \quad \text{if } |i-j| > k, \tag{1.1}$$

where k is an integer with $k < n$ (usually $k \ll n$). In the usual terminology, k is said to be the half-bandwidth of A; the bandwidth is $2k+1$. Such systems arise in many numerical algorithms, for example, in the numerical solution of partial differential equations and of optimal control problems. Often the system has other characteristics which can be exploited, e.g., it is actually block-tridiagonal, or is known to be symmetric positive definite, in which case more specialized algorithms than the one to be described here may be appropriate.

Much effort has been devoted to solving special cases of (1.1) on parallel machines; see, for example, Johnsson [10]; van der Vorst [15]; Buzbee, Golub, and Nielson [3]; Hockney and Jesshope [9]; Sameh and Kuck [14]; and Berry and Sameh [1] for tridiagonal and block-tridiagonal systems, and Lawrie and Sameh [12] and Dongarra and Johnsson [5] for banded systems. Most of these papers describe algorithms which can fail to produce valid solutions even when the problem is well conditioned and exact arithmetic is used. Two exceptions are the algorithm in [14], and the "double-width separator" algorithm described by Gilbert and Schreiber [6] and Conroy [4], which will be discussed later. Most of the algorithms can be expressed in terms of a "tearing" of the initial matrix, and subsequent application of the Sherman-Morrison-Woodbury formula for inverse updating. In the case of the well-known cyclic reduction method (see Buzbee, Golub, and Nielson [3]), this tearing is applied recursively, and the size of the remaining subsystem is halved at each level. The problem with many of these methods arises from the fact that the tearing strategy corresponds to choosing a predetermined elimination ordering for the variables, which is based on considerations of efficient implementation rather than numerical validity. For many special cases of (1.1) (e.g., those in which A is symmetric positive definite) this presents no difficulty. However, it is our aim here to specify an algorithm which will, at least in principle, produce a valid solution for *all* problems of the form (1.1).

* Received by the editors March 8, 1989; accepted for publication (in revised form) May 9, 1990. This research was supported in part by National Science Foundation grants ASC-8714009, DMS-8619903, and DMS-8900984, and Air Force Office of Scientific Research grant AFOSR-ISSA-87-0092.

† Mathematics Department, Box 8205, North Carolina State University, Raleigh, North Carolina 27695-8205. This research was performed while the author was visiting the Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439.

Our algorithm is based on that of Dongarra and Johnsson [5], in which the matrix is broken up along the diagonal into a number (say, p) of blocks, each separated by a small dense $k \times k$ block. Each main block has the same bandwidth as the original matrix A, and each is factorized using Gaussian elimination without pivoting. A reduced system of dimension approximately $(p-1)k$ is then formed from the rows of A corresponding to the separating blocks. This is solved, and the remaining variables are recovered using data stored during the initial factorization stage.

The main difference in our method is to allow pivoting to occur in the factorization of each of the main blocks. Row pivoting alone is not enough, as these blocks can be rank deficient—they can have nullity as high as $2k$. Considerable complication is therefore introduced in the formation of the reduced system. When singularity is detected in any of the blocks, the dimension and bandwidth of the reduced system is increased. After solution of the reduced system, the remaining variables are recovered as before. Despite its greater complexity, the potential for speedup of the algorithm, when implemented on a multiprocessor, remains high. Most of the computation takes place in the initial factorization of the main diagonal blocks, which is "embarrassingly parallel."

The remainder of the paper is organized as follows. In § 2, the algorithm is described under assumptions of exact arithmetic, and its validity is proved. In § 3, we specify a practical version, perform an error analysis, and estimate the computational requirements. The double-width separator method is discussed briefly in § 4. Implementation of both algorithms on shared-memory multiprocessors at the Argonne Advanced Computing Research Facility is described in § 5.

Throughout the paper, superscripts on vector and matrix quantities are used to denote partition number, while subscripts are used for components. On scalar quantities, a subscript is used for partition number. The notation $\|\cdot\|$ denotes $\|\cdot\|_\infty$ unless otherwise specified.

2. Fundamentals.

2.1. An outline. We start by imposing the same partitioning on the system as that used in Dongarra and Johnsson [5, Fig. 7]. The matrix is broken up diagonally into p main blocks, which are separated along the diagonal by smaller $k \times k$ separator blocks (see Fig. 1). We make the implicit assumption throughout that n is substantially greater than $p^2 k$. This ensures that the size of the reduced system is significantly smaller than the size of the main diagonal blocks which are factorized concurrently, and so efficient multiprocessor implementation of the algorithm will be possible. For convenience, we assume that all main blocks A^ℓ have the same dimension m. The dimension of the entire system is therefore $n = pm + (p-1)k$.

Before describing the factorization of each A^ℓ, we note that these submatrices may be singular according to the following lemma.

LEMMA 2.1. *Under the assumptions in* (1.1), *each* A^ℓ, $\ell = 2, \cdots, p-1$ (*the "internal" partitions*) *has rank at least* $m - 2k$. *Any collection of* t_ℓ *columns taken from columns* $1, \cdots, m-k$ *of* A^ℓ *has rank at least* $t_\ell - k$. *Similarly, any collection of* t_ℓ *rows taken from rows* $1, \cdots, m-k$ *of* A^ℓ *has rank at least* $t_\ell - k$.

Proof. By extracting columns $(\ell-1)m + (\ell-1)k + 1, \cdots, \ell m + (\ell-1)k$ from the matrix A, and stripping off the zero rows, we obtain

$$\begin{bmatrix} T^{\ell 1} & 0 & \cdots & 0 \\ & & A^\ell & \\ 0 & 0 & \cdots & T^{\ell 2} \end{bmatrix}, \tag{2.1}$$

which has dimensions $(m \times 2k) \times m$ and rank m, according to (1.1). When we remove

the first k and last k rows, we are left with A^ℓ, which must therefore have rank at least $m-2k$.

By choosing a selection of t_ℓ columns from columns $1, \cdots, m-k$ of the matrix (2.1), we have

$$\begin{bmatrix} \tilde{T}^{\ell 1} \\ \tilde{A}^\ell \\ 0 \end{bmatrix}, \tag{2.2}$$

where $\tilde{T}^{\ell 1} \in R^{k \times t_\ell}$ consists of columns of $T^{\ell 1}$ and zero columns, while $\tilde{A}^\ell$ consists of t_ℓ columns from A. Again from (1.1), the matrix (2.2) has full column rank t_ℓ, and so by removing $\tilde{T}^{\ell 1}$ we conclude that $\tilde{A}^\ell$ has rank at least $t_\ell - k$. An identical argument is used to prove the last statement of the lemma. □

Similar results apply to the "boundary" partitions A^1 and A^p, which both have rank at least $m-k$. For simplicity we restrict most of the following discussion to the internal partitions.

The factorization of each A^ℓ uses an enhanced row pivoting strategy. Lemma 2.1 implies that there may be stages of the factorization at which no suitable pivot is to be found in the current column. In this case it is necessary to look ahead to subsequent columns. As we show later, this look-ahead, denoted in the algorithm by *look*(i), is at most k columns, except possibly in the final stages of the factorization. We also show that, in the absence of rounding error, the values chosen for *look*(i) below are sufficient to ensure a valid factorization of A^ℓ under the assumptions (1.1). If it is found at stage i of the elimination that columns $i, i+1, \cdots, j-1$ (for some $j > i$) contain no suitable pivots, then a cyclic column permutation is performed: these columns are moved to the end of the matrix, and columns $j, j+1, \cdots, m$ are shifted $j-i$ places to the left.

The factorization algorithm makes use of a logical test *reject*$_i$, whose purpose is to test whether a prospective pivot element is too small to be used. In this section we define it very simply as follows:

$$\textit{reject}_i(\alpha) = \text{true} \Leftrightarrow \alpha = 0.$$

In the next section a more practical definition is supplied.

In the following statement of the algorithm, we use the integer $d(i)$ to denote the number of zero eigenvalues detected *before* stage i of the factorization. The notation *largest*$(\alpha_1, \alpha_2, \cdots)$ denotes the element of largest magnitude in the given argument list $\alpha_1, \alpha_2, \cdots$.

ALGORITHM **factor** (A^ℓ).
Initialize: $d(1) \leftarrow 0$.
for $i = 1, \cdots, m$
  if $i \leqq m-2k$
  then *look*$(i) \leftarrow k - d(i)$
  else *look*$(i) \leftarrow m - d(i) - i$;
  if *look*$(i) < 0$
  then STOP (A^ℓ has rank $i-1$)
  for $j = i, i+1, \cdots, i+$*look*(i)
    (look for pivot element in column j)
    $J \leftarrow \min(m, j+k+d(i))$;
    $A^\ell_{gj} \leftarrow$ *largest*$(A^\ell_{ij}, A^\ell_{i+1,j}, \cdots, A^\ell_{Jj})$;
    if *reject*$_i(A^\ell_{gj})$ **then** go to next j;
    if $j \neq i$
    then reorder the columns $(i, i+1, \cdots, m)$ of A^ℓ

```
        to (j, ···, m, i, i+1, ···, j−1);
        d(i+1) ← d(i)+(j−i);
    if g ≠ i
    then swap rows g and i;
    (now perform the elimination)
    for i_1 = i+1, ···, m
        m^ℓ_{i_1,i} ← A^ℓ_{i_1,i} / A^ℓ_{ii}; A^ℓ_{i_1,i} ← 0;
        for i_2 = i+1, ···, m
            A^ℓ_{i,i_2} ← A^ℓ_{i_1 i_2} − m^ℓ_{i_1,i} A^ℓ_{i,i_2}
        end (for i_2)
    end (for i_1)
    go to next i;
end (for j)
(at this stage, all possible pivots have been rejected)
if i > m − 2k
then STOP (A^ℓ has rank i−1)
end (for i).
```

Before proving that the above algorithm is valid under the assumptions (1.1), we note that the integer J is defined in such a way that the elements $A^{\ell}_{J+1,j}, \cdots, A^{\ell}_{mj}$ are all zero. This is because

- The original matrix A^{ℓ} has lower bandwidth k,
- At stage i, the columns $i+1, \cdots, i+look(i)$ have been shifted a total of $d(i)$ places to the left on previous iterations, thus increasing the lower bandwidth to $k+d(i)$,
- Row pivoting and elimination at earlier stages has not increased the lower bandwidth beyond this.

THEOREM 2.2. *Under the assumptions* (1.1), *and given the partitioning in Fig.* 1, *the algorithm* **factor** *terminates successfully in the absence of rounding error, and produces a factorization of the form*

$$P^{\ell}A^{\ell}\Pi^{\ell} = \begin{bmatrix} L^{\ell} & 0 \\ L^{\ell}_{(2)} & I \end{bmatrix} \begin{bmatrix} U^{\ell} & U^{\ell}_{(2)} \\ 0 & 0 \end{bmatrix}, \tag{2.3}$$

where P^{ℓ} and Π^{ℓ} are permutation matrices, L^{ℓ} and U^{ℓ} have dimensions $m_{\ell} \times m_{\ell}$, m_{ℓ} is the rank of A^{ℓ}, and the other submatrices in the partition are dimensioned accordingly. L^{ℓ} and U^{ℓ} are upper and lower triangular, respectively.

Proof. We first show that for all $i \leqq m-2k$, we have $look(i) \geqq 0$, and a pivot will be found somewhere in columns $i, i+1, \cdots, i+look(i)$ during the inner loop. This is done by contradiction. Suppose for the moment that $look(i) \geqq 0$, and that after $i-1$ stages, $A^{\ell}_{gj} = 0$, for all $g = i, \cdots, m$ and $j = i, \cdots, i+look(i)$, where $look(i) = k - d(i)$. Let $\hat{A}^{\ell}$ be the matrix consisting of the first $i+k$ columns of A^{ℓ}. By Lemma 2.1, $\hat{A}^{\ell}$ has rank at least i. Let L^{ℓ}_{i-1} be the $m \times m$ product of the first $i-1$ Gauss transformations which were applied at previous elimination steps. Since L^{ℓ}_{i-1} is nonsingular, the rank of $L^{\ell}_{i-1}\hat{A}^{\ell}$ will also be at least i. However, each column of the latter matrix has zeros in positions $i, \cdots, m$ because either

- It has been a pivot column at an earlier stage of the factorization,
- It had been pivoted to the end at an earlier stage, say stage $j < i$, because it had zeros in positions $j, \cdots, m$. This property will not have changed on subsequent steps, or
- It falls within the range of the current look-ahead. Columns $i, \cdots, i+k-d(i)$ at stage i were originally in positions $i+d(i), \cdots, i+k$ of the unreduced A^{ℓ}.

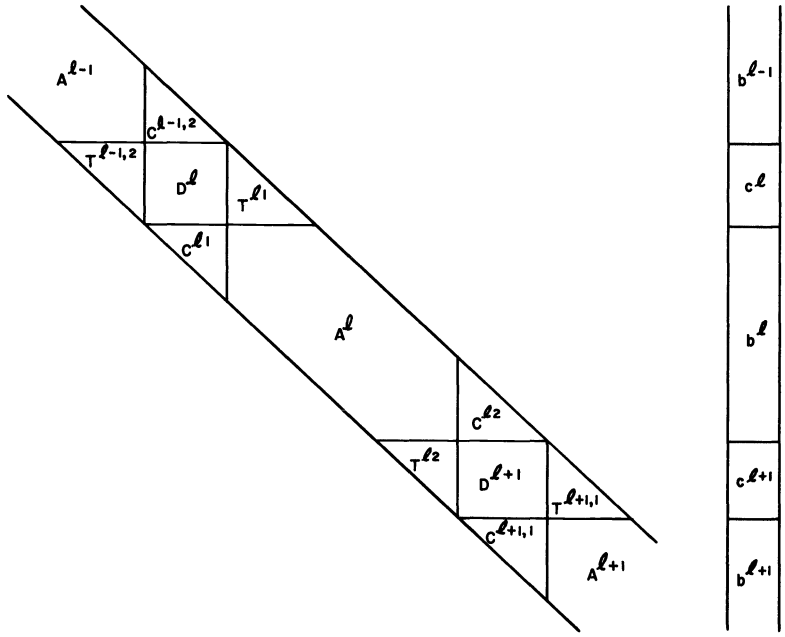


FIG. 1. *Initial partitioning of the matrix A and right-hand side b.*

Hence $L_{i-1}^{\ell}\hat{A}^{\ell}$ has only $i-1$ rows which contain nonzeros, and so has rank at most $i-1$, giving a contradiction.

If $look(i)<0$ for some $i \leqq m-2k$ (i.e., $d(i)>k$ at stage i), we proceed exactly as above in showing that $L_{i-1}^{\ell}\hat{A}^{\ell}$ has only zeros in rows $i, \cdots, m$. The only difference is that the third case no longer applies. A contradiction is obtained again.

It follows from this that **factor** does not terminate until $i > m-2k$. It only remains to show that the triangular factors have the form indicated in (2.3). From the usual properties of Gaussian elimination, it is clear that the factors are indeed triangular. We only need show that the lower right blocks of both factors have the desired structure.

When $m_\ell = m$, no column pivoting is necessary, and the algorithm reduces to standard Gaussian elimination with row partial pivoting. In this case, the conclusion of the theorem is clearly true.

When $m_\ell < m$, one of the two premature exits may be taken. Suppose an exit occurs at stage i. From the first part of the proof, $i > m-2k$ and $look(i) = m-d(i)-i$. Because columns $m-d(i)+1, \cdots, m$ of the remaining matrix (i.e., the U factor) are columns which have been moved to the end at previous stages, they clearly contain zeros in row positions $i, \cdots, m$. If $look(i) \geqq 0$, it is also true (from the failure of the j-loop in **factor**) that columns $i, \cdots, m-d(i)$ also contain only zeros in row positions $i, \cdots, m$. In any case, there is a square zero block of dimension $(m-i+1)$ in the lower right corner of the U factor which, since $m_\ell = i-1$, is the desired form (2.3). For the L factor, the lower right block is the identity since no Gauss transformations are applied during stages $i, \cdots, m$. □

When A^{ℓ} has full rank, it is well known that when Gaussian elimination with row partial pivoting is applied, the factor L^{ℓ} can be stored in a data structure with lower bandwidth k, and U^{ℓ} will have upper bandwidth $2k$. The analogous result when our pivot strategy is used is as follows.

THEOREM 2.3. *Under the assumptions in* (1.1), *the Gauss transformations which make up* L^ℓ *and* $L^\ell_{(2)}$ *in* (2.3) *each contain at most* $2k$ *subdiagonal elements, and* U^ℓ *has upper bandwidth* $2k$.

Proof. From (2.3), we see that L^ℓ, $L^\ell_{(2)}$, and U^ℓ can be considered as arising from applying Gaussian elimination with row partial pivoting to the $m \times m_\ell$ matrix

$$A^\ell \bar{\Pi}^\ell, \tag{2.4}$$

where $\bar{\Pi}^\ell$ consists of the first m_ℓ columns of Π^ℓ. The result for the lower triangular factor will follow if we can show that the lower bandwidth of the matrix (2.4) is at most $2k$. It is clearly sufficient to establish this for the first $m-2k$ columns of the matrix. To see this, observe that columns $1, \cdots, m-2k$ of (2.4) consist of columns of A^ℓ which have been shifted at most $d(m-2k+1)$ places to the left (where d is as defined in **factor**). Since $d(m-2k+1) \leqq k$, the lower bandwidth of these columns cannot have increased to more than $2k$, giving the result for L^ℓ.

We prove the result for U^ℓ inductively by rows, again with reference to (2.4). More specifically, it is shown that after each step i of the elimination, the remaining $(m-i) \times (m_\ell - i)$ submatrix satisfies the following property:

> "For each row of the submatrix, if a nonzero appears in column position j, then all columns after column position $j+2k$ will contain zeros."

This property simply means that there are at most $2k+1$ nonzeros in each row, and that they occur within a band. If the property holds for each of the m_ℓ submatrices which arise during the elimination process, we have the desired result. This is because, possibly after a row swap, the pivot position (i.e., the top left position of the submatrix) contains a nonzero, and so the pivot row has upper bandwidth at most $2k$.

Clearly the property is satisfied by the initial matrix (2.4). Assuming it is true for the submatrix which remains at stage i, we aim to show that it is true for the next step.

After finding a pivot at stage i by looking down the first column of the submatrix, and doing a row swap if necessary, we have already observed that the pivot row has upper bandwidth at most $2k$. For each nonzero in column 1 of the submatrix, we eliminate by adding a multiple of the pivot row. This does not disturb the property. When we form the new submatrix for stage $i+1$ by deleting the first row and column, it is easy to see that the property still holds. Hence the upper bandwidth of U^ℓ is at most $2k$. □

The algorithm **factor** is of course only a part of the overall method for solving (1.1). We now show how it is embedded into the remainder of the outer algorithm. Each major computational step is referred to as a "phase."

Phase 1. Apply **factor** to A^ℓ, $\ell = 1, \cdots, p$, and simultaneously apply the row pivots and forward-elimination steps to the right-hand side b^ℓ and the left and right flanking matrices $C^{\ell 1}$ and $C^{\ell 2}$. In addition, apply the column permutations to the top and bottom flanking matrices $T^{\ell 1}$ and $T^{\ell 2}$, and after each step i of **factor**, eliminate the elements in column i of the top and bottom flanks. This step is explained in more detail below. Figure 2 gives an indication of the possible appearance of a partition at the end of this phase.

Phase 2. Form a reduced system by taking the "separating" rows and columns and the "degenerate" rows and columns (positions $m_\ell+1, \cdots, m$) from each partition. The resulting system has the block-pentadiagonal form shown in Fig. 3. Its dimension is

$$(p-1)k + \sum_{\ell=1}^{p} (m - m_\ell),$$

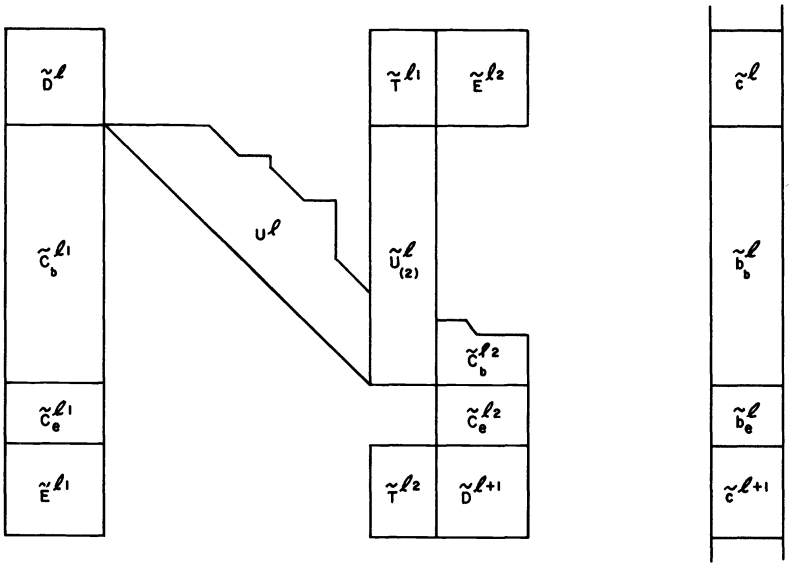


FIG. 2. *Partitioning ℓ after Phase 1 of the algorithm.*

$$\begin{bmatrix} \ddots & \ddots & \ddots & & & & & \\ \tilde{E}^{\ell-1,1} & \tilde{T}^{\ell-1,2} & \tilde{D}^{\ell} & \tilde{T}^{\ell 1} & \tilde{E}^{\ell 2} & & & \\ & & \tilde{C}_e^{\ell 1} & \tilde{S}^{\ell} & \tilde{C}_e^{\ell 2} & & & \\ & & \tilde{E}^{\ell 1} & \tilde{T}^{\ell 2} & \tilde{D}^{\ell+1} & \tilde{T}^{\ell+1,1} & \tilde{E}^{\ell+1,2} & \\ & & & & \tilde{C}_e^{\ell+1,1} & \tilde{S}^{\ell+1} & \tilde{C}_e^{\ell+1,2} & \\ & & & & & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ y_c^{\ell} \\ y_e^{\ell} \\ y_c^{\ell+1} \\ y_e^{\ell+1} \\ y_c^{\ell+2} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \tilde{c}^{\ell} \\ \tilde{b}_e^{\ell} \\ \tilde{c}^{\ell+1} \\ \tilde{b}_e^{\ell+1} \\ \vdots \end{bmatrix}$$

FIG. 3. *Structure of the reduced system.*

while its bandwidth is

$$2k-1+\max_{\ell=1,\cdots,p}(m-m_{\ell}).$$

(In the absence of rank-deficiency in the blocks A^{ℓ}, the $\tilde{C}_e$, $\tilde{S}$, and $\tilde{T}$ blocks in Fig. 3 disappear, leaving a block-tridiagonal reduced system.) Solve the reduced system to obtain the vector

$$(\cdots,(y_c^{\ell})^T,(y_e^{\ell})^T,(y_c^{\ell+1})^T,\cdots)^T,$$

where $y_c^{\ell}\in R^k$ and $y_e^{\ell}\in R^{m-m_{\ell}}$.

Phase 3. Back-substitute to recover the eliminated variables:

$$y_b^{\ell}=(U^{\ell})^{-1}(\tilde{b}_b^{\ell}-\tilde{C}_b^{\ell 1}y_c^{\ell}-\tilde{C}_b^{\ell 2}y_c^{\ell+1}-\tilde{U}_{(2)}^{\ell}y_e^{\ell}). \tag{2.5}$$

Perform column permutations to recover the solution x:

$$x_c^{\ell}=y_c^{\ell}, \qquad x^{\ell}=\Pi^{\ell}\begin{bmatrix} y_b^{\ell} \\ y_e^{\ell} \end{bmatrix}.$$

The obvious way to execute this algorithm in a parallel environment is to assign each partition A^{ℓ} to a processor. Each factorization in Phase 1 and each back-substitution in Phase 3 can then be carried out concurrently. Phase 2, however, is usually carried out on a single processor.

As described in Phase 1, elimination of the elements in the top and bottom flanks is interleaved with the Gaussian elimination process applied to the main blocks. However, we can conceptually view this process as happening *after* the algorithm **factor** has done its work, that is, after U^ℓ has been found, and the left and right flanks and right-hand side have been modified to produce the matrices $\tilde{C}^{\ell 1}$, $\tilde{C}^{\ell 2}$, and $\tilde{b}^\ell$, partitioned as shown in Fig. 2. Elimination of the top flank $T^{\ell 1}$ is achieved by first performing the column pivot sequence used by **factor**:

$$\bar{T}^{\ell 1} = [T^{\ell 1} \quad 0 \quad \cdots \quad 0]\Pi^\ell = [\bar{T}_b^{\ell 1} \quad \bar{T}_e^{\ell 1}]$$

(where $\bar{T}_b^{\ell 1}$ consists of the first m_ℓ columns), and then finding a $k \times m_\ell$ matrix $M_b^{\ell 1}$ such that

$$M_b^{\ell 1} U^\ell + \bar{T}_b^{\ell 1} = 0.$$

The other top-flank matrices in Fig. 2 are then produced by the formulae

$$\begin{aligned} \tilde{D}^\ell &\leftarrow D^\ell + M_b^{\ell 1} \tilde{C}_b^{\ell 1}, \\ \tilde{T}^{\ell 1} &= \bar{T}_e^{\ell 1} + M_b^{\ell 1} U_{(2)}^\ell, \\ \tilde{E}^{\ell 2} &= M_b^{\ell 1} \tilde{C}_b^{\ell 2}, \\ \tilde{c}^\ell &\leftarrow c^\ell + M_b^{\ell 1} \tilde{b}_b^\ell. \end{aligned} \tag{2.6}$$

For the bottom flank, we find another matrix of multipliers $M_b^{\ell 2}$ and obtain $\tilde{E}^{\ell 1}$, $\tilde{T}^{\ell 2}$, $\tilde{D}^{\ell+1}$, and $\tilde{c}^{\ell+1}$ in the same way. Note the $\tilde{D}^\ell$, $\tilde{D}^{\ell+1}$, $\tilde{c}^\ell$, and $\tilde{c}^{\ell+1}$ are also updated by the partitions $\ell - 1$ and $\ell + 1$ during Phase 1.

Phase 1 may also be described in terms of an initial red-black reordering of the matrix, which takes the initial form

$$A = \begin{bmatrix} A^1 & C^{12} & & & & & & & \\ T^{12} & D^1 & T^{21} & & & & & & \\ & C^{21} & A^2 & C^{22} & & & & & \\ & & T^{22} & D^2 & T^{31} & & & & \\ & & & C^{31} & A^3 & C^{32} & & & \\ & & & & \ddots & \ddots & \ddots & & \\ & & & & & & T^{p-1,2} & D^{p-1} & T^{p,1} \\ & & & & & & & C^{p1} & A^p \end{bmatrix}$$

to the reordered form

$$P_0 A \Pi_0 = \left[\begin{array}{ccccc|ccccc} A^1 & & & & & C^{12} & & & & \\ & A^2 & & & & C^{21} & C^{22} & & & \\ & & \ddots & & & & \ddots & \ddots & & \\ & & & A^{p-1} & & & & C^{p-1,1} & C^{p-1,2} & \\ & & & & A^p & & & & C^{p1} & \\ \hline T^{12} & T^{21} & & & & D^1 & & & & \\ & T^{22} & T^{31} & & & & D^2 & & & \\ & & \ddots & \ddots & & & & \ddots & & \\ & & & T^{p-1,2} & T^{p1} & & & & D^{p-1} & \end{array}\right].$$

During Phase 1, additional row and column pivoting is needed during the factorization of each A^ℓ. Also, the blocks created by the "rejected" columns need to be

pivoted out of the first part. Using the notation introduced above, the net result is a factorization

$$P_1 A \Pi = \bar{L}\bar{U},$$

where

$$\bar{L} = \left[\begin{array}{cccc|c} L^1 & & & & \\ & L^2 & & & 0 \\ & & \ddots & & \\ & & & L^p & \\ \hline L^1_{(2)} & & & & I \\ M_b^{12} & M_b^{21} & & & \\ & \ddots & & & \ddots \\ & & M_b^{p-1,2} & M_b^{p1} & \\ & & & L^p_{(2)} & I \end{array}\right],$$

$$\bar{U} = \left[\begin{array}{cccc|ccc} U^1 & & & & U^1_{(2)} & \tilde{C}_b^{12} & \\ & U^2 & & & & \tilde{C}_b^{21} & \\ & & \ddots & & & & \ddots \\ & & & & & & \tilde{C}_b^{p-1,2} \\ & & & U^p & & & \tilde{C}_b^{p1} \quad U^p_{(2)} \\ \hline & 0 & & & & \hat{A} & \end{array}\right].$$

The Schur complement $\hat{A}$, which is the coefficient matrix of the reduced system (Fig. 3), is then factorized using row partial pivoting:

$$P_2 \hat{A} = \hat{L}\hat{U},$$

yielding an overall factorization

$$PA\Pi = \tilde{L}\tilde{U}, \tag{2.7}$$

where

$$P = \begin{bmatrix} I & 0 \\ 0 & P_2 \end{bmatrix} P_1.$$

3. Practical implementation.

3.1. Pivot suitability. Clearly, the test *reject*$_i$ for suitability of a candidate pivot element is insufficient if we want the algorithm to retain numerical stability properties. For instance, during the elimination of a column-i element in a top or bottom flank, a large multiple will be needed if this element is much larger than the pivot being used to eliminate it. This means that the norm of the lower triangular factor in the LU decomposition which we are implicitly forming in Phase 1 will be too large. Formally, let $\bar{T}^{\ell 1,i}$ and $\bar{T}^{\ell 2,i}$ be the $k \times m$ top and bottom flanking matrices, after $i-1$ elimination steps. If a_{ii}^{ℓ} is the prospective pivot for the ith stage of the Phase 1 elimination, *reject*$_i(a_{ii}^{\ell})$ is set to *false* only if the following inequalities hold:

$$\begin{aligned} \left|\frac{\bar{T}_{ji}^{\ell 1,i}}{a_{ii}^{\ell}}\right| &\leqq \mu, \qquad j = 1, \cdots, k, \\ \left|\frac{\bar{T}_{ji}^{\ell 2,i}}{a_{ii}^{\ell}}\right| &\leqq \mu, \qquad j = 1, \cdots, k, \end{aligned} \tag{3.1}$$

where $\mu > 1$ is a chosen constant. If (3.1) fails, then column i is swapped to the end of the matrix as described in **factor**, causing the dimension of the reduced system to increase by one. Because a_{ii}^{ℓ} and the elements below it may not be exactly zero, the block A^{ℓ} is not necessarily rank-deficient, since the lower right block of the U factor in (2.3) need not be zero. However, **factor** will still have the behavior discussed in § 2 provided μ is not chosen so small that more than k columns are rejected during the first $m-2k$ stages of the factorization. As expected, we find in the error analysis that a smaller μ leads to better numerical stability, while the increased rejection rate causes loss of efficiency (see § 5).

3.2. Error analysis. Since the algorithm can be thought of as Gaussian elimination with a certain restricted pivot strategy, we can use the standard backward error analysis framework. The following theorem can be immediately deduced from, for example, Theorem 4.3-2 of Golub and Van Loan [7].

THEOREM 3.1. *Let $\tilde{L}\tilde{U}$ be the factorization of $PA\Pi$ computed by Gaussian elimination, where P and Π are arbitrary permutation matrices. Then the solution $\hat{x}$ of the system $Ax = b$ which is computed using this factorization is the exact solution of a perturbed system $(A+E)\hat{x} = b$, where*

$$\|E\|_\infty \leqq \phi_1(n)[\|A\|_\infty + \|\tilde{L}\|_\infty \|\tilde{U}\|_\infty]u, \tag{3.2}$$

where $\phi_1(n) = O(n)$ and u is unit roundoff.

Since our aim is to bound the second term in (3.2) in terms of $\|A\|_\infty$, we need to examine element growth during the factorization. Phases 1 and 2 are considered separately.

Referring to the notation in (2.7), a bound for $\|\tilde{L}\|_\infty$ can be deduced by noting that the elements of the L^{ℓ}, $L_{(2)}^{\ell}$, and $\hat{L}$ blocks are all bounded in magnitude by one, while from (3.1), those in $M_b^{\ell 1}$ and $M_b^{\ell 2}$ are bounded by μ. Since each $M_b^{\ell 2}$ has at most k nonzero columns, we deduce that

$$\|\hat{L}\|_\infty \leqq (m+k)\mu + m_r, \tag{3.3}$$

where m_r is the order of the reduced system.

Growth in the $\bar{U}$ factor during the first phase can, unfortunately, be exponential when $k > 1$. To be more specific, we need to define the order-k Fibonacci sequence: For $k \geqq 1$, define

$$F(i, k) = 2^{i-1} \quad \text{for } i = 1, \cdots, k,$$

$$F(i, k) = \sum_{j=1}^{k} F(i-j, k) \quad \text{for } i > k.$$

We also observe that for any $m \geqq k > 1$,

$$\sum_{i=1}^{m} F(i, k) \leqq \left\lceil \frac{\log km}{\log k} \right\rceil F(m, k). \tag{3.4}$$

(Here, $\lceil z \rceil$ represents the smallest integer which is larger than z.) This can be proved by first extending the definition above to set $F(i, k) = 0$ for $i \leqq 0$. Then for any integer i,

$$\sum_{j=1}^{k} F(i-j, k) \leqq F(i, k).$$

Choosing t so that $k^{t-1} < m \leqq k^t$, we find, by successively grouping together sets of k consecutive terms, that

$$\begin{aligned}\sum_{i=1}^{m} F(i,k) &= \sum_{j=0}^{k^t} F(m-j,k) \\ &= F(m,k) + \sum_{j=0}^{k^{t-1}} F(m-jk,k) \\ &\leqq 2F(m,k) + \sum_{j=0}^{k^{t-2}} F(m-jk^2,k) \\ &\vdots \\ &\leqq (t+1)F(m,k).\end{aligned}$$

In the tridiagonal case ($k=1$), we have $F(i,1)=1$ for $i>0$, and hence $\sum_{i=1}^{m} F(i,1) = m$.

First we consider the case in which no column pivoting is needed during the factorizations in Phase 1. The number $F(m,k)$ bounds the element growth in the matrices $\tilde{C}_b^{\ell 1}$ and $\tilde{C}_b^{\ell 2}$, during Phase 1. This bound is sharp—the worst case occurs when initially $C_{i,k}^{\ell 1} = 1$ for $i = 1, \cdots, k$, and all multipliers on subsequent elimination steps are one. Element growth can also occur in the blocks which make up the reduced system, because of the elimination of top and bottom flanks (see (2.6)). At each elimination step, a multiple of up to μ of row i of $\tilde{C}_b^{\ell 1}$ and $\tilde{C}_b^{\ell 2}$ is added to D^{ℓ} and $D^{\ell+1}$, respectively. This yields an upper bound on growth in these blocks of

$$1 + 2\mu \sum_{i=1}^{m} F(i,k) \leqq 1 + 2\mu \left\lceil \frac{\log km}{\log k} \right\rceil F(m,k). \tag{3.5}$$

Growth in the blocks $\tilde{E}^{\ell 1}$ and $\tilde{E}^{\ell 2}$ is also bounded by this quantity. Growth in the upper triangular factors U^{ℓ} themselves is not as significant. Bohte [2, p. 140] shows that when Gaussian elimination with partial pivoting is applied to a matrix with half-bandwidth k, the growth factor is bounded by

$$R_k \overset{\text{def}}{=} 2^{2k-1} - (k-1)2^{k-2}. \tag{3.6}$$

Since we assume $m \gg k$, this number is clearly bounded by $F(m,k)$.

In the no-column-pivot case, the matrix for the reduced system is block-tridiagonal with $k \times k$ blocks. Since the bandwidth is $2k-1$, the result (3.6) could be used to bound additional element growth during Phase 2. A slightly better bound can be deduced by noting that when row partial pivoting is applied to this matrix, no nonzero can be swapped to a position more than $3k-1$ places above the main diagonal (i.e., $\hat{U}$ has upper bandwidth at most $3k-1$). Since the maximum element size at most doubles whenever elements are updated, and since each element is updated at most $3k-1$ times, the growth during Phase 2 is bounded by 2^{3k-1}. The results can be summarized as follows.

THEOREM 3.2. *If $k>1$, and no column pivoting is performed on any block during Phase 1 of the factorization, then the growth factor g defined by*

$$g \overset{\text{def}}{=} \frac{\max_{i,j=1,\cdots,n} \tilde{U}_{ij}}{\max_{i,j=1,\cdots,n} A_{ij}}$$

is bounded by

$$2^{3k-1}\left[1 + 2\mu \left\lceil \frac{\log km}{\log k} \right\rceil F(m,k)\right]. \tag{3.7}$$

The computed solution $\tilde{x}$ *then satisfies* $(A+E)\tilde{x}=b$, *where*

$$\|E\|_\infty \leqq \phi_2(n, p, k)\mu^2 2^{3k}\left\lceil \frac{\log km}{\log k}\right\rceil F(m, k)\|A\|_\infty u,$$

and $\phi_2(n, p, k)=O(kn^2/p)$.

Proof. Since each row of $\tilde{U}$ has at most $4k$ nonzeros, the ratio $\|\tilde{U}\|_\infty/\|A\|_\infty$ is bounded by $4kg$. The result is then obtained by combining (3.2), (3.3), and (3.7), and using the relation $m \approx n/p$. □

The result can be adapted to the tridiagonal case, by noting the following facts. From (3.6), growth of at most two can occur during factorization of a tridiagonal matrix with row partial pivoting; no growth occurs in left and right flanks, while growth in the elements which make up the reduced system is at most $1+2m\mu$. The bounds

$$\|\tilde{L}\|_\infty \leqq 2(p-1)+(m+1)\mu,$$
$$\|\tilde{U}\|_\infty \leqq 6(1+2m\mu)\|A\|_\infty$$

apply, and so we can state the following result.

THEOREM 3.3. *If a tridiagonal matrix is factorized and no column pivoting is necessary in any block, the computed solution* $\tilde{x}$ *is the exact solution of the perturbed system* $(A+E)\tilde{x}=b$, *where*

$$\|E\|_\infty \leqq \phi_3(n, p)\mu^2\|A\|_\infty u,$$

where $\phi_3(n, p)=O(n^3/p^2)$.

In the case in which column pivoting occurs, the increase in lower bandwidth of the rearranged matrix makes it necessary to consider order-$2k$ Fibonacci bounds. Growth in the left and right flanks, and in the "rejected" columns of A^ℓ is bounded by $F(m, 2k)$. Since the U^ℓ block has upper bandwidth at most $2k$, it can be shown that growth in these matrices is bounded by the smaller quantity 2^{2k}. Phase 1 growth in the blocks which make up the reduced system is bounded by

$$1+2\mu\left\lceil \frac{\log 2km}{\log 2k}\right\rceil F(m, 2k).$$

In Phase 2, the block-banded reduced system has half-bandwidth at most $4k-1$, and if it is factorized using row partial pivoting, the upper triangular factor produced will have half-bandwidth at most $7k-1$. Growth is thus bounded by 2^{7k-1}. The net result can be summarized as follows.

THEOREM 3.4. *If column pivoting is performed in Phase* 1, *then the growth factor* g *is bounded by*

$$2^{7k-1}\left[1+2\mu\left\lceil \frac{\log 2km}{\log 2k}\right\rceil F(m, 2k)\right]. \tag{3.8}$$

The computed solution $\tilde{x}$ *then satisfies* $(A+E)\tilde{x}=b$, *where*

$$\|E\|_\infty \leqq \phi_2(n, p, k)\mu^2 2^{7k}\left\lceil \frac{\log 2km}{\log 2k}\right\rceil F(m, 2k)\|A\|_\infty u,$$

and $\phi_2(n, p, k)=O(kn^2/p)$.

A few observations on these results are in order. First, the presence of the term $F(m, k)$ makes the growth factor exponential in the size of the submatrices. This contrasts with the situation for the one-processor algorithm of choice, namely, Gaussian elimination with row partial pivoting, which Bohte [2] shows yields a growth factor which is exponential in the bandwidth (3.6). It also contrasts with the situation for

general $n \times n$ matrices, for which the growth term can be exponential in n in the worst case. As in that case, we expect that this term is overly pessimistic and that the average-case performance should be much better, an assertion that the numerical results appear to support.

Second, we note the presence of the term μ^2 in the bound on $\|E\|_\infty$. As stated earlier, a larger choice of μ will decrease the need for column pivoting while it does not appear to have too deleterious an effect on the error bound. In fact, the presence of this term is the main difference between the bounds obtained in Theorems 3.2 and 3.4 and the bounds obtained by Conroy [4] for the double-width separator algorithm. (The log term can be discounted, and the remaining point of difference is a term 2^{4k} in his analysis, which compares to a term 2^{3k} in our best case and 2^{7k} in our worst case. Since this is a pessimistic term, and since we are most interested in smaller values of k, it is not particularly significant.)

3.3. Operation counts. Operation counts for banded system solvers are complicated by the use of pivoting and the resulting fill-in. Their usefulness as predictors of run time is limited because loop overheads, conditional tests, and data movements can add significantly to the run time, particularly when k is small. There is an additional complication in our case because the boundary partitions take less time to process than the interior partitions. The total operation count is a function of the number of partitions p, which increases sharply as p increases from one to four (because the savings due to the boundary conditions are shrinking), but only displays a marginal increase for larger values of p.

Below, we count the operations in the parallelizable phases (Phases 1 and 3), and in the serial phase (Phase 2) under certain conditions, and make a comparison with the operation count for one-processor Gaussian elimination with partial pivoting. In the expressions below, each arithmetic operation $(+, -, \times, /)$ counts as one "flop."

The simplest case is the one in which no row or column pivoting is required. Here, approximately $(4k+2)mk$ operations are needed to factor each interior block and apply the Gauss transformations to the left and right flanks. An additional $2mk^2$ may be needed to eliminate the top flank. In Phase 2, the block tridiagonal solve takes about $(\frac{14}{3}k^3+6k^2)(p-1)$ operations when pivoting is not used. Recovery of the eliminated variables in Phase 3 can be accomplished in about $4mk$ operations. In summary:

Partitioned algorithm, no pivoting:
parallel: $(6k+6)mk$ operations per partition
serial: $(\frac{14}{3}k^3+6k^2)(p-1)$ operations.

A corresponding count for one-processor Gaussian elimination yields:

Serial algorithm, no pivoting:
serial: $(2k+5)nk$ operations.

This tends to suggest a factor-of-three increase in total operation count in cases in which not much pivoting is needed, for example, when the matrix is close to being diagonally dominant.

If we consider the case in which row pivoting takes place, but no column pivoting, we find that in the worst case (i.e., the U^ℓ factor filling out to an upper bandwidth of $2k$), the Phase-1 operation count jumps to approximately $(10k+2)mk$ operations, and Phase 3 may require up to about $6mk$ operations. If we also allow row pivoting in the solution of the reduced system, a total of about $(\frac{23}{3}k^3-5k^2)(p-1)$ operations may be

needed. Summarizing:

Partitioned algorithm, row pivoting only:
parallel: $(10k+8)mk$ operations per partition
serial: $(\frac{23}{3}k^3-5k^2)(p-1)$ operations.

Serial algorithm, row pivoting:
serial: $(4k+7)nk$ operations.

When column pivoting is also allowed, the operation count increases substantially, due to both the increased lower bandwidth of the remaining submatrix and the presence of the rejected columns in the last k positions of the submatrix, to which the Gauss transformations should also be applied. In addition, the reduced system increases in size and bandwidth, so in the worst case we obtain:

Partitioned algorithm, row and column pivoting:
parallel: $(20k+12)mk$ operations per partition
serial: $(87k^3-7k^2)(p-1)$.

4. The double-width separator algorithm. An algorithm which doubles the size of the separator in order to more completely decouple the main blocks was apparently first proposed by Gilbert and Schreiber [6] in the context of decomposition of two-dimensional domains for elliptic partial differential equations. Since then it has been further developed and implemented on a hypercube architecture by Oleson and Gilbert [13]. The coefficient matrix A in this case is a discretized differential operator, and hence has additional structure within the band, but their description applies equally well to the general case. An independent derivation and error analysis of the algorithm appears in Conroy [4].

The advantage of using double-width separators is that the rows in the top and bottom flanks can be used as pivot rows during the factorization of each block, without affecting the top and bottom flanks of adjoining partitions. This removes the need for column pivoting. However, the operation count may be increased appreciably, since it is now possible for the left and right flanks to have $2k$ nonzero columns. The reduced system is block tridiagonal, with blocks of size $2k$ and constant dimension $2(p-1)k$ (possibly twice as large as the reduced system produced by the single-width separator algorithm of §§ 2 and 3), though the off-diagonal blocks are only half full. In [13], a parallel algorithm for solving the reduced system is proposed, but it can be unstable for general matrices A.

To be more specific, we give operation counts for the double-width separator algorithm in the case in which no row pivoting is needed either in the factorization of the individual blocks or in the factorization of the reduced system, and in the worst case in which the row pivoting creates maximal fill-in. In the former case, the parallel count is the same as in the no-pivot case in § 3, but the serial count may be slightly larger:

Double-width separator algorithm, no pivoting:
parallel: $(6k+6)mk$ operations per partition
serial: $(\frac{46}{3}k^3-4k^2)(p-1)$ operations.

Double-width separator algorithm, row pivoting:
parallel: $(12k+10)mk$ operations per partition
serial: $(\frac{76}{3}k^3-6k^2)(p-1)$ operations.

These counts indicate that when no column pivoting is needed in the single-width separator, the serial part of the code requires about a third as many operations as for the double-width separator case, while execution time for the parallel parts are similar.

Therefore, if the reduced system is *not* small relative to the size of each partition, the double-width separator algorithm may be slower.

5. Numerical results. The single-width separator algorithm of §§ 2 and 3, and the double-width separator algorithm were implemented on two shared-memory machines at the Advanced Computing Research Facility, at Argonne National Laboratory. They were the Alliant FX/8 (8 processors) and the Sequent Balance (24 processors). On the Alliant, each processor has vector capabilities; these are not much help for our problem, except during the back substitution in Phase 3, which accounts for only a small part of the computation.

The cautionary remarks in § 3.3, regarding the usefulness of operation counts as predictors of runtime, are borne out by our numerical experience. The codes for the partitioned algorithms are not optimal in terms of their flop counts. To achieve this it is necessary to execute a number of comparisons and conditional tests, which usually use more cpu time than they save. Loop overheads are also significant, since for narrow-banded systems many loops are short and execute for only a few iterations.

For similar reasons, block algorithms and BLAS3 primitives are of limited applicability for narrow-banded systems. Any matrix-matrix operations would involve small blocks of dimension k or $2k$. Data locality in these algorithms is also good, as factorization of each block, and the reduced system, takes place in a reasonably sequential fashion. In the parallel implementation, the amount of working storage needed at any point in the execution of Phase 1 is a fairly small multiple of pk^2, and for the examples given here, the Alliant cache can easily meet these needs.

On both machines, Phases 1 and 3 were usually executed in parallel by assigning one block A^ℓ to each processor. Phase 2 (solution of the reduced system) was carried out on a single processor, using the LINPACK routines `dgbfa` and `dgbsl`. On the Alliant, these routines were compiled with the `-Ogv -DAS` options. Full optimization was found to be a hindrance rather than a help, as the results obtained by applying the LINPACK routines to solve the entire system (Table 8) show. Compiler directives were used extensively on the Alliant to "tune" both the single-width and double-width separator codes. Run time was improved by disabling vectorization on many short loops. The value $\mu = 10^4$ was used for the pivot tolerance in the single-width separator code.

The codes were tested on seven problems for which each partition A^ℓ in the single-width separator case was nonsingular, and so no column pivoting was necessary. The dimensions of these problems were as follows:

1. $n = 2{,}000$, $k = 2$,
2. $n = 2{,}000$, $k = 5$,
3. $n = 5{,}000$, $k = 2$,
4. $n = 5{,}000$, $k = 5$,
5. $n = 10{,}000$, $k = 2$,
6. $n = 10{,}000$, $k = 5$,
7. $n = 10{,}000$, $k = 8$.

The nonzero elements of the matrices were chosen from a uniform distribution in the interval $[-1, 1]$, except for the diagonals, which were chosen from the interval $[-.1, .1]$. The solution x was chosen to be the vector $(1, 2, 3, \cdots)^T$.

Results from the Balance on problems 1-4, using both the single-width and double-width separator codes, are given in Tables 1 and 2. In all cases, the number of processors p_n and number of partitions p were chosen to be identical. Unfortunately, core storage limits made it impossible to run the larger problems on this machine. The

TABLE 1
Balance results, single-width separator algorithm, nondegenerate banded problems. Times in seconds.

$p = p_n$	Prob. 1	Prob. 2	Prob. 3	Prob. 4
1	3.13	8.38	7.83	20.8
2	3.38	9.67	8.33	24.3
4	1.80	5.22	4.50	12.9
8	.98	2.88	2.35	6.75
16	.65	1.98	1.37	3.93
Max speedup over $p = 1$	4.8	4.2	5.7	5.3

TABLE 2
Balance results, double-width separator algorithm, nondegenerate banded problems. Times in seconds.

$p = p_n$	Prob. 1	Prob. 2	Prob. 3	Prob. 4
1	3.13	8.38	7.83	20.8
2	3.60	11.52	8.87	29.1
4	1.85	6.00	4.48	14.8
8	1.03	3.45	2.40	7.83
16	.70	2.80	1.40	4.93
Max speedup over $p = 1$	4.5	3.0	5.6	4.2

results follow the predicted behavior from the operation counts in §§ 3.3 and 4 quite closely, except that the multiprocessor counts are somewhat pessimistic for both codes. As noted in those sections, the total fill-in (and hence total operation count) increases sharply as p increases from 1 to 4, as the savings made in processing the first and last partitions diminish. In all cases the total cpu requirements more than double from $p = 1$ (standard Gaussian elimination with row partial pivoting) to $p = 2$, though this is ameliorated by spreading the work over two processors. For higher values of p, near-linear speedup is seen, until the serial bottleneck created by the reduced system results in a decrease in efficiency. For the pentadiagonal systems ($k = 2$) this is less of a problem than for the larger-bandwidth systems, which is not surprising as the size of the reduced system is about pk in the single-width case and $2pk$ in the double-width case. Comparing runtimes for the single- and double-width cases, we see that there are only minor differences for the $k = 2$ problems, while for the $k = 5$ problems the single-width code holds a distinct advantage which tends to grow as p increases. Table 4 gives the timings for the LINPACK routines on a single processor. These are similar to the 1-processor implementation reported in Tables 1 and 2.

Timings obtained by running problems 1, 2, 4, 5, 6, and 7 on the Alliant are reported in Tables 5 and 6. The first line of each table shows results for the serial algorithm (these can be compared with the LINPACK timings in Table 8). The second line shows results for an eight-partition version as run on a single processor, and the third line shows an eight-partition version on eight processors. Comparison of the second and third lines gives an indication of the multiprocessing efficiency; this is typically around 84 percent for the single-width code and between 70 and 81 percent for the double-width code. In each case, single-processor solution of the reduced system prevents higher efficiencies from being attained. Despite the fact that the same code was used for the Phase 1 and Phase 3 routines as on the Balance, the relative

TABLE 3
Balance results, single-width separator algorithm, degenerate banded problems. Times in seconds.

$p = p_n$	Prob. 1a	Prob. 2a	Prob. 4a
1	3.13	8.38	20.8
2	3.38	9.73	24.4
4	1.80	5.20	12.9
8	1.35	3.45	8.07
16	.83	2.18	4.40

TABLE 4
Balance results, LINPACK, 1 *processor, times in seconds.*

	Prob. 1	Prob. 2	Prob. 3	Prob. 4
LINPACK (1 processor)	3.47	8.35	8.67	20.7

TABLE 5
Alliant results, single-width separator algorithm, nondegenerate banded problems. Times in seconds.

p	p_n	Prob. 1	Prob. 2	Prob. 4	Prob. 5	Prob. 6	Prob. 7
1	1	.32	.78	1.93	1.61	3.89	7.14
8	1	.67	1.85	4.56	3.33	9.18	18.14
8	8	.10	.28	.68	.49	1.37	2.64
Max speedup over $p = 1$		3.2	2.7	2.8	3.2	2.8	2.7

TABLE 6
Alliant results, double-width separator algorithm, nondegenerate banded problems. Times in seconds.

p	p_n	Prob. 1	Prob. 2	Prob. 4	Prob. 5	Prob. 6	Prob. 7
1	1	.32	.78	1.93	1.61	3.89	7.14
8	1	.47	1.56	3.89	2.39	7.81	16.42
8	8	.08	.28	.63	.38	1.21	2.59
Max speedup over $p = 1$		4.0	2.7	3.1	4.2	3.2	2.8

TABLE 7
Alliant results, single-width separator algorithm, degenerate banded problems. Times in seconds.

p	p_n	Prob. 1a	Prob. 2a	Prob. 4a
1	1	.32	.77	1.93
8	1	.69	1.90	4.73
8	8	.14	.35	.83

TABLE 8
Alliant results, LINPACK, *times in seconds.*

	Prob. 1	Prob. 2	Prob. 4	Prob. 5	Prob. 6	Prob. 7
LINPACK (-Ogv-DAS)	.56	1.02	2.57	2.81	5.13	6.29
LINPACK (-O-DAS)	.71	1.52	3.83	3.56	7.56	7.98

timings are quite different. A preference is shown for the double-width separator algorithm except in the larger-bandwidth problems, that is, problems 2 and 7. This is probably due to nonnumerical computational overhead of the type discussed above; the double-width code is shorter and less complex than the single-width code. Apparently, arithmetic processing on the Alliant is sufficiently fast that this overhead is more significant than on the Balance.

Tables 3 and 7 show results for test problems in which certain elements of the matrices were chosen to create singular partitions in the single-width algorithm. The dimensions are

1a. $n = 2{,}000$, $k = 2$,
2a. $n = 2{,}000$, $k = 5$,
4a. $n = 5{,}000$, $k = 5$.

Again, using m to denote the dimension of each partition and m_ℓ to denote its rank, each of these problems had $m_\ell = m$, except that $m_2 = m - 2$ when $p = 8$ and $m_3 = m - 1$ when $p = 16$. The rank-deficient partitions take considerably longer to factor than the nonsingular ones, and hence a load-balancing problem is created. Table 3 shows a sharp increase in runtime for the cases $p = 8$ and $p = 16$ over the corresponding entries in Table 1. However, for the larger-bandwidth matrices, the runtimes are still competitive with, and in some cases somewhat better than, the corresponding results for the double-width algorithm in Table 2. On the Alliant (Table 7), it is seen that the eight-partition algorithm is only slightly slower than in the nondegenerate case when run on a single processor. Unfortunately, all of the increase in runtime takes place on a single processor, as can be seen in the timings for the eight-processor version.

In all cases, the relative error in the computed answer was very small (at least 10 digits of accuracy in double-precision), and was apparently independent of p.

6. Conclusions. We have discussed and analysed partitioned Gaussian elimination algorithms for solving banded linear systems on multiprocessors. The results of tests on shared-memory machines with small to moderate numbers of processing elements confirms their usefulness.

For larger numbers of processors, the method as described above is not suitable. To see this, we let the problem size and number of processors p_n increase as in the definition of "scaled speedup" (see Gustafson, Montry, and Benner [8]). When $p = p_n$ is increased, while m (the blocksize) and k (half-bandwidth) are held constant, the size of the reduced system is $O(pk)$, and the cost of solving it on a single processor eventually dominates the cost of factoring each $m \times m$ partition concurrently. (In practice, it is reasonable to expect problems to scale in this way. For example, a user might double the number of gridpoints in a discretization of an optimal control problem if twice as many processors are available. This does not affect the bandwidth but doubles the dimension of the system.) A strategy for handling this decrease in efficiency would be to apply the method *recursively*, that is, to solve the *reduced* system in the same way as the original system, but on a smaller number of processors. (This concept is similar to cyclic reduction.) We do not explore the proposal further here, but simply

note that its difficulties include the possible compounding of the element growth discussed in § 3.2, and the fact that the reduced system has larger bandwidth than the original system.

Acknowledgments. I thank Andrea Hudson for her help with the Sequent Balance implementation, and a referee for some helpful comments.

REFERENCES

[1] M. BERRY AND A. H. SAMEH, *Multiprocessor schemes for solving block-tridiagonal linear systems*, Internat. J. Supercomput. Appl., 2 (1988), pp. 37–57.

[2] Z. BOHTE, *Bounds for rounding errors in the Gaussian elimination for band systems*, J. Inst. Math. Appl., 16 (1975), pp. 133–142.

[3] B. L. BUZBEE, G. H. GOLUB, AND C. W. NIELSON, *On direct methods for solving Poisson's equations*, SIAM J. Numer. Anal., 7 (1970), pp. 627–656.

[4] J. M. CONROY, *Parallel algorithms for the solution of narrow banded systems*, Appl. Numer. Math., 5 (1989), pp. 409–421.

[5] J. J. DONGARRA AND S. L. JOHNSSON, *Solving banded systems on a parallel processor*, Parallel Comput., 5 (1987), pp. 219–246.

[6] J. R. GILBERT AND R. SCHREIBER, *Nested dissection with partial pivoting*, in Proc. Sparse Matrix Symposium, Fairfield Glade, TN, 1982.

[7] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[8] J. L. GUSTAFSON, G. R. MONTRY, AND R. E. BENNER, *Development of parallel methods for a 1024-processor hypercube*, SIAM J. Sci. Statist. Comput., 9 (1984), pp. 609–638.

[9] R. W. HOCKNEY AND C. R. JESSHOPE, *Parallel Computers*, Adam Hilger, Bristol, U.K., 1981.

[10] S. L. JOHNSSON, *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 354–392.

[11] H. B. KELLER, *Accurate difference methods for nonlinear two-point boundary value problems*, SIAM J. Numer. Anal., 11 (1974), pp. 305–320.

[12] D. H. LAWRIE AND A. H. SAMEH, *The computation and communication complexity of a parallel banded system solver*, ACM Trans. Math. Software, 10 (1984), pp. 185–195.

[13] T.-H. OLESON AND J. R. GILBERT, *One-way dissection with pivoting on the hypercube*, submitted for publication, 1989.

[14] A. H. SAMEH AND D. J. KUCK, *On stable parallel linear system solvers*, J. Assoc. Comput. Mach., 25 (1978), pp. 81–91.

[15] H. A. VAN DER VORST, *Large tridiagonal and block tridiagonal linear systems on vector and parallel computers*, Parallel Comput., 5 (1987), pp. 45–54.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 843-858, July 1991
© 1991 Society for Industrial and Applied Mathematics
008

CONDITIONS AT THE DOWNSTREAM BOUNDARY FOR SIMULATIONS OF VISCOUS, INCOMPRESSIBLE FLOW*

THOMAS HAGSTROM†

Abstract. The proper specification of boundary conditions at artificial boundaries for the simulation of time-dependent fluid flows has long been a matter of controversy. In this work, the general theory of asymptotic boundary conditions for dissipative waves, developed in [T. Hagstrom, *Math. Comp.*, submitted], is applied to the design of simple, accurate conditions at a downstream boundary for incompressible flows. For Reynolds numbers far enough below the critical value for linear stability, a scaling is introduced which greatly simplifies the construction of the asymptotic conditions. Numerical experiments with the nonlinear dynamics of vortical disturbances to plane Poiseuille flow are presented which illustrate the accuracy of our approach for low to moderate Reynolds numbers. The consequences of directly applying the scalings to the equations are also considered.

Key words. Navier-Stokes equations, incompressible flow, boundary conditions

AMS(MOS) subject classifications. 76D05, 41A60, 65P05

1. Introduction.

Problems posed on unbounded domains are ubiquitous in theoretical and applied fluid mechanics. Examples include exterior domains, as arise in aerodynamics, and cylindrical domains, as are found in the study of internal flows. Numerical simulations require a finite computational mesh which is typically obtained by introducing an artificial boundary. The proper specification of boundary conditions there has long been a matter of controversy. Their choice is often made on the basis of ad hoc reasoning and their effect on the solution not fully determined.

In [8] Hagstrom develops a general theory of asymptotic boundary conditions for linear, dissipative wave propagation problems. The basic approach is to numerically identify dominant wave groups and construct linear approximations to the dispersion relation which are valid in the neighborhood of the dominant waves. These lead to simple, local boundary operators. Error estimates for the truncated problem are derived which establish convergence as the size of the computational domain is increased. The goal of this work is to apply the general theory to the linearized, incompressible, Navier-Stokes equations and to test it in fully nonlinear simulations. A question of particular importance which we will address is for what range of Reynolds numbers can simple, local conditions be expected to yield accurate results.

In § 2 we outline the construction of the asymptotic expansions and boundary conditions for the linearized Navier-Stokes equations. Simplified approximations to the full theory, valid for moderate Reynolds numbers, are also constructed. These lead to boundary conditions involving fixed constants and a parametric dependence on Reynolds numbers. The simplifications are based on a simple scaling argument. In § 3 we carry out the numerical computation of the asymptotic boundary operators for the special case of linearizations about plane Poiseuille flow. Both the simplified and full theory are considered, the latter to validate the approximations leading to the former.

* Received by the editors December 11, 1989; accepted for publication (in revised form) May 4, 1990. This research was supported in part by the Institute for Computational Mechanics in Propulsion (ICOMP), National Aeronautics and Space Administration, Lewis Research Center, Cleveland, Ohio, and by National Science Foundation grant DMS-8905314.

† Department of Applied Mathematics and Statistics, State University of New York at Stony Brook, Stony Brook, New York 11794. Present address, Department of Mathematics and Statistics, University of New Mexico, Albuquerque, New Mexico 87131.

Numerical experiments with the nonlinear dynamics of vortical disturbances at moderate Reynolds numbers are presented. These verify the accuracy of our approach even for large amplitude perturbations. Comparisons with the solution on an extended domain are given to study the decay of the error with increasing domain size. Finally, in § 4, we develop an alternate approach based on the direct introduction of the scalings used to simplify the asymptotic analysis. This leads to a simple set of boundary conditions previously considered by Naughton [14].

Boundary conditions for similar equations based on the analysis of linearizations about homogeneous base flows have been derived by Halpern [10] and Halpern and Schatzman [11]. Related techniques have been employed by Abarbanel and coworkers [1] for the compressible Navier-Stokes equations, by Fasel [7] for simulations of transition, and by Ache [2] for steady flows. More recently, Danabasoglu, Biringen, and Street [5] have investigated the effect of outflow conditions on the simulation of spatially evolving instabilities.

2. Construction of asymptotic boundary conditions for the incompressible Navier–Stokes equations.

2.1. Asymptotic analysis. We consider the incompressible Navier-Stokes equations linearized about plane parallel flow:

$$\frac{\partial \omega}{\partial t}+U(y)\frac{\partial \omega}{\partial x}-U''(y)\frac{\partial \psi}{\partial x} = \frac{1}{\mathrm{Re}_l}\left(\frac{\partial^2\omega}{\partial x^2}+\frac{\partial^2\omega}{\partial y^2}\right)+\frac{1}{\mathrm{Re}_t}\frac{\partial^2}{\partial y^2}(D(y)\omega)-\frac{1}{\mathrm{Re}_t}D''(y)\frac{\partial^2\psi}{\partial x^2}, \tag{1}$$

$$\omega=\frac{\partial^2\psi}{\partial x^2}+\frac{\partial^2\psi}{\partial y^2}, \tag{2}$$

$$(x, y)\in(0,\infty)\times(y_0, y_1), \tag{3}$$

$$\psi=\frac{\partial\psi}{\partial y}=0, \qquad y=y_0, y_1, \tag{4}$$

$$\psi(0, y, t)=g_0(y, t), \qquad \omega(0, y, t)=g_1(y, t). \tag{5}$$

Here, ψ is the stream function perturbation, ω is the vorticity perturbation, and $U(y)$ is the parallel base flow. For generality we have included eddy viscosity terms consistent with a simple mixing length model. (See, e.g., Stanisic [15].) Re_l is, then, a Reynolds number based on the mean flow and the kinematic viscosity while Re_t is a turbulence Reynolds number. An unbounded domain in y could also be considered.

Equations (1)-(5) are meant to model the "far field" for a more complicated and general class of problems. Our goal is to derive reasonably simple but accurate boundary conditions at a downstream boundary $x=\tau$. The general approach to the construction of asymptotic boundary conditions given in [8] may be directly applied in this case. To begin we consider the eigenvalue problem:

$$s\hat{\omega}+\lambda U(y)\hat{\omega}-\lambda U''(y)\hat{\psi} = \frac{1}{\mathrm{Re}_l}\left(\lambda^2\hat{\omega}+\frac{\partial^2\hat{\omega}}{\partial y^2}\right)+\frac{1}{\mathrm{Re}_t}\frac{\partial}{\partial y^2}(D(y)\hat{\omega})-\frac{\lambda^2}{\mathrm{Re}_t}D''(y)\hat{\psi}, \tag{6}$$

$$\hat{\omega}=\lambda^2\hat{\psi}+\frac{\partial^2\hat{\psi}}{\partial y^2}. \tag{7}$$

The equations above are well known in the theory of hydrodynamic stability and are usually referred to as the spatial Orr-Sommerfeld equations. Techniques for their

numerical and asymptotic analysis have been extensively developed. (See, e.g., Drazin and Reid [6].)

Solutions to the linearized problem may be expressed in terms of pairs,

$$\left(\lambda_l(s), \begin{pmatrix} \hat{\omega}_l(y; s) \\ \hat{\psi}_l(y; s) \end{pmatrix}\right), \tag{8}$$

satisfying $\Re(\lambda_l(s)) < 0$ for $\Re(s)$ sufficiently large. We have

$$\begin{pmatrix} \omega \\ \psi \end{pmatrix} = \sum_l \Phi_l, \tag{9}$$

$$\Phi_l = \int_0^t c_l(p) q_l(x, y, t-p)\, dp, \tag{10}$$

$$q_l(x, y, t) = \frac{1}{2\pi i} \int_C e^{st+\lambda_l(s)x} \begin{pmatrix} \hat{\omega}_l(y; s) \\ \hat{\psi}_l(y; s) \end{pmatrix} ds. \tag{11}$$

Here C is an appropriate inversion contour.

By a standard steepest descent computation we formally obtain

$$q_l(x, y, t) \sim \frac{e^{x(s^*(\gamma)\gamma+\lambda_l(s^*(\gamma)))}}{\sqrt{2\pi\lambda_l''(s^*(\gamma))x}} \begin{pmatrix} \hat{\omega}_l(y; s^*(\gamma)) \\ \hat{\psi}_l(y; s^*(\gamma)) \end{pmatrix}, \tag{12}$$

where

$$\gamma = \frac{t}{x}, \qquad x \gg 1, \tag{13}$$

and s^* satisfies

$$\lambda_l'(s^*) = -\gamma, \qquad \Re(\gamma) \geqq 0, \qquad \Im(\gamma) = 0. \tag{14}$$

Substituting the asymptotic expansion of q_l into (10) yields an approximation to Φ_l. The expansion itself may be interpreted as a description of wave packets moving at their group velocity $1/\gamma$. The formula for Φ_l is then a superposition of wave groups generated at various times.

In contrast with the usual problems in dispersive wave propagation, our generalized wave groups decay (or grow) exponentially as they propagate with a rate:

$$\mu_l(\gamma) = \Re(s^*(\gamma)\gamma + \lambda_l(s^*(\gamma))) \neq 0. \tag{15}$$

We exploit this property to restrict our attention to a small number of dominant wave groups. That is, we expect the signal far downstream to be dominated by wave groups for which μ_l is maximized. Differentiating with respect to γ and using (14), we find a necessary condition for a local maximum:

$$\mu_l'(\gamma) = \Re(s^*(\gamma)) = 0. \tag{16}$$

Let $\tilde{\gamma}_l$ correspond to a global maximum of μ_l. The exact, nonlocal boundary condition satisfied by q_l is given in transform space by

$$\left(\frac{\partial}{\partial x} - \lambda_l(s)\right) \hat{q}_l = 0. \tag{17}$$

We approximate it by replacing λ_l by the linear part of its Taylor series at $s^*(\tilde{\gamma}_l)$. This leads to a local condition:

$$\left(\frac{\partial}{\partial x} - \lambda_l(s^*(\tilde{\gamma}_l)) - \lambda_l'(s^*(\tilde{\gamma}_l))\left(\frac{\partial}{\partial t} - s^*(\tilde{\gamma}_l)\right)\right) \Phi_l = 0. \tag{18}$$

This operator may be directly applied to $\binom{\omega}{\psi}$ if it corresponds to a maximum over all l. More generally, an operator may be constructed as the product of operators taken from a few modes and local maxima.

We summarize below a procedure for constructing the asymptotic boundary conditions:

- Compute local minima of the decay rates $-\mu_l$. That is, compute solutions of (14) and (16).
- Choose from these a small subset $(\tilde{s}_j, l_j)$ including the overall minimum decay rate as well as any other wave groups whose decay rate is close enough to the minimum.
- Use the following as the boundary condition at $x = \tau$:

$$\prod_j \left(\frac{\partial}{\partial x} - \lambda_{l_j}(\tilde{s}_j) - \lambda'_{l_j}(\tilde{s}_j)\left(\frac{\partial}{\partial t} - \tilde{s}_j\right)\right)\binom{\omega}{\psi} \equiv B\binom{\omega}{\psi} = \binom{0}{0}. \tag{19}$$

In [8] an error analysis is developed for this procedure. The main result is that the difference e between the solution on the truncated domain and the true unbounded domain solution satisfies

$$\|e\| = O\left(\frac{e^{\mu_0 \tau}}{\tau}\right), \qquad \tau \to \infty. \tag{20}$$

Here,

$$\mu_0 = \max_{l,\gamma} \mu_l(\gamma). \tag{21}$$

Although this result was obtained under fairly general hypotheses, we have been unable to verify them in this case. The error estimate is, however, consistent with the computational experiments we will present.

In order to directly complete the search for the dominant wave groups, the repeated numerical solution of (6)-(7) in conjunction with a line search procedure is required. We have implemented an algorithm to accomplish this using a spectral discretization of the Orr-Sommerfeld problem [4]. This will be discussed in greater detail in the context of the numerical experiments.

2.2. Simplified approximations. Despite the availability of a number of appropriate techniques, the direct numerical approach to the construction of the asymptotic boundary conditions has certain drawbacks. Among these are the expense of the computation and the need to recompute the boundary condition as the Reynolds number(s) change. Consequently, we have developed an alternate procedure based on the asymptotic analysis of (6)-(7) for $\mathrm{Re}_{t,l}$ large. The assumption underlying this analysis follows.

Assumption 1. The dominant wave groups satisfy

$$\tilde{s} = 0. \tag{22}$$

We note that this condition does *not* necessarily hold for all Reynolds numbers. In particular, near the critical Reynolds number from linear stability theory the dominant wave group corresponds to the critical Tollmien-Schlichting waves which, in general, do not have $s^* = 0$. For the special case of Poiseuille flow, which we have used for numerical experimentation, we find that Assumption 1 holds for a wide range of subcritical values of Re_l.

By itself Assumption 1 allows the construction of the asymptotic boundary conditions using a single solution of the Orr-Sommerfeld equations. Furthermore, the *necessary* condition for its validity, (14), can be directly verified. Further simplification

occurs when we consider an expansion of the solution of (6)-(7) in powers of $1/\mathrm{Re}_t$. (For laminar flows this is replaced by an expansion in $1/\mathrm{Re}_l$.) We take

$$\lambda = \frac{\bar{\lambda}}{\mathrm{Re}_t}, \qquad s = \frac{\bar{s}}{\mathrm{Re}_t}. \tag{23}$$

To leading order the eigenvalue problem becomes

$$\bar{\lambda}\left(U(y)\frac{\partial^2}{\partial y^2} - U''(y)\right)\hat{\psi} = \left(\frac{\partial^2}{\partial y^2}(D(y)+\varepsilon_l)\frac{\partial^2}{\partial y^2} - \bar{s}\frac{\partial^2}{\partial y^2}\right)\hat{\psi}, \tag{24}$$

where we have made the substitution

$$\hat{\omega} = \frac{\partial^2 \hat{\psi}}{\partial y^2}, \tag{25}$$

and

$$\varepsilon_l = \frac{\mathrm{Re}_t}{\mathrm{Re}_l}. \tag{26}$$

We may now proceed to construct the asymptotic boundary conditions based on a numerical analysis of (24) with Assumption 1. We emphasize the advantages of this simplification:

- We need only solve (24) once for $\bar{s} = 0$ and then compute the required derivatives $d\bar{\lambda}/d\bar{s}$.
- The resulting boundary conditions depend algebraically on the Reynolds number.

Unfortunately, (24) with $\bar{s} = 0$ is not self-adjoint, so it is difficult to obtain general results on its solutions. An ideal result would guarantee the existence of solutions with real, positive, group velocity. We note that a real group velocity is guaranteed for real eigenvalues. It is tempting to rewrite the troublesome term $U(\partial^2/\partial y^2)$ as its symmetric part plus a perturbation, but the unbounded "perturbation" is too large for the direct application of the general theory (Kato [13]). For the special case of Poiseuille flow, $y_0 = -1$, $y_1 = 1$, and $U = 1 - y^2$, our numerical solution of (24) indicates that *all* eigenvalues are real and negative and that the group velocities are positive. Again, these conditions may be checked in the general case by a single solution of (24). The computational cost of this procedure is then negligible in comparison with that of the full simulations.

3. Numerical experiments: The dynamics of some vortical disturbances to plane Poiseuille flow. In this section we describe the results of numerical experiments with the asymptotic boundary conditions we have developed. We take our base flow to be laminar channel flow:

$$U(y) = 1 - y^2, \qquad y \in (-1, 1), \tag{27}$$

and consider (1) without the eddy viscosity terms. (That is, $D = 0$ and $\mathrm{Re} = \mathrm{Re}_l$.) We divide our discussion into two parts considering first the eigenvalue problems used to construct the asymptotic boundary conditions and, then, the use of these conditions in the simulation of the full nonlinear equations. A more detailed description of the discretization may be found in [9].

3.1. Numerical analysis of the full and reduced eigenvalue problems. We have developed two separate programs for the computation of the various quantities defining

the boundary conditions. The first solves a discrete approximation to the reduced eigenvalue problem (24) with $\bar{s}=0$ to find λ and $d\lambda/ds$. It is the simplified condition defined by this reduced problem which is used in the simulations. The second solves a discrete (pseudospectral) approximation to the full spatial Orr-Sommerfeld equations, (6)-(7), and also computes the group velocities. The results of the latter are used to validate the use of the reduced equations for moderate Reynolds numbers. In practice, the point of the approximations leading to (24) is to avoid the expense of solving (6)-(7), and we show that this is justified in the case of Poiseuille flow for Re large enough compared to 1 but far enough below $\mathrm{Re}_{\mathrm{crit}} \approx 5775$. For larger values of Re the full eigenvalue problem must be used.

Equation (24) was approximated on a uniform mesh using centered, second-order, finite differences. The eigenvalues and eigenvectors of the resulting generalized matrix eigenvalue problem were computed using the IMSL subroutine EIGZF. To compute $d\bar{\lambda}/d\bar{s} = d\lambda/ds$ we differentiate (24) with respect to $\bar{s}$ to obtain

$$(\bar{\lambda}_l L - M)\frac{d\hat{\psi}_l}{d\bar{s}} = -\frac{d\bar{\lambda}_l}{d\bar{s}} L\hat{\psi}_l - \frac{\partial^2 \hat{\psi}_l}{\partial y^2}. \tag{28}$$

Here L and M are, respectively, the partial differential operators appearing on the left- and right-hand sides of (24) with $\bar{s}=0$. Multiplying this equation by an adjoint eigenfunction and integrating leads to an expression for $d\bar{\lambda}/d\bar{s}$. For the discrete approximation we simply use the original difference approximation to L and D_+D_- for $\partial^2/\partial y^2$. Left eigenvectors $\hat{\Upsilon}$ were computed by inverting the matrix of right eigenvectors and were then used to compute $d\lambda/ds$:

$$\frac{d\lambda}{ds} = -\frac{\sum_i \hat{\Upsilon}_i D_+ D_- \hat{\Psi}_i}{\sum_i \hat{\Upsilon}_i (2\hat{\Psi}_i + (1-y_i^2) D_+ D_- \hat{\Psi}_i)}. \tag{29}$$

All eigenvalues were found to be real and negative. Tabulated are the six smallest eigenvalues along with the corresponding values of $d\lambda/ds$. (See Table 1.) For these we have $N=39$. Their accuracy may be verified by comparing with solutions of the full Orr-Sommerfeld equation for $s=0$ published by Bramley and Dennis [3].

TABLE 1
Dominant eigenvalues.

λ	$\frac{d\lambda}{ds}$
−21.659	−1.999
−28.111	−1.384
−72.694	−1.991
−85.178	−1.430

The first and third eigenvalues correspond to even stream function perturbations while the second and fourth correspond to odd perturbations. It is interesting to note that the group velocities $-(d\lambda/ds)^{-1}$ are nearly equal within each symmetry class. In the experiments which follow a product boundary condition based on the first two modes is employed. This is formally justified if $e^{-28.111\tau/\mathrm{Re}} \gg e^{-72.694\tau/\mathrm{Re}}$. This suggests reasonable accuracy, in the context of the linearized problem, for moderate τ and Re up to about 1000. However, the near equality of the group velocities is likely to extend the range for which the two-mode condition is reasonably accurate.

Although the solutions of the reduced eigenvalue problem described above satisfy *necessary*, local conditions for dominant wave groups, *sufficient* conditions are global and more difficult to check. Indeed, it is well known that Poiseuille flow becomes linearly unstable for $\mathrm{Re} \approx 5775$ [6]. Then the dominant wave groups are expected to grow in space, while those above decay.

In order to further investigate the dominance of the solutions of the reduced problem for subcritical Reynolds numbers, as well as compute the asymptotic boundary conditions for higher Reynolds numbers, approximate solutions of the full spatial Orr-Sommerfeld equation were computed. These employed Chebyshev pseudospectral discretizations of the derivatives in (24) to solve for λ given s. Approximations to $d\lambda/ds$ were also computed following the same ideas used for the simplified eigenproblem. A (coarse) sampling of Reynolds numbers between 100 and 6000 was taken. For each value of Re solutions for $s = i(k-1)/10$, $k = 1, \cdots, 21$ were found. This choice is motivated by (16) and the fact that neutral modes in the classical theory satisfy $|\Im(s)| < 1$ [6]. If the imaginary part of the derivative of any eigenvalue appeared to pass through 0 and the real part of λ was of the same order as the solutions of the reduced problem, the s increments were refined in the appropriate neighborhood. Note that a real $d\lambda/ds$ corresponds to a maximum or minimum of $\Re(\lambda)$ as s varies on the imaginary axis. On the basis of these computations we tentatively conclude that the modes approximated by the reduced eigenvalue problem are dominant for a range of Reynolds numbers including at least the interval $100 \leqq \mathrm{Re} \leqq 4000$.

It is somewhat surprising, and perhaps of some interest in the study of subcritical transition phenomena, that the reduced problem still provides the dominant modes at so high a Reynolds number. For example, for $\mathrm{Re} = 4000$ we find that the least damped mode with $s^* \neq 0$ has a decay rate more than one and a half times that of the least damped mode with $s^* = 0$. For $\mathrm{Re} = 6000$, on the other hand, there is an eigenvalue with positive real part for $\Im(s)$ in an interval including $(.25, .28)$. The dominant wave group is then described by

$$s^* \approx .27i, \tag{30}$$

$$\lambda \approx 9.11 \times 10^{-4} - 1.027i, \tag{31}$$

$$\frac{d\lambda}{ds} \approx -.380. \tag{32}$$

We emphasize that the effectiveness of simple two or three mode conditions for these Reynolds numbers is doubtful even for the linearized problem due to the presence of many modes with comparable decay/growth rates.

3.2. Numerical method. In order to test the proposed boundary conditions, we consider the following problem:

$$\frac{\partial \omega}{\partial t} - \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} + \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} = \frac{1}{\mathrm{Re}}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right), \tag{33}$$

$$\omega = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}, \tag{34}$$

$$(x, y) \in (0, \tau) \times (-1, 1), \tag{35}$$

with boundary and initial conditions

$$\psi(x, y, 0) = -y + \frac{y^3}{3}, \qquad \omega(x, y, 0) = 2y, \tag{36}$$

$$\psi(x,-1,t)=\frac{2}{3},\qquad \psi(x,1,t)=-\frac{2}{3},\qquad \frac{\partial\psi}{\partial y}(x,\pm1,t)=0, \tag{37}$$

$$\psi(0,y,t)=g_0(y,t),\qquad \omega(0,y,t)=g_1(y,t), \tag{38}$$

$$B\psi(\tau,y,t)=B\left(-y+\frac{y^3}{y}\right),\qquad B\omega(\tau,y,t)=B(2y). \tag{39}$$

Here the asymptotic boundary operator B is given by

$$B=\left(\frac{\partial}{\partial x}-\lambda_1-\alpha_1\frac{\partial}{\partial t}\right)\left(\frac{\partial}{\partial x}-\lambda_2-\alpha_2\frac{\partial}{\partial t}\right), \tag{40}$$

$$\lambda_1=-\frac{21.6593}{\mathrm{Re}},\qquad \alpha_1=-1.9990265, \tag{41}$$

$$\lambda_2=-\frac{28.11134}{\mathrm{Re}},\qquad \alpha_2=-1.383905. \tag{42}$$

A uniform mesh,

$$y_i=-1+ih,\quad i=1,\cdots,N,\qquad x_j=jh,\quad j=1,\cdots,M, \tag{43}$$

was introduced and the spatial derivatives replaced by the standard second-order central differencing formulas; D_+D_- for the second derivatives and D_0 for the first. Time differencing was Crank-Nicholson for the viscous terms and Adams-Bashforth for the transport terms so that for interior nodes we have

$$\left(1-\frac{\delta t}{2\,\mathrm{Re}}\tilde{\nabla}^2\right)\omega^{(t+\delta t)}=\left(1+\frac{\delta t}{2\,\mathrm{Re}}\tilde{\nabla}^2\right)\omega^{(t)}-\frac{\delta t}{2}(3T^{(t)}-T^{(t-\delta t)}), \tag{44}$$

$$\omega^{(t+\delta t)}=\tilde{\nabla}^2\psi^{(t+\delta t)}. \tag{45}$$

Here $\tilde{\nabla}^2$ is the discrete Laplacian and $T^{(t)}$ is given by

$$T^{(t)}=-D_{0y}\psi^{(t)}D_{0x}\omega^{(t)}+D_{0x}\psi^{(t)}D_{0y}\omega^{(t)}. \tag{46}$$

For all our simulations we have taken $\delta t/h=.125$. The wall boundary conditions on ψ are also directly discretized and ω at the wall is computed by (45). The asymptotic boundary condition (39) was replaced by a product of discrete boundary operators following Higdon [12]. A typical term,

$$B_l=\left(\frac{\partial}{\partial x}-\lambda-\alpha\frac{\partial}{\partial t}\right), \tag{47}$$

is approximated by

$$\begin{aligned}\tilde{B}_l(v_{(x)}^{(t)})=&\frac{1}{2h}(v_+^+ + v_+^- - v_-^+ - v_-^-)-\frac{\lambda}{4}(v_+^+ + v_+^- + v_-^+ + v_-^-)\\ &-\frac{\alpha}{2\delta t}(v_+^+ + v_-^+ - v_+^- - v_-^-),\end{aligned} \tag{48}$$

where

$$v_\pm^\pm=v_{(x\pm h/2)}^{(t\pm\delta t/2)}. \tag{49}$$

Then $\prod B_l$ becomes $\prod \tilde{B}_l$. In the present case we have the product of two first-order operators. The resulting discrete operator then involves three mesh lines in x and three

time levels. Although we have not considered the implementation of these conditions in conjunction with spectral or pseudospectral discretizations, Higdon's principle of employing products of stable approximations to first-order operators should still apply. To solve the linear system required by the implicit timestepping we employed a direct method based on discrete separation of variables. That is a method similar to fast solvers for elliptic equations.

3.3. Results. We first present results for $\mathrm{Re} = 400$. We take $N = 39$ and consider $\tau = 2, 4, 6$. ($M = 40, 80, 120$.) For purposes of comparison, we also compute a solution with $\tau = 15$, which we will refer to as "exact," and for $\tau = 4$ with Neumann boundary conditions, $\partial\psi/\partial x = \partial\omega/\partial x = 0$. The inflow perturbations are taken to be

$$g_0 = -y + \frac{y^3}{3} + \begin{cases} A\, e^{-6(y^2+(1-2t)^2)}, & 0 \leqq t \leqq 1, \\ 0, & t > 1, \end{cases} \tag{50}$$

$$g_1 = 2y + \begin{cases} 24A\, e^{-6(y^2+(1-2t)^2)}, & 0 \leqq t \leqq 1, \\ 0, & t > 1. \end{cases} \tag{51}$$

Here A is an adjustable amplitude parameter.

Figure 1 depicts vorticity contours at $t = 2.5$ for $\tau = 15$ and $A = \frac{1}{2}$. Note that the flow certainly cannot be considered a small perturbation of Poiseuille flow, so that we cannot expect the linearized analysis leading to the boundary conditions to be valid. We are nonetheless interested in testing the conditions in this nonlinear setting. Figures 2–4 depict the same flow at $t = 6$ as well as simulations with $\tau = 4$ using the asymptotic boundary conditions and Neumann conditions. We see that the exact solution and the one computed with the boundary conditions we propose are virtually indistinguishable, while the flow computed with the Neumann conditions is greatly distorted. Note that the displacement of the vortex causes a large error at the upper wall which extends well back into the computational domain. Figures 5–7 depict the same computations at $t = 7.5$. Although the main disturbance has passed outside the computational domain, the error at the walls in the Neumann computation is still quite evident. Keep in mind that Figs. 1, 2, and 5 show the restriction to the small domain of a solution computed on a large domain.

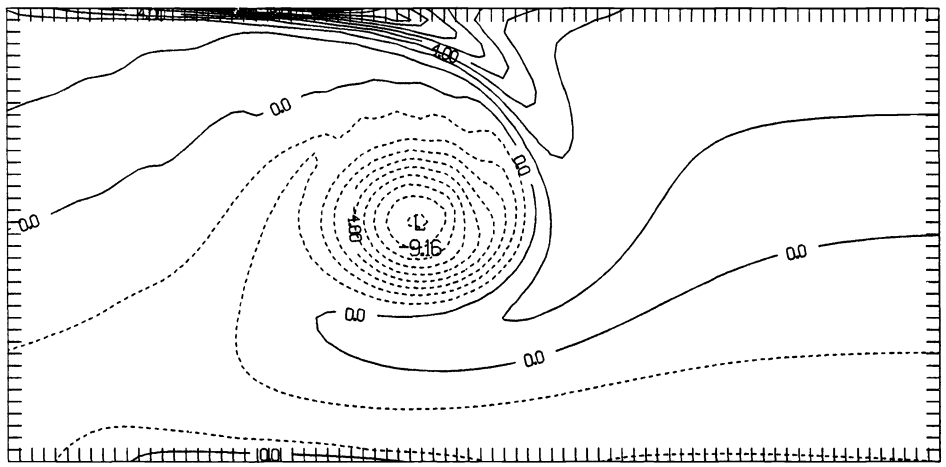


FIG. 1. *Vorticity contours at* $t = 2.5$, $\tau = 15$.

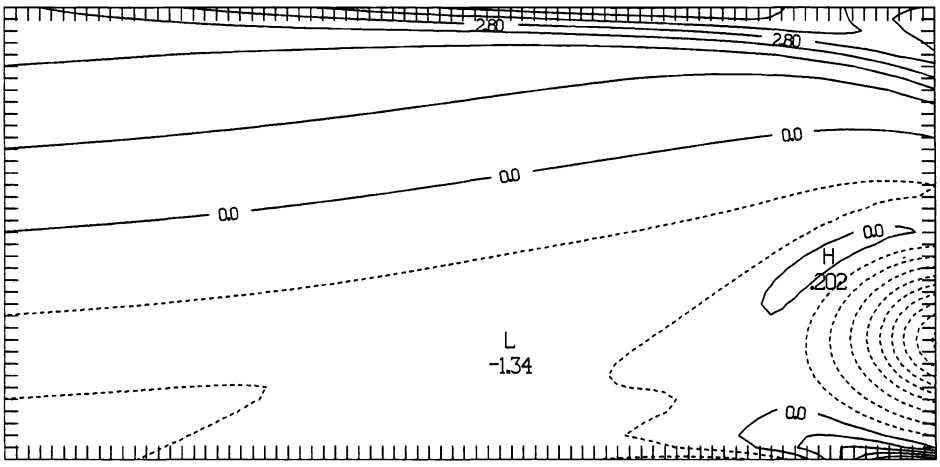


FIG. 2. *Vorticity contours at* $t = 6$, $\tau = 15$.

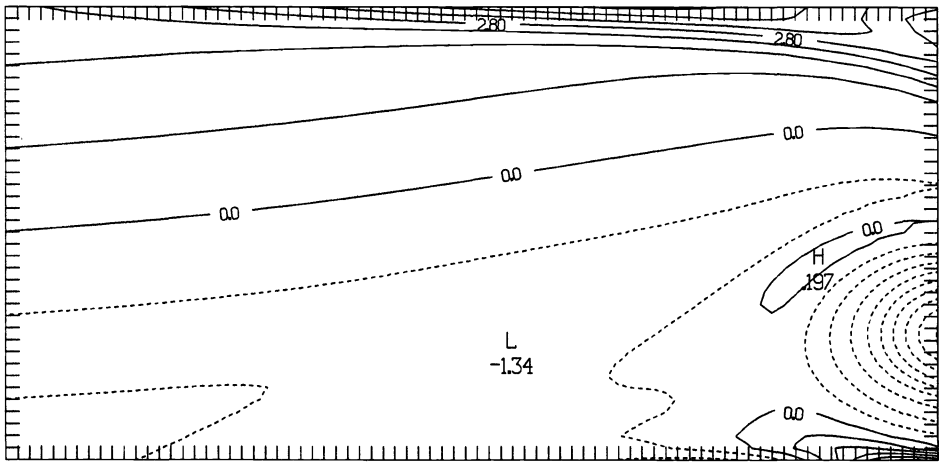
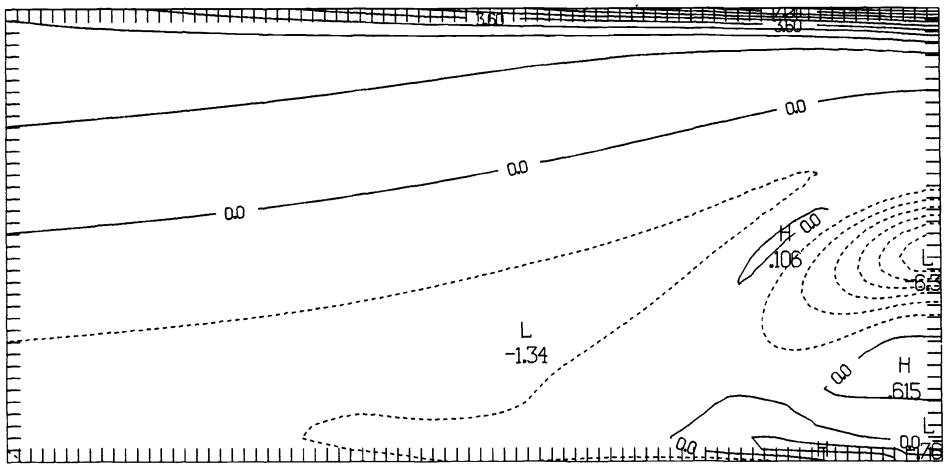


FIG. 3. *Vorticity contours at* $t = 6$, $\tau = 4$, *asymptotic conditions.*

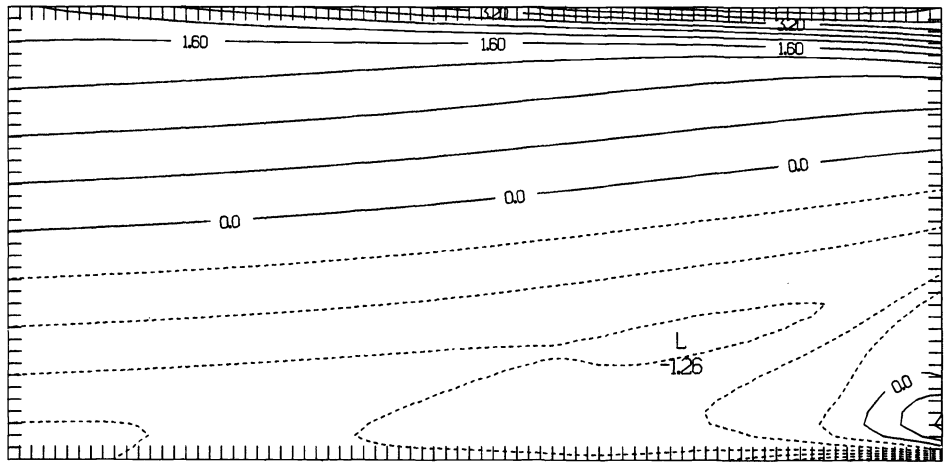
It is also interesting to look at the maximum errors as a function of τ. Here, again, we take the error to be the difference between the given solution on the truncated domain and the solution for $\tau = 15$. The tabulated results clearly show the decay of the error with increasing τ. (See Table 2.) Indeed the theory predicts a decrease in error of approximately $\frac{2}{3}$ as τ increases from 4 to 6, and the results show this precisely. This conclusion can only be tentative, however, as at the end of the simulation the disturbance had not fully passed through the boundary in the case of $\tau = 6$. The error decreases much more rapidly as τ increases from 2 to 4, but this is not surprising as our entire construction is dependent on large τ asymptotics.

Nonlinear effects are demonstrated by comparing results for $A = \frac{1}{2}$ with those for $A = 1/16$. Although the amplitude is scaled by a factor of 8, the errors scale by factors of 15 to 30. Nonetheless, we are quite surprised by the success of our method in the



CONTOUR FROM -5.4000 TO 8.1000 CONTOUR INTERVAL OF 0.90000

FIG. 4. *Vorticity contours at* $t = 6$, $\tau = 4$, *Neumann conditions.*

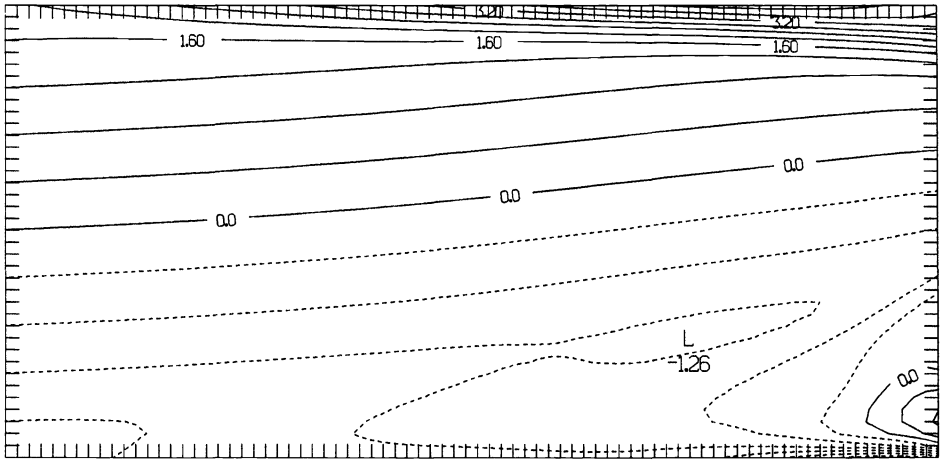


CONTOUR FROM -2.4000 TO 4.0000 CONTOUR INTERVAL OF 0.40000

FIG. 5. *Vorticity contours at* $t = 7.5$, $\tau = 15$.

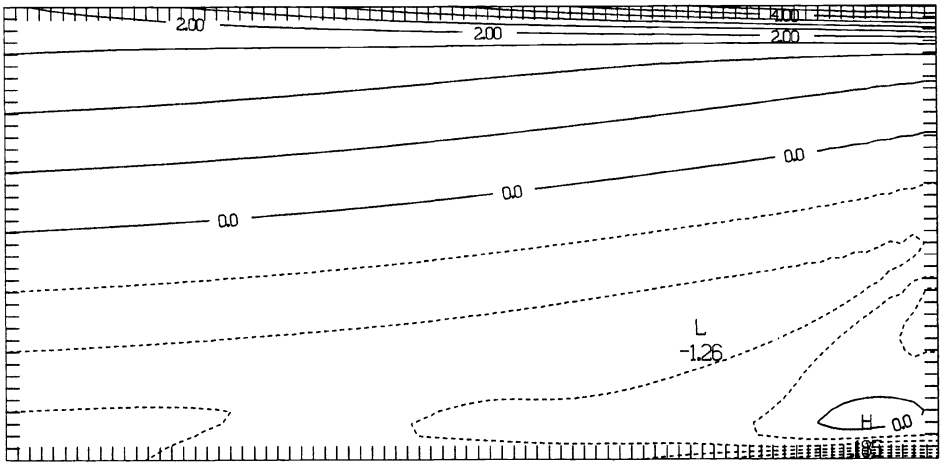
nonlinear regime. It cannot be explained by our linear analysis. It seems a nonlinear analysis of disturbance propagation is required. This is an interesting topic for future research.

A limited number of simulations have also been carried out at $\mathrm{Re} = 2000$ for the same inflow disturbances. It should be noted that even on the basis of linear theory this is beyond the range of Reynolds numbers for which the two mode conditions are expected to be effective. Figures 8 and 9 show vorticity contours at $t = 5.75$ for simulations using, respectively, the asymptotic and Neumann conditions at $\tau = 4$. In this case we have $N = 79$ and $M = 160$. Due to the expense involved, we were unable to carry out a similar simulation on a large domain. Nonetheless, the differences in the vortex dynamics for the two downstream conditions is clear and, apparently, analogous with the results at lower Reynolds number.



CONTOUR FROM -2.4000 TO 4.0000 CONTOUR INTERVAL OF 0.40000

FIG. 6. *Vorticity contours at* $t = 7.5$, $\tau = 4$, *asymptotic conditions.*



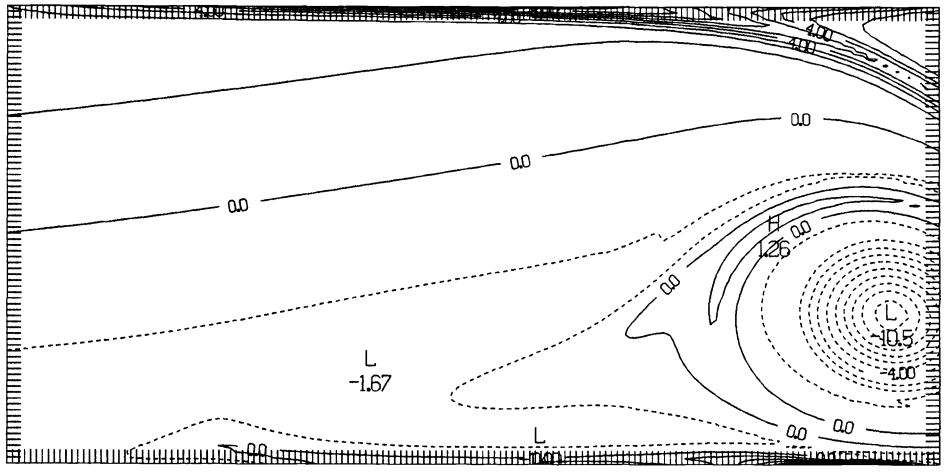
CONTOUR FROM -3.0000 TO 5.0000 CONTOUR INTERVAL OF 0.50000

FIG. 7. *Vorticity contours at* $t = 7.5$, $\tau = 4$, *Neumann conditions.*

TABLE 2

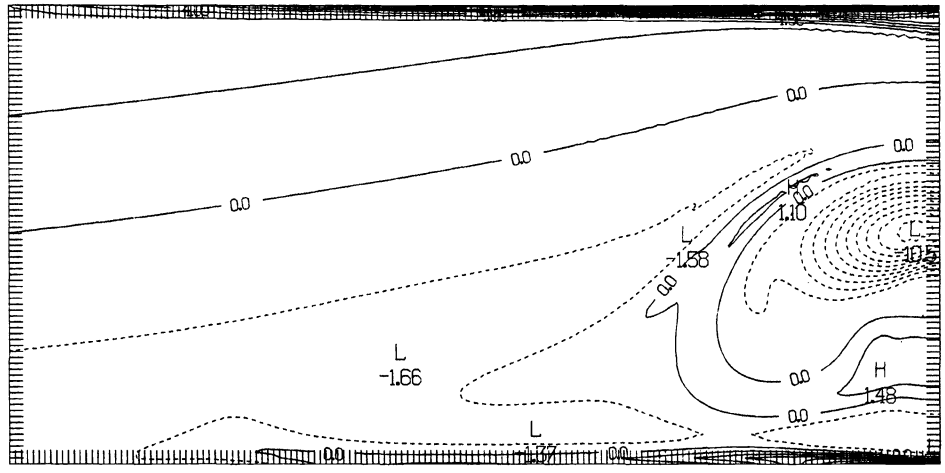
Vorticity errors for $0 \leqq t \leqq 10$. *The maximum vorticity is approximately* 10 *for* $A = \frac{1}{2}$.

A	Boundary condition	τ	Maximum error
$\frac{1}{2}$	Asymptotic	2	1.8
$\frac{1}{2}$	Asymptotic	4	.65
$\frac{1}{2}$	Neumann	4	7.0
$\frac{1}{2}$	Asymptotic	6	.42
$\frac{1}{16}$	Asymptotic	2	.12
$\frac{1}{16}$	Asymptotic	4	.018
$\frac{1}{16}$	Asymptotic	6	.013



CONTOUR FROM -10.000 TO 9.0000 CONTOUR INTERVAL OF 1.0000

FIG. 8. *Vorticity contours at* $t = 5.75$, $\tau = 4$, *asymptotic conditions.*



CONTOUR FROM -10.000 TO 18.000 CONTOUR INTERVAL OF 1.0000

FIG. 9. *Vorticity contours at* $t = 5.75$, $\tau = 4$, *Neumann conditions.*

4. Primitive variable formulation and direct application of the scalings. In this section we discuss the application of the ideas in this paper to the derivation of asymptotic boundary conditions for the primitive variable formulation of the Navier–Stokes equations. We are guided by the theory of boundary conditions for primitive variable formulations which is developed in the doctoral dissertation of Naughton [14]. This theory is based on the construction of energy estimates for the linearized equations. We emphasize that the conditions we will describe have not yet been fully analyzed or implemented.

Linearizing about a parallel flow and assuming, for turbulent flows, a simple eddy viscosity we have

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0, \tag{52}$$

$$\frac{\partial u}{\partial t}+U(y)\frac{\partial u}{\partial x}+U'(y)v=-\frac{\partial p}{\partial y}+\frac{1}{\mathrm{Re}_l}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+\frac{1}{\mathrm{Re}_t}\left(D(y)\frac{\partial^2 u}{\partial x^2}+\frac{\partial}{\partial y}\left(D(y)\frac{\partial u}{\partial y}\right)\right), \tag{53}$$

$$\frac{\partial v}{\partial t}+U(y)\frac{\partial v}{\partial x}=-\frac{\partial p}{\partial y}+\frac{1}{\mathrm{Re}_l}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)+\frac{1}{\mathrm{Re}_t}\left(D(y)\frac{\partial^2 u}{\partial x\,\partial y}+\frac{\partial}{\partial y}\left(D(y)\frac{\partial v}{\partial y}\right)\right). \tag{54}$$

It is not entirely clear from the form of the equations what the correct number of boundary conditions is at an outflow boundary. However, following either the analysis of Halpern and Schatzman [11] or Naughton [14] we seek two conditions. Often in numerical algorithms the somewhat awkward divergence equation is replaced by a Poisson equation for the pressure. Then, an additional boundary condition is needed. To guarantee that the velocity field is divergence free, this condition must in general ensure that the divergence is zero at the boundary. (For outflow boundaries Naughton has shown that this may be relaxed to a Neumann condition on the divergence.)

One possibility is to apply the boundary operator derived in the sections above to two of the perturbed variables. For example,

$$Bu=0, \tag{55}$$

$$Bv=0. \tag{56}$$

An alternate approach, which could also be applied to the stream function—vorticity formulation—is to introduce directly into the equations the scalings used to approximate the spatial Orr-Sommerfeld equation. In terms of the physical variables these are:

$$\frac{\partial}{\partial x}=O\left(\frac{1}{\mathrm{Re}}\right), \tag{57}$$

$$\frac{\partial}{\partial t}=O\left(\frac{1}{\mathrm{Re}}\right). \tag{58}$$

(Here Re is the turbulent or kinematic Reynolds number as appropriate.) The continuity equation (52) may be integrated from a wall boundary to yield

$$v=O\left(\frac{1}{\mathrm{Re}}\right). \tag{59}$$

Differentiating (59) with respect to x yields

$$\frac{\partial v}{\partial x}=O\left(\frac{1}{\mathrm{Re}^2}\right), \tag{60}$$

while substituting it into (54) along with the scalings leads to

$$\frac{\partial p}{\partial y}=O\left(\frac{1}{\mathrm{Re}^2}\right). \tag{61}$$

Dropping the $O(1/\mathrm{Re}^2)$ terms results in equations which could be used as boundary conditions. Unfortunately, the pressure condition is difficult to analyze using energy

methods. To rectify this we add to it another $O(1/\mathrm{Re}^2)$ term and integrate with respect to y:

$$p-\left(\frac{1}{\mathrm{Re}_l}+\frac{1}{\mathrm{Re}_t}D(y)\right)\frac{\partial u}{\partial x}=\Phi(t)+O\left(\frac{1}{\mathrm{Re}^2}\right). \tag{62}$$

This corresponds to the so-called normal constraint of [11]. The function $\Phi(t)$ corresponds to the arbitrary constant which may be added to p. For simplicity we take $\Phi(t)=0$. This leads to the following boundary conditions:

$$\frac{\partial v}{\partial x}=0, \tag{63}$$

$$p-\left(\frac{1}{\mathrm{Re}_l}+\frac{1}{\mathrm{Re}_t}D(y)\right)\frac{\partial u}{\partial x}=0. \tag{64}$$

For solvers which use the Poisson equation for p this must be supplemented by either $(\partial u/\partial x)+(\partial v/\partial y)=0$ or $\partial^2 u/\partial x^2=0$.

Equations (63)-(64) are discussed in [14], where they are called Type 3 conditions. Energy estimates independent of Re are given there for the half-channel laminar flow problem ($D=0$) using these conditions and linearizations about an arbitrary velocity field with outflow. A simple generalization of Naughton's estimates holds in the case of turbulent flow.

It is interesting to compare (63)-(64) with the conditions given in [11]. The equation for v is common to both works. In [11] equation (64) is replaced by a more complicated relationship between p and the velocities which is nonlocal in space. It is worth noting that linear theory suggests that these conditions will be effective for a wider range of Reynolds numbers than the product conditions as they are accurate for all modes satisfying the simplified Orr-Sommerfeld equation.

In the future we plan to carry out numerical experiments which test these conditions as well as to consider the problem of deriving error estimates. Among other interesting questions to be studied are the application of these ideas to viscous, compressible flows, their use in the simulation of interesting unsteady flows, and their generalization to problems in three dimensions. We note that the reduction of domain size allowed by the use of asymptotic boundary conditions is of increasing importance for three-dimensional simulations. In the three-dimensional case the reduction in effort resulting from the use of the approximate eigenvalue problem or the direct use of the scalings would also be most keenly felt.

REFERENCES

[1] S. ABARBANEL, A. BAYLISS, AND L. LUSTMAN, *Non-reflecting boundary conditions for the compressible Navier-Stokes equations*, ICASE Report No. 86-9, ICASE-NASA Langley Research Center, Hampton, VA, 1986.

[2] G. ACHE, *Computational boundary conditions for the incompressible Navier-Stokes equations in channels and pipes*, CMS TSR No. 88-12, Department of Computer Sciences, University of Wisconsin, Madison, WI, 1987.

[3] J. BRAMLEY AND S. DENNIS, *The calculation of eigenvalues for the stationary perturbation of Poiseuille flow*, J. Comput. Phys., 47 (1982), pp. 179-198.

[4] C. CANUTO, M. HUSSAINI, A. QUARTERONI, AND T. ZANG, *Spectral Methods in Fluid Dynamics*, Springer-Verlag, Berlin, New York, 1988.

[5] G. DANABASOGLU, S. BIRINGEN, AND C. L. STREET, *Numerical simulation of spatially evolving instability*, NASA Langley Transition Workshop Proceedings, Springer-Verlag, Berlin, New York, 1989.

[6] P. DRAZIN AND W. REID, *Hydrodynamic Stability*, Cambridge University Press, Cambridge, U.K., 1981.
[7] H. FASEL, *Investigation of the stability of boundary layers by a finite-difference model of the Navier-Stokes equations*, J. Fluid Mech., 78 (1976), pp. 355-383.
[8] T. HAGSTROM, *Asymptotic boundary conditions for dissipative waves: General theory*, Math. Comp., submitted.
[9] ———, *Conditions at the downstream boundary for simulations of viscous incompressible flow*, NASA Tech. Memorandum 102510, ICOMP-90-05, Institute for Computational Mechanics in Propulsion, NASA-Lewis Research Center, Cleveland, OH, 1990.
[10] L. HALPERN, *Artificial boundary conditions for the linear advection diffusion equation*, Math. Comp., 46 (1986), pp. 425-438.
[11] L. HALPERN AND M. SCHATZMAN, *Artificial boundary conditions for viscous incompressible flows*, SIAM J. Math. Anal., 20 (1989), pp. 308-353.
[12] R. HIGDON, *Numerical absorbing boundary conditions for the wave equation*, Math. Comp., 49 (1987), pp. 65-90.
[13] T. KATO, *Perturbation Theory for Linear Operators*, Springer-Verlag, Berlin, New York, 1980.
[14] M. NAUGHTON, *On numerical boundary conditions for the Navier-Stokes equations*, Ph.D. thesis, Department of Applied Mathematics, California Institute of Technology, Pasadena, CA, 1986.
[15] M. STANISIC, *The Mathematical Theory of Turbulence*, Springer-Verlag, Berlin, New York, 1988.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 859-866, July 1991

A BIVARIATE INTERPOLATION ALGORITHM FOR DATA THAT ARE MONOTONE IN ONE VARIABLE*

R. E. CARLSON† AND F. N. FRITSCH†

Abstract. This paper describes an algorithm for interpolating data on a rectangular mesh by piecewise bicubic functions. In [*SIAM J. Numer. Anal.*, 26 (1989), pp. 1-9] the authors presented a monotonicity-preserving algorithm for data that are monotone in both variables. The present paper shows how the algorithm can be modified to treat data that are monotone in only one variable.

Key words. bivariate interpolation, bicubic Hermite surfaces, shape-preserving interpolation

AMS(MOS) subject classifications. 65D05, 65D07

1. Introduction. In [1] Carlson and Fritsch developed a piecewise bicubic Hermite (PBH) interpolation algorithm which produced a bivariate monotone interpolant to monotone data. The original algorithm was improved and simplified in [2]. The algorithm described there requires the data to be monotone in both variables, even if it is only required that the interpolant be monotone in one direction. Carlson and Fritsch also have encountered applications in which the underlying data are monotone in only one variable, and these applications have led to the development of a new PBH interpolation algorithm which is described here.

The basic problem may be defined as follows. Suppose we are given data $f_{i,j} = f(x_i, y_j)$ which are monotone in x on a rectangular grid $x_1 < x_2 < \cdots < x_{nx}$, $y_1 < y_2 < \cdots < y_{ny}$. That is, along each horizontal grid line $y = y_j$,

$$\text{sign}\,(f_{i+1,j} - f_{i,j}) = s_x, \qquad i = 1, 2, \cdots, ny. \tag{1}$$

Our objective is to construct a PBH interpolant $p(x, y)$ with domain $[x_1, x_{nx}] \times [y_1, y_{ny}]$, which is monotone in x.

The function p is uniquely defined by the Hermite coordinates

$$\{p_{i,j}, px_{i,j}, pxy_{i,j}: i = 1, 2, \cdots, nx, j = 1, 2, \cdots, ny\}, \tag{2}$$

where we use the abbreviations $px = \partial p/\partial x$, $py = \partial p/\partial y$, $pxy = \partial^2 p/\partial x\,\partial y$, and $g_{i,j} = g(x_i, y_j)$. Since function values are determined by the interpolation conditions $p_{i,j} = f_{i,j}$, a PBH interpolation algorithm consists of computing the $3 \cdot nx \cdot ny$ unknown Hermite derivatives in (2).

Our development closely parallels that of [2], however complications arise because there are no restrictions on the sign of $py_{i,j}$. This difference, discussed in § 3, requires a change in the sweep algorithms used to modify the initial set of partial derivatives obtained by using one-dimensional methods along grid lines. Section 4 outlines the complete algorithm. An example that demonstrates how the algorithm performs is included in § 5.

2. The system of inequalities. In [2] the following set of inequalities was given for a PBH function $p(x, y)$ to be monotone in x:

$$0 \leqq s_x px_{k,j} \leqq 3 s_x (p_{i+1,j} - p_{i,j})/h_i, \qquad k = i, i+1, \tag{3a}$$

* Received by the editors February 8, 1989; accepted for publication (in revised form) March 1, 1990. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract W-7405-Eng-48.

† Computing and Mathematics Research Division, Lawrence Livermore National Laboratory, Livermore, California 94550.

$$-b_{i,j}/k_j \leqq s_x(py_{i+1,j} - py_{i,j}) \leqq b_{i,j}/k_{j-1}, \tag{3b}$$

$$\begin{Bmatrix} -3px^+_{i,j} \\ 3(C_{i,j} + px^-_{i,j}) \\ 3(C_{i-1,j} + px^-_{i,j}) \end{Bmatrix} \leqq s_x pxy_{i,j} \leqq \begin{Bmatrix} 3px^-_{i,j} \\ 3(D_{i,j} - px^+_{i,j}) \\ 3(D_{i-1,j} - px^+_{i,j}) \end{Bmatrix}, \tag{3c}$$

where

$$h_i = x_{i+1} - x_i, \qquad k_j = y_{j+1} - y_j,$$

$$px^+_{i,j} = s_x px_{i,j}/k_j, \qquad px^-_{i,j} = s_x px_{i,j}/k_{j-1},$$

$$b_{i,j} = 3s_x(p_{i+1,j} - p_{i,j}) \geqq 0,$$

$$C_{i,j} = s_x[(py_{i+1,j} - py_{i,j}) - 3(p_{i+1,j} - p_{i,j})/k_{j-1}]/h_i,$$

$$D_{i,j} = s_x[(py_{i+1,j} - py_{i,j}) + 3(p_{i+1,j} - p_{i,j})/k_j]/h_i.$$

If we force zero to be an acceptable value for $pxy_{i,j}$ in (3c), then we have the following additional restrictions on $(py_{i+1,j} - py_{i,j})$:

$$-B_{i,j}/k_j \leqq s_x(py_{i+1,j} - py_{i,j}) \leqq B_{i,j}/k_{j-1}, \tag{4}$$

where

$$B_{i,j} = 3s_x(p_{i+1,j} - p_{i,j}) - h_i \max\{s_x px_{i+1,j}, s_x px_{i,j}\}.$$

Note that $B_{i,j}$ is nonnegative if (3a) is satisfied, and if (4) is satisfied, so is (3b).

3. Satisfying a special system of inequalities. For a fixed j, inequalities (3b) and (4) are of the form

$$-\text{lhs}_i \leqq d_{i+1} - d_i \leqq \text{rhs}_i, \tag{5}$$

where lhs_i and rhs_i are nonnegative. These inequalities are similar to those in (5.1) of [2], except that in (5) there is no restriction on the signs of the elements of $\mathfrak{G} = \{d_1, d_2, \cdots, d_n\}$. Define a *string* to be a set of consecutive elements from $\mathfrak{G}$. There is a unique, natural decomposition of $\mathfrak{G}$ into positive, negative, and zero strings $\mathfrak{S}_1, \mathfrak{S}_2, \cdots, \mathfrak{S}_m$. That is, each element of the string $\mathfrak{S}_j$ is of the same sign, or each element is zero, and consecutive strings cannot be of the same type. The first element of each nonzero string (except for $\mathfrak{S}_1$) and the last element of each nonzero string (except for $\mathfrak{S}_m$) are said to be *transition* elements.

The basic idea is to modify the original set $\mathfrak{G}$, if necessary, in order to satisfy (5). As in the original algorithm, each modification preserves the sign of d_i, but may decrease its magnitude. The first part of the algorithm involves only the transition elements. The second part involves each nonzero string separately. (Note that (5) is trivially satisfied if $d_{i+1} = d_i = 0$, so zero strings can be left alone.) The following lemma justifies this partitioning.

LEMMA. *Suppose that d_i and d_{i+1} are transition elements, so that* $\text{sgn}(d_i) \neq \text{sgn}(d_{i+1})$, *where*

$$\text{sgn}(x) = \begin{cases} +1 & \text{if } x > 0, \\ 0 & \text{if } x = 0, \\ -1 & \text{if } x < 0, \end{cases}$$

and suppose that (5) *is satisfied. If* δ_i *and* δ_{i+1} *are any numbers such that* $d_k \cdot \delta_k \geqq 0$, $0 \leqq |\delta_k| \leqq |d_k|$, $k = i, i+1$, *then*

$$-\text{lhs}_i \leqq \delta_{i+1} - \delta_i \leqq \text{rhs}_i.$$

Proof. Case 1. $\text{sgn}(d_i) = +1$ or $\text{sgn}(d_{i+1}) = -1$.

$$\delta_i = |\delta_i| \leqq |d_i| = d_i \quad \text{and} \quad -\delta_{i+1} = |\delta_{i+1}| \leqq |d_{i+1}| = -d_{i+1}$$

imply

$$\delta_{i+1} - \delta_i \geqq d_{i+1} - d_i \geqq -\text{lhs}_i.$$

On the other hand, $\delta_i \geqq 0$ and $\delta_{i+1} \leqq 0$ imply

$$\delta_{i+1} - \delta_i \leqq 0 \leqq \text{rhs}_i.$$

The other case [$\text{sgn}(d_i) = -1$ or $\text{sgn}(d_{i+1}) = +1$] follows similarly. □

Part 1. Modify transition elements. Let d_k be a transition element. For illustrative purposes let us assume that d_k is the last element of a positive string (not $\mathfrak{S}_m$). Thus, $d_k > 0$ and d_{k+1} is either a transition element of a negative string, or $d_{k+1} = 0$. Under these assumptions the inequality

$$d_{k+1} - d_k \leqq \text{rhs}_k \tag{6}$$

is *always* satisfied, but

$$-\text{lhs}_k \leqq d_{k+1} - d_k \tag{7}$$

may be violated. If (7) is violated, both d_k and d_{k+1} are multiplied by

$$\rho_k = -\text{lhs}_k/(d_{k+1} - d_k) \in [0, 1); \tag{8}$$

i.e., update the transition elements by

$$d_k := \rho_k \cdot d_k, \qquad d_{k+1} := \rho_k \cdot d_{k+1}. \tag{9}$$

Similarly, if $d_k < 0$, then (7) is always satisfied, but (6) may be violated. If (6) is violated, we again multiply by a number in [0, 1):

$$\rho_k = \text{rhs}_k/(d_{k+1} - d_k). \tag{10}$$

To complete Part 1 of the algorithm, we need to consider the case where d_{k+1} is a transition element of a nonnegative string and $d_k = 0$. This is accomplished by proceeding as if $d_k > 0$ when $d_{k+1} < 0$ or by proceeding as if $d_k < 0$ when $d_{k+1} > 0$.

With these changes to the transition elements, inequalities that involve elements from adjacent strings will be satisfied, even if d_k or d_{k+1} is reduced in magnitude in Part 2. Note that the magnitude of d_k and d_{k+1} have been changed (reduced), but the signs remain the same.

Remark. All of the inequality checks and changes (8)-(10) must be accomplished using the original values from $\mathfrak{G}$. In this way the final values are independent of the order in which the changes were made. This is discussed further in the Appendix.

Part 2. Process nonzero strings. The second part of the algorithm involves upsweeps and downsweeps of each nonzero string of length greater than one. (A string of length one has already been completely processed in Part 1.) This requires a modification of Algorithm (5.4) described in [2] to allow negative d-values.

If $\mathfrak{S}_j = \{d_\beta, d_{\beta+1}, \cdots, d_{\beta+\lambda-1}\}$ is a positive string of length $\lambda > 1$, then the inequalities

$$d_{k+1} - d_k \leqq \text{rhs}_k, \qquad k = \beta, \beta+1, \cdots, \beta+\lambda-2 \tag{11}$$

are checked. If (11) is violated, then d_{k+1} is "too large," and a new value is obtained using the formula

$$d_{k+1} := d_k + \text{rhs}_k. \tag{12}$$

This type of checking is called an *upsweep*. The inequalities

$$-\text{lhs}_k \leqq d_{k+1} - d_k, \qquad k = \beta+\lambda-2, \cdots, \beta+1, \beta \tag{13}$$

are checked on the *downsweep*, with corrections

$$d_k := d_{k+1} + \text{lhs}_k. \tag{14}$$

For a negative string, (13) is checked on the upsweep, with correction

$$d_{k+1} := d_k - \text{lhs}_k, \tag{15}$$

and (11) is checked on the downsweep, with correction

$$d_k := d_{k+1} - \text{rhs}_k. \tag{16}$$

As shown in [2], the resulting set of values $\mathfrak{S}_j'$ satisfies (5) and is independent of the order of the sweeps. Because of our lemma, if the transition elements satisfied (5) at the beginning of Part 2, they will satisfy (5) after application of the above two-sweep algorithm to each nonzero string.

Remark. The algorithm described in this section requires three temporary arrays of length $m-1$ for storage of the string decompositions. In the worst case, m can be as large as nx.

4. The algorithm. As in [1] and [2], the algorithm for determining a reasonable set of Hermite derivatives that satisfy (3a), (3b), (3c), and (4) involves several steps.

Step 1. Verify that the data are monotone in x and compute s_x.

Step 2. Compute initial values for $px_{i,j}$ and $py_{i,j}$ such that the $px_{i,j}$-values satisfy (3a).

Step 3. Construct an interval of acceptable values (containing zero) for $pxy_{i,j}$, changing py-values as necessary to satisfy (4). Note: Since (4) implies (3b) and the bounds do not involve py, it is not necessary to introduce a separate step for (3b) as was done in [2].

Step 4. Compute values for $pxy_{i,j}$ that satisfy (3c).

Step 2 can be accomplished by using subroutine PCHIC from the PCHIP software package [3]. The values of $px_{i,j}$ computed this way satisfy (3a) if the data are monotone in x. Step 3 can be done using the algorithm described in § 3, where for a fixed j, $d_i = s_x \cdot py_{i,j}$ and the bounds lhs_i and rhs_i can be read from (4). To compute $pxy_{i,j}$, as in [2], use the standard three-point difference approximation applied to $p_x(x_i, y)$ and $p_y(x, y_j)$. If the average of these two values does not satisfy (3c), use the closer endpoint value.

Remark. A similar algorithm for data that are monotone only in y can be obtained by applying the algorithm of § 3 to the y-inequalities of [2].

5. Example. To illustrate this algorithm, we have modified the transpose of the 4×4 example in Appendix B of [2] to be slightly nonmonotonic in y, so that BIMOND3 would refuse to deal with it. The results from the algorithm described here are given in Table 1.

TABLE 1
A sample problem that is monotone only in x.

Data				*py* before	*py* after	
x	*y*	*f*	*px*	Step 3	Step 3	*pxy*
1	1	0	4.498	0	0	−4.24875
2	1	2.999	1.999333 (−3)	0	0	2.587536
3	1	3	1.999600 (−3)	3.5	3.5	1.712536
4	1	8	7.4995	0	0	−9.124875
1	2	2	0	3.599960	1.287714	0
2	2	3	1.714286	1.999882 (−3)	1.999882 (−3)	0
3	2	9	1.714286	7.764706	4.619103	0
4	2	10	0	3.333389	3.333389	0
1	3	19.998	1.0 (−3)	1.999889 (−3)	1.999889 (−3)	2.500111 (−4)
2	3	19.999	1.0 (−3)	1.999882 (−3)	1.999882 (−3)	−3.0 (−3)
3	3	20	1.0 (−3)	1.999818 (−3)	1.999818 (−3)	−3.0 (−3)
4	3	20.001	1.0 (−3)	1.999880 (−3)	1.999800 (−3)	2.500024 (−4)
1	4	19.997	1.0 (−3)	−3.0 (−3)	−3.0 (−3)	−2.5 (−4)
2	4	19.998	1.0 (−3)	−3.0 (−3)	−3.0 (−3)	3.0 (−3)
3	4	19.999	1.0 (−3)	−3.0 (−3)	−3.0 (−3)	3.0 (−3)
4	4	20	1.0 (−3)	−3.0 (−3)	−3.0 (−3)	−2.5 (−4)

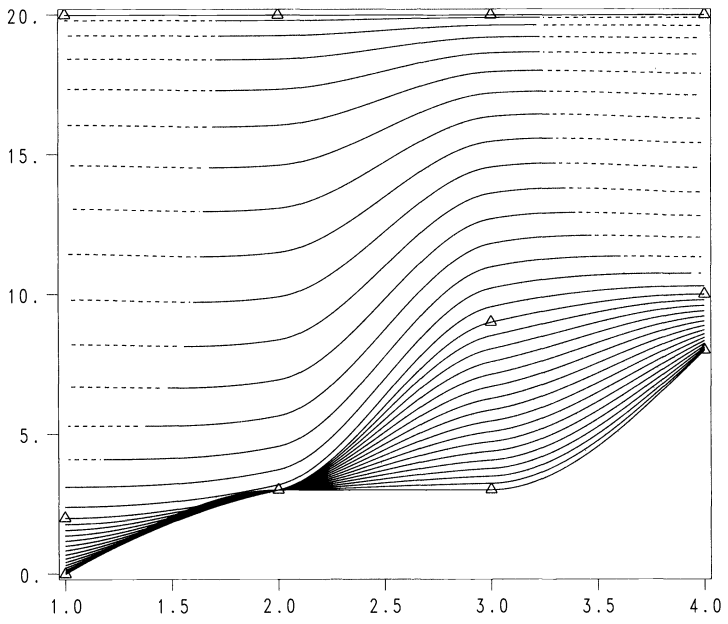


FIG. 1. *Sections y = constant of the surface resulting from omitting Step 3. Dashed lines are segments on which* $\partial p/\partial x < 0$.

Several cross sections in the x-direction of the surface obtained by omitting Step 3 are shown in Fig. 1. It is nonmonotonic between $y = 2$ and $y = 3$. Corresponding sections of the result from the full algorithm are given in Fig. 2. In both figures the data points are marked by triangles.

Appendix. Implementation of the transition element processor.

The naïve algorithm. While the transition element processor described in Part 1 of § 3 is correct, it is dependent on the order of processing. (There is no inherent

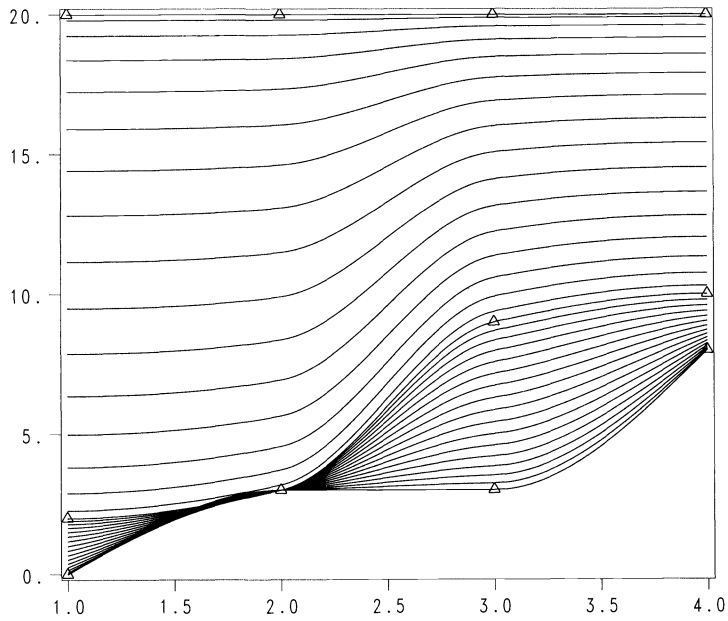


FIG. 2. *Sections of the surface from the full algorithm, showing that the desired monotonicity in the x-direction has been achieved.*

reason to run k from 1 to $m-1$ rather than running the index in the reverse order.) The following totally artificial problem with $n=8$ illustrates the situation:

$$\mathfrak{G} = \{-2, -2, -2, 2, 2, -1, 1, 1\},$$

$$\textbf{lhs} = \textbf{rhs} = \{3, 3, 3, 2, 2, 1, 1\}.$$

Clearly the string decomposition of $\mathfrak{G}$ consists of $m=4$ strings. If β_j is the index of the start of string $\mathfrak{S}_j$, λ_j its length, and s_j its sign, the string decomposition of $\mathfrak{G}$ is given by

$$\beta = \{1, 4, 6, 7\}, \qquad \lambda = \{3, 2, 1, 2\}, \qquad \mathbf{s} = \{-1, +1, -1, +1\}.$$

Let us call the algorithm as described in § 3 the "naïve algorithm." If applied to $\mathfrak{G}$ in a left-to-right sweep, the result is

$$\mathfrak{G}^+ = \{-2, -2, -3/2, 3/2, 4/3, -2/5, 3/5, 1\}.$$

On the other hand, if the naïve algorithm is applied in a right-to-left sweep, the result is

$$\mathfrak{G}^- = \{-2, -2, -3/2, 3/2, 8/5, -2/5, 1/2, 1\},$$

and we note that the elements with indices 5 and 7 are different. (Both sequences satisfy the inequalities.)

A symmetric algorithm. The lack of symmetry in the naïve algorithm can be eliminated by using **Algorithm A1**.

ALGORITHM A1.
for $j := 1$ **step** 1 **until** $m-1$ **do begin**
    $k := \beta_{j+1} - 1$;
    $\Delta_j := d_{k+1} - d_k$
end;

for $j := 1$ **step** 1 **until** $m-1$ **do begin**
    $k := \beta_{j+1} - 1$;
    $\rho := 1$;
    if ($s_j > 0$ or $s_{j+1} < 0$) **then**
        if ($\Delta_j < -\text{lhs}_k$) $\rho := -\text{lhs}_k / \Delta_j$
    else
        if ($\Delta_j > \text{rhs}_k$) $\rho := \text{rhs}_k / \Delta_j$
    endif;
    $d_k := \rho \cdot d_k$;
    $d_{k+1} := \rho \cdot d_{k+1}$
end;

Note that we have precomputed the d-differences across the transitions with the original values and used these for the tests and ρ-computations. At the end of the main **if-else-endif** clause, $0 \leqq \rho \leqq 1$, with $\rho = 1$ if no change is required.

When applied to our example (in either direction), Algorithm A1 produces $\Delta = \{4, -3, 2\}$ and

$$\mathfrak{G}^1 = \{-2, -2, -3/2, 3/2, 4/3, -1/3, 1/2, 1\}.$$

The validity of this algorithm can be established as a corollary to the proof of our lemma. If an inequality is violated with the new (smaller) d-values, then it surely was also violated with the original ones. The result is that we are possibly making greater changes to some d-values in exchange for the desired symmetry property.

If we carefully study how the original lack of symmetry arose, it becomes clear that it is due to the possibility of a d-value being changed more than once. A short analysis reveals that d_k can be changed more than once only if it is the sole element of a string of length one. Algorithm A1 is symmetric because $\rho_{k+1}(\rho_k \cdot d_k) = \rho_k(\rho_{k+1} \cdot d_k)$, and the same ρ's are computed in either case.

A better algorithm. Making explicit checks for strings of length one results in a slightly more complicated algorithm that makes smaller changes to the original d-values. The basic idea is to change each d-value at most once. Instead of changing the element of a string of length one twice, a single change is made that satisfies both inequalities in which the element is involved. The result is **Algorithm A2.**

ALGORITHM A2.
for $j := 1$ **step** 1 **until** $m-1$ **do begin**
    $k := \beta_{j+1} - 1$;
    $\Delta := d_{k+1} - d_k$;
    $\rho_j := 1$;
    if ($s_j > 0$ or $s_{j+1} < 0$) **then**
        if ($\Delta < -\text{lhs}_k$) $\rho_j := -\text{lhs}_k / \Delta$
    else
        if ($\Delta > \text{rhs}_k$) $\rho_j := \text{rhs}_k / \Delta$
    endif
end;
for $j := 1$ **step** 1 **until** $m-1$ **do begin**
    $k := \beta_{j+1} - 1$;
    if ($j > 1$ and $\lambda_j = 1$) **then**
        $\rho := 1$
    else
        $\rho := \rho_j$

```
        endif
        d_k := ρ · d_k ;
        if (j+1 < m and λ_{j+1} = 1) then
            ρ := min {ρ_j, ρ_{j+1}}
        else
            ρ := ρ_j
        endif;
        d_{k+1} := ρ · d_{k+1}
    end;
```

At the end of the first phase we have $0 \leqq \rho_j \leqq 1$, $j = 1, \cdots, m-1$, with $\rho_j = 1$ if no change is required. In the second phase, a different ρ-value may be used to change d_{k+1} from that used for d_k.

When applied to our example (in either direction), Algorithm A2 produces $\rho = \{3/4, 2/3, 1/2\}$ and

$$\mathfrak{G}^2 = \{-2, -2, -3/2, 3/2, 4/3, -1/2, 1/2, 1\}.$$

Note that now $d_6 = (1/2) \cdot (-1) = -1/2$, rather than $(2/3) \cdot (1/2) \cdot (-1) = -1/3$. Algorithm A2 has been used in the version of BIMOND that produced the results in § 5.

Remark. The two phases of Algorithm A2 can be combined in such a way that only four (rather than $m+1$) real variables are required for the storage of intermediate results.

REFERENCES

[1] R. E. CARLSON AND F. N. FRITSCH, *Monotone piecewise bicubic interpolation*, SIAM J. Numer. Anal., 22 (1985), pp. 386–400.

[2] ———, *An algorithm for monotone piecewise bicubic interpolation*, SIAM J. Numer. Anal., 26 (1989), pp. 1–9.

[3] F. N. FRITSCH, PCHIP *final specifications*, Computer Documentation Report UCID-30194, Lawrence Livermore National Laboratory, Livermore, CA, 1982.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 867-883, July 1991
© 1991 Society for Industrial and Applied Mathematics
010

ASYMPTOTIC BEHAVIOR OF THE NUMBER OF REGRESSION QUANTILE BREAKPOINTS*

STEPHEN PORTNOY†

Abstract. In the general regression model $y_i = x_i'\beta + e_i$, for $i = 1, \cdots, n$ and $\beta \in \mathbf{R}^p$, the "regression quantile" $\hat{\beta}(\theta)$ estimates the coefficients of the linear regression function parallel to $x_i'\beta$ and roughly lying above a fraction θ of the data. As introduced by Koenker and Bassett [*Econometrica*, 46 (1978), pp. 33-50], these regression quantiles are analogous to order statistics and provide a natural and appealing approach to the analysis of the general linear model. Computation of $\hat{\beta}(\theta)$ can be expressed as a parametric linear programming problem with J_n distinct extremal solutions as θ goes from zero to one. That is, there will be J_n breakpoints $\{\theta_j\}$, for $j = 1, \cdots, J_n$, such that $\hat{\beta}(\theta_j)$ is obtained from $\hat{\beta}(\theta_{j-1})$ by a single simplex pivot. Each $\hat{\beta}(\theta_j)$ is characterized by a specific subset of p observations. Although no previous result restricts J_n to be less than the upper bound $\binom{n}{p} = \mathbf{O}(n^p)$, practical experience suggests that J_n grows roughly linearly with n. Here it is shown that, in fact, $J_n = \mathbf{O}(n \log n)$ in probability, where the distributional assumptions are those typical of multiple regression situations. The result is based on a probabilistic rather than combinatoric approach which should have general application to the probabilistic behavior of the number of pivots in (parametric) linear programming problems. The conditions are roughly that the constraint coefficients form random independent vectors, and that the number of variables is fixed while the number of constraints tends to infinity.

Key words. linear models, regression quantiles, parametric linear programming

AMS(MOS) subject classifications. primary 90C05, 62J05; secondary 65K05, 49D35, 62J10

1. Introduction.

1.1. Background on regression quantiles. Order statistics, or sample quantiles, provide a powerful approach to the statistical analysis of a single sample. However, in multiple dimensions the lack of a linear order has hampered the extension of such ideas beyond the one sample problem. In 1978, Koenker and Bassett developed a natural and extremely useful generalization of the notion of sample quantile to the general linear model. The generalization is based on the *implicit* characterization of the sample θth quantile as a value $\xi = \xi_\theta$, solving the minimization problem

$$\min_{\xi} \sum \rho_\theta(y_i - \xi) \tag{1.1}$$

where $\theta \in [0, 1]$ and $\rho_\theta(u) = \theta u^+ + (1-\theta)^- u$ for $u^+ = \max\{u, 0\}$ and $u^- = \max\{-u, 0\}$. For example, the sample median corresponds to the case $\theta = \frac{1}{2}$.

To generalize this approach, consider the linear model

$$y_i = x_i'\beta + e_i, \qquad i = 1, \cdots, n, \quad \beta \in \mathbf{R}^p \tag{1.2}$$

where $\{e_i\}$ are a sample from the error distribution function F_e. In matrix form we have

$$\mathbf{Y} = \mathbf{X}\beta + \mathbf{E}$$

where $\mathbf{Y}$ is $n \times 1$, $\mathbf{X}$ is $n \times p$ and has rows x_i', β is $p \times 1$, and $\mathbf{E}$ is $n \times 1$. Though there is no natural notion of an ordering of the sample observations (y_i, x_i), the *implicit* approach to defining the sample quantiles extends (1.1) directly, yielding the "regression quantiles" $\hat{\beta}(\theta)$, which solve the minimization problem

$$\min\{\textstyle\sum \rho_\theta(y_i - x_i'\beta) : \beta \in \mathbf{R}^p\}. \tag{1.3}$$

* Received by the editors March 10, 1988; accepted for publication (in revised form) May 24, 1990. This research was partially supported by National Science Foundation grants DMS 85-03785 and DMS 88-02555, and by Air Force Office of Scientific Research grant AFOSR 87-0041.

† Department of Statistics, University of Illinois, Urbana, Illinois 61801.

In fact, $\hat{\beta}(\theta)$ need not be unique for a given θ; and so we define more generally the regression quantile set

$$B_\theta = \{\beta \in \mathbf{R}^p : \sum \rho_\theta(y_i - x_i\beta) = \min\}.$$

In general, there will be J_n breakpoints,

$$0 = \theta_0 < \theta_1 < \cdots < \theta_{J_n} = 1, \tag{1.4}$$

such that B_θ contains a single fixed estimator for $\theta_j < \theta < \theta_{j+1}$. That is, nonuniqueness occurs only at the breakpoints. In the one sample case ($p = 1$ and all $x_i = 1$) the breakpoints are i/n and $J_n = n$. However, in general, J_n may be larger or smaller than n, as shown in the example below and in the empirical results of § 4. The asymptotic behavior of the number of breakpoints J_n (as $n \to \infty$) will be the primary focus of this paper.

The general regression quantile function $\{\hat{\beta}(\theta) : \theta \in [0, 1]\}$ is by definition the piecewise constant function equal to the unique element of B_θ on the intervals (θ_j, θ_{j+1}) and chosen to be an arbitrary element of B_θ for $\theta = \theta_j$, $j = 1, \cdots, J_n$. An illustrative example given in Koenker and Portnoy (1987) consists of a historical data set analyzed by Boscovich in 1755 (see Stigler (1986, pp. 41–50)). The independent variable x is $\sin^2(\alpha)$, where α is the latitude of five locations on the earth's surface, and the dependent variable y is the measured length of an arc of one degree latitude. Using the model $y = \beta_0 + \beta_1 x$, Boscovich estimated the earth's ellipticity by the equation $1/\text{ellipticity} = 3\beta_0/\beta_1$. The data and the $J_n = 4$ distinct regression quantiles are plotted in Fig. 1.1. The line through Quito and Rome solves (1.3) for $0 \leqq \theta \leqq .213$, the line through Quito and Paris solves (1.3) for $.213 \leqq \theta \leqq .478$, and so forth. At the breakpoints (.213, .478, and .732), any linear combination of the two "adjacent" quantiles will solve (1.3).

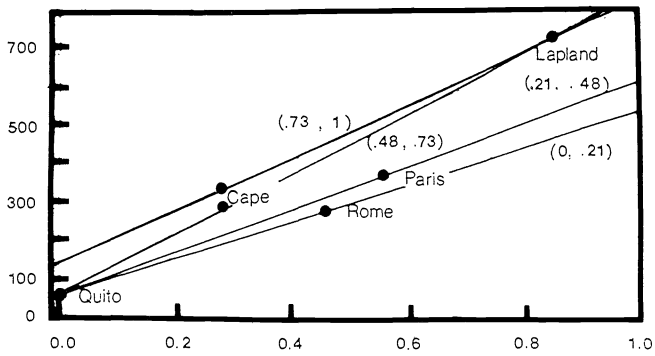


FIG. 1.1. *Boscovich ellipticity data. The x-axis gives* $\sin^2$ (*latitude*) *while the y-axis gives the length of one degree latitude* (*in* Toise) *minus* 56,700. *The* $J_n = 4$ *distinct regression quantile lines are plotted with the interval of corresponding* θ *values given in parentheses near each line. The length of these intervals is denoted by* $D(\theta)$ *in* § 1.3.

Note that $x_i'\hat{\beta}(\theta)$ estimates the parallel conditional quantile functions $x_i'\beta(\theta) + F_e^{-1}(\theta)$, although (as the example demonstrates) the $x_i'\hat{\beta}(\theta)$ themselves need not be parallel. Furthermore, Bassett and Koenker (1982) show that indeed $y_i \leqq x_i'\beta(\theta)$ for very nearly a fraction θ of the data: specifically, $n\theta - p \leqq \#\{i : y_i \leqq x_i'\beta(\theta)\} \leqq n\theta$. They further show that the analogy between $\hat{\beta}(\theta)$ and one-sample quantiles extends to a complete asymptotic theory. Not only does $(\hat{\beta}(\theta))$ have an asymptotic normal distribution which mimics that for one-sample quantiles, but a sample quantile function

analogue $Q(\theta)$ can be defined as $Q(\theta) = \bar{x}'\hat{\beta}(\theta)$, where $\bar{x}$ is the average of the design vectors x_i (see Bassett and Koenker (1982)). This quantile function can be inverted to define an empirical c.d.f. estimator

$$\hat{F}_n(u) = \inf\{\theta: u \leqq \bar{x}'\hat{\beta}(\theta)\},$$

which estimates the error distribution F_e. The asymptotic theory also applies to the c.d.f. estimator $\hat{F}_n$ showing that the process $\sqrt{n}\,(\hat{F}_n(x) - F(x))$ has exactly the same asymptotic behavior as the usual one sample empirical c.d.f. (see Portnoy (1984)). These distributional results provide a basis for statistical inference based on regression quantiles. Ruppert and Carroll (1980) discussed "trimming" observations on the basis of regression quantiles, and Koenker (1987) discussed hypothesis testing. Koenker and Portnoy (1987) provide a general theory of L-estimators for the linear model. Portnoy and Koenker (1987) show how to use regression quantiles to construct fully adaptive estimators of "slope" parameters in the linear model, and Portnoy (1987) discusses their application for identifying outliers.

1.2. Algorithmic considerations: Use of parametric linear programming. The regression quantile estimators $\hat{\beta}(\theta)$ are computable by standard linear programming methods. Indeed, for fixed θ, problem (1.3) is equivalent to minimizing over β the linear function

$$\text{(1.5)} \qquad \sum_{i=1}^{n} [\theta u_i + (1-\theta)v_i]$$

$$\text{subject to} \quad y_i - x_i'\beta = u_i - v_i, \quad u_i \geqq 0, \quad v_i \geqq 0, \quad \text{for } i = 1, \cdots, n.$$

This is not difficult to show since, at a minimum, either $u_i = 0$ or $v_i = 0$ (or both) for $i = 1, \cdots, n$, thus making u_i and v_i the positive and negative parts of $(y_i - x_i'\beta)$. The extreme solutions $\hat{\beta}(0)$ and $\hat{\beta}(1)$ solve especially simple linear programming problems: $\hat{\beta}(0)$ minimizes (over $\beta \in \mathbf{R}^p$) $\sum_{i=1}^{n} (y_i - x_i'\beta)$ (or, equivalently, $-\bar{x}'\beta$) subject to $x_i'\beta \leqq y_i$ for $i = 1, \cdots, n$, and $\hat{\beta}(1)$ maximizes the same linear function subject to the reverse inequalities. As θ varies over $[0, 1]$, (1.5) defines a parametric linear programming problem (as described by Gal (1979)) consisting of a linear family of linear programming problems connecting the two extreme problems.

An efficient algorithm for computing the regression quantile process $\{\hat{\beta}_\theta : \theta \in [0, 1]\}$ is presented in Koenker and d'Orey (1985), based on the well-known L_1 algorithm of Barrodale and Roberts (1974) and standard methods of parametric programming. Given $\hat{\beta}(\theta_j)$ at a fixed breakpoint θ_j, a single simplex pivot will produce $\hat{\beta}(\theta_{j+1})$ at the next larger breakpoint θ_{j+1}. Thus, J_n is the number of simplex pivots needed to solve the complete parametric programming problem (1.5) over $0 \leqq \theta \leqq 1$.

A fundamental question about performance of methods based on the entire regression quantile function concerns the behavior of J_n, the number of different breakpoints $\{\theta_i\}$. As long as J_n is not much larger than the sample size n, regression quantile methodology requires relatively modest computational resources and can be strongly recommended in practice. In fact, experience on a wide variety of real and simulated data sets suggests that J_n grows roughly linearly in n. However, at least in principle, there could be a separate θ_i-value for each subset of p observations (where p is the dimension of the model). This would require $J_n = \binom{n}{p} = \mathbf{O}(n^p)$, and computation would be prohibitive for all but the very smallest problems. Indeed, Murty (1980) provides an example where J_n is as large as possible. However, Murty's example does not permit p to remain fixed, and it is not known whether the example can be adjusted to apply to the specific problem here in (1.3). Thus, it is of utmost importance to

identify the linear model situations, where J_n is sufficiently small so that all regression quantiles can be readily computed.

To study the asymptotic statistical behavior of J_n, approximations $\tilde{J}_n$ will be constructed so that $J_n = \tilde{J}_n$ except for finitely many values of n (with probability one) and such that the expected value of $\tilde{J}_n$ satisfies

$$E\tilde{J}_n \leqq cn \log n \tag{1.6}$$

for some constant c. Consequently, $J_n = \mathbf{O}_p(n \log n)$. That is, for any $\varepsilon > 0$ there are constants B and N such that $P\{J_n/(n \log n) > B\} \leqq \varepsilon$ for $n \geqq N$. For these results, it is essential that standard distributional assumptions for the regression model (1.2) be imposed in developing the appropriate probabilistic behavior for J_n. Since the regression quantile problem is a parametric program, we may have hoped to apply known results on the efficiency of appropriate simplex algorithms in random situations. Indeed, there are results on the probabilistic behavior of the number of pivots in parametric linear programming problems (see Smale (1983), Shamir (1984), Borgwardt (1987) and Blair (1986)). However, most of these results require rather artificial uniform distributions on spheres, which are completely unreasonable in statistical applications. Although Blair's results permit somewhat general discrete distributions (of form (1.2)), his results treat asymptotics in terms of p (and *not* n). None of the previous results apply directly to the particular parametric programming problem here (1.3).

An important idea presented here is that the probabilistic behavior of algorithms should be developed under distributional models, which include those appropriate for the specific application at hand. Although appropriate distributional assumptions clearly depend on the specific application, it is hoped that the present methods might be applicable to a wide range of linear programming problems. From the discussion following (1.5), the results of Theorem 3.1 should apply whenever the number of variables is fixed and the coefficients of the constraint equations can be considered as a sample of size n from some multivariate distribution (in $\mathbf{R}^{p+1}$) satisfying the hypotheses of § 2.

1.3. Basic ideas of the mathematical approach. Because the proof of (1.6) is extremely complicated, a brief sketch of the main steps and basic ideas will be presented. First define the length of the interval, $D(\theta)$, where $\hat{\beta}(\theta)$ remains constant:

$$\begin{aligned} D(\theta) &= \bar{D}(\theta) - \underline{D}(\theta), \\ \bar{D}(\theta) &= \sup\{t: \hat{\beta}(t) = \hat{\beta}(\theta)\}, \\ \underline{D}(\theta) &= \inf\{t: \hat{\beta}(t) = \hat{\beta}(\theta)\}. \end{aligned} \tag{1.7}$$

Thus, $\int_{\theta_i}^{\theta_{i+1}} (1/D(\theta))\, d\theta = 1$, and, hence,

$$J_n = \int_0^1 \frac{1}{D(\theta)}\, d\theta. \tag{1.8}$$

So to analyze J_n, we need the distribution of $D(\theta)$, $F_{nD}(w) \equiv P\{nD(\theta) \leqq w\}$. Furthermore, it suffices to compute $P\{A(w, w)\}$, where

$$A(u, u') = \{\theta: \underline{D}(\theta) \leqq \theta - u/n \wedge \bar{D}(\theta) \geqq \theta + u'/n\}. \tag{1.9}$$

Now to bound the probability of this event, we follow the asymptotic analysis of Koenker and Bassett (1978). Assuming the data are nondegenerate, there must be exactly p zero residuals in order for $\hat{\beta}(\theta)$ to optimize the linear programming problem. That is, letting H be the collection of all subsets of $\{1, 2, \cdots, n\}$ of size exactly p, then $\hat{\beta}(\theta) = \beta_h$ for some $h \in H$, where β_h satisfies

$$Y_i = x_i'\beta_h = (X_h'\beta_h)_i \quad \text{for } i \in h. \tag{1.10}$$

Here X_h is the $p \times p$ matrix with columns x_i for $i \in h$. Furthermore, the optimality conditions imply that $\hat{\beta}(\theta) = \beta_h$ if and only if $n^{1/2} X_h^{-1} V \in S$, where S is a rectangle in $\mathbf{R}^p$ depending on θ and W (see (2.1)) and where V and W are defined by

$$V = V_h = n^{-1/2} \sum_{i \notin h} I(Y_i \leqq x_i' \beta_h) x_i,$$
$$W = W_h = n^{-1/2} \sum_{i \notin h} x_i. \tag{1.11}$$

Koenker and Bassett then argue that an extension of the result of Stone (1965) (based on the central limit theorem and subsequently provided by Mukhin (1984)) implies

$$P\{\hat{\beta} = \beta_h\} = P\{n^{1/2} V \in X_h S\} \approx c \operatorname{Vol}(n^{-1/2} S) \det(X_h) \tag{1.12}$$

where c is a constant depending on the asymptotic normal distribution of V (given by the central limit theorem).

This Koenker-Bassett argument can be related to $P(A)$ in (1.9) as follows. Some algebra (see Lemma 2.1) shows that

$$\{\hat{\beta}(\theta) = \beta_h\} \wedge A(u, u') \Leftrightarrow n^{1/2} V \in X_h T(W, u, u')$$

where T is a subrectangle of S (depending on W and having volume $(1+2w)^p - 1 \leqq c_0 w$). Since $F_n D(w)$ can be calculated as $\sum_{h \in H} P\{\hat{\beta} = \beta_h \wedge A(w, w)\}$, the Koenker-Bassett analysis (along with much additional computation) will show that

$$\begin{aligned} F_{nD}(w) \equiv P\{nD(\theta) \leqq w\} &\leqq c \sum_{h \in H} (E_h \det(X_h) \operatorname{Vol}(T) + \text{error}) \\ &\leqq c' w + \sum_{h \in H} \text{error}. \end{aligned} \tag{1.13}$$

The major problem requiring much new work is that the error must be small enough so that the sum over $h \in H$ (a sum of $\binom{n}{p}$ summands) remains small. For this, the error (for each h) should be of order $\mathbf{O}(n^{-(p+2)})$, and, unfortunately, the original analysis of Koenker and Bassett (using the Stone-Mukhin results) only provides an error of order $\mathbf{O}(n^{-p/2})$. To obtain the needed order, the distribution of V must be approximated using higher-order normal approximations (that is, Edgeworth expansions; see Feller (1966)). However, V does not satisfy the Cramér condition (which is needed for the Edgeworth expansions) unless x_i are also taken to be random. Thus, we will assume that both Y_i and x_i are random (satisfying conditions given in § 2), apply Edgeworth expansions to the pair (V, W) (see Lemma 2.3), and then analyze J_n (using (1.8)). An additional complication arises since $nD(\theta)$ is approximately uniform (see (1.13)); and, hence, $E(1/D(\theta)) = \infty$. To circumvent this problem, we define

$$\tilde{J}_n = n \int_0^1 \min\left(\frac{1}{nD(\theta)}, n^{p+2}\right) d\theta, \tag{1.14}$$

the expectation of which can be bounded (using (1.13)); and show that $D(\theta) > 1/n^{p+3}$ except for finitely many values of n (with probability one). The main results are given in § 3 following the statement of hypotheses and some preliminary analysis presented in § 2. Section 4 presents a small-scale simulation experiment showing quite reasonable agreement with theoretical results. Empirically, the discrepancy from theory depends on the number of parameters p, but is nearly independent of distributional assumptions (at least in smooth cases).

2. Hypotheses and preliminary results. As noted at the end of § 1, it will be necessary to assume that the independent variables $\{x_i\}$ are random. It is desirable to permit two types of randomness for the independent variables. The first type is where $\{x_i\}$ form a sample from some distribution in $\mathbf{R}^p$, as would be appropriate in a wide variety of multiple regression settings. The second case, appropriate for ANOVA, ANOCOVA, or related models, is where it is desired to consider the $\{x_i\}$ fixed. Here, the requisite smoothness can be achieved by adding randomization externally; that is, let $x_i = u_i + \varepsilon_i$, where u_i are fixed and ε_i form a random sample with zero mean and fixed covariance matrix. This external randomization, sometimes called "dithering," is extremely useful in the regression quantile setting both for precluding algorithmic degeneracy and for identifying outliers (see Portnoy (1987)). Since the first case can be modeled by the second case (by taking $u_i = 0$), formal conditions for the second case will be listed below. In either case, the conditional distribution of Y_i will be given by the standard linear model (1.2). The conditions here will also assume that the average mean of the independent variables is zero; when the model design contains a constant, this occurs without loss of generality. Condition (B) below posits continuity of the error distribution and is needed to apply Theorem 3.4 of Koenker and Bassett (1978) (see Lemma 2.3).

The specific assumptions follow:

(A) $x_i = (1, u_i) + (0, \varepsilon_i)$, $i = 1, \cdots, n$, where

(A.1) $\{\varepsilon_i\}$ form an independent and identically distributed sample in $\mathbf{R}^{p-1}$ satisfying:

(a) $E\varepsilon_i = 0$,
(b) $E\|\varepsilon_i\|^{12p+13} < +\infty$,
(c) The distribution of ε_i has an absolutely continuous component.

(A.2) For some constant b, $\|u_i\| < b$ for $i = 1, \cdots, n$.

(A.3) $\sum_{i=1}^{n} u_i = 0$.

(B) The error distribution F_e is continuous.

Some preliminary results can now be presented. First consider the random variables (V, W) defined by (1.11) and $(D, \bar{D}, \underline{D})$ defined by (1.7). Koenker and Bassett (1978) show that for each $h \in H$, the event $\{\hat{\beta} = \beta_h\}$ holds if and only if $(V, W) \in S$, where S is the set

$$S_h = \{(v, w): \theta - 1 \leqq A_j - B_j\theta \leqq \theta, j = 1, \cdots, p\}. \tag{2.1}$$

Here A_j and B_j are the jth coordinates of the vectors

$$A = n^{1/2}X_h^{-1}V = n^{1/2}X_h^{-1}\sum_{h \notin H} I(Y_i \leqq x_i'\beta)x_i, \qquad B = n^{1/2}X_h^{-1}W = n^{1/2}X_h^{-1}\sum_{h \notin H} x_i$$

where X_h is the $p \times p$ matrix with columns $\{x_i': i \in h\}$.

The first result below (Lemma 2.1) shows how to express the event $\{nD(\theta) \leqq w\}$ in terms of a subrectangle of S to which the Koenker-Bassett argument applies. Lemma 2.3 then expresses the probability of this rectangle using the Edgeworth expansion. Since the expansion holds only on a subset of the observation space, Lemma 2.3 also shows how to restrict the calculations to the subset on which the expansion holds to within a negligible error.

LEMMA 2.1. *Let $u>0$, and let $u'>0$ be given and define the set*

$$T_h(u,u')=\Big\{(v,w)\colon (B_j+1)\Big(-\frac{u}{n}+I_j\Big(\frac{u+u'}{n}\Big)\Big)+\theta-1\leqq A_j-B_j\theta \leqq (B_j+1)\Big(\frac{u'}{n}-I_j\Big(\frac{u+u'}{n}\Big)\Big)+\theta, j=1,\cdots,p\Big\} \tag{2.2}$$

where I_j is the indicator function $I(B_j+1>0)$. (Note that $S=T(0,0)$.) Then the event $\{\hat\beta=\beta_h \wedge \underline{D}\leqq\theta-u/n\wedge\bar D\geqq\theta+u'/n\}$ occurs if and only if $(V,W)\in T_h$.

Proof. Solving the inequalities in (2.1) gives

$$\frac{A_j+(1-I_j)}{B_j+1}\leqq\theta\leqq\frac{A_j+I_j}{B_j+1},\qquad j=1,\cdots,p.$$

Therefore,

$$\underline{D}(\theta)=\max_j\Big\{\frac{A_j+1-I_j}{B_j+1}\Big\},\qquad \bar D(\theta)=\min_j\Big\{\frac{A_j+I_j}{B_j+1}\Big\}.$$

Some straightforward algebra now shows that both $\underline{D}(\theta)\leqq\theta-u/n$ and $\bar D(\theta)\geqq\theta+u'/n$ hold if and only if the inequalities in (2.2) hold. □

Now define the following notation.

DEFINITION 2.1. Let $P_h(\cdot)$ denote the conditional probability of a set (depending on (V,W)) conditional on $Z\equiv\{(x_i,Y_i)\colon i\in h\}$; let E_h denote conditional expectation given Z; and let E^h denote the (marginal) expectation of a function of Z.

LEMMA 2.2. *Let (V,W) be defined by* (1.11) *and assume that condition* (A) *holds. Define*

$$\mu\equiv\mu_h=E_hV\quad\text{and}\quad\eta\equiv\eta_h=E_hW. \tag{2.3}$$

Then there are constants $b_0>0$ and $b_n\to0$, with $b_n\geqq n^{-\varepsilon}$ for some $\varepsilon\in(0,\frac16)$, such that, except with P_h-probability at most $\mathbf{O}(n^{-(2p+2)})$,

$$\|W-\eta\|\leqq b_nn^{1/6}, \tag{2.4}$$

$$\lambda_{\max}(X_h'X_h)\leqq p\max_{1\leqq i\leqq n}\|x_i\|^2\leqq b_0^2n^{1/2} \tag{2.5}$$

where $\lambda_{\max}$ denotes the maximum eigenvalue.

Remark. Note that (2.5) implies that for any $z\in\mathbf{R}^p$, $\|X_hz\|\leqq b_0n^{1/4}\|z\|$.

Proof. First note that by condition (A.1),

$$E_h\Big\|\sum_{i=1}^n(x_i-E_hx_i)\Big\|^k\leqq cn^{k/2}\quad\text{for } k=12p+13.$$

Whence, $E_h\|W-\eta\|^k\leqq c$; and Chebyshev's inequality implies

$$P_h(\|W-\eta\|\geqq b_nn^{1/6})\leqq cb_n^{-(12p+13)}n^{-(2p+2+1/6)}\leqq\mathbf{O}(n^{-(2p+2)})$$

as long as b_n is such that $b_n^{12p+13}n^{1/6}$ is bounded away from zero. Note that $b_n=n^{-\varepsilon}$ works here with $\varepsilon=(6(12p+13))^{-1}$.

Similarly, from conditions (A.1) and (A.2), $E_h\|x_i\|^{8p+12}$ is bounded. Hence,

$$P_h\Big(\max_i\|x_i\|\geqq b_0n^{1/4}\Big)\leqq cnb_0^{-(8p+12)}n^{-(2p+3)}=\mathbf{O}(n^{-(2p+2)}).$$

The inequality on $\lambda_{\max}$ follows immediately. □

LEMMA 2.3. *Let (V,W,μ,η) be given by* (1.11) *and* (2.3). *Let $p(v-\mu,w-\eta)\exp\{-\frac12A(v-\mu,w-\eta)\}$ denote the Edgeworth expansion of the distribution of*

(V, W) with error of order $n^{-(2p+2)}$:

$$p(v, w) = (2\pi)^{-p/2}(\det \Sigma)^{-1/2}\left(1 + \sum_{j=1}^{4p+6} \frac{Q_j(v, w)}{n^{j/2}}\right), \tag{2.6}$$

$$A(v, w) = (v'\ w')\Sigma^{-1}(v\ w)'$$

where $\Sigma = \mathrm{Cov}\,(V, W)$ and Q_j are polynomials of degree $j+2$. Define the event

$$R(u, u') = \{\underline{D}(\theta) \leqq \theta - u/n \wedge \bar{D}(\theta) \geqq \theta + u'/n\}. \tag{2.7}$$

Using (2.1), (2.2) *and* (2.4)-(2.5), *define sets*

$$\begin{aligned} \tilde{T} &= \{v: (v, w) \in S_h - T_h\}, \\ D_h^* &= \{w: \|w - \eta_h\| \leqq b_n n^{1/6}\}, \\ D_h &= \{x_i: i \in h \wedge \lambda_{\max}(X_h' X_h) \leqq b_0^2 n^{1/2}\}. \end{aligned} \tag{2.8}$$

Then under conditions (A) *and* (B), *there is $H_1 \subset H$ and $\theta_0 \in (0, \frac{1}{2})$ (see* (2.10)) *such that for all u and u' the probability of the complement of $R(u, u')$ is*

$$P(R^c) = \sum_{h \in H_1} E^h U_h + \mathbf{O}(n^{-(p+2)})$$

uniformly in $\theta \geqq \theta_0$ where (using (2.8)),

$$U_h = I(D_h) \int_{D_h^*} \int_{\tilde{T}} p(v - \mu, w - \eta)\, e^{-A(v-\mu, w-\eta)/2}\, dv\, dw.$$

Here $I(D_h)$ denotes the indicator function of the set D_h.

Proof. First note that by Lemma 2.1,

$$R^c = \bigcup_h \{\hat{\beta}(\theta) = \beta_h\} \cap R^c = \bigcup_h (S_h - T_h).$$

Also, by Lemma 2.2,

$$E^h P_h\{S_h - T_h\} = E^h I(D_h) P_h\{(S_h - T_h) \cap D_h^*\} + \mathbf{O}(n^{-(2p+2)}).$$

Therefore, since the cardinality of H is less than n^p,

$$P(R^c) = \sum_{h \in H} E^h I(D_h) P_h\{(S_h - T_h) \cap D_h^*\} + \mathbf{O}(n^{-(p+2)}). \tag{2.9}$$

Now let $q_h = P_h\{Y_i \leqq x_i'\beta_h\}$. Define the binomial random variable $N_h = \#\{i: Y_i \leqq x_i'\beta_h\}$, and choose constants $p_1 < p_0 < \frac{1}{2}$ and a such that for $q_h < p_1$

$$P\{N_h \geqq p_0 n\} \leqq e^{-an}.$$

Now define the set H_1 and the constant θ_0 as follows:

$$H_1 = \{h: q_h \geqq p_1\}, \qquad \theta_0 = p_0 + \frac{p}{n}. \tag{2.10}$$

(Note. $\theta_0 < \frac{1}{2}$ for n large enough.) Now if $(V_h, W_h) \in S$, then $\hat{\beta}(\theta) = \beta_h$. Hence, since F_e is continuous, Koenker-Bassett (1978, Thm. 3.4 and following) yields $N_h \geqq n\theta - p \geqq p_0 n$, for $\theta \geqq \theta_0$. Therefore,

$$\begin{aligned} \sum_{h \notin H_1} E^h I(D_h) P_h\{(S_h - T_h) \cap D_h^*\} &\leqq \sum_{h \notin H_1} P\{N_h \geqq p_0 n\} \\ &\leqq \binom{n}{p} e^{-an} = \mathbf{O}(n^{-(2p+2)}) \end{aligned} \tag{2.11}$$

uniformly for $\theta \geqq \theta_0$. Thus, $P(R^c)$ is given by (2.9) with the sum over H_1 instead of H.

Now for $h \in H_1$ apply the Edgeworth expansion given by Theorem 20.6 of Bhattacharya and Rao (1976) to the distribution of (V_h, W_h). Since $q_h \geqq p_1$, condition (A.1)(c) implies that Cramér's condition holds; and the required moment condition also holds by condition (A.1)(b). Using arguments similar to those in Bhattacharya and Rao, the additional error term in Theorem 20.6 is bounded by the standard normal probability (in $\mathbf{R}^{2p}$) of an ε-neighborhood of the boundary of $(S_h - T_h) \cap D_h^*$ with $\varepsilon = 4\, e^{-an}$, which is in turn bounded by the volume of this boundary set. For each $w \in D_h^*$, consider the w-section of this boundary set. Since the w-section of $S_h - T_h$ is a difference of the images of rectangles under X_h, the volume is bounded by a constant times $\det(X_h)$ times the volume of a rectangular face times ε. Since the length of a rectangular side is bounded by $n^{-1/2} \leqq 1$, and $|\det X_h| \leqq \lambda_{\max}(X_h' X_h)^{p/2}$, the volume of the w-section is bounded by a constant times $n^{p/4}\, e^{-an}$. Thus, since the volume of the boundary set is $\mathrm{Vol}(D_h^*)$ times the volume of the w-section, and $\mathrm{Vol}(D_h^*) \leqq n^{(p-1)/6}$, the additional error term is bounded by a constant times $n^{p/2}\, e^{-an} = \mathbf{O}(n^{-(2p+2)})$. Therefore, for $h \in H_1$ and $\theta \geqq \theta_0$, the Edgeworth expansion holds with error of the desired order. Thus, the probability in (2.9) can be replaced by the integral given by the lemma, where the domain splits as indicated since D_h^* depends only on w. □

3. Main results. The main result bounding J_n in probability will be developed as Corollary 3.1 to Theorem 3.1. The theorem gives an appropriate bound on the distribution function of $nD(\theta)$ (to within a negligible error), and it follows from Lemma 2.3 and some technical computations given in Lemmas 3.1–3.3. Lemmas 3.1 and 3.2 provide bounds on the terms of the Edgeworth expansion, and Lemma 3.3 analyzes the volume of the rectangle $\tilde{T}$ appearing in the result of Lemma 2.3. Conditions (A) and (B) will be assumed throughout this section.

THEOREM 3.1. *Let $D(\theta)$ be given by* (1.9). *Then,*

$$P\{nD(\theta) \leqq w\} \leqq c^* w + \mathbf{O}(n^{-(p+2)})$$

uniformly for $\theta \in [0, 1]$, for some constant c^*.

Proof. Let $\tilde{H}$ be defined by (3.9). Starting from the conclusion of Lemma 2.3 and applying Lemmas 3.1 and 3.3, for $h \in H_1 \cap \tilde{H}$ and $\theta \geqq \theta_0$, we have

$$\frac{U_h}{I(D_h)} = \int_{D_h^*} \int_{\tilde{T}} (2\pi)^{-p/2} (\det \Sigma)^{-1/2} (1 + \mathbf{o}(1))\, e^{-g(w)+\mathbf{o}(1)}\, dv\, dw$$

$$\leqq c \int_{D_h^*} \mathrm{Vol}(\tilde{T})\, e^{-g(w)}\, dw$$

$$\leqq c n^{-p/2} \left(\sum_{j=1}^{p} |\det X_{hj}| + |\det X_h| \right) (u + u') \int_{D_h^*} e^{-g(w)}\, dw$$

where c is a generic constant, u and u' appear in the definition of the set R^c in Lemma 2.3, and X_{hj} denotes the submatrix of X_h obtained by deleting the first column and the jth row (as in Lemma 3.3).

Now, the expectation E^h need be applied only to the determinants above, since $I(D_h) \leqq 1$ and the other factors above do not depend on X_h. By Lemma 3.4, $E^h|\det X_h|$ is bounded, and since X_{hj} is a submatrix, the same argument bounds its expected

absolute value. Therefore, Lemma 2.3 yields

$$P(R^c) \leqq c(u+u')n^{-p/2} \sum_{h \in H_1 \cap \tilde{H}} E^h I(D_h) \int_{D_h^*} e^{-g(w)}\, dw + \varepsilon + \mathbf{O}(n^{-(p+2)}) \tag{3.1}$$

where

$$\varepsilon \leqq \sum_{h \in \tilde{H}^c} E^h I(D_h) \int_{D_h^*} \int_{\tilde{T}} p(v-\mu, w-\eta)\, e^{-A(v-\mu, w-\eta)/2}\, dv\, dw.$$

By Lemma 3.2, for c a positive generic constant and $d = \frac{1}{6} - \varepsilon$,

$$\begin{aligned} \varepsilon &\leqq c \sum_{h \in H} E^h \operatorname{Vol}(S_h) \int_{D_h^*} (1 + \|\theta\eta - \mu\|)^{4p+8}\, e^{-c\|\theta\eta - \mu\| - cn^d}\, dw \\ &\leqq cn^p n^{p/6}\, e^{-cn^d} = \mathbf{O}(n^{-(p+2)}), \end{aligned}$$

since $\operatorname{Vol}(S_h) \operatorname{Vol}(D_h^*) \leqq cn^{p/6}$. Hence,

$$P(R^c) \leqq c(u+u')n^{-p/2} \sum_{h \in H_1 \cap \tilde{H}} \int_{D_h^*} e^{-g(w)}\, dw + \mathbf{O}(n^{-(p+2)}). \tag{3.2}$$

In an entirely analogous manner,

$$\begin{aligned} 1 &= \sum_{h \in H} P\{\beta = \beta_h\} = \sum_{h \in H_1 \cap \tilde{H}} E^h P_h(S_h) + \mathbf{O}(n^{-(p+2)}) \\ &\geqq c' \sum_{h \in H_1 \cap \tilde{H}} (E^h I(D_h)) \operatorname{Vol}(S_h) \int_{D_h^*} e^{-g(w)}\, dw + \mathbf{O}(n^{-(p+2)}) \\ &\geqq c' n^{-p/2} \sum_{h \in H_1 \cap \tilde{H}} \int_{D_h^*} e^{-g(w)}\, dw + \mathbf{O}(n^{-(p+2)}) \end{aligned}$$

where Lemmas 3.2–3.4 are applied as above, and the fact that $P(D_h) = 1 - \mathbf{o}(1)$ is used. Combining this with (3.2),

$$P(R^c) \leqq c^*(u+u') + \mathbf{O}(n^{-(p+2)}) \tag{3.3}$$

for some constant c^*, uniformly in $\theta \geqq \theta_0$.

Lastly, using (3.3) and (2.7) (for the definition of R),

$$\begin{aligned} P\{nD(\theta) \leqq w\} &\leqq P\left\{ \underline{D}(\theta) \geqq \theta - \frac{w}{n} \wedge \bar{D}(\theta) \leqq \theta + \frac{w}{n} \right\} \\ &\leqq P\left\{ \underline{D}(\theta) > \theta - \frac{w}{n} \vee \bar{D}(\theta) < \theta + \frac{w}{n} \right\} = P(R^c) \leqq cw. \end{aligned}$$

This gives the result of Theorem 3.1 for $\theta \geqq \theta_0$. By replacing e_i by $-e_i$ and β by $-\beta$ in the linear model (1.2), the distribution of $\{D(\theta)\colon \theta \leqq \theta_0\}$ becomes the same as the distribution of the transformed random function $\{D^-(1-\theta)\colon \theta \geqq 1-\theta_0\}$. Since the conditions for the theorem also hold for the model with $-e_i$ and $-\beta$, the result holds for all $\theta \in [0, 1]$. □

Recalling (1.14), define the following approximation to J_n:

$$\tilde{J}_n = \int_0^1 \min\left\{ \frac{1}{D(\theta)}, n^{p+3} \right\} d\theta. \tag{3.4}$$

COROLLARY 3.1. *Under conditions* (A) *and* (B), *there is a constant* $\tilde{c}$ *such that*

$$E\tilde{J}_n \leqq \tilde{c} n \log n, \tag{3.5}$$

and furthermore, $\tilde{J}_n = J_n$ except for finitely many values of n with probability one. Consequently, $J_n = \mathbf{O}_p(n \log n)$.

Proof. Factoring out n in (3.4), and letting $F_n(w) \equiv F_n(w; \theta)$ be the distribution function for $nD(\theta)$, we have

$$
\begin{aligned}
E\tilde{J}_n &= n \int_0^1 E \min\left\{\frac{1}{nD(\theta)}, n^{p+2}\right\} d\theta \\
&= n \int_0^1 \left\{ \int_{n^{-(p+2)}}^{n} \frac{1}{w}\, dF_n(w) + n^{p+2} \int_0^{n^{-(p+2)}} dF_n(w) \right\} d\theta \qquad (3.6) \\
&\equiv n \int_0^1 \{I_1 + I_2\}\, d\theta
\end{aligned}
$$

where I_1 and I_2 are the inner integrals above. Now, by Theorem 3.1,

$$
I_2 = n^{p+2} F_n(n^{-(p+2)}) \leqq c^* \qquad (3.7)
$$

uniformly in θ. Furthermore, integrating by parts and using Theorem 3.1,

$$
\begin{aligned}
I_1 &= \frac{1}{w} F_n(w)\Big|_{n^{-(p+2)}}^{n} + \int_{n^{-(p+2)}}^{n} \frac{1}{w^2} F_n(w)\, dw \\
&\leqq \frac{1}{n} + c^* \int_{n^{-(p+2)}}^{n} \frac{1}{w}\, dw + \mathbf{O}(n^{-(p+2)}) \int_{n^{-(p+2)}}^{n} \frac{1}{w^2}\, dw \leqq \tilde{c} \log n,
\end{aligned}
$$

again, uniformly in θ. Hence, from (3.6) and (3.7), equation (3.5) holds. Thus, since $\tilde{J}_n$ is positive, the first-order Chebyshev inequality implies that

$$
\tilde{J}_n = \mathbf{O}_p(n \log n). \qquad (3.8)
$$

Lastly, compute the probability that $\tilde{J}_n$ differs from J_n. Let $\{\theta_i\}$ denote the breakpoints, and recall that since each breakpoint corresponds to at most one subset h, there are fewer than n^p breakpoints. Thus, from (1.7) and (3.4), using Theorem 3.1,

$$
\begin{aligned}
P\{\tilde{J}_n \neq J_n\} &= P\{\inf_\theta \{D(\theta)\} \leqq n^{-(p+3)}\} \leqq P\{\inf_i \{D(\theta_i)\} \leqq n^{-(p+3)}\} \\
&\leqq n^p P\{D(\theta_i) \leqq n^{-(p+3)}\} \leqq n^p F_n(n^{-(p+2)}) \leqq \frac{c^*}{n^2}.
\end{aligned}
$$

Therefore, $\sum_{n=1}^{\infty} P\{\tilde{J}_n \neq J_n\} < +\infty$, and the Borel–Cantelli lemma implies that $\tilde{J}_n = J_n$ except finitely often. Thus, J_n and $\tilde{J}_n$ have the same stochastic order, and the remainder of the corollary follows from (3.8). □

LEMMA 3.1. *Let D_h^* and D_h be defined by* (2.8), *and let p and A give the Edgeworth expansion* (2.6). *Define*

$$
\tilde{H} = \{h: |\eta_h - \theta\mu_h| \leqq b_n^{1/2} n^{1/6}\} \qquad (3.9)
$$

where the variables in (3.9) *are defined in Lemma* 2.3. *Then there is a function $g(w)$, depending only on w, such that for $h \in \tilde{H}$, $X_h \in D_h$, and uniformly for $(v, w) \in S_h \cap D_h^*$,*

$$
\begin{aligned}
&A(v - \mu, w - \eta) = g(w) + \mathbf{o}(1), \quad \textit{and} \\
&p(v - \mu, w - \eta) = (2\pi)^{-p/2} \det(\Sigma)^{-1/2}(1 + \mathbf{o}(1)).
\end{aligned}
$$

Proof. First note that for $(v, w) \in S_h$, $v - \theta w$ lies in the image under X_h of a rectangle of side $n^{-1/2}$. Thus, on S_h and D_h,

$$
\|v - \theta w\| \leqq n^{-1/2} |\lambda_{\max}(X_h)| \leqq c n^{-1/4}.
$$

Now, expanding $v - \mu$ and defining the vector $k(w)$,

$$
v - \mu = v - w\theta + \theta(w - \eta) + \theta\eta - \mu \equiv v - \theta w + k(w). \qquad (3.10)
$$

Note that $\|v-\mu\| = \mathbf{O}(n^{1/6})$. Then, from (2.6),

$$\begin{aligned} A(v, w) &= (v'\ w')\Sigma^{-1}(v\ w)' \\ &= ((w-\eta)'k'(w))\Sigma^{-1}((w-\eta)k(w))' + (0'(v-\theta w)')\Sigma^{-1}(0(v-\theta w))' \\ &\quad + 2(0'(v-\theta w)')\Sigma^{-1}((w-\eta)k(w))' \\ &= g(w) + \mathbf{O}(\|v-\theta w\|(\|w-\eta\| + \|\theta\eta-\mu\| + \|v-\theta w\|)) \\ &= g(w) + \mathbf{O}(n^{-1/4}(n^{1/6}+n^{1/6}+n^{-1/4})) = g(w) + \mathbf{o}(1) \end{aligned}$$

where $g(w)$ is defined in the third equality above.

Similarly, the polynomials Q_j satisfy

$$Q_j(v-\mu, w-\eta) = \mathbf{O}(\|v-\mu\|^{j+2} + \|w-\eta\|^{j+2}) = \mathbf{O}(b_n^{1/2}n^{1/6})^{j+2}.$$

Therefore, since $j \geqq 1$,

$$n^{-j/2}Q_j = \mathbf{O}(b_n^{(j+2)/2}n^{-j/3+1/3}) = \mathbf{O}(b_n^{3/2}) = \mathbf{o}(1).$$

The remainder of the lemma follows from (2.6). □

LEMMA 3.2. *Using the definitions of Lemma* 3.1, *for* $h \in \tilde{H}^c$ (*the complement of* $\tilde{H}$; *see* (3.9)), $X_h \in D_h$, *and* $(v, w) \in S_h \cap D_h^*$,

$$A(v-\mu, w-\eta) \geqq cn^d + c\|\theta\eta-\mu\|, \quad \textit{and}$$
$$p(v-\mu, w-\eta) \leqq c(1+\|\theta\eta-\mu\|^{4p+8})$$

where c *is a generic positive constant and* $d = \frac{1}{6}-\varepsilon$ (*with* ε *given by Lemma* 2.2).

Proof. From (2.6) and the order relations in Lemma 3.1,

$$\begin{aligned} A(v-\mu, w-\eta) &\geqq \lambda_{\min}(\Sigma^{-1})(\|v-\mu\|^2 + \|w-\eta\|^2) \\ &\geqq c(\|\theta\eta-\mu\|^2 - 2\|\theta\eta-\mu\|(\|v-\theta w\| + \theta\|w-\eta\|)) \\ &\geqq c\|\theta\eta-\mu\|(b_n^{1/2}n^{1/6} - (\theta b_n n^{1/6} + \mathbf{O}(n^{-1/4}))) \\ &\geqq cb_n^{1/2}n^{1/6}\|\theta\eta-\mu\| \geqq \tfrac{1}{2}cb_n n^{1/6} + \tfrac{1}{2}c\|\theta\eta-\mu\| \end{aligned}$$

from which the first part of the lemma follows since $b_n \geqq n^{-\varepsilon}$.

Again, analogously to Lemma 3.1, for $h \in \tilde{H}^c$,

$$\begin{aligned} Q_j(v-\mu, w-\eta) &= \mathbf{O}(\|w-\eta\| + \|\theta\eta-\mu\| + \|v-\theta w\|)^{j+2} \\ &\leqq c(b_n n^{1/6} + \|\theta\eta-\mu\|^{4p+8}). \end{aligned}$$

The remainder of the lemma now follows directly from (2.6). □

LEMMA 3.3. *Let* $\tilde{T} = \tilde{T}(w)$ *be given by* (2.8) (*see also* (2.1) *and* (2.2)). *Then there is a constant* c *such that for* $w \in D_h^*$,

$$\text{Vol}\,(\tilde{T}(w)) \leqq cn^{-p/2}\sum_{j=1}^{p}(|\det X_{hj}| + |\det X_h|)(u+u').$$

Proof. First consider T_h given by (2.2) and analyze the factor $B_j + 1 = n^{1/2}(X_h^{-1}W)_j + 1$. Since $\sum_{i=1}^n u_i = 0$, with $e = (1, 0, \cdots, 0)'$ and $\underline{1} = (1, \cdots, 1)'$,

$$\begin{aligned} B + \underline{1} &= n^{1/2}X_h^{-1}(W-\eta) + X_h^{-1}\sum_{i\notin h}(1\ u_i)' + \underline{1} \\ &= nX_h^{-1}e + n^{1/2}X_h^{-1}(W-\eta) - \sum_{i\in h}X_h^{-1}(1\ u_i)' + \underline{1} \qquad (3.11) \\ &\equiv nX_h^{-1}e + c(w) \end{aligned}$$

where

$$c(w) = n^{1/2}X_h^{-1}(W-\eta) - X_h^{-1}\sum_{i\in h}(1\ u_i)' + \underline{1} \tag{3.12}$$
$$= \mathbf{O}(n^{1/2+1/6+1/4}) = \mathbf{o}(n)$$

for $w \in D_h^*$ by conditions (A.2) and (A.3) and the remark after Lemma 2.2.

Now note that T_h is the result of applying the matrix $n^{-1/2}X_h$ to a rectangle whose sides have length $1-L_j$ (equal to the difference between the left and right sides of the inequalities defining T_h in Lemma 2.2), where

$$L_j = \left\{[n(X_h^{-1}\ e)_j + c_j(w)]\frac{(u+u')}{n}\right\}. \tag{3.13}$$

Furthermore, T_h is empty unless $0 \leqq (B_j+1)(u+u')/n \leqq 1$ for $j = 1, \cdots, p$. Therefore, letting the notation $Z \wedge (0,1)$ denote the term Z truncated below zero and above one:

$$\begin{aligned}\text{Vol}\,(T_h) &= n^{-p/2}|\det X_h| \prod_{j=1}^{p}\{1-[L_j \wedge (0,1)]\} \\ &= n^{-p/2}|\det X_h| \prod_{j=1}^{p}\left\{1-\left[\left(\frac{\det X_{hj}}{\det X_h}+\frac{c_j(w)}{n}\right)(u+u')\right]\wedge(0,1)\right\}\end{aligned} \tag{3.14}$$

where Cramer's rule is used to compute $(X_h^{-1}\ e)_j$. Note that by (3.12) $n^{-1}c_j(w) = \mathbf{O}(1)$. Now, let $G \subset \{1, \cdots, p\}$, and consider an arbitrary term in the expansion of the product above where the factors not equal to one are indexed by $j \in G$. If $\#G \geqq 1$, bound all but one of the factors by one. Then

$$\begin{aligned}\left|\prod_{j\in G} L_j \wedge (0,1)\right| &= \left|\prod_{j\in G}\left[\left(\frac{\det X_{hj}}{\det X_h}+\mathbf{O}(1)\right)(u+u')\right]\wedge(0,1)\right| \\ &\leqq \left|\frac{\det X_{hj_0}}{\det X_h}+c\right|(u+u')\end{aligned}$$

for some j_0. Therefore, from (3.14), since there are 2^p such terms in the p-fold product,

$$\text{Vol}\,(T_h) \geqq n^{-p/2}|\det X_h| - \frac{2^p}{p}n^{-p/2}\sum_{j=1}^{p}(|\det X_{hj}| + c|\det X_h|)(u+u').$$

Therefore, the lemma follows since $\text{Vol}\,(S_h) = n^{-p/2}|\det X_h|$, and $\tilde{T}(w) = S_h - T_h$. □

LEMMA 3.4. *Under condition* (A.3) *there are constants* $0 < c_1 < c_2 < +\infty$ (*independent of* h) *such that*

$$c_1 < E^h|\det X_h| < c_2.$$

Proof. Let U_h be the $p \times p$ matrix with rows $(0\ u_i)'$, and note that

$$E^h|\det X_h| = E^h|\det((X_h - U_h) + U_h)|. \tag{3.15}$$

For a given $p \times p$ matrix U, define $g(U) = E|\det(Z+U)|$, where $Z \sim X_h - U_h$ has a fixed distribution independent of h (by condition (A.1)). Since $\det(Z+U) = 0$ on a null set, condition (A.1)(c) implies that $g(U)$ is strictly positive for all U. Using condition (A.2), define the compact set $C = \{U \in \mathbf{R}^{p^2}: \text{tr}\,(U\,U') \leqq pb\}$. By condition (A.2), $U_h \in C$ for all $h \in H$. Since g is continuous on C, g is bounded above zero and below $+\infty$ on C, and the lemma follows from (3.15). □

4. Empirical results. A small-scale simulation experiment was carried out on the SUN network at the Statistics Department of the University of Illinois. The experimental design specified four fully crossed factors: the main independent factor is the sample size and the other factors are the number of parameters p (including the intercept) and distributional assumptions on x and Y. For the distributional assumptions, the coordinates of x are taken to be independent, and an asymmetric distribution of the form $U+.3V$, where U is negative exponential (with mean 1) and $V \sim N(0,1)$, is introduced. The precise design follows:

n (5 levels): 20, 50, 100, 200, 400
p (3 levels): 2, 5, 10
x distribution (2 levels): $N(0,1)$, $U+.3V$
Y distribution (4 levels): $N(0,1)$, $t(4\ df)$, $U+.3V$, logistic

The simulation used the Koenker-d'Orey (1987) algorithm with 1000 Monte Carlo replications except for the seven cases where $n=400$ and one of the distributions was not normal (when 250 replications were used). The dependent variables considered were the following Monte Carlo sample statistics: the average value of the number of breakpoints ($\bar{J}_n$), the sample standard error of $\bar{J}_n$, and the maximum and minimum J_n values in the sample. Three basic results can be summarized:

(1) Growth in J_n is between n and $n \log n$ over the range of the experiment.

(2) The behavior is very nearly independent of specific distributional assumptions if distributions are smooth.

(3) The maximum number of breakpoints in the Monte Carlo sample does not grow substantially faster than the mean number.

Result (2) is the most surprising one. Not only is the pattern of mean values the same over the various distributional assumptions, but the actual values are almost identical. For example, the ratio of the mean for any distributional pair to the corresponding mean for normal assumptions ranged from .96 to 1.04 over all n and p values. That is, differences were generally only in the third significant figure. Given this agreement over distributions, detailed analysis is presented below only for the case where both x and Y are normal.

Result (3) is very reassuring. Not only does the maximum grow at roughly the same rate as the mean, but it is rarely substantially greater. This probably reflects the empirical result that the standard deviation of J_n appears to grow no faster than rate $n \log n$. In particular, over the entire experimental range, the ratio of the maximum to the mean was never more than 1.81, and was generally less than 1.2. The largest ratio occurs for $n=20$, $p=10$ when x is normal and $Y \sim t_4$ (when the mean is actually less than n). For $n=50$, 100, 200, or 400, the ratio was always less than 1.39. Furthermore, in the entire experimental range, the maximum number of breakpoints was never more than $2.11n$. Thus an upper bound of $3n$ (as suggested by Koenker and d'Orey (1987)) appears to be sufficiently conservative for practical work, particularly if extra randomization is added ("dithering").

Detailed weighted regression analyses were performed when both x and Y were normal. A SUN version of S-plus distributed by Statistical Sciences, Inc. (see Becker and Chambers (1984)) was used to regress the mean number of breakpoints against both n and $n \log n$ separately for $p=2, 5$, and 10, where weights were inversely proportional to the square of Monte Carlo standard errors. For $p=2$, the means were extremely linear in n (see Fig. 4.1). The linear coefficient was 1.125 (with t-value 148) and the coefficient of $n \log n$ was .2 ($t=15$). Note however that the residuals show a slightly convex pattern, and I believe that the $n \log n$ behavior will become more

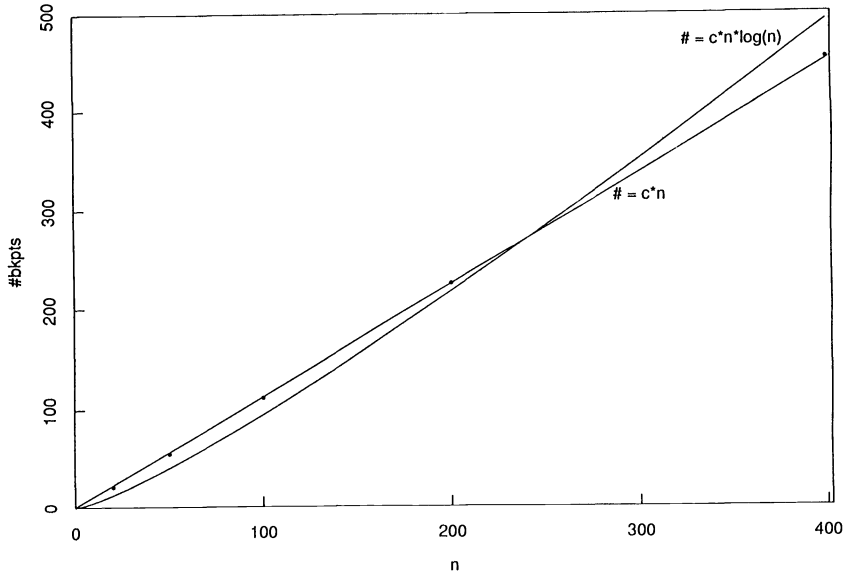


FIG. 4.1. *Number of breakpoints versus n for* $p = 2$.

evident for n somewhat larger. For $p = 5$ and $p = 10$, the growth was between n and $n \log n$ (see Figs. 4.2 and 4.3). Clear nonlinearity is apparent, particularly for large n. For $p = 5$, the coefficient of n is 1.50 ($t = 31$) and that for $n \log n$ is .28 ($t = 19$). For $p = 10$, the coefficient of n is 1.84 ($t = 10$) while that for $n \log n$ is .36 ($t = 27$). Note that the simulation standard error is somewhat smaller than the size of the "•" indicating observations in the figures. It seems clear that EJ_n grows roughly linearly for small n and as $n \log n$ for larger n, showing remarkable agreement with theoretical results.

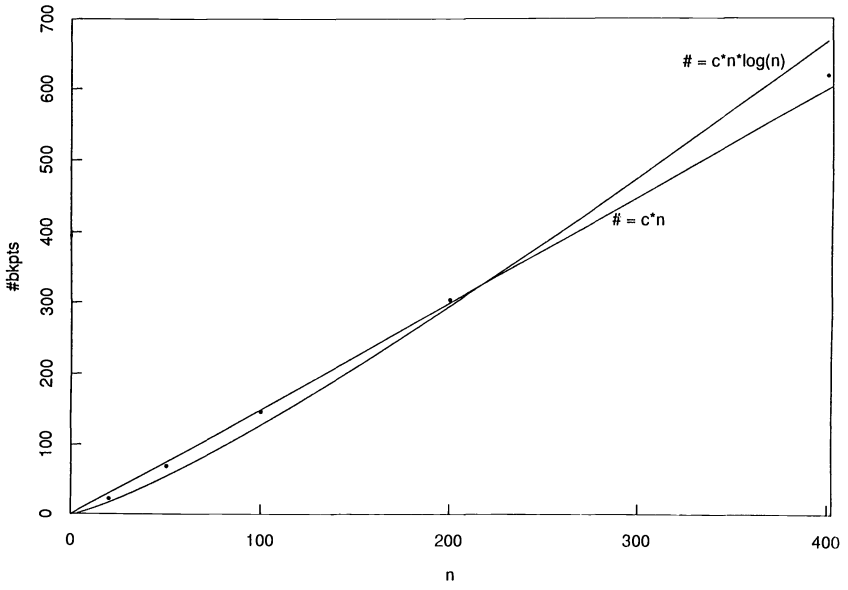


FIG. 4.2. *Number of breakpoints versus n for* $p = 5$.

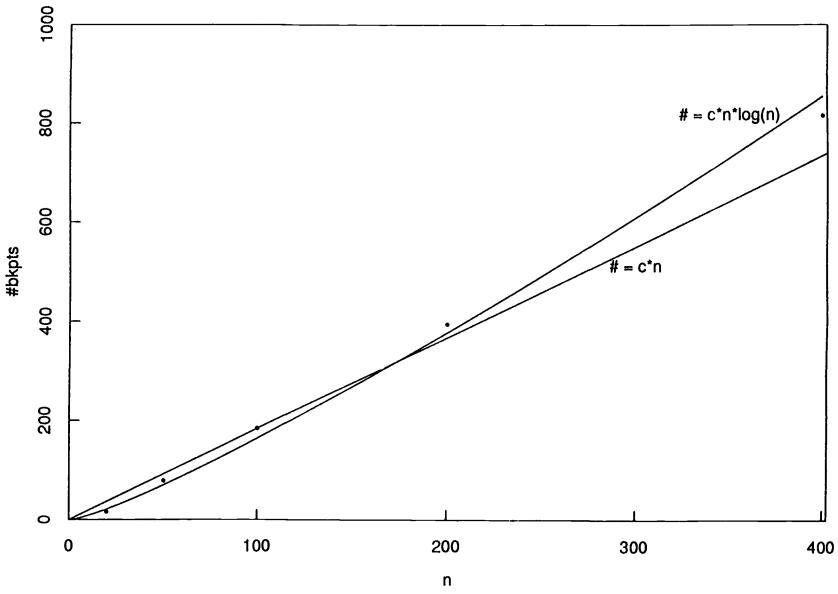


FIG. 4.3. *Number of breakpoints versus n for p = 10.*

REFERENCES

I. BARRODALE AND F. D. K. ROBERTS (1974), *Solution of an overdetermined system of equations in the l_1 norm*, Commun. Assoc. Comput. Mach., 17, pp. 319-320.

G. W. BASSETT AND R. W. KOENKER (1982), *An empirical quantile function for linear models with iid errors*, J. Amer. Statist. Assoc., 77, pp. 407-415.

R. A. BECKER AND J. M. CHAMBERS (1984), *S: An Interactive Environment for Data Analysis and Graphics*, Wadsworth, Belmont, CA.

R. N. BHATTACHARYA AND R. RANGA RAO (1976), *Normal Approximation and Asymptotic Expansions*, John Wiley, New York.

C. BLAIR (1986), *Random linear programs with many variables and few constraints*, Math. Programming, 34, pp. 62-71.

K. H. BORGWARDT (1987), *The Simplex Method: A Probabilistic Analysis*, Springer-Verlag, Berlin.

W. FELLER (1966), *An Introduction to Probability Theory and Its Applications*, II, John Wiley, New York, pp. 504-515.

T. GAL (1979), *Postoptimal Analysis, Parametric Programming, and Related Topics*, John Wiley, New York.

R. W. KOENKER (1987), *A comparison of asymptotic testing methods for l_1 regression*, in Statistical Data Analysis Based on the L_1 Norm and Related Methods, Y. Dodge, ed., North Holland, Amsterdam, pp. 287-295.

R. W. KOENKER AND G. W. BASSETT (1978), *Regression quantiles*, Econometrica, 46, pp. 33-50.

R. W. KOENKER AND D'OREY (1987), *Computing regression quantiles*, Appl. Statist., 36, pp. 383-393.

R. W. KOENKER AND S. PORTNOY (1987), *L-estimation for the linear model*, J. Amer. Statist. Assoc., 82, pp. 851-857.

M. B. MUKHIN (1984), *Local limit theorems for distributions of sums of independent random vectors*, Theory Probab. Appl., 29, pp. 369-375.

G. K. MURTY (1980), *Computational complexity of parametric linear programming*, Math. Programming, 19, pp. 213-219.

S. PORTNOY (1984), *Tightness of the sequence of* c.d.f. *processes defined from regression fractiles*, Robust and Nonlinear Time Series Analysis, Franke, Härdle, and Martin, eds., Springer-Verlag, New York, pp. 231-246.

——— (1987), *Using regression quantiles to identify outliers*, Statistical Data Analysis Based on the L_1 Norm and Related Methods, Y. Dodge, ed., North Holland, Amsterdam, pp. 345-356.

S. PORTNOY AND R. W. KOENKER (1987), *Adaptive L-estimation of linear models*, Ann. Statist., 17, pp. 362-381.

D. RUPPERT AND R. J. CARROLL (1980), *Trimmed least squares estimation in the linear model*, J. Amer. Statist. Assoc., 75, pp. 828–838.

R. SHAMIR (1984), *On the efficiency of the simplex method*, Ph.D. thesis, Department of Industrial Engineering and Operational Research, University of California, Berkeley, CA.

S. SMALE (1983), *On the average speed of the simplex method of linear programming*, Math. Programming, 27, pp. 241–262.

S. STIGLER (1986), *The History of Statistics*, Harvard University Press, Cambridge MA.

C. STONE (1965), *A local limit theorem for nonlattice multidimensional distribution functions*, Ann. Math. Statist., 36, pp. 546–551.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 884-910, July 1991
© 1991 Society for Industrial and Applied Mathematics
011

A NUMERICAL STUDY OF AN AUGMENTED LAGRANGIAN METHOD FOR THE ESTIMATION OF PARAMETERS IN ELLIPTIC SYSTEMS*

K. ITO†, M. KROLLER‡, AND K. KUNISCH§

Abstract. A method based on an augmented Lagrangian formulation is developed which allows coefficients in an elliptic differential equation to be estimated from measurements of the state. This is a hybrid method combining the output-least-squares and the equation-error technique. Seminorm regularization is employed, and convergence and stability properties are discussed. Several aspects of an efficient implementation are described. Finally, the effectiveness of the method is demonstrated by means of one- and two-dimensional examples.

Key words. augmented Lagrangian method, inverse problems in elliptic equations, numerical studies

AMS(MOS) subject classifications. 49D29, 65K05

1. Introduction. In this paper we discuss our numerical experience with the hybrid method [IK2] based on the augmented Lagrangian formulation for estimating the coefficient q in the elliptic equation

$$\begin{aligned} -\operatorname{div}(q \operatorname{grad} u) &= f \quad \text{in } \Omega, \\ u|_\Gamma &= 0 \end{aligned} \tag{1.1}$$

from a measurement z of the solution $u(\hat{q})$ which corresponds to the "true" coefficient $\hat{q}$. Here $f \in H^{-1}$ is given, Ω is a bounded open set in $\Re^n$, $n = 1, 2$, or 3 with piecewise smooth boundary Γ, and q is a scalar valued function of the spatial variable x. Frequently $u(\hat{q})$ is not known exactly; this may be due to model and/or measurement error and also to the fact that measurements may be known only at discrete points x_i in the domain Ω so that z is constructed by interpolation of pointwise data. A motivation for developing the hybrid method is to combine the output least squares (OLS) method and the equation error method [C] into one algorithm while retaining the favorable properties of both. We recall the OLS and the equation error formulation for the problem under consideration. Let X be a Hilbert space such that the embedding of X into L^∞ is compact. Then, the OLS-formulation of the estimation of q in (1.1) can be stated, for instance, as the minimization problem in X:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}|u(q) - z|^2_{H_0^1} + \frac{\beta}{2} N(q) \\ &\text{over} \quad Q_{\text{ad}} = Q \cap \{q \in L^\infty : q \geqq \alpha > 0\}, \end{aligned} \tag{1.2}$$

where Q is a closed, bounded, convex subset of X, $\beta \geqq 0$ and $\alpha > 0$ are a priori chosen constants, and $u(q) \in H_0^1$ denotes the solution of (1.1) for given $q \in Q_{\text{ad}}$. Here $N(q)$

*Received by the editors March 13, 1989; accepted for publication (in revised form) July 27, 1990.

†Center for Applied Mathematical Sciences, Department of Mathematics, University of Southern California, Los Angeles, California 90089-1113. The research of this author was supported in part by the Air Force Office of Scientific Research under contract F49620-86-C-0111, and by the National Science Foundation under grant UINT-8521208 and DMS-8818530.

‡Institut für Mathematik, Universität Graz, Elisabethstrasse 12, 8010 Graz, Austria. The research of this author was supported in part by the Fonds zur Foerderung der wissenschaftlichen Forschung under grants S3206 and P6005.

§Institut für Mathematik, Technische Universität Graz, Kopernikusgasse 24, 8010 Graz, Austria. The research of this author was supported in part by the Fonds zur Foerderung der wissenschaftlichen Forschung under grants S3206 and P6005.

denotes the square of a seminorm X. The conditions on Q_{ad} guarantee that (1.2) has a solution, provided that $Q_{ad} \neq 0$. The OLS approach is used most frequently for the estimation of coefficients in partial differential equations and for many classes of problems it is successful. Apparently these do not include the case of estimating q in (1.1) for $n \geqq 2$. Our own implementation was not successful and we are not aware of any work where the OLS formulation has been used successfully for estimating a variable coefficient q in (1.1) when $n \geqq 2$.

In the equation error approach of estimating q in (1.1) from knowledge of z we replace u in (1.1) by z. This results in a hyperbolic equation for q, which is singular at the zeros of ∇z. Again we can choose an optimization-theoretic formulation, which is given by

$$\text{minimize} \quad |\nabla(q\nabla z)+f|_H^2 \quad \text{over} \quad q \geqq \alpha, \tag{1.3}$$

where $H = H^0$ or $H = H^{-1}$, for example. Due to possible zeros of ∇z, the cost functions in (1.3) are not radially unbounded, and further constraints on q or a regularization term have to be added to (1.3) to guarantee existence of a solution of (1.3). We observe that (1.3) is quadratic in q and that it requires differentiation of possibly error-corrupted data.

The primary purpose of this work is to establish a method that allows us to determine a spatially varying diffusion coefficient q in (1.1) from data z in a stable way. While the method that we propose has its specific form when applied to (1.1), it is of general nature and can be applied to a wide class of estimation problems for partial differential equations, as for instance to the estimation of a temporally and spatially varying diffusion coefficient in a parabolic partial differential equation [KP]. The estimation of q in (1.1) is of particular interest in groundwater flow modelling [Y], where u represents the pressure of the fluid, and q is the porosity of the aquifer. The algorithm that we purpose is currently being tested with real data available for piezometric head measurements at discrete locations within a rather complicated domain Ω. Currently the method that is proposed in this paper is also investigated for a problem arising in impedance computed tomography [J], which in our notation consists of determining q from measurements of u on the boundary.

The augmented Lagrangian formulation that we propose, just as the equation error method, reduces the parameter estimation problem to a sequence of quadratic programming problems and, like the output least squares method, it is versatile with regards to availability of observations as will be demonstrated in § 4. The method is based upon formulating the problem as a constrained and regularized minimization problem (P^β) for *two* independent variables $(q, u) \in H^2(\Omega) \times H_0^1(\Omega)$. Let us consider the following:

$$(P^\beta) \qquad \begin{aligned} &\text{minimize} \quad \frac{1}{2}|u-z|_{H_0^1}^2+\frac{\beta}{2}N(q) \\ &\text{subject to} \quad e(q,u)=(-\Delta)^{-1}(\operatorname{div}(q \operatorname{grad} u)+f)=0 \\ &\qquad\qquad q \in Q \cap \{q \in L^\infty(\Omega): q(x) \geqq \alpha > 0\}, \end{aligned}$$

where we assume that $f \in H^{-1}(\Omega)$ and $z \in H_0^1(\Omega)$, and where Δ denotes the Laplacian with Dirichlet boundary condition as an operator from $H_0^1(\Omega)$ onto $H^{-1}(\Omega)$. The problems (P^β) will be solved by an augmented Lagrangian approach to incorporate the infinite-dimensional equality constraint $e(q, u) = 0$, which describes the partial differential equation. In the case of one-dimensional domain, $(q, u) \in H^2(\Omega) \times H_0^1(\Omega)$ can be replaced by $(q, u) \in H^1(\Omega) \times H_0^1(\Omega)$.

It will be shown in § 2 that there exists a Lagrange multiplier $\lambda^* \in H_0^1(\Omega)$ associated with the equality constraint $e(q, u) = 0$, which is characterized by

$$\langle q^* \nabla \lambda^*, \nabla \psi \rangle + \langle \nabla u^* - \nabla z, \nabla \psi \rangle = 0 \quad \text{for all } \psi \in H_0^1, \tag{1.4}$$

where (q_β^*, u_β^*) denotes a solution of (P^β). If an augmentability condition holds at (q_β^*, u_β^*), then there exists $c_0 > 0$ such that for all $c \geqq c_0$, (q_β^*, u_β^*) is also a solution of

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}|u - z|_{H_0^1}^2 + \frac{\beta}{2} N(q) + \langle \lambda^*, e(q, u) \rangle_{H_0^1} + \frac{c}{2}|e(q, u)|_{H_0^1}^2, \\ &\text{subject to} \quad (q, u) \in Q_{\text{ad}} \times H_0^1. \end{aligned} \tag{1.5}$$

In particular, the solution (q_β^*, u_β^*) of (P^β) can also be realized as a solution of an optimization problem, where the equality constraint is eliminated. Unlike the penalty technique, it is not required that c become arbitrarily large. In (1.5) the Lagrange multiplier λ^* is unknown. It is determined iteratively as part of the augmented Lagrangian algorithm, which involves a sequence of minimizations of the functionals

$$\begin{aligned} &L_{c_k}(q, u, \lambda^k) = \frac{1}{2}|u - z|_{H_0^1}^2 + \frac{\beta}{2} N(q) + \langle \lambda^k, e(q, u) \rangle_{H_0^1} + \frac{c_k}{2}|e(q, u)|_{H_0^1}^2 \\ &\text{subject to} \quad q \in Q_{\text{ad}}, \end{aligned} \tag{1.6}$$

where the multiplier sequence $\{\lambda^k\}$ in $H_0^1(\Omega)$ is generated by the updating rule

$$\lambda^{k+1} = \lambda^k + (c_k - c_0) e(q_k, u_k), \tag{1.7}$$

$k = 1, 2, \cdots$, and $c_0 > 0$. Here the pair (q_k, u_k) is a minimizer of $L_{c_k}(\cdot, \cdot, \lambda_k)$. To carry out this iterative scheme a sequence of monotonically nondecreasing, positive, real numbers $\{c_k\}_{k=1}^\infty$, $c_1 > c_0$ and a startup value $\lambda^1 \in H_0^1(\Omega)$ for the Lagrange multiplier for the equality constraint $e(q, u) = 0$ need to be chosen. If the data z were free of error and if no regularization was used ($\beta = 0$), then we would have $u(q_\beta^*) = z$ and in view of (1.4) this would lead to $\lambda^* = 0$. We therefore suggest as a startup value for λ, the choice $\lambda^1 = 0$, but convergence will be guaranteed for any other choice of λ^1 as well. The hybrid property of our algorithm is now evident since the term $c_k/2|e(q, u)|_{H_0^1}^2$ involves the equation error

$$|\text{div}\,(q \,\text{grad}\, u) + f|_{H^{-1}}^2$$

and the term $\frac{1}{2}|u - z|_{H_0^1}^2$ represents an output least squares criterion. We shall return to this point in § 2.

A number of remarks are in order (see also § 2 for further discussion).

(1) Since u is uniquely determined from q through the equality $e(q, u) = 0$, the objective functional can be considered to be dependent on q only. In this way (P^β) is reduced to the regularized least squares method (1.2) given by

$$\text{minimize} \quad |u(q) - z|_{H_0^1}^2 + \frac{\beta}{2} N(q),$$

where $u(q)$ satisfies

$$-\text{div}\,(q \,\text{grad}\, u(q)) = f \quad \text{and} \quad q \in Q_{\text{ad}}.$$

From a theoretical point of view, (P^β) and the least squares method are equivalent. From the point of view of numerical optimization, these two formulations are very different. This is due to the fact that in (P^β), q and u are treated as independent variables. In the OLS-formulation the pair $(q^k, u^k) = (q^k, u(q^k))$ at any stage of the iteration of an algorithm will satisfy (1.1) (feasibility). This is not the case for the

implementation of (P^β) by the augmented Lagrangian technique, where the iterates (q_k, u_k) satisfy $e(q, u) = 0$ only in the limit.

(2) The pointwise constraint $q(x) \geqq \alpha$ guarantees the strong ellipticity and the boundedness of Q in $X \subset L^\infty$ allows us to argue the existence of solutions of (P^β), $\beta \geqq 0$. Without the boundedness of Q, (P^β) may not have a solution in general.

(3) In [IK2] we studied the cases $X =$ finite-dimensional, and $X = H^1(\Omega)$ with $N(q) = |q_x|^2_{L^2}$ if $n = 1$, and $X = H^2(\Omega)$ with

$$N(q) = |\nabla q|^2_{L^2} + \sum_{1 \leqq i,j \leqq n} |q_{x_i x_j}|^2_{L^2} \tag{1.8}$$

for $n = 2$, $n = 3$. For an appropriate choice of $\beta \geqq 0$ the use of a regularization term in (P^β) induces the continuity of the mapping from the observation $z \in H^1_0$ to the solution $(q^*_\beta(z), u^*_\beta(z))$ of (P^β) (see [CK1], [CK2]). In general β cannot be taken equal to zero if continuous dependence of the solutions of (P^β) on z is desired.

(4) The theory which will be developed in § 2 (see also [IK2]) requires that X is compactly embedded into $L^\infty(\Omega)$ and the functional $N(q)$ appearing (P^β) defines the square of a seminorm which basically bounds the L^∞-norm. For the one-dimensional case the choice of X and $N(q)$ described in remark (3) satisfies this requirement, and the discretized parameter space $\mathcal{H}^m$ by piecewise linear elements (see § 3 for the details) is in X for each m. For multidimensional ($n = 2$ or 3) case, a possible choice of X may be a finite-dimensional subspace of $L^\infty(\Omega)$ which involves a priori parameterization of X like the space of piecewise constant functions defined on a fixed grid, as in Type III of § 3. But it is by no means satisfactory in order to estimate pure-functional coefficients. As described in remark (3), X can be taken as $H^2(\Omega)$. The numerical realization of the algorithm requires a discretization of $H^2(\Omega)$. In our calculations we did not choose the subspaces $\mathcal{H}^m$, in which the approximations q^m to q lie, in $H^2(\Omega)$. Rather we experimented with the following three choices of $\mathcal{H}^m$: piecewise linear tensor splines, piecewise linear basis functions on a triangular grid, and piecewise constant functions where $\Omega = (0, 1) \times (0, 1)$. Thus the theoretical smoothness requirement is not satisfied. Even though the theory in § 2 applies for $X = \mathcal{H}^m$ with appropriate choice of $N(q)$ for fixed m, since $\mathcal{H}^m$ is embedded compactly in $L^\infty(\Omega)$, convergence property of q^m as $m \to \infty$ should be analyzed in $H^2(\Omega)$. In the light of the successful computations it is in our interest to fill this gap between the theory and numerical implementations. The choice of the spaces $\mathcal{H}^m$ as above is motivated by our interest in finding out whether the proposed algorithm would also be capable of identifying discontinuous coefficients and, in the long run, of detecting cracks.

2. Theoretical considerations. In this section we will discuss some of the theoretical issues with regard to estimation problems in elliptic partial differential equations and their solution by means of the augmented Lagrangian technique.

2.1. Convergence of algorithm. First we address the question of existence of solutions to (P^β).

LEMMA 2.1 (existence of solutions to (P^β), $\beta \geqq 0$). *There exists at least one solution pair* $(q^*_\beta, u^*_\beta) \in Q_{\text{ad}} \times H^1_0$ *of* (P^β), $\beta \geqq 0$.

Proof. Since the performance index is bounded below by zero, there exists a minimizing sequence $(q^n, u^n) \in Q_{\text{ad}} \times H^1_0$. Due to boundedness of Q in X, there exist a weakly convergent subsequence $\{q^{n_k}\}$ and $q^*_\beta \in X$ such that $q^{n_k} \to q^*_\beta \in X$. Moreover, Q_{ad} is closed and convex in X and therefore $q^*_\beta \in Q_{\text{ad}}$. Since X is embedded compactly into L^∞ we find $q^{n_k} \to q^*_\beta$ in L^∞. Note that the solution map $q \in L^\infty \to u(q) \in H^1_0$ of (1.1) is continuous, that $\ker(N)$ is a closed linear subspace of X, and that N defines a

norm on the quotient space $X/\ker(N)$. Since norms are weakly sequentially lower semicontinuous,

$$\liminf_{k\to\infty} J(q^{n_k}) \geqq J(q^\beta),$$

where

$$J(q)=\frac{1}{2}|u(q)-z|^2_{H_0^1}+\frac{\beta}{2}N(q),$$

and hence $(q^*_\beta, u^*_\beta)=(q^*_\beta, u(q^*_\beta))$ is a solution of (P^β).

The solutions (q^*_β, u^*_β) of (P^β) depend on β. This dependence is indicated only when it is required by the context. Next, we establish the first-order necessary condition for a solution (q^*_β, u^*_β) of (P^β), $\beta \geqq 0$.

LEMMA 2.2 (first-order necessary condition). *There exists a Lagrange multiplier* $\lambda^* \in H_0^1(\Omega)$ *such that for the Lagrange functional*

$$L(q, u; \lambda^*)=\frac{1}{2}|u-z|^2_{H_0^1}+\frac{\beta}{2}N(q)+\langle\lambda^*, e(q, u)\rangle_{H_0^1},$$

we have

$$\nabla L(q^*_\beta, u^*_\beta; \lambda^*)(q-q^*_\beta, v)\geqq 0 \tag{2.1}$$

for all $v\in H_0^1$ *and* $q\in Q_{\text{ad}}$.

Proof. First of all, note that Q_{ad} defines a closed convex set in X. We use the constraint qualification condition in [ZK]. Let

$$E(u, v)=\nabla e(h, v)=-\nabla\cdot(h\nabla u^*_\beta+q^*_\beta\nabla v)\in H^{-1}.$$

Then we need to show the regular point condition

$$0\in \operatorname{int}\{E(h, v)\in H^{-1}: h\in Q_{\text{ad}}-q^*_\beta \text{ and } v\in H_0^1\}.$$

This condition holds, since the map $v\in H_0^1\to -\nabla\cdot(q^*_\beta\nabla v)\in H^{-1}$ is onto, and hence there exists a Lagrange multiplier λ^* satisfying (2.1).

COROLLARY 2.3. *The Lagrange multiplier* λ^* *satisfies*

$$\langle q^*_\beta\nabla\lambda^*, \nabla\psi\rangle+\langle\nabla u^*_\beta-\nabla z, \nabla\psi\rangle=0 \tag{2.2}$$

for all $\psi\in H_0^1$.

Proof. If we set $q=q^*_\beta$ in (2.1), we obtain

$$\langle q^*_\beta\nabla\lambda^*, \nabla v\rangle+\langle\nabla u^*_\beta-\nabla z, \nabla v\rangle\geqq 0$$

for all $v\in H_0^1$. Thus, the corollary follows from the fact that with $v\in H_0^1$ also $-v\in H_0^1$.

Next, we argue the augmentability condition. We assume that $N(q)$ is a nonnegative quadratic form on X such that $\ker(N)=\{1\}$ and that N defines a norm on the quotient space $X/\mathcal{R}=\{h\in X: \langle 1, h\rangle_X=0\}$ that is equivalent to the X-norm. Thus, there exists a positive constant k_1 such that

$$|h|_{L^\infty}\leqq k_1\sqrt{N(h)} \quad \text{for all } h\in X/\mathcal{R}. \tag{2.3}$$

Since Q is bounded in X, there exists a positive constant k_2 such that

$$|q|_{L^\infty}\leqq k_2 \quad \text{for all } q\in Q.$$

PROPOSITION 2.4. *Define the augmented Lagrange functional* J_c, $c\geqq 0$:

$$J_c(q, u; \lambda^*)=L(q, u; \lambda^*)+\frac{c}{2}|e(q, u)|^2_{H_0^1},$$

where e is defined in (P^β) and λ^* satisfies (2.2). *Then, there exist a nontrivial interval I in R^+ and positive constants c_0 and σ such that if $c \geqq c_0$ and $\beta \in I$,*

$$J_c(q, u; \lambda^*) \geqq J_c(q_\beta^*, u_\beta^*; \lambda^*) + \sigma(|q - q_\beta^*|_X^2 + |u - u_\beta^*|^2) \tag{2.4}$$

for all $(q, u) \in Q_{\mathrm{ad}} \times H_0^1$, provided the following condition is satisfied for some $\beta^0 > 0$:

$$|u_0^* - z|_{H_0^1}^2 \leqq \beta^0 \left[\frac{2\alpha^2}{5k_1^2} \left(1 - \frac{4}{5} \frac{k_2^4}{k_1^2 |f|_{H^{-1}}^2 \beta^0} \right) - (N(q_0^*) - N(q_{\beta^0}^*)) \right]. \tag{2.5}$$

Here (q_0^*, u_0^*) is a minimum norm solution of (P^0) satisfying $N(q_0) \leqq N(q)$ for all solutions (q, u) of (P^0).

Condition (2.5) is satisfied, for example, if z is attainable, i.e., if there exists $q_0^* \in Q_{\mathrm{ad}}$ such that $u(q_0^*) = z$. The proof of Proposition 2.4 is given in the Appendix.

COROLLARY 2.5. *Under the condition of Proposition* 2.4, *(q_β^*, u_β^*) with $\beta \in I$ is a solution to the minimization problem*

$$\textit{minimize} \quad J_c(q, u; \lambda^*) \qquad \textit{over} \quad (q, u) \in Q_{\mathrm{ad}} \times H_0^1.$$

Remark 2.6. In [IK2], we studied among others the case when $n = 2$ or 3, $X = H^2(\Omega)$, and $N(q) = |q|_{H^2}^2 - |q|_{L^2}^2$, and established results corresponding to Proposition 2.4 and Corollary 2.5. However, the results on [IK2] are of a local nature, requiring, in addition, that (q, u) is in a neighbourhood of (q_β^*, u_β^*).

Now, we state the convergence result for the augmented Lagrangian method. We recall the definition of functional

$$L_c(q, u, \lambda) = \frac{1}{2}|u - z|_{H_0^1}^2 + \frac{\beta}{2} N(q) + \langle \lambda, e(q, u) \rangle_{H_0^1} + \frac{c}{2} |e(q, u)|_{H_0^1}^2. \tag{2.6}$$

THEOREM 2.7. *Assume that* (2.4) *holds for $\beta \in I$, and let $\{c_k\}_{k=1}^\infty$, a monotone nondecreasing sequence with $c_1 > c_0$, and $\lambda^1 \in H_0^1$ be given. Suppose that for $k \geqq 1$, $(q_k, u_k) \in Q_{\mathrm{ad}} \times H_0^1$ satisfies*

$$L_{c_k}(q_k, u_k; \lambda^k) \leqq L_{c_k}(q_\beta^*, u_\beta^*; \lambda^k) = J(q_\beta^*, u_\beta^*) \tag{2.7}$$

and let the sequence $\{\lambda^k\}$ in H_0^1 be given by

$$\lambda^{k+1} = \lambda^k + (c_k - c_0) e(q_k, u_k).$$

Then we find

$$\sigma(|q_k - q_\beta^*|_X^2 + |u_k - u_\beta^*|_{H_0^1}^2) + \frac{1}{2\hat{c}_k} |\lambda^{k+1} - \lambda^*|_{H_0^1}^2 \leqq \frac{1}{2\hat{c}_k} |\lambda^k - \lambda^*|_{H_0^1}^2$$

where $\hat{c}_k = c_k - c_0$ and $\sigma > 0$ is defined in Proposition 2.4. *In particular, this implies that*

$$|\lambda^k - \lambda^*|_{H_0^1} \leqq |\lambda^1 - \lambda^*|_{H_0^1} \quad \text{for all } k \geqq 1,$$

and

$$\sum_{k=1}^\infty \sigma(|q_k - q_\beta^*|_X^2 + |u_k - u_\beta^*|_{H_0^1}^2) \leqq \frac{1}{2\hat{c}_1} |\lambda^1 - \lambda^*|_{H_0^1}^2.$$

Proof. For every $(q, u) \in X \times H_0^1$, $\lambda \in H_0^1$ and $c > c_0$. We have

$$L_c(q, u; \lambda) = K_{c_0}(q, u; \lambda^*) + \langle e(q, u), \lambda - \lambda^* \rangle_{H_0^1} + \frac{1}{2}(c - c_0)|e(q, u)|_{H_0^1}^2$$

$$= J_{c_0}(q, u; \lambda^*) + \frac{1}{2(c - c_0)} |\lambda + (c - c_0) e(q, u) - \lambda^*|_{H_0^1}^2 - \frac{1}{2(c - c_0)} |\lambda - \lambda^*|_{H_0^1}^2.$$

Thus, the theorem follows from (2.4) and from this equality with $(q, u, \lambda) = (q_k, u_k, \lambda^k)$, $c = c_k$.

We observe that (q_k, u_k) satisfies (2.7), for example, if it is a solution of

$$\text{minimize} \quad L_{c_k}(q, u; \lambda_k) \qquad \text{over} \quad (q, u) \in Q_{\text{ad}} \times H_0^1.$$

2.2. Seminorm regularization. The use of the regularization term $\beta N(q)$ in (P^β) is common in solving ill-posed inverse problems. In [CK1], [CK2], and [KS], for example, the stability of the solutions to the regularized output least squares method with respect to perturbations in the problem data in (P^β) is studied. We point out the fact that the regularization term that we propose here is only a seminorm on X. In practice this amounts to regularization norms that involve only derivatives of q. If the L_2-norm of q is included in the regularization term, we normally obtain underestimated solutions.

2.3. Identifiability and singular set. Convergence of (q_β^*, u_β^*) to a minimum norm solution of (P^0) (that is, a solution of (P^0) that minimizes $N(q)$) as $\beta \to 0+$ was studied in [IK2] and [EKN]. Uniqueness of the solutions of (P^0) and (P^β), $\beta > 0$, cannot be guaranteed in general. We mention here a related result concerning the injectivity of the mapping $q \to u(q)$ and refer to [K] for further discussion of these matters. Injectivity of $q \to u(q)$ is a necessary but not a sufficient condition to obtain the continuity of the inverse of the map $q \to u(q)$. Let $\hat{q}$ and q satisfy the constraints in (P^β) where $\hat{q}$ is the "true" reference coefficient and q is a perturbed one. Let us denote by $u(\hat{q})$ and $u(q)$ the corresponding solutions. Then we obtain the estimate [K],

$$\int_\Omega |q - \hat{q}||\nabla u(\hat{q})|^2 \, dx \leqq K |u(\hat{q}) - u(q)|_{W^{2,1}(\Omega)}, \tag{2.8}$$

where the constant $K > 0$ depends on

$$|\hat{q}|_{W^{1,p}}, |q|_{W^{1,p}}, \qquad p > n \quad \text{and} \quad |u(\hat{q})|_{L^\infty}.$$

Let us define the singular set

$$S = \{x \in \bar{\Omega}, \nabla u(\hat{q})(x) = 0\}. \tag{2.9}$$

Clearly, if meas $(S) = 0$ and $u(\hat{q}) = u(q)$, then $\hat{q} = q$ almost everywhere, so that the injectivity of the map $q \to u(q)$ holds at $\hat{q}$.

2.4. Choice of the output least squares criterion. In our formulation of the augmented Lagrangian method, we use the H_0^1-norm as an output least squares criterion. Such a choice is based on the facts that (i) the H_0^1-topology for the output least squares criterion and the H^{-1}-topology for the equation error term are natural from the point of view of the second-order sufficient optimality condition for (P^β) (see [IK2] for details), (ii) the choice of these topologies also leads to a method that requires the same amounts of numerical differentiations in both the equation error and the output least squares term, and (iii) it enhances the sensitivity of the map $q \to u(q)$. In terms of computational efforts there are no differences between the use of the L_2-norm criterion and that of the H_0^1-norm criterion.

2.5. A general class of inverse problems in elliptic partial differential equations. In this paper we restricted our attention to the Dirichlet boundary value problem (1.1). Without difficulty we can modify our formulation to treat other types of boundary conditions as well. For example, consider the problem of determining $q(x)$ in

$$-\text{div}\,(q \,\text{grad}\, u) + p(x)u = f$$

with the boundary condition $q(\partial/\partial n)u = g$ on Γ, where $(\partial/\partial n)u$ denotes the outward normal derivative, $p(x) \in L^\infty$ with $p(x) \geqq w > 0$ and $g \in H^{-1/2}(\Gamma)$. In this case, we define the equation error term by

$$e(q, u) = (-\Delta + 1)^{-1}(-\operatorname{div}(q \operatorname{grad} u) + pu - f),$$

where Δ is the Laplacian with $\operatorname{dom}(\Delta) = \{u \in H^2(\Omega) \text{ and } \partial/\partial n|_\Gamma = 0\}$ and H_0^1 is replaced by H^1 in (P^β). While we are concerned with estimating the diffusion coefficient in this paper, our formulation enables us to consider the estimation problem for other coefficients in elliptic partial differential equations (e.g., $p(x)$ in the above example).

2.6. Inequality constraints. As we pointed out in the Introduction, inequality constraints (a pointwise lower bound and a norm constraint for g) are required for existence of solutions to (P^β). In our calculations, we ignored the pointwise lower bound without harm. The optimization problems at the iterative stages of Algorithm 1 and 2 in the next section do not require this constraint to guarantee existence of solution (q_k^m, u_k^n). The relevance of the norm constraint was investigated for the one-dimensional case in [KK1]. It was observed that its use improved the results if the interior of the singular set $S = \{x \in [0, 1], u_x(\hat{q}) = 0\}$ is not empty and it did not change them significantly if meas $S = 0$, provided, of course, that γ is chosen such that $|\hat{q}|_{H^1(0,1)} \leqq \gamma$. We did not use the norm constraint in our implementation for the case $\Omega = [0, 1] \times [0, 1]$ since we do not expect that any new phenomena would arise. Moreover, the regularization term also provides a seminorm bound for the solutions.

3. Algorithm and implementation. We carried out our computations for the choices $\Omega = [0, 1]$ and $\Omega = [0, 1] \times [0, 1]$. For the computations, we discretized the problem using the finite element method [AB]; i.e., we chose approximations q^m and u^m to q and u in finite-dimensional subspaces $\mathcal{H}^m$ and V^n in the form

$$q^m(x) = \sum_{i=1}^{m} q_i^m \psi_i^m(x) \in \mathcal{H}^m, \tag{3.1}$$

$$u^n(x) = \sum_{j=1}^{n} u_j^n \phi_j^n(x) \in V^n, \tag{3.2}$$

where $\phi_i^n \in H_0^1(\Omega)$ and $\psi_i^m \in L^\infty(\Omega)$. We used the following types of basis functions for ψ_i^m and ϕ_i^n in the two-dimensional case.

Type I. The functions ψ_i^m and ϕ_i^n are piecewise linear tensor splines [Sch]; i.e., $m = (M+1)^2$

$$\psi_{j(M+1)+i+1}^m = B_i^M(x_1) B_j^M(x_2) \quad \text{for } i, j = 0, \cdots, M$$

and $n = (N-1)^2$

$$\phi_{(j-1)(N-1)+i}^n = B_i^N(x_1) B_j^N(x_2) \quad \text{for } i, j = 1, \cdots, N-1$$

where

$$B_i^N(x) = \begin{cases} N\left(x - \dfrac{i-1}{N}\right) & \text{on } \left[\dfrac{i-1}{N}, \dfrac{i}{N}\right], \\ N\left(\dfrac{i-1}{N} - x\right) & \text{on } \left[\dfrac{i}{N}, \dfrac{i+1}{N}\right], \\ 0 & \text{elsewhere.} \end{cases}$$

Type II. The functions ψ_i^m and ϕ_i^n are piecewise linear basis functions on triangular elements with nodes at $(i/M, j/M)$, $i, j = 0, \cdots, M$ and nodes at $(i/N, j/N)$, $i, j = 1, \cdots, N-1$, respectively, where $m = (M+1)^2$ and $n = (N-1)^2$.

Type III. The functions ϕ_i^n are the same as in Type II and the functions ψ_i^m are piecewise constant; i.e.,

$$\psi^m_{(j-1)(M+i)} = \chi_{((i-1)/M,i/M)}(x_1)\chi_{((j-1)/M,j/M)}(x_2) \quad \text{for } i,j = 1,\cdots,M,$$

where $\chi_{(a,b)}$ is the indicator function of the interval (a, b) and $m = M^2$. In the one-dimensional case we used linear splines [Sch] for the approximation of both q^m and u^n.

As a regularization term we took $\int_0^1 |q_x|^2\,dx$ in case of dimension one and

$$N(q) = \int_0^1 \int_0^1 (|q_{x_1}|^2 + |q_{x_2}|^2)\,dx_1\,dx_2, \tag{3.3}$$

for dimension two.

This choice of seminorm will guarantee (2.3) when X is finite-dimensional, for example, $X = \mathcal{H}^m$ for fixed m. In dimension one, the choice $X = H^1$ implies $X \in L^\infty$ and the seminorm (3.3) will satisfy (2.3). In the case of dimensions two and three the second-order terms would have to be added to the seminorm in (3.3) to guarantee that (2.3) holds for the choice $X = H^2 \subset L^\infty$. Our choice of seminorm in (2.3) is directly applicable to basis elements of Type I and II, and it requires finite-difference approximations to the derivative terms in (2.3) for Type III basis elements.

With these specifications made, the description of (P^β), $\beta \geqq 0$, is given by

$$(P_\beta^{m,n}) \qquad \text{minimize} \quad J^{m,n}(\underline{q}^m, \underline{u}^n) = \frac{1}{2}\underline{u}^{n^T} H \underline{u}^n + \underline{u}^{n^T}\underline{b} + \frac{1}{2}|z|^2_{H_0^1} + \frac{\beta}{2}\underline{q}^{m^T} W \underline{q}^m$$

$$\text{subject to} \quad \tilde{e}_i^{m,n}(\underline{q}^m, \underline{u}^n) := \langle q^m \nabla u^n, \nabla \phi_i^n\rangle - \langle f, \phi_i^n\rangle = 0, \qquad 1 \leqq i \leqq n,$$

$$q_j^m \geqq \alpha \quad \text{for } 1 \leqq j \leqq m,$$

$$\underline{q}^{m^T} \tilde{W} \underline{q}^m \leqq \gamma^2,$$

where $\underline{q}^m = \text{col}\,(q_1^m, \cdots, q_m^m) \in \mathcal{R}^m$ is the coordinate vector of $q^m(x) \in L^\infty(\Omega)$ in (3.1) and $\underline{u}^n = \text{col}\,(u_1^n, \cdots, u_n^n) \in \mathfrak{R}^n$ is the coordinate vector of $u^n(x) \in H_0^1(\Omega)$ in (3.2). The coordinate vector $\underline{e}^{m,n}$ in $\mathfrak{R}^n$ of $e^{m,n}$ is given by $\underline{e}^{m,n} = H^{-1}\tilde{\underline{e}}^{m,n}$. In what follows we will use the symbol q^m for both the element in $L^\infty(\Omega)$ and its vector coordinate in $\mathcal{R}^m$ and similarly for u^n. Moreover the bar is deleted from the notation of $\tilde{e}^{m,n}$. The matrices $H \in \mathfrak{R}^{n\times n}$ and $W \in \mathfrak{R}^{m\times m}$ and the vector $b \in \mathfrak{R}^n$ are given by

$$H_{i,j} = \langle \nabla\phi_i^n, \nabla\phi_j^n\rangle, \qquad i,j = 1,\cdots,n,$$

$$W_{i,j} = \langle \nabla\psi_i^m, \nabla\psi_j^m\rangle, \qquad i,j = 1,\cdots,m,$$

$$b_i^n = \langle \nabla\phi_i^n, \nabla z\rangle, \qquad i = 1,\cdots,n.$$

As already mentioned in § 2, we did not implement a norm bound for the vectors q^m, except in some experiments in dimension one, where we took the H^1-norm. If we were to implement an H^2-norm bound in higher dimension, then an efficient way to realize the constraint $|q|^2_{H^2} \leqq \gamma^2$ can be to require $\underline{q}^{m^T}\tilde{W}\underline{q}^m \leqq \gamma^2$, where we choose $\tilde{W}$ as the symmetric positive-definite matrix $\tilde{W}$ on $\mathfrak{R}^m$ given by

$$\tilde{W} = W^2 + W + \tilde{Q}$$

with $\tilde{Q}_{ij} = \langle \psi_i^m, \psi_j^m\rangle_{L^2}$, $i,j = 1,\cdots,m$.

The augmented Lagrangian function associated with $(P_\beta^{m,n})$, with only the equality constraint augmented, has the form

$$L_{c_k}^{m,n}(q^m, u^n, \lambda_k) = \frac{1}{2} u^{n^T} H u^n + u^{n^T} b + \frac{1}{2} |z|_{H_0^1}^2 + \frac{\beta}{2} q^{m^T} W q^m + \frac{c_k}{2} (\tilde{e}^{m,n})^T H^{-1} \tilde{e}^{m,n} + \lambda_k^{n^T} \tilde{e}^{m,n}. \tag{3.4}$$

The Lagrange multiplier $\lambda_k^n \in \mathfrak{R}^n$ is updated by

$$\lambda_{k+1}^n = \lambda_k^n + c_k H^{-1} \tilde{e}^{m,n}(q_k^m, u_k^n), \tag{3.5}$$

where $\lambda_k^n(x) = \sum_{i=1}^n (\lambda_k^n)_i \phi_i^n(x) \in H_0^1$ approximates the Lagrange multiplier $\lambda^* \in H_0^1$ for the equality constraint $e(q, u) = 0$ in (P^β). In the case of Type III basis elements for $\mathcal{H}^m$, the matrix W was approximated via a finite-difference approximation.

Next we describe two algorithms to solve $(P_\beta^{m,n})$.

ALGORITHM 1.

(1) Choose $\lambda_1 = 0$ and $\{c_k\}$ monotonically nondecreasing with c_1 sufficiently large.

(2) Set $k = 1$.

(3) Determine (q_k^m, u_k^n) from

$$\text{minimize} \quad L_{c_k}^{m,n}(q^m, u^n, \lambda_k^n) \qquad \text{subject to} \quad q^m \geqq \alpha.$$

(4) If convergence is achieved, then stop. Otherwise, put

$$\lambda_{k+1}^n = \lambda_k^n + c_k H^{-1} \tilde{e}^{m,n}(q_k^m, u_k^n).$$

Return to (3) with $k = k + 1$.

The second algorithm takes advantage of the quadratic properties of $L_{c_k}^{m,n}$; i.e., $L_{c_k}^{m,n}$ is quadratic in q^m (respectively, u^n) if u^n is fixed (respectively, if q^m is fixed).

ALGORITHM 2.

(1) Same as in Algorithm 1.

(2) Set $k = 1$ and $u_0^n = H^{-1} b$.

(3) Determine q_k^m from

$$(P_{\text{equ}}) \quad \text{minimize} \quad \frac{c_k}{2} \tilde{e}^{m,n}(q^m, u_{k-1}^n)^T H^{-1} \tilde{e}^{m,n}(q^m, u_{k-1}^n) + \lambda_k^{n^T} \tilde{e}^{m,n}(q^m, u_{k-1}^n)$$

$$\text{over} \quad q^m$$

$$\text{subject to} \quad q^m \geqq \alpha.$$

(4) Determine u_k^n from

$$(P_{\text{out}}) \quad \text{minimize} \quad \frac{1}{2} u^{n^T} H u^n + u^{n^T} b + \lambda_k^{n^T} \tilde{e}^{m,n}(q_k^m, u^n) + \frac{c_k}{2} \tilde{e}^{m,n}(q_k^m, u^n)^T H^{-1} \tilde{e}^{m,n}(q_k^m, u^n) \quad \text{over} \quad u^n$$

(5) is identical to (4) in Algorithm 1.

Concerning a stopping criterion, several obvious choices involving consecutive values of λ_k^n, u_k^n, and q_k^m, and combinations thereof can be made. Our calculations were stopped either after L_2-difference of successive values of q_k^m was smaller than a certain tolerance level, or after a fixed amount of iteration.

From the first iteration of Algorithm 2 the hybrid nature of the augmented Lagrangian approach is most apparent. In fact, with our choice for $\lambda_1 = 0$ and $u_0^n = H^{-1}b$, the solution q_1^m of (P_{eqn}) with $k=1$ is the solution of the equation error formulation in H^{-1}. The subsequent optimization problem (P_{out}) is a least square step balanced with the equation error penalty term.

The numerical solution of the minimization problem in (3) of Algorithm 1 or of (P_{equ}) and (P_{out}) of Algorithm 2 require the gradients of $L_{c_k}^{m,n}$ with respect to q^m and u^n, and we specify them next. The gradient of $L_{c_k}^{m,n}$ with respect to q^m is given by

$$\beta W q^m + \Phi^T(u^n)[\lambda_k^n + c_k H^{-1}(\Phi(u^n)q^m - f^n)]$$

and the gradient of $L_{c_k}^{m,n}$ with respect to u^n is given by

$$Hu^n + b^n + H(q^m)[\lambda_k^n + c_k H^{-1}(H(q^m)u^n - f^n)]$$

where $\tilde{e}^{m,n}(q^m, u^n) = \Phi(u^n)q^m - f^n = H(q^m)u^n - f^n$ and the matrices $\Phi(u^n) \in \mathcal{R}^{n\times m}$ and $H(q^m) \in \mathcal{R}^{n\times n}$ are defined by

$$\Phi(u^n)_{k,l} = \langle \psi_l^m \nabla u^n, \nabla \phi_k \rangle, \qquad k = 1, \cdots, n, \quad l = 1, \cdots, m,$$

$$H(q^m)_{i,j} = \langle q^m \nabla \phi_i, \nabla \phi_j \rangle, \qquad i, j = 1, \cdots, n.$$

With our choice of basis functions for ψ_i^m and ϕ_i^n, the matrices $\Phi(u^n)$ and $H(q^m)$ are sparse. In the two-dimensional case, the gradient calculations can be performed by the order of $N^2 + M^2$ operations. Since H is a symmetric band matrix with band width N, the calculation of $H^{-1}\tilde{e}$ for given $\tilde{e} \in \mathcal{R}^m$ using Cholesky decomposition without pivoting requires the order of N^3 operations [DB, § 5.4].

Remarks. (1) In Algorithm 2, steps (3) and (4) are a quadratic minimization in q^m and u^n, respectively. This is due to the fact that the pair consisting of the coefficient q and the state variable u appear in a bilinear form in (1.1) and that q and u are treated as independent variables in our approach. In the output least squares formulation, when the state variable u is treated as a function of q, this property is lost.

(2) We used the conjugate gradient method [AB] to perform step (3) in algorithm 1 and steps (3) and (4) in Algorithm 2 (we used the ZXCGR-routine in the IMSL library in our calculations).

(3) Our theory requires that the optimization problem of step (3) of Algorithm 1, i.e.,

$$\text{minimize} \quad L_{c_k}^{m,n}(q^m, u^n, \lambda_k^n) \qquad \text{subject to} \quad q^m \geqq \alpha, \tag{3.6}$$

be solved sufficiently accurately in the sense of (2.7) of Theorem 2.7 to achieve convergence of the iterates $(q_k^m, u_k^n, \lambda_k^n)$ as $k \to \infty$. Of course, steps (3) and (4) of Algorithm 2 only provide an approximate solution to (3.6). We therefore suggest to repeat steps (3) and (4) of Algorithm 2 a few times at the initial stages of the algorithm (i.e., $k=1, 2$) before updating the Lagrange multiplier in step (5). (Our numerical experience shows that even if this suggestion is not followed the algorithm will converge, but at each level of the iteration the numerical solution q_k^m is a less accurate fit to the "true" coefficient $\hat{q}$ than when steps (3) and (4) are repeated as explained above.)

(4) Based on our theoretical work in [IK1], [IK2], and Proposition 2.4, the choice of c_1 must be made such that the Lagrangian function of (P^β) is uniformly convex at the solution (q_β^*, u_β^*). Thus c_1 depends on z, f, γ, β, α and embedding constants. In our calculations, we chose c_1 heuristically and found that the algorithm is quite insensitive with respect to the choice of c_k. We thus took $c_k = 1$ for all k in most of our calculations. The relative insensitivity of the algorithm to the choice of $\{c_k\}$ can

probably be attributed to the fact that the regularization term also adds to the required local convexity of the cost-functional in the neighborhood of the solution. We refer the interested reader to [PT] for general guidelines on choosing the coefficients c_k.

(5) The matrix operation H^{-1} (which approximates $(-\Delta)^{-1}$) is a consequence of the specific norm in which the equation error term is incorporated in the augmented Lagrangian functional. From a numerical point of view it plays the role of (pre-) conditioning the cost functional. In fact, (P_{out}) can be written as

$$\text{(3.7)} \qquad \text{minimize } \tfrac{1}{2}u^{n^T}(H + c_k H(q_k^m)H^{-1}H(q_k^m))u^n + u^{n^T}(b^n + H(q_k^m)(\lambda_k^n - c_k H^{-1}f^n)) + \text{const.}$$

Since $H(q_k^m)$ is the approximation to $-\nabla(q_k^m(\nabla\cdot))$ on V^n, the matrices H and $H(q_k^m)$ possess the same order of spectral condition number. Without the operation H^{-1} the equation error term $\frac{1}{2}c_k u^{n^T}H(q_k^m)H^{-1}(q_k^m)u^n$ in (P_{out}) would exhibit the square of the condition number of H. On the other hand, in the multidimensional case the operation $H^{-1}e$, for given $e \in \mathfrak{R}^n$, is the most costly calculation in our algorithms. Hence we may consider to replace H^{-1} by a smoothing matrix V based on the SSOR (symmetric successive overrelaxation) method [AB] or on a multigrid method [HT] so that less calculation (order of N^2) is required. We refer to [AB] for a discussion of the condition number of $H(q_k^m)VH(q_k^m)$ in this case.

(6) It is recommended to use a preconditioned conjugate gradient algorithm for the minimization of (3.6) since the condition number of the matrix $H(q_k^m)H^{-1}H(q_k^m)$ is still of the order of N^2 in dimension two. The preconditioned conjugate gradient method can be formulated as follows: setting

$$Q = H + c_k H(q_k^m)H^{-1}H(q_k^m),$$

the jth step consists of

$$\alpha_j = g_j^T h_j / d_j^T Q d_j, \qquad u_{j+1} = u_j + \alpha_j d_j,$$
$$g_{j+1} = g_j + \alpha_j Q d_j, \qquad h_{j+1} = H^{-1}g_{i+1},$$
$$\beta_j = g_{j+1}^T h_{j+1} / g_j^T h_j, \qquad d_{j+1} = -h_{j+1} + \beta_j d_j,$$

where u_j is the jth iterate for the minimizer, g_j is the corresponding gradient vector, and d_j is the conjugate direction. Initially we choose u_0 and put $g_0 = Hu_0 + b^n + H(q_k^m)(\lambda_k^n - c_k H^{-1}f^n)$, $h_0 = H^{-1}g_0$, and $d_0 = -h_0$. Again, we could replace the smoothing operation H^{-1} by one of the alternates described above.

(7) An implicit regularization is often achieved by choosing coarser basis elements for representing $q(x)$ than for $u(x)$. For example, in some of our calculations, we took $N = 2M$ (see [KK1]-[KK3]).

4. Numerical results. Numerous numerical experiments were carried out for the two algorithms that we explained in § 2. The tests were conducted in the following way. For all examples except Example 7, we chose the "true" parameter $\hat{q}$ as well as the solution $\hat{u} = u(\hat{q})$ and calculated f from (1.1). For Example 7 we chose $\hat{q}$ and f and calculated $\hat{u} = u(\hat{q})$ numerically. The data z were determined from $u(\hat{q})$ either by putting $z = u(\hat{q})$ ("distributed data") or by evaluating $u(\hat{q})$ at points $x_i \in \Omega$ and determining z as a cubic Hermite spline interpolation of $z_i = u(\hat{q}, x_i)$. Noisy data were obtained by specifying

$$z_i = u(\hat{q}, x_i) + \delta\xi_i,$$

with ξ_i a uniformly distributed random number in $[-1, 1]$, and by choosing z as the cubic Hermite spline interpolation of these z_i. The numerical examples to be discussed

below are divided into two groups and are based on two different packages. The first six examples demonstrate the performance of the algorithm while a specific aspect, for instance, the behavior near the singular set, is under investigation and the number of unknowns is no larger than 81. In Examples 7 and 8 we show that the algorithm we propose also works very well for a very fine resolution (64×64) of the unknown coefficient. For the calculations of Examples 1–6 we took Type I basis functions with the grid for the state approximation twice as fine as the grid for the coefficient approximation ($N = 2M$). This choice introduces some inherent regularization. The $(-\Delta)^{-1}$ operation is resolved by Cholesky factorization and the optimization problems are solved with the IMSL version ZXCGR of the conjugate gradient algorithm. The discretization for Examples 7 and 8 used Type III basis functions with $N = M$. The $(-\Delta)^{-1}$ operation is realized by a multigrid step involving one V-cycle with finite-element injection, while the optimization problems are solved by a preconditioned conjugate gradient algorithm as explained in (5) of § 2. Moreover, Algorithm 2 was slightly modified by iterating several times between steps (3) and (4) before going to step (5). The results from a coarser mesh are then used as startup values for the next finer mesh. The calculations for Examples 1–6 were carried out on an IBM-AT and those for Examples 7–8 on an IBM 3090, both in double precision. For further specification of the numerical aspects and for additional test examples we refer to [KK1]–[KK3].

We summarize some general observations.

(1) In all examples the augmented Lagrangian algorithm in the form of Algorithm 1 as well as Algorithm 2 performed well, by which we mean that our algorithms recover the unknown coefficient in a manner in which the graph of the true coefficient and that of the numerical solution are (almost) identical in the error-free case and where the shape of the numerical solution is still qualitatively correct in the presence of error. We remind the reader here that our interest lies in the inverse problem, and hence the primary measures of success are less ambitious than those for the forward problem.

(2) In all examples where complete data were available (i.e., $z = u(\hat{q})$), where $\hat{q}$ was a smooth function, and where the singular set S was small, the first or second iteration gave a good approximation to both $\hat{q}$ and $u(\hat{q})$. If these requirements were not satisfied, and m was sufficiently large, then higher iterations gave an improvement over the first one (see, e.g., Example 2). Recall that for Algorithm 2, the determination of the first iteration q_1^m coincides with the solution of the equation error algorithm.

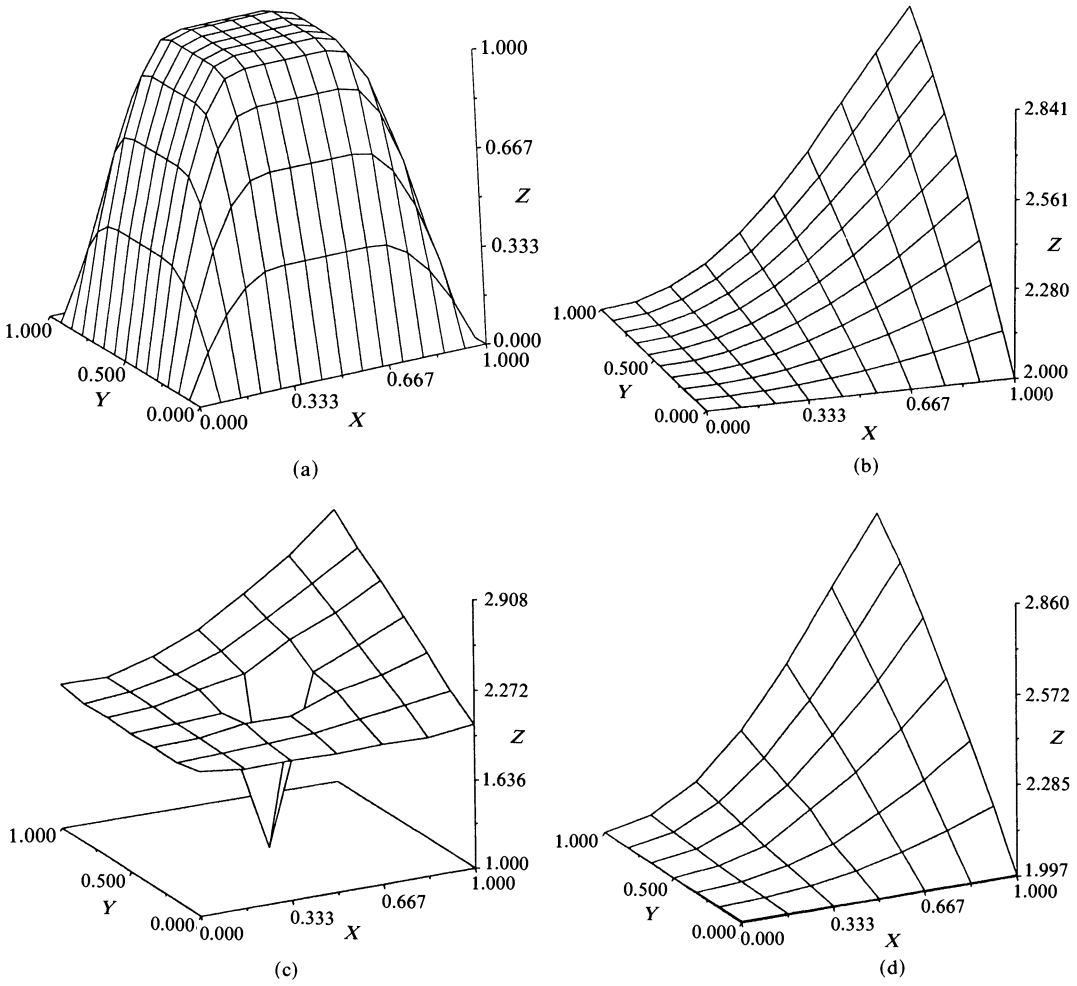
(3) Algorithm 2 is significantly faster than Algorithm 1 (see Example 3).

(4) For the estimation of $\hat{q}$ the form of the singular set S is of great importance. If the dimension of S is smaller than the dimension of Ω (and $\beta = 0$ or β is very small), then the maximum error of the approximation to $\hat{q}$ generally occurs within a small neighborhood of S. If S contains an open set, then $\hat{q}$ cannot be estimated there (see, e.g., Examples 1 and 2).

(5) We could observe convergence and rates of convergence of the solutions to $(P_\beta^{m,n})$ as $m \to \infty$, $n \to \infty$ (see Example 3).

(6) The use of a regularization term, especially if the same grid is chosen for both the coefficient and the state space approximation, or to avoid oscillations in the neighborhood of a singular set, is useful (see Examples 1 and 4). We are currently working on an adaptive algorithm to specify the level of regularization that is required for a specific problem.

(7) The augmented Lagrangian algorithm that we propose also works very well when applied to problems with discontinuous unknown coefficients (see Examples 4 and 7). Comparing the results where a gridpoint of the discretization for the coefficient



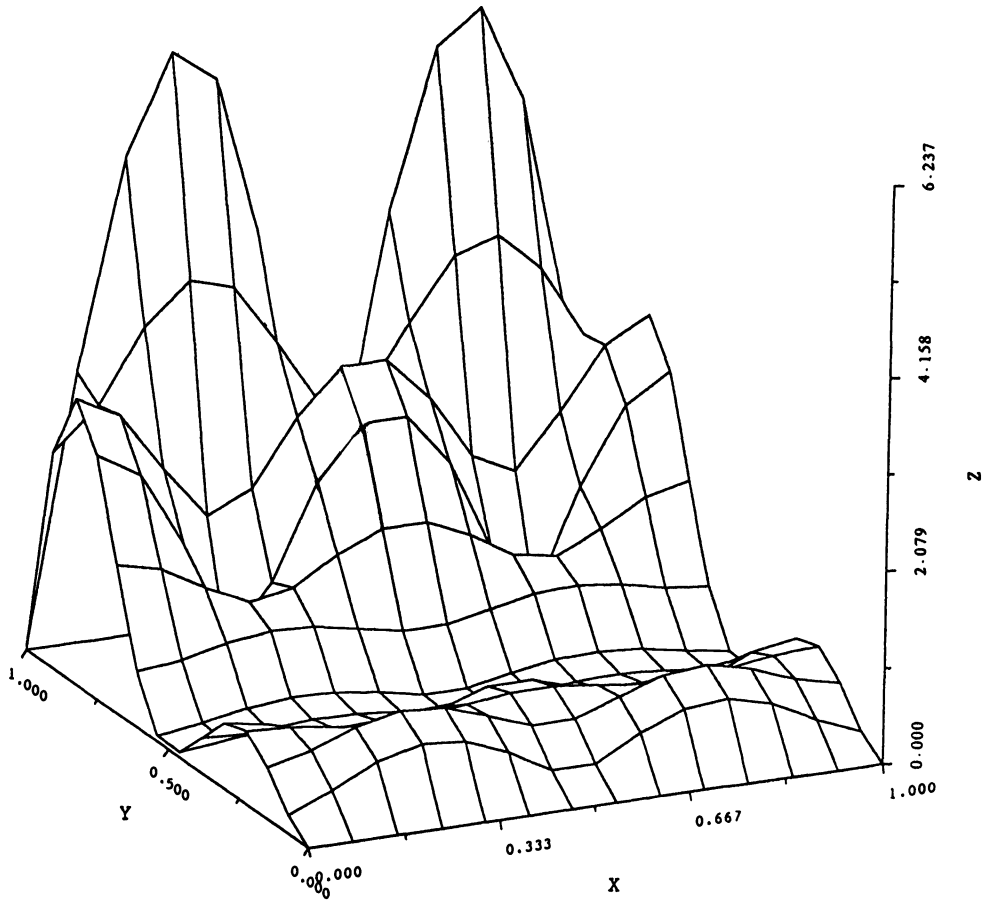
EXAMPLE 1. *Plot* 1: (a) *function* z; (b) *function* q^*; (*c*) $\beta = 0$, $M = 6$, $k = 8$, $L^2 = 0.12$; (*d*) $\beta = 10^{-5}$, $M = 6$, $k = 8$, $L^2 = 0.0036$.

coincides with a discontinuity of $\hat{q}$, to those where discontinuities and gridpoints do not coincide, the latter were clearly better (see Example 4).

Most of the plots that we present below are endowed with the following information: Example X, $\beta = X$, $M = X$, $k = X$, $L^2 = x$, additional comments. Here L^2 stands for the L^2-error $|q_k^M - \hat{q}|_{L^2}$. Unless otherwise specified, the data were taken to be error-free distributed observations, $z = u(\hat{q})$. Examples 3-6 were calculated with Algorithm 1, the remaining examples with Algorithm 2. Since the numerical results always produced a very good approximation to $u(\hat{q})$, we only show the numerical results for the approximation of the coefficient $\hat{q}$ (except in Example 6). In the one-dimensional examples, $\Omega = (0, 1)$, and in the two-dimensional examples, $\Omega = (0, 1) \times (0, 1)$. The startup value for q^m for the finite-dimensional approximating problems $(P_\beta^{m,n})$ was chosen to be identically one in Examples 1-6 and 1.5 in Examples 7 and 8. For all calculations c_k was taken to be equal to one.

Example 1. Here we take

$$z(x, y) = u(\hat{q})(x, y) = w(x)w(y), \qquad \hat{q} = 2 + \sin(x^2 y)$$



EXAMPLE 2. *Plot 2: function* $|\nabla z|$.

where

$$w(x)=\begin{cases}-9x^2+6x, & x\in[0,\tfrac{1}{3}],\\ 1, & x\in(\tfrac{1}{3},\tfrac{2}{3}),\\ -9x^2+12x+3, & x\in[\tfrac{2}{3},1],\end{cases}$$

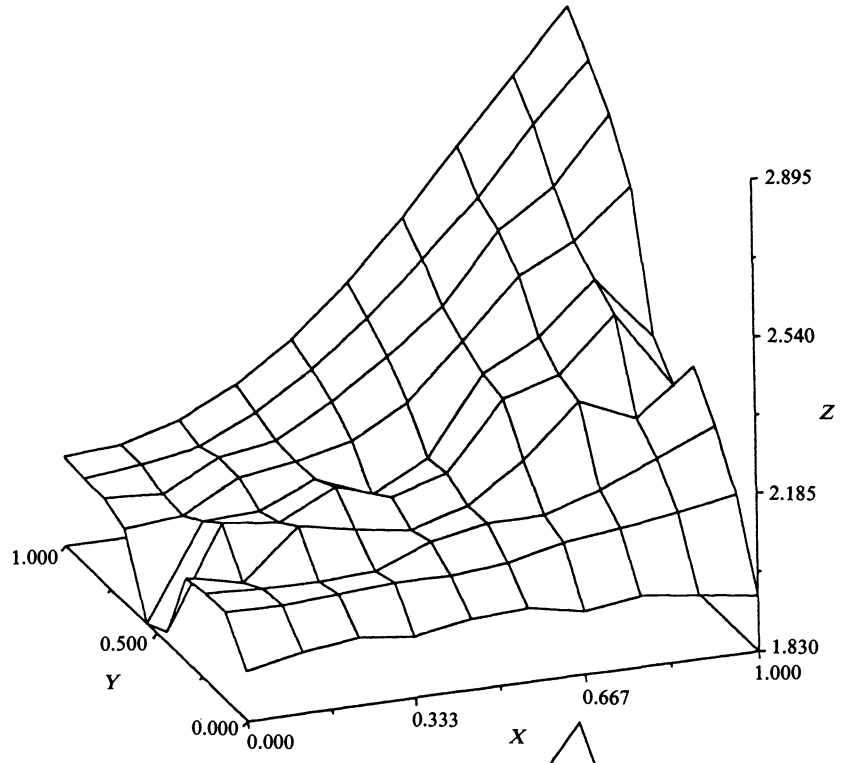
with f calculated from (1.1). In Plot 1 we give the graph for z and $\hat{q}$, as well as the numerical result for q after eight iterations without and with the use of a regularization term. Here the singular set has nonempty interior. Observe that regardless of the use of a regularization term, $\hat{q}$ is identified well over the complement of the singular set.

Example 2. Here we took

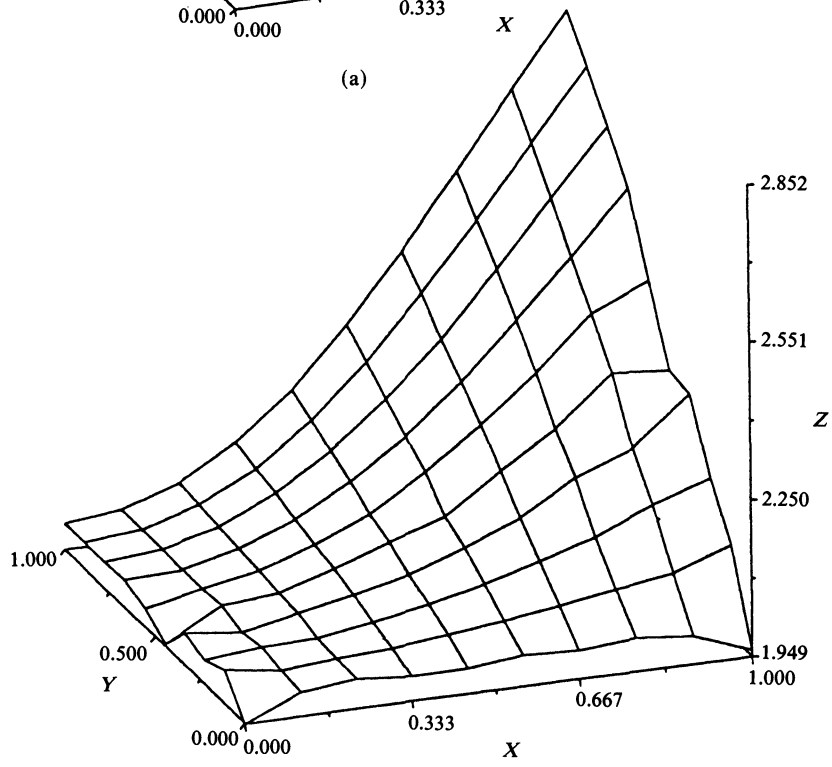
$$z(x,y)=u(\hat{q})(x,y)=(y-2y^2)\sin 2\pi x\sin 2\pi y,\qquad \hat{q}=2+\sin(x^2y),$$

and f is calculated from (1.1).

The singular set for this example consists of isolated points and the line characterized by $y=0$ and $y=\frac{1}{2}$. Plot 2 gives a graph of $|\nabla z|$ and Plot 3 shows the results after the first and eighth iteration. As pointed out in § 2 the solution for q after the first iteration of Algorithm 2 coincides with the equation error formulation in H^{-1}. This

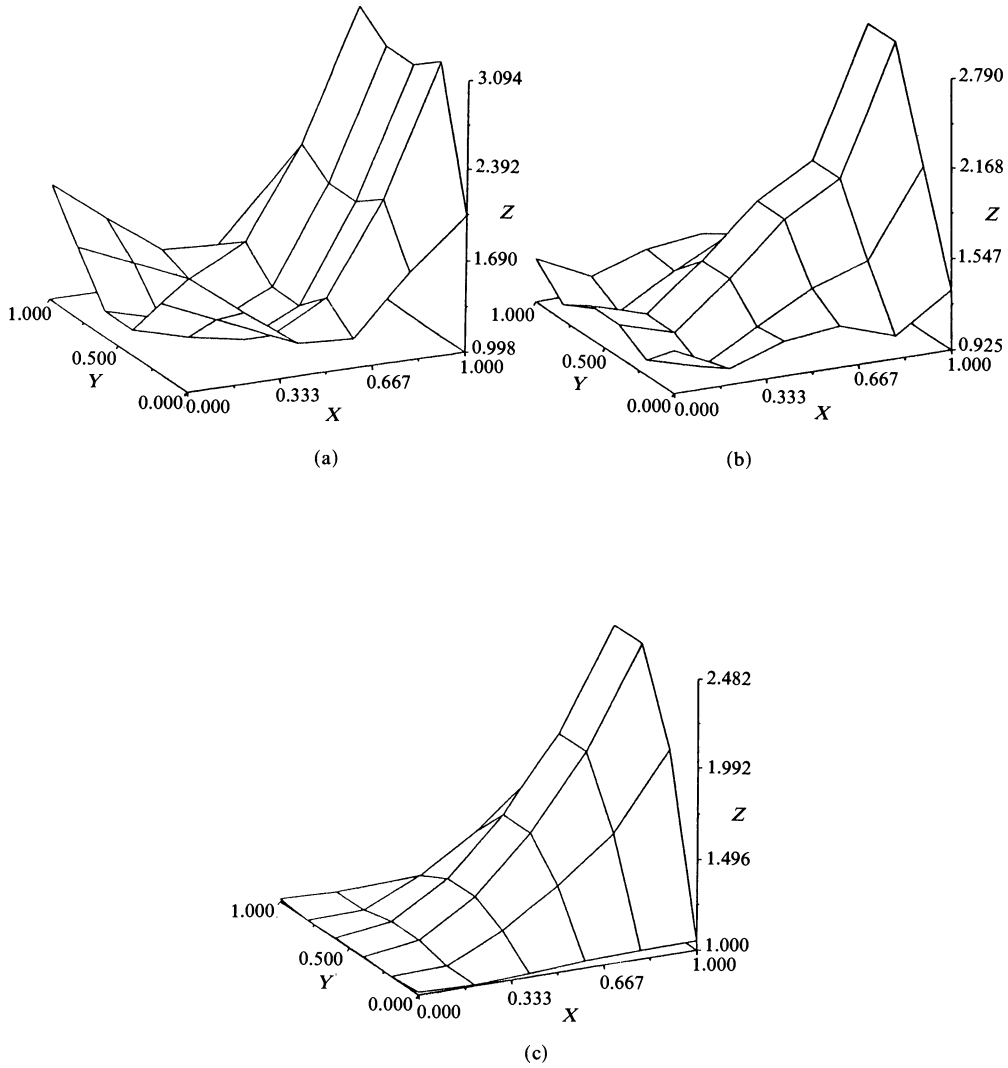


(a)



(b)

EXAMPLE 2. *Plot* 3: (a) $\beta = 0$, $M = 9$, $k = 1$, $L^2 = 0.034$; (b) $\beta = 0$, $M = 9$, $k = 8$, $L^2 = 0.0065$.



EXAMPLE 3. *Plot* 4: (a) $\beta = 0$, $M = 5$, $k = 1$ (*interpolated data at* $\{\frac{i}{3}, \frac{j}{3}\}$); (b) $\beta = 0$, $M = 5$, $k = 1$ (*interpolated data at* $\{\frac{i}{4}, \frac{j}{4}\}$); (c) $\beta = 0$, $M = 5$, $k = 1$ (*interpolated data at* $\{\frac{i}{5}, \frac{j}{5}\}$).

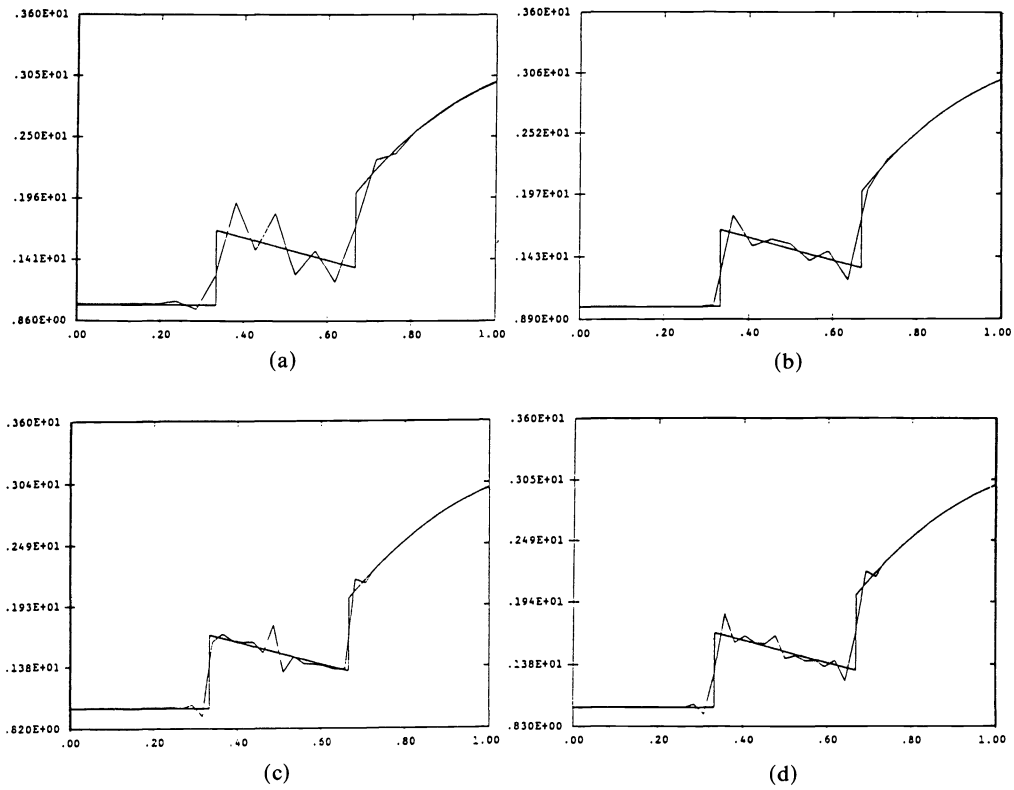
example illustrates that, even with error-free data, the augmented Lagrangian method essentially improves the result of the equation error method.

The decrease of the L^2-error $|q_k^M - \hat{q}|_{L^2}$ for $M = 8$ as k increases is documented in Table 1. Table 1 also shows the number of function evaluations (FUNCEVA) of the cost functional needed by the conjugate gradient algorithm ZXCGR in Algorithm 2. The count for FUNCEVA combines the number of function evaluations needed for the minimization with respect to q and with respect to u.

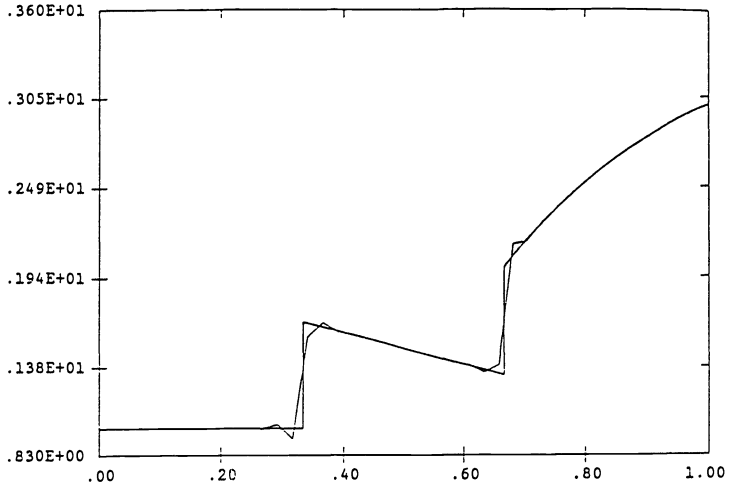
Example 3. Here we took

$$u(\hat{q}) = \sin 2\pi x \sin 2\pi y, \qquad \hat{q} = 1 + 6x^2 y(1-y),$$

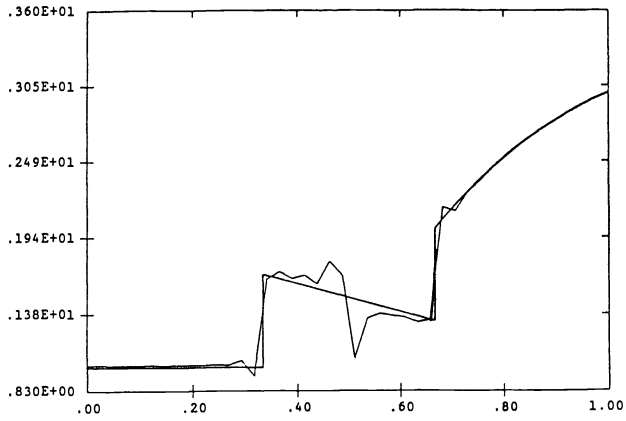
and calculated f from (1.1). Then tests were carried out assuming that successively more data points become available. The results after one iteration of Algorithm 1 with



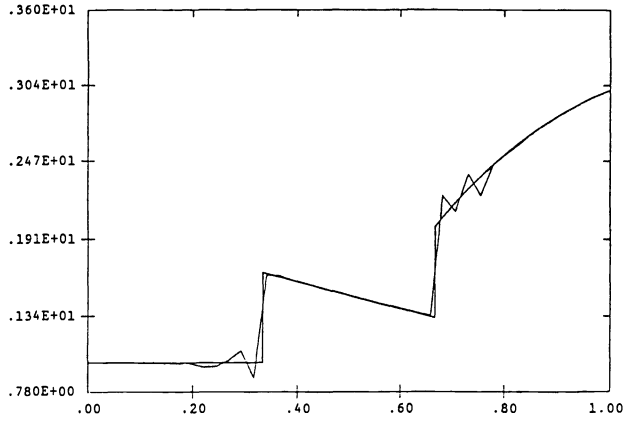
EXAMPLE 4. *Plot* 5: (a) $\beta = 0$, $M = 21$, $k = 2$, $L^2 = 0.101$; (b) $\beta = 0$, $M = 22$, $k = 2$, $L^2 = 0.064$; (c) $\beta = 0$, $M = 41$, $k = 2$, $L^2 = 0.055$; (d) $\beta = 0$, $M = 42$, $k = 2$, $L^2 = 0.060$.



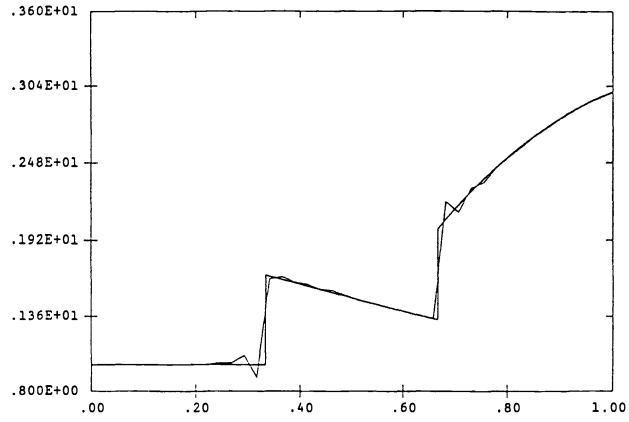
EXAMPLE 4. *Plot* 6: $\beta = 10^{-5}$, $M = 41$, $k = 3$, $L^2 = 0.0445$.



(a)

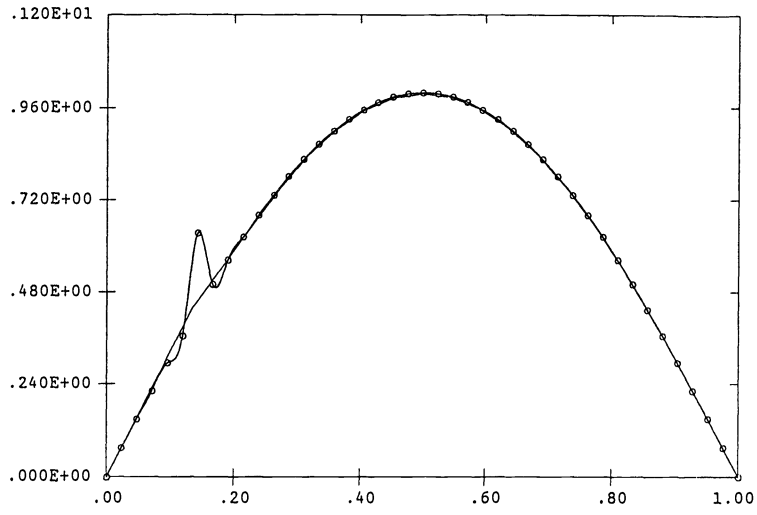


(b)

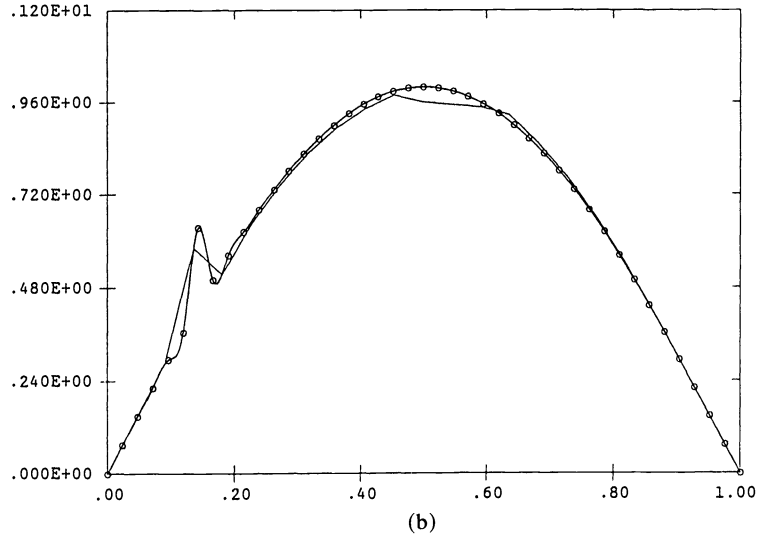


(c)

EXAMPLE 5. (a) *Plot* 7: $\beta = 0$, $M = 41$, $k = 3$, $L^2 = 0.0525$ (*observation* $z_1 = \sin(\pi x)$); (b) $\beta = 0$, $M = 41$, $k = 3$, $L^2 = 0.047$ (*observation* $z_2 = \sin(2x)$); (c) $\beta = 0$, $M = 41$, $k = 3$, $L^2 = 0.0447$ (*two distributional observations*).



(a)



(b)

EXAMPLE 6. *Plot* 8: (a) $\beta = 0$, $M = 6$, $k = 3$, $W^{1,1}$*-criterion*; (b) $\beta = 0.1$, $M = 6$, $k = 3$, H^1*-criterion.*

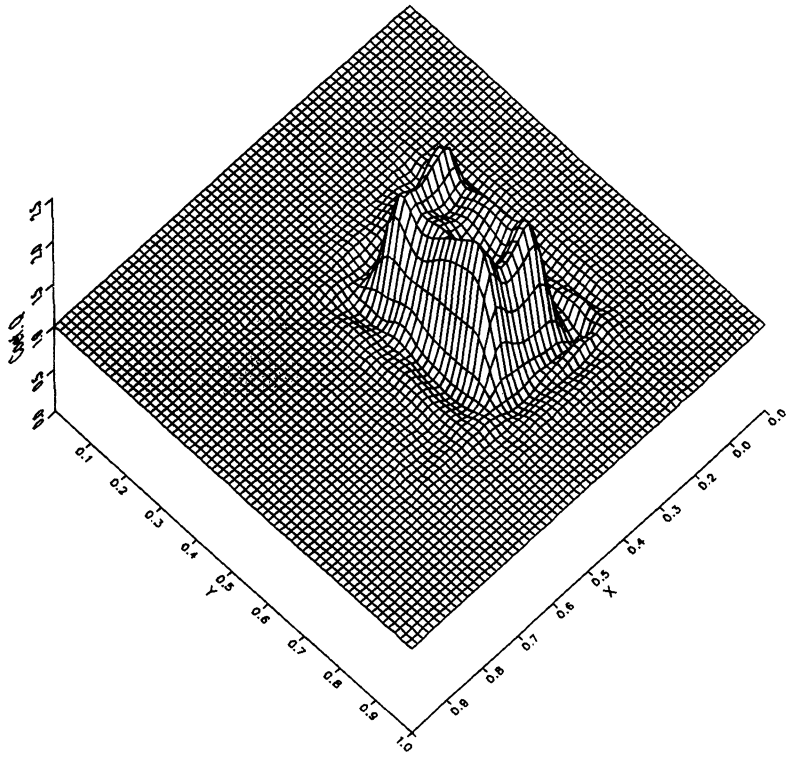
$M = 5$ are shown in Plot 4. Further iterations do not change the results significantly for this example. This is probably due to the fact that the discretization of q is rather coarse. For this example we also compared the L^2-error of the solution q_8^M of the eighth iteration of Algorithm 2 to $\hat{q}$, as M and $N(=2M)$ increased. The results can be found in Table 2, where $\hat{\delta}$ and $\hat{k}$ are calculated from

$$|q_8^M - \hat{q}|_{L^2} / |q_8^{M-1} - \hat{q}|_{L^2} = \left(\frac{M}{M+1}\right)^{\hat{\delta}}$$

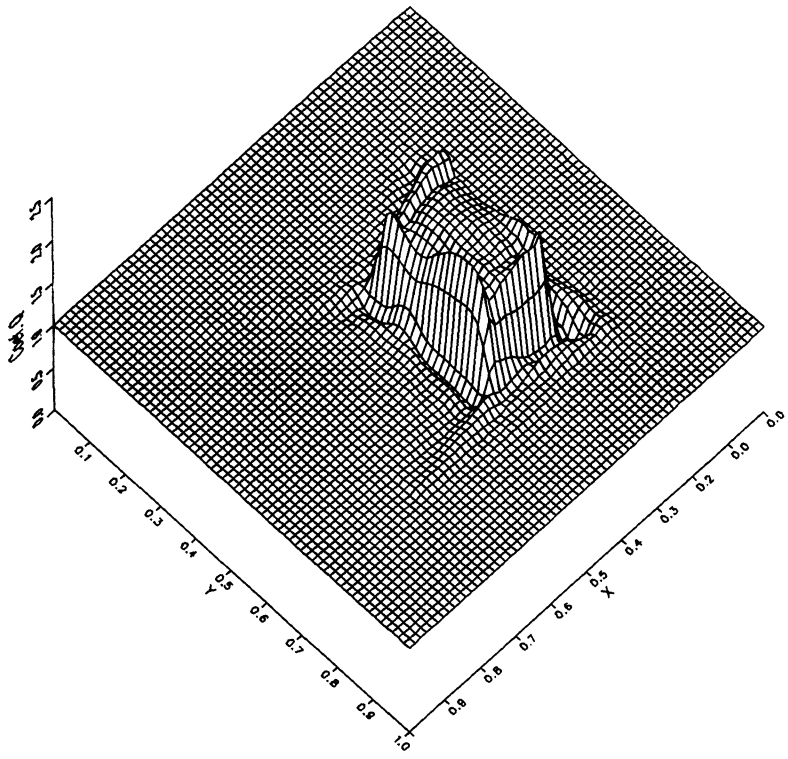
and

$$|q_8^M - \hat{q}|_{L^2} = \hat{k}(M+1)^{-\hat{\delta}}.$$

At iteration 8 the algorithm was considered to have converged, since the numerical result did not change in four significant digits between iterations 7 and 8. Tables 3 and 4 show the L^2-error $|q_k^M - \hat{q}|_{L^2}$ for $M = 8$ and the number of function evaluations of the cost functional required by the conjugate gradient algorithm. As before, the count for FUNCEVA for Algorithm 2 combines the number of function evaluations needed

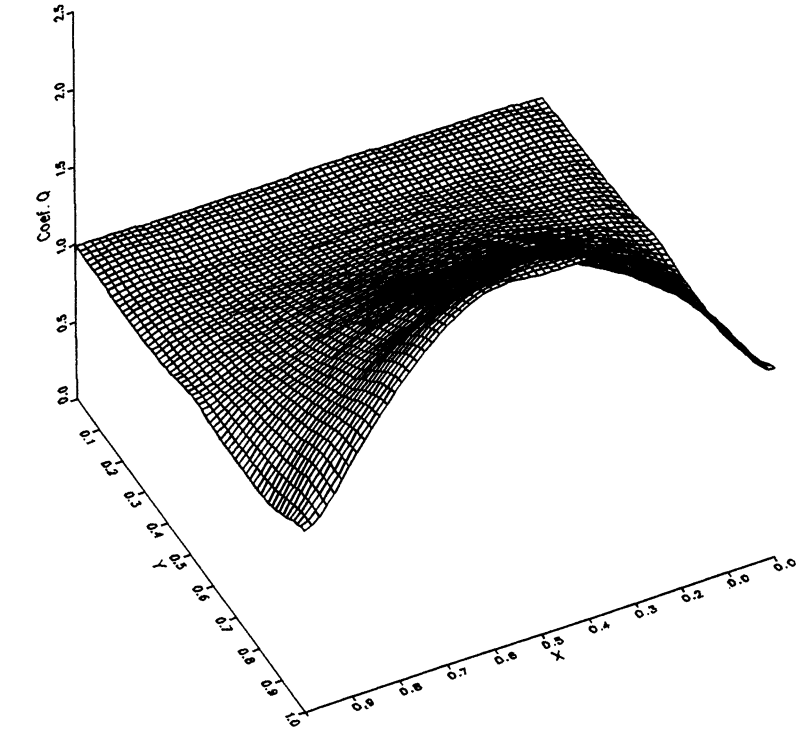


(a)

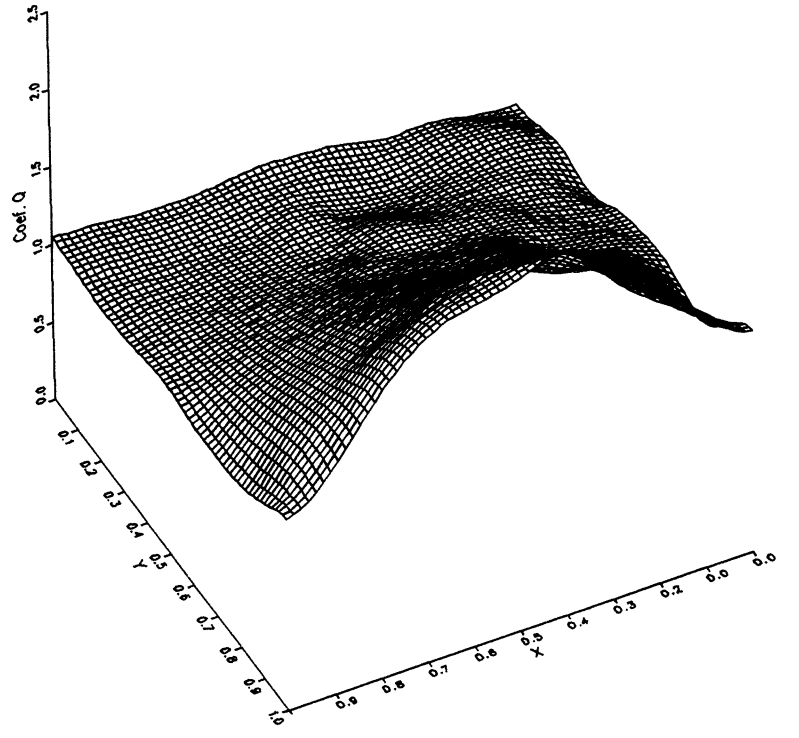


(b)

EXAMPLE 7. *Plot* 9: (a) $\beta = 10^{-4}$, $M = 64$ (20×20 *point measurements*); (b) $\beta = 10^{-4}$, $M = 64$ (40×40 *point measurements*).

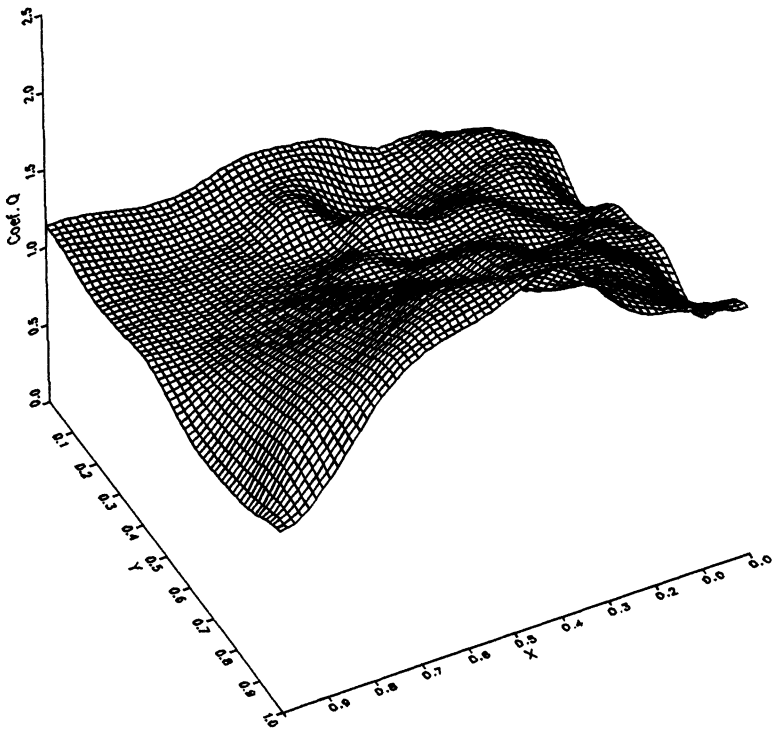


(a)



(b)

EXAMPLE 8. *Plot* 10: (a) $\beta = 10^{-3}$, $M = 64$ (10×10 *point data with* 0% *noise*); (b) $\beta = 5 \times 10^{-3}$, $M = 64$ (10×10 *point data with* 1% *noise*).



(c)

EXAMPLE 8 (*cont.*). *Plot* 10: (c) $\beta = 5 \times 10^{-3}$, $M = 64$ (10×10 *point data with* 3% *noise*).

TABLE 1
(*Example* 2)
L^2-*error, Algorithm* 2, ($M = 8$).

k	L^2-error	FUNCEVA
1	.0344	387
2	.0154	164
3	.0122	96
4	.0097	99
5	.0084	74
6	.0077	72
7	.0067	86
8	.0065	62

TABLE 2
(*Example* 3)

M	L^2-error	$\hat{\delta}$	$\hat{k}$
4	0.00977	2.1	0.3
5	0.00663	1.9	0.2
6	0.00495	2.0	0.2
7	0.00380	2.0	0.2
8	0.00300		

TABLE 3
(*Example* 3)
Algorithm 1 ($M=8$).

k	L^2-error	FUNCEVA
1	.003007	1156
2	.003007	152
3	.003006	190

TABLE 4
(*Example* 3)
Algorithm 2 ($M=8$).

k	L^2-error	FUNCEVA
1	.014066	114
2	.004022	93
3	.003063	80
4	.002846	87
5	.002830	66
6	.002859	69

for the minimization with respect to q and with respect to u. The calculation for the first six iterations of Algorithm 1 took about seven times longer than for Algorithm 2.

Example 4. This is an example with a discontinuous coefficient:

$$z = u(\hat{q}) = \sin \pi x,$$

$$\hat{q} = \begin{cases} 1 & \text{for } x \in [0, \tfrac{1}{3}], \\ 2-x & \text{for } x \in (\tfrac{1}{3}, \tfrac{2}{3}), \\ -9x^2+(21/2)x-3 & \text{for } x \in [\tfrac{2}{3}, 1]. \end{cases}$$

The results for $M=21$, $M=22$, and $M=42$ are given in Plot 5. For $M=21$ and $M=42$, the discontinuities are also gridpoints of the discretization for q. We point out that these estimates are obtained without a priori assumptions on the location of the discontinuity and the size of the jump. The results can be improved by using regularization (see Plot 6).

Example 5. Here $\hat{q}$ is chosen as in Example 4. We compare the numerical results for two different observations $z_1 = \sin \pi x$ and $z_2 = \sin 2\pi x$. Observe that the maximum of the error occurs in a neighborhood of the singular set which for z_1 is $S(z_1) = \frac{1}{2}$ and for z_2 is $S(z_2) = \{\frac{1}{4}, \frac{3}{4}\}$. The last graph in Plot 7 shows the numerical result, when Algorithm 1 was modified in the obvious way to account for two observations simultaneously.

Example 6. Our motivation for the choice of H_0^1 as an output norm was discussed in § 3. The question arises, whether a norm in the family of $W^{1,p}$-norms, $p \in [1, \infty]$, should be preferred. While we have not pursued a detailed study concerning this question, we would at least like to point out by means of a specific example that the choice can have a significant influence on the result, if the data are not error free. As an example,

$$u(\hat{q}) = \sin \pi x, \qquad \hat{q} = 1 + x.$$

We put $z(x_i) = u(\hat{q})(x_i)$ with $x_i = i/42$, for $i = 0, \cdots, 7, 9, \cdots, 42$, and $z(x_8) = u(\hat{q})(x_8) + \frac{1}{4}$, thus producing an outlier at 4/21. Then we passed a cubic Hermite spline

interpolate through z_i (see the thicker lines in Plot 8). To estimate the unknown parameter $\hat{q}$ we used Algorithm 1 as well as Algorithm 1 with the $|u-z|_{H_0^1}$ output criterion replaced by an $|u-z|_{W_0^{1,1}}$ criterion. The second curve in the graphs of Plot 8 give the numerical result for u^n. Differently from the H^1-criterion, the outlier is essentially ignored by the numerical solution u^n, when the $W^{1,1}$-criterion is used. In this case the approximation q^m to $\hat{q}$ is still qualitatively correct while for the H^1-criterion without regularization it produces a useless estimate for $\hat{q}$. Further examples can be found in [KK3].

Example 7. Here we took

$$f = \sin 2\pi x \sin 2\pi y,$$
$$\hat{q} = \begin{cases} 2 & \text{for } (x, y) \in (.3, .3) = (.3, .6), \\ 1 & \text{elsewhere,} \end{cases}$$

and data were assumed to be available on a 20×20 grid for the first result and on a 40×40 grid for the second result. Observe that adding more data points results in a sharper resolution of the discontinuous coefficient $\hat{q}$.

Algorithm 2 is employed in this and the next examples. In the initial step ($k = 1$) of Algorithm 2, steps (3) and (4) are repeated twice before going to step (5). The (64×64) estimate is constructed starting with coarse grid (16×16) problem and using an exterpolate value of its solution as a startup for the next grid. In this example we used three levels (16×16, 32×32, and 64×64). The startup value is 1.5 (constant function) and execution time on IBM 3090 was 8.65 sec. Table 5 is a summary of convergence history.

TABLE 5
(*Example* 7)

Grids	No. of augmented Lagrangian steps	Total no. of conjugate gradient steps	Total no. of step (3)	Total no. of step (4)
16×16	3	55	38	17
32×32	4	49	19	30
64×64	2	29	8	21

For each grid problem, the algorithm is terminated when the L_2-norm of difference of successive estimates $q^{(k)}$ is less then 1×10^{-5}.

Example 8. For this example, with

$$u(\hat{q}) = \sin 2\pi x \sin 2\pi y, \qquad \hat{q} = 1 + 6x^2 y(1-y),$$

consecutively more noise was added to $u(\hat{q})(x_i)$ at the observation points x_i which were assumed at a uniform 10×10 grid. Observe that we are referring to absolute, rather than relative noise here.

Appendix.

Proof of Proposition 2.4. For $(q, u) \in Q_{ad} \times H_0^1$, we find

$$\begin{aligned} J_c(q, u; \lambda^*) - J_c(q_\beta^*, u_\beta^*; \lambda^*) &= \nabla L(q_\beta^*, u_\beta^*; \lambda^*)(h, v) - \langle h \nabla \lambda^*, \nabla v \rangle \\ &\quad + \frac{1}{2}|v|_{H_0^1}^2 + \frac{\beta}{2} N(h) + \frac{c}{2}|e|_{H_0^1}^2 \\ &\geqq \frac{1}{2}|v|_{H_0^1}^2 + \frac{\beta}{2} N(h) + \frac{c}{2}|e|_{H_0^1}^2 + \langle h \nabla \lambda^*, \nabla v \rangle, \end{aligned}$$

where $v = u - u_\beta^*$ and $h = q - q_\beta^*$, and we used Lemma 2.2. The rest of the proof is almost parallel to the one of Proposition 3.4 in [IK2]. Thus, we only give a sketch. First, note that with $P = \nabla \cdot (\Delta)^{-1} \nabla$, we find

$$\begin{aligned} |e|_{H_0^1}^2 &= \langle P(q\nabla u - q_\beta^* \nabla u_\beta^*),\, q\nabla u - q_\beta^* \nabla u_\beta^* \rangle \\ &= \langle P(h\nabla u_\beta^* + q\nabla v),\, h\nabla u_\beta^* + q\nabla v \rangle \\ &\geqq \frac{1}{2} \langle P(h\nabla u_\beta^*),\, h\nabla u_\beta^* \rangle - \langle P(q\nabla v),\, q\nabla v \rangle \\ &\geqq \frac{1}{2} \langle P(h\nabla u_\beta^*),\, h\nabla u_\beta^* \rangle - k_2^2 |v|_{H_0^1}^2, \end{aligned}$$

where we used the fact that P is an orthogonal projection of $(L^2)^n$. Since $h \in X$ can be uniquely represented as

$$h = h^1 + h^2 \quad \text{with } h^1 = \text{const.} \quad \text{and} \quad \langle 1, h_2 \rangle = 0,$$

we have

$$\langle P(h\nabla u_\beta^*),\, h\nabla u_\beta^* \rangle \geqq |h_1|^2 |u_\beta^*|_{H_0^1}^2 - 2k_1 |h_1| \sqrt{N(h_2)}\, |u_\beta^*|_{H_0^1}^2.$$

Next, note that from Corollary 2.3

$$|\langle h\nabla \lambda^*, \nabla v \rangle| \leqq (|h_1| + k_1 \sqrt{N(h_2)})\, |v|_{H^1} |u_\beta^* - z|_{H_0^1} / \alpha.$$

From [CK1] and [IK2] it follows that

$$|u_\beta^* - z|_{H_0^1}^2 \leqq |u^0 - z|_{H_0^1}^2 + \beta(N(q_0^*) - N(q_\beta^*)), \tag{A1}$$

where $N(q_\beta^*) \uparrow N(q_0^*)$ as $\beta \downarrow 0$.

Let $c = \delta\beta$ with $\delta > 0$. Then, we obtain

$$\begin{aligned} I_c &:= \frac{1}{2} |v|_{H_0^1}^2 + \frac{\beta}{2} N(h) + \frac{c}{2} |e|_{H_0^1}^2 + \langle h\nabla \lambda^*, \nabla v \rangle \\ &\geqq \frac{1}{2} (1 - \delta\beta k_2^2) |v|_{H_0^1}^2 - \frac{1}{\alpha} (|h_1| + k_1 \sqrt{N(h_2)})\, |u_\beta^* - z|_{H_0^1} |v|_{H_0^1} \\ &\quad + \frac{1}{2} \beta \left(N(h_2) + \frac{\delta}{2} |u_\beta^*|_{H_0^1}^2 |h_1|^2 - k_1 \delta |h_1| \sqrt{N(h_2)}\, |u_\beta^*|_{H_0^1}^2 \right). \end{aligned}$$

A calculation shows that

$$N(h_2) + \frac{\delta}{2} |u_\beta^*|_{H_0^1}^2 |h_1|^2 - k_1 \delta |h_1| \sqrt{N(h_2)}\, |u_\beta^*|_{H_0^1}^2 \geqq \frac{1}{5k_1^2} (|h_1|^2 + k_1^2 N(h_2))$$

provided that we choose δ such that $\delta |u_\beta^*|_{H_0^1}^2 k_1^2 = 4/5$.

We note that

$$|f|_{H^{-1}} = \sup_{v \in H_0^1} \frac{\langle q_\beta^* \nabla u_\beta^*, \nabla v \rangle}{|v|_{H_0^1}} \leqq k_2 |u_\beta^*|_{H_0^1}$$

and therefore $|u_\beta^*|_{H_0^1} \geqq |f|_{H^{-1}} / k_2$ for all $\beta \geqq 0$. Using these estimates we obtain

$$\begin{aligned} I_c &\geqq \frac{1}{2} \left(1 - \frac{4}{5} \frac{k_2^4 \beta}{|f|_{H^{-1}}^2 k_1^2} \right) |v|_{H_0^1}^2 + \frac{\beta}{10k_1^2} (|h_1|^2 + k_1^2 N(h_2)) \\ &\quad - \frac{\sqrt{2}}{\alpha} \sqrt{|h_1|^2 + k_1^2 N(h_2)}\, |v|_{H_0^1} |u_\beta^* - z|_{H_0^1}. \end{aligned}$$

From (A1) and (2.5) it then follows that

$$I_c \geqq \varepsilon(|v|^2_{H_0^1} + |h_1|^2 + k_1^2 N(h_2))$$

for some $\varepsilon > 0$. Since $\sqrt{|h_1|^2 + k_1^2 N(h_2)}$ defines an equivalent norm of X, this proves (2.4) at $\beta = \beta_0$. This implies that a second-order sufficient optimality condition holds for (P^β) at β_0, and consequently $\beta \to (q^*_\beta, u^*_\beta) \in X \times H_0^1$ is Hölder continuous with exponent $\frac{1}{2}$ at $\beta = \beta_0$ [A]. Therefore there exists a nontrivial interval I containing β_0 such that (2.4) holds for all $\beta \in I$.

REFERENCES

[A] W. ALT, *Lipschitzian perturbations of infinite optimization problems*, in Mathematical Programming with Data Perturbations III, A. V. Fiacco, ed., Marcel Dekker, New York, 1983.

[AB] O. AXELSSON AND V. A. BARKER, *Finite Element Solution of Boundary Value Problems, Theory and Computation*, Academic Press, New York, 1984.

[C] G. CHAVENT, *Identification of distributed parameter systems: About the least square method, its implementation and identifiability*, in Proc. 5th IFAC Symposium on Identifiability and System Parameter Estimation, Darmstadt, Germany, Pergamon Press, Oxford, 1979, pp. 85-97.

[CK1] F. COLONIUS AND K. KUNISCH, *Output least squares stability in elliptic systems*, Appl. Math. Optim., 19(1988), pp. 33-63.

[CK2] ———, *Stability of perturbed optimization problems with applications to parameter estimation*, manuscript.

[DB] G. DAHLQUIST AND Å. BJÖRCK, *Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[EKN] H. W. ENGL, K. KUNISCH, AND A. NEUBAUER, *Tikhonov regularization for the solution of nonlinear ill-posed problems* I, Inverse Problems, submitted.

[HT] W. HACKBUSCH AND U. TROTTENBERG, EDS., *Multigrid Methods*, Lecture Notes in Mathematics 960, Springer-Verlag, Berlin, 1982.

[IK1] K. ITO AND K. KUNISCH, *The augmented Lagrangian method for equality and inequality constraints in Hilbert spaces*, Math. Programming, to appear.

[IK2] ———, *The augmented Lagrangian method for parameter estimation in elliptic systems*, SIAM J. Control Optim., 28 (1990), pp. 113-136.

[J] X. JIANG, *Augmented Lagrangian method for impedence computed tomography*, Ph.D. thesis, Brown University, Providence, RI, 1989.

[K] K. KUNISCH, *Inherent identifiability of parameters in elliptic equations*, J. Math. Anal. Appl., 132 (1988), pp. 453-472.

[KP] K. KUNISCH AND G. PEICHL, *Estimation of a temporally and spatially varying diffusion coefficient in a parabolic system by an augmented Lagrangian technique*, Appl. Math. Optim., submitted.

[KS] C. KRAVARIS AND J. H. SEINFELD, *Identification of parameters in distributed systems by regularization*, SIAM J. Control Optim., 23 (1985), pp. 217-241.

[KK1] M. KROLLER AND K. KUNISCH, *A numerical study of an augmented Lagrangian method for the estimation of parameters in a two point boundary value problem*, Tech. Report, No. 87, Institute for Mathematics, Technical University of Graz, Graz, Austria, March 1987.

[KK2] ———, *A numerical study of an augmented Lagrangian method for the estimation of parameters in elliptic systems*, Tech. Report, No. 101, Institute for Mathematics, Technical University of Graz, Graz, Austria, September 1987.

[KK3] ———, *A numerical study of an augmented Lagrangian method for the estimation of parameters in elliptic systems: Noisy data, outliers and discontinuous coefficients*, Tech. Report, No. 123, Institute for Mathematics, Technical University of Graz, Graz, Austria, September 1988.

[PT] V. T. POLYAK AND N. Y. TRET'YAKOV, *The method of penalty estimates for conditional extremum problems*, Zh. Vychisl. Mat. i Mat. Fiz., 13 (1973), pp. 34-36.

[Sch] M. H. SCHULTZ, *Spline Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[Y] W. W.-G. YEH, *Review of parameter identification procedures in groundwater hydrology: The inverse problem*, Water Res. Rev., 22 (1986), pp. 95-108.

[ZK] J. ZOWE AND S. KURCYUSZ, *Regularity and stability for the mathematical programming problem in Bausch spaces*, Appl. Math. Optim., 5 (1979), pp. 49-62.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 911-939, July 1991
© 1991 Society for Industrial and Applied Mathematics
012

ON THE GENERALIZED SCHUR DECOMPOSITION OF A MATRIX PENCIL FOR PARALLEL COMPUTATION*

A. BUNSE-GERSTNER† AND H. FASSBENDER†

Abstract. An algorithm to solve the generalized eigenvalue problem $Ax = \lambda Bx$ for arbitrary complex $n \times n$ matrices A, B is developed. It is a generalization of an unsymmetric Jacobi method of Eberlein. Each iteration step needs only local information about the current matrix, thus admitting efficient parallel implementations on certain parallel architectures. Versions for an $(n/2) \times (n/2)$ square array of mesh-connected processors and a ring of $n/2$ processors are discussed. Convergence performance of the algorithm is compared with a recently proposed Jacobi-like method for matrix pencils based on the unsymmetric Jacobi method of Stewart. The numerical experiments suggest that the method presented here is considerably faster for matrix pencils which are not near to normality.

Key words. generalized Schur decomposition, matrix pencil, Jacobi method, ring processor system, mesh-connected processors, parallel algorithms

AMS(MOS) subject classification. 65F15

1. Introduction.

Jacobi methods for the solution of the Hermitian matrix eigenvalue problem have not, except for special applications, been considered for practical use since the introduction of the highly efficient QR algorithm.

With the advent of multiprocessor computer systems, however, interest in Jacobi methods has revived due to their apparently high inherent parallelism. Cyclic Jacobi methods for the Hermitian eigenvalue problem and the singular value problem have been modified to work on different parallel architectures [3]-[5], [9], [10], [12], [13], [19], [28], [33], [41]. Comparisons of performance results with corresponding adapted versions of the QR method on the Alliant FX/8, Cray X-MP, and Cyber 205 seem to show that the parallel Jacobi methods can in fact be faster [1], [2], [11]. This suggests that Jacobi methods might be competitive or superior to conventional methods for nonhermitian problems as well.

An elementary step of the Jacobi method [27] for an $n \times n$ Hermitian matrix A consists in choosing a pair of indices (k, l), the pivot pair, and transforming the matrix A such that the (k, l) and (l, k) entries are annihilated by a similarity transformation with a rotation U in the k, l-plane. U differs from the identity only in the four matrix elements $u_{k,k} = u_{l,l} = \cos \alpha$, $u_{k,l} = -e^{-i\theta} \sin \alpha$, and $u_{l,k} = e^{i\theta} \sin \alpha$. These four essential entries of U form a 2×2 unitary matrix, which diagonalizes the 2×2 submatrix of A consisting of the corresponding entries in the k, l plane. From the two angles α in $(-\pi/2, \pi/2]$, which solve this 2×2 diagonalization problem, the one smaller in modulus is chosen. Various Jacobi methods differ in the way the pivot indices are chosen. The parallel versions are always based on a *cyclic Jacobi method*, i.e., Jacobi methods in which all the indices below or above the diagonal are cyclically traversed in a special order. Such a traverse is called a *sweep*. Convergence can be proved under certain conditions and can be shown to be quadratic [26], [29], [34], [35], [38], [39], [42]. Some of these ideas also carry over to normal matrices [21].

Most of the generalizations to nonhermitian eigenvalue problems are of the so-called *norm-reducing* type [14], [15], [17], [31], [40]. These methods employ non-

* Received by the editors September 13, 1989; accepted for publication (in revised form) June 12, 1990. This work was supported by the FSP Mathematisierung of the University of Bielefeld.

† Fakultät für Mathematik, Universität Bielefeld, Postfach 8640, D-4800 Bielefeld 1, Germany.

unitary elementary transformations in addition to the rotations to minimize the Euclidean norm of the current matrix. The matrices thus become asymptotically normal and convergence can be proved to be finally quadratic for nondefective complex matrices [15], [32]. However, these methods seem generally not to be suitable for distributed memory machines because each step needs global information about the current matrix in order to compute the Euclidean norm.

For numerical stability reasons it is desirable to confine ourselves to using only unitary matrices for the transformations. Two different methods of this type for arbitrary complex matrices have been proposed which use only local information for setting up the rotations in each step. One is a method proposed by Stewart [37], and the other, in a simple form first introduced by Greenstadt [7], [20], [23], [24], was suggested by Eberlein [16].

Stewart's method has been adapted to compute the generalized Schur decomposition of an arbitrary complex matrix pencil by Charlier and Van Dooren [8], who were able to prove global convergence under a restriction on the rotation angles. An adaptation of Stewart's method for Hamiltonian matrices was given by Byers [6]. All authors point out that they do not consider these methods, as they now stand, as suitable for practical purposes, but rather as a step in studying the possibilities for parallel computation of nonhermitian eigenvalue problems. Experience suggests that the speeds of convergence of the methods are very unsatisfactory for problems which are not near to normality.

In this paper we develop a Jacobi-like algorithm for the generalized eigenvalue problem analogous to the method of Eberlein. It transforms the initial matrix pencil $A-\lambda B$ to generalized Schur form by a sequence of equivalence transformations with rotations. Section 2 describes the basic idea of the method and § 3 describes the algorithm and the computation of the rotation parameters.

In § 4 we present numerical test results comparing this method with that of Charlier and Van Dooren. These examples demonstrate that this method converges much faster for problems which are not near to normality. They also suggest that the number of rotations necessary to achieve the generalized Schur form with the method here is a function of the dimension of the problems, whereas for the Charlier–Van Dooren method there appears to be no relation. This is important for a possible parallel implementation, because often we would like to run the method there for a fixed number of rotations instead of terminating it based on the size of the lower triangle of the current matrices. The latter way of determining when to stop would need global information about the current matrices, an undesirable requirement for many parallel architectures. As a special case we show corresponding performance results for the standard eigenvalue problem using the Eberlein and the Stewart methods.

As a result of this section we see that we are in the curious situation already known for the Stewart and the Eberlein methods in the case of single matrices: For the Charlier–Van Dooren algorithm a convergence result can be proved, thus making it much more satisfying from a purely mathematical point of view than this method for which, as for the Eberlein method, we do not yet have a convergence theory. However, the performance of our method on the experimental data is much superior to the Charlier–Van Dooren method.

In § 5 we sketch the possibilities of parallel implementations on a ring processor system and on a mesh-connected array of processors suggested by Brent and Luk [3].

There are still many issues to be considered before we can decide whether or not such a method is of practical value for use on parallel machines. For instance, the method as it stands now employs complex arithmetic, even if we start with a real

matrix pencil. Moreover, we observed that for real pencils having an all-real generalized Schur decomposition, the convergence slows down considerably with the use of the complex transformations in the sense that the decrease of the moduli of entries below the diagonal suddenly slows down. An important question is whether the method can be modified to allow only the use of real arithmetic.

Another important question relates to the rate of convergence. In all numerical experiments with this method the Frobenius norm of the lower triangles of the iterates is observed to finally decrease monotonically, and the convergence seems to be between linear and quadratic. However, it has not yet been possible to prove this.

The purpose of this paper is to provide insight and report experience about the performance of these Jacobi-like methods for the solution of nonhermitian eigenvalue problems. Such knowledge is important because of their apparent suitability for certain parallel architectures.

2. Basic considerations. For any pair of $n \times n$ complex matrices $A = (a_{kl})$, $B = (b_{kl})$ there exists a pair of unitary matrices such that $T = UAV^H$ and $S = UBV^H$ are both upper triangular matrices. The pencil $T - \lambda S$ is called *generalized Schur form* of $A - \lambda B$ and $U^H(T - \lambda S)V$ *generalized Schur decomposition* of $A - \lambda B$. A generalized Schur decomposition displays eigenvalues of the pencil and corresponding deflating subspaces (see, e.g., [22]). For sequential machines the QZ algorithm of Stewart and Moler [30] is an efficient way to compute such a decomposition.

Here we want to construct U and V as products of rotations, where a rotation is defined to be a matrix of the following form:

$$
(2.1)\qquad G_{k,l}(\alpha,\beta) = \begin{bmatrix}
1 & & & & & & & & \\
& \ddots & & & & & & & \\
& & 1 & & & & & & \\
& & & \cos\alpha & \cdots & -e^{-i\beta}\sin\alpha & & & k\text{th row} \\
& & & & 1 & & & & \\
& & & \vdots & \ddots & \vdots & & & \\
& & & & & 1 & & & \\
& & & e^{i\beta}\sin\alpha & \cdots & \cos\alpha & & & l\text{th row} \\
& & & & & & 1 & & \\
& & & & & & & \ddots & \\
& & & & & & & & 1
\end{bmatrix}
$$

with $\alpha, \beta \in \mathbb{R}$, $\alpha \in (-\pi/2, \pi/2]$.

The following is a general approach. We set $A_0 := A$ and $B_0 := B$, and for $j \in \mathbb{N}$ we determine the *pivot indices* (k_j, l_j), $k_j < l_j$, and rotations $Q_j := G_{k_j,l_j}(\gamma_j, \delta_j)$ and $Z_j := G_{k_j,l_j}(\alpha_j, \beta_j)$, such that in $A_{j+1} := Q_j^H A_j Z_j$ and $B_{j+1} := Q_j^H B_j Z_j$ the entries in position (l_j, k_j), the *pivot elements*, are zero.

Obviously in the jth step the angles must be chosen to compute the generalized Schur form of the 2×2 subproblem in the k_j, l_j-plane, i.e. (omitting the index j and setting $A_j = (a_{rs})$, $B_j = (b_{rs})$, $A_{j+1} = (\hat{a}_{rs})$, $B_{j+1} = (\hat{b}_{rs})$),

$$
(2.2)\qquad \begin{bmatrix} \cos\gamma & e^{-i\delta}\sin\gamma \\ -e^{i\delta}\sin\gamma & \cos\gamma \end{bmatrix} \begin{bmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{bmatrix} \begin{bmatrix} \cos\alpha & -e^{-i\beta}\sin\alpha \\ e^{i\beta}\sin\alpha & \cos\alpha \end{bmatrix} = \begin{bmatrix} \hat{a}_{kk} & \hat{a}_{kl} \\ 0 & \hat{a}_{ll} \end{bmatrix},
$$

$$
\begin{bmatrix} \cos\gamma & e^{-i\delta}\sin\gamma \\ -e^{i\delta}\sin\gamma & \cos\gamma \end{bmatrix} \begin{bmatrix} b_{kk} & b_{kl} \\ b_{lk} & b_{ll} \end{bmatrix} \begin{bmatrix} \cos\alpha & -e^{-i\beta}\sin\alpha \\ e^{i\beta}\sin\alpha & \cos\alpha \end{bmatrix} = \begin{bmatrix} \hat{b}_{kl} & \hat{b}_{kl} \\ 0 & \hat{b}_{ll} \end{bmatrix}.
$$

Remark 2.3. This 2×2 generalized Schur decomposition corresponds to two 2×2 Schur decompositions, if the matrices are invertible, because both

$$\begin{bmatrix} \cos\gamma & e^{-i\delta}\sin\gamma \\ -e^{i\delta}\sin\gamma & \cos\gamma \end{bmatrix}\begin{bmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{bmatrix}\begin{bmatrix} b_{kk} & b_{kl} \\ b_{lk} & b_{ll} \end{bmatrix}^{-1}\begin{bmatrix} \cos\gamma & -e^{-i\delta}\sin\gamma \\ e^{i\delta}\sin\gamma & \cos\gamma \end{bmatrix},$$

and

$$\begin{bmatrix} \cos\alpha & e^{-i\beta}\sin\alpha \\ -e^{i\beta}\sin\alpha & \cos\alpha \end{bmatrix}\begin{bmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{bmatrix}^{-1}\begin{bmatrix} b_{kk} & b_{kl} \\ b_{lk} & b_{ll} \end{bmatrix}\begin{bmatrix} \cos\alpha & -e^{-i\beta}\sin\alpha \\ e^{i\beta}\sin\alpha & \cos\alpha \end{bmatrix}$$

are upper triangular matrices. However, none of the matrices above are submatrices of $A_jB_j^{-1}$ or $A_j^{-1}B_j$. Thus a method of this kind is not just an implicit version of a Jacobi-like method for AB^{-1} or $A^{-1}B$.

The matrix pair (Q_j, Z_j) used in the jth step is called *generalized Schur rotation.* In general there will be more than one generalized Schur rotation.

Example 2.4(i)

$$A=\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \text{and} \quad B=\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}. \tag{2.4a}$$

$A-\lambda B$ has only one generalized Schur rotation, namely,

$$Q=Z=\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad \text{with} \quad Q^H(A-\lambda B)Z=\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}-\lambda\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

This situation arises, when the 2×2 pencil has only one eigenvalue (here ∞) with geometric multiplicity one. Note that the uniqueness of this generalized Schur rotation comes from normalizing the $(1, 1)$-entry of Q and Z to be real nonnegative.

(ii)

$$A=\begin{bmatrix} 1 & i \\ i & 0 \end{bmatrix} \quad \text{and} \quad B=\begin{bmatrix} 1 & i \\ -i & 0 \end{bmatrix}. \tag{2.4b}$$

Here we have two generalized Schur rotations:

$$Q_1=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad Z_1=\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad \text{with } Q_1^H(A-\lambda B)Z_1=\begin{bmatrix} -1 & -i \\ 0 & 1 \end{bmatrix}-\lambda\begin{bmatrix} -1 & -i \\ 0 & -1 \end{bmatrix},$$

$$Q_2=\begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix}, \quad Z_2=\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix} \quad \text{with } Q_2^H(A-\lambda B)Z_2=\frac{1}{\sqrt{2}}\left(\begin{bmatrix} -1 & -i \\ 0 & 2 \end{bmatrix}-\lambda\begin{bmatrix} 1 & i \\ 0 & 2 \end{bmatrix}\right).$$

The 2×2 problem is diagonalizable. This is the generic case. The second Schur rotation leads to a generalized Schur form, where the eigenvalues occur in reversed order on the diagonals.

(iii)

$$A=0 \quad \text{and} \quad B\in\mathbb{C}^{2\times 2} \text{ arbitrary.} \tag{2.4c}$$

Then for **any** rotation Z_1 a rotation Q_1 can be chosen such that $Q_1(BZ_1)$ is an upper triangular matrix, i.e., $Q_1(A-\lambda B)Z_1$ is an upper triangular matrix. And likewise for **any** rotation Q_2, a rotation Z_2 can be chosen, such that $(Q_2B)Z_2$ is an upper triangular matrix and therefore also $Q_2(A-\lambda B)Z_2$.

(iv)

$$A=B=0. \tag{2.4d}$$

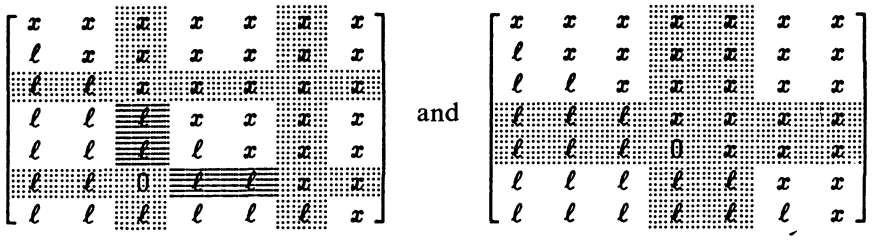
All pairs of rotations are Schur rotations.

Passing from A_j, B_j to A_{j+1}, B_{j+1} only rows and columns k_j and l_j are changed. With the notation above setting $c = \cos\alpha$, $s = e^{i\beta}\sin\alpha$, $p = \cos\gamma$, $q = e^{i\delta}\sin\gamma$, we get the new entries $\hat{a}_{rs}$ and $\hat{b}_{rs}$ as follows:

$$
\begin{aligned}
&\hat{a}_{rs} = a_{rs} \quad \text{for } r, s = 1, \cdots, n && r, s \notin \{k, l\},\\
&\hat{a}_{kr} = ca_{kr} + \bar{s}a_{lr};\ \hat{a}_{lr} = -sa_{kr} + ca_{lr} && \text{for } r \notin \{k, l\},\\
&\hat{a}_{rk} = pa_{rk} + qa_{rl};\ \hat{a}_{rl} = -\bar{q}a_{rk} + pa_{rl} && \text{for } r \notin \{k, l\},\\
&\hat{a}_{kk} = pca_{kk} + p\bar{s}a_{lk} + qca_{kl} + q\bar{s}a_{ll},\\
&\hat{a}_{ll} = q\bar{s}a_{kk} - \bar{q}ca_{lk} - psa_{kl} + pca_{ll},\\
&\hat{a}_{kl} = -\bar{q}ca_{kk} - \overline{qs}\,a_{lk} + pca_{kl} + p\bar{s}a_{ll},\\
&\hat{a}_{lk} = 0.
\end{aligned}
\tag{2.5}
$$

The entries $\hat{b}_{rs}$ of B_{j+1} are given by the same formulas, replacing each a_{rs}, $\hat{a}_{rs}$ by b_{rs} and $\hat{b}_{rs}$, respectively.

Example 2.6. Let A_j, B_j be 7×7 matrices. Then for the pivot indices $(3, 6)$ and $(4, 5)$ the new iterates A_{j+1}, B_{j+1} change their entries as follows:



N.B. The ℓ's in this figure are given as l's in text.

respectively. Here x and l denote entries in the upper and lower triangular part of the matrices, respectively. The light shading indicates entries which are altered in this step. The dark shading marks entries in the lower triangular part, which are constructed as a combination of entries from the lower **and** upper triangular part of $A_{j+1}(B_{j+1})$.

If we had already achieved iterates A_{j+1}, B_{j+1}, which have only very small entries in their lower triangular parts, then we would not want to increase them again by combining them with considerable parts of possibly big entries from the upper triangulars.

For subdiagonal pivot elements, i.e., for elements with indices $(k+1, k)$, $k = 1, \cdots, n-1$, this never happens. With any choice of Schur transformation corresponding to subdiagonal pivot elements we get new iterates with a smaller Frobenius norm of the lower triangular parts. This is the motive for the Stewart and Charlier-Van Dooren methods to use only subdiagonal pivot elements.

The confinement to subdiagonal pivots makes it necessary to employ in any step the Schur rotation which is farthest away from the identity, measured in the Frobenius norm. For $B = I$, the standard eigenvalue problem, we take $Z_j = Q_j = G_{k_j,l_j}(\alpha_j, \beta_j)$. If there are two solutions to the elementary 2×2 problem, the rotation with the angle α_j of larger modulus has the larger off-diagonal element. It is called *outer rotation.* The rotation with the angle of smaller modulus is called *inner rotation.* Stewart's method uses outer rotations throughout. This in turn leads to a special order in which the

subdiagonal pivot elements are traversed, namely,

$$
\begin{array}{l}
(1,2),(2,3),(3,4),\cdots,(n-2,n-1),(n-1,n)\\
(1,2),(2,3),(3,4),\cdots,(n-2,n-1)\\
\vdots\\
(1,2),(2,3)\\
(1,2)
\end{array}
\tag{2.7}
$$

which is called a *forward sweep* and subsequently

$$
\begin{array}{l}
(n-1,n),(n-2,n-1),(n-3,n-2),\cdots,(2,3),(2,1)\\
(n-1,n),(n-2,n-1),(n-3,n-2),\cdots,(2,3)\\
\vdots\\
(n-1,n),(n-2,n-1)\\
(n-1,n)
\end{array}
\tag{2.8}
$$

which is called a *backward sweep.*

For a detailed motivation of these choices, see [37].

If $B \neq I$, then taking the angle α_j with larger absolute value causes Q_j to be farthest away from the identity, i.e., to be outer, but there is no guarantee that the corresponding Z_j will also be outer. It might be nearer to the identity than a $\tilde{Z}_j$ from a different generalized Schur rotation for this step (see Example (2.4b)). The Charlier–Van Dooren method, using the pivot ordering (2.7), (2.8), decides for the Q_j as outer rotations in forward sweeps and the Z_j as outer in backward sweeps. In a preparatory step the initial B is reduced to upper triangular form, which is preserved for all iterates B_j. It is shown in [8] that this method is convergent if $\gamma_j \notin [-\varepsilon, \varepsilon]$ through any forward sweep and $\alpha_j \notin [-\varepsilon, \varepsilon]$ through any backward sweep. (Note that we restricted the angles to $(-\pi/2, \pi/2]$.)

For the standard eigenvalue problem $B = I$, Eberlein [16] considers a traverse of all entries below the diagonal in a sweep. Looking at the formulas (2.5) we note that upper triangular elements appear in the new lower triangular elements $\hat{a}_{r,s}$ multiplied by a factor $s = e^{i\beta} \sin \alpha$ (here $s = q$ and $c = p$). Thus choosing s as small as possible, i.e., choosing the inner rotation, keeps the influence of the upper triangular elements as small as possible.

We closely follow the considerations in [16] for the development of a method for matrix pencils: In contrast to the Charlier–Van Dooren method we will traverse all entries below the diagonal in a sweep and use only inner rotation, either for all Q_j or all Z_j.

3. A Jacobi-like method for $A - \lambda B$. For the matrix pencil problem we see from (2.5) for a nonsubdiagonal pivot element, that entries from the upper triangular parts will enter the lower triangular parts with the multiplicative factors $q = e^{i\delta} \sin \gamma$ and $s = e^{i\beta} \sin \alpha$. As we want to keep the upper triangular entries from increasing the lower triangular parts, which might already be small, we would like to choose q and s as small as possible.

Unfortunately, once we decide for a special q or an s all the other quantities of the Schur rotation are essentially uniquely determined. A Q_j chosen to be the inner rotation may correspond to an outer Z_j and vice versa, if we are not already near

enough to the final form. Here we first confine ourselves to choose all Q_j as inner rotations. By *inner generalized Schur rotation* we will denote the generalized Schur rotation (Q, Z) for which Q is closest to the identity in the Frobenius norm. (Its computation is discussed later.) This leads to a first simple version of our algorithm. $\|\cdot\|_F$ denotes the Frobenius norm of a matrix and L_M denotes the strictly lower triangular part of a matrix M.

ALGORITHM 3.1.
INPUT
$A, B \in \mathbb{C}^{n\times n}$, $eps \in \mathbb{R}_+$ (deflation bound)
OUTPUT
The generalized Schur decomposition $A-\lambda B = U^H(T-\lambda S)V$. A, B is overwritten by the upper triangular matrices T and S, respectively.
Initialize $U := I$, $V := I$.
WHILE $\|L_A\|_F/\|A\|_F + \|L_B\|_F/\|B\|_F > eps$
  FOR $k = 1, \cdots, n-1$
    FOR $l = k+1, \cdots, n$.
      IF $|a_{kl}|+|b_{kl}| > 2eps$ THEN
        Compute the inner generalized Schur rotation $(G_{k,l}(\gamma, \delta), G_{k,l}(\alpha, \beta))$
        Compute $A := G_{k,l}^H(\gamma, \delta)AG_{k,l}(\alpha, \beta)$, $B := G_{k,l}^H(\gamma, \delta)BG_{k,l}(\alpha, \beta)$
                $U := UG_{k,l}(\gamma, \delta)$, $V := VG_{k,l}(\alpha, \beta)$
      END IF
    END FOR l
  END FOR k
END WHILE

Remark 3.2. We might also decide to choose $G_{k,l}(\alpha, \beta)$ always as inner rotation. This works as well. However, numerical tests have shown that mixing these two strategies does not work. We might, for instance, think of deciding in each step for the inner rotation which is nearest to the identity. Thus we might hope to keep the influence of the upper triangular part on the lower triangular part as small as possible. However, surprisingly, it turns out that all mixing of this kind leads to slow or no convergence at all. As of yet we have no theoretical explanation for this behavior.

The iterates of the algorithm will in most cases converge to a generalized Schur form (see § 4). But there are examples, where the iteration shows a cycling behavior.

Example 3.3 (see also [16]). Let

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ \varepsilon & 0 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ \eta & 0 & 1 \end{bmatrix}.$$

Applying Algorithm 3.1, we would have the first nontrivial step for the pivot pair (1, 3). The 2×2 subproblem to be solved is here:

$$\begin{bmatrix} 0 & 0 \\ \varepsilon & 0 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ \eta & 1 \end{bmatrix}.$$

Up to diagonal transformation it has the unique Schur decomposition

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \left(\begin{bmatrix} 0 & -\varepsilon \\ 0 & 0 \end{bmatrix} - \lambda \begin{bmatrix} 1 & -\eta \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

Thus here we have only one generalized Schur rotation for this step, $(G_{1,3}(\pi/2, 0), G_{1,3}(\pi/2, 0))$, i.e., inner and outer rotation coincide. Moreover, $Q = G_{1,3}(\pi/2, 0)$ is as far away from the identity as possible. Proceeding with Algorithm 3.1 we find that each subproblem is of this kind. After six sweeps the pencil is transformed back to the pencil we started with. So even if ε, η are small and $A - \lambda B$ is near to generalized Schur form already, we will not get convergence.

The crucial problem in the example above is that the rotations $G_{k,l}(\gamma, \delta)$ cannot be chosen "near to the identity" if we insist on eliminating the entry in position k, l. The large angle $\pi/2$ makes the entries above the diagonal keep coming back to the lower triangulars. Therefore it seems reasonable to avoid these large angles for off-subdiagonal pivots even if we have to relinquish the annihilation of the pivot elements. If each time the angle γ of the Schur rotation is $\pi/2$, we would set γ to $\pi/4$, then for the example above we would get convergence.

The following algorithm incorporates such a rule. If for a subdiagonal pivot the angle γ turns out to be $\pi/2$, then it is set to $\pi/4$. For an off-subdiagonal pivot, no γ with absolute value larger than $\pi/4$ is accepted. Recall that here we try to govern convergence by controlling the left-hand side rotations to be inner rotations, letting the right-hand side rotations be whatever they have to be then. Likewise we do not control the size of α, as we do for γ. In further, more detailed, experiments we could also try to incorporate such a rule.

ALGORITHM 3.4.

INPUT

$A, B \in \mathbb{C}^{n\times n}$, $eps \in \mathbb{R}_+$ (deflation bound), $\mu \leqq \pi/4$ (angle bound)

OUTPUT

The generalized Schur decomposition $A - \lambda B = U^H(T - \lambda S)V$. A, B is overwritten by the upper triangular matrices T and S, respectively.

Initialize $U := I$, $V := I$.

WHILE $\|L_A\|_F/\|A\|_F + \|L_B\|_F/\|B\|_F > eps$

  FOR $k = 1, \cdots, n-1$

    FOR $l = k+1, \cdots, n$.

      IF $|a_{kl}| + |b_{kl}| > 2eps$ THEN

        Compute the inner generalized Schur rotation $(G_{k,l}(\gamma, \delta), G_{k,l}(\alpha, \beta))$.

        IF $|\gamma| = \pi/2$ THEN $\gamma := \gamma/2$ END IF

        IF $|\gamma| \geqq \mu$ and $l > k+1$ THEN $\gamma := \gamma/2$ END IF

        Compute $A := G_{k,l}^H(\gamma, \delta)AG_{k,l}(\alpha, \beta)$, $B := G_{k,l}^H(\gamma, \delta)BG_{k,l}(\alpha, \beta)$

                $U := UG_{k,l}(\gamma, \delta)$, $V := VG_{k,l}(\alpha, \beta)$

      END IF

    END FOR l

  END FOR k

END WHILE

As usual the rotations are not computed via the angles. The nontrivial entries are computed directly. The two rotations in a generalized Schur rotation can be represented by just one parameter each, the tangents or cotangents of the angles. In a parallel implementation it is then only this pair of parameters which will have to be passed through the processors, which have to carry out the transformation.

In the following we describe the computation of the inner generalized Schur rotation.

The basic problem which has to be solved is the following.

Given $A, B \in \mathbb{C}^{2\times 2}$, find $c, s, p, q \in \mathbb{C}$ with $|p|^2+|q|^2=|c|^2+|s|^2=1$, such that

$$\begin{aligned}[-q \quad p]\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22}\end{bmatrix}\begin{bmatrix} c \\ s\end{bmatrix}&=0,\\ [-q \quad p]\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22}\end{bmatrix}\begin{bmatrix} c \\ s\end{bmatrix}&=0.\end{aligned} \tag{3.5}$$

For

$$Q=\begin{bmatrix} p & -\bar{q} \\ q & \bar{p}\end{bmatrix} \quad \text{and} \quad Z=\begin{bmatrix} c & -\bar{s} \\ s & \bar{c}\end{bmatrix}$$

the matrices Q^HAZ and Q^HBZ are then upper triangular. If we normalize Q and Z to have p and c real and nonnegative, then (Q, Z) is a generalized Schur rotation for A, B. We drop this normalization for the following.

If B is singular, then it is very easy to find all solutions of (3.5). If $\left[\begin{smallmatrix} a_{21} & a_{22} \\ b_{21} & b_{22}\end{smallmatrix}\right]$ or $\left[\begin{smallmatrix} a_{11} & b_{11} \\ a_{21} & b_{21}\end{smallmatrix}\right]$ are singular, then (3.5) immediately shows that a solution is given by $q=0$ or $s=0$, respectively. With $q=0$ the matrix Q is the identity and we have the desired inner generalized Schur rotation. The solution with $s=0$ might not be the inner one. The other generalized Schur rotation that possibly exists can be computed from the quantities below.

If $q\neq 0$ and $s\neq 0$ in (3.5), then the equations can be reformulated using the corresponding cotangent expressions. Setting $t:=p/q$ and $r:=c/s$ (3.5) implies

$$\begin{aligned} rta_{21}+ta_{22}-ra_{11}-a_{12}&=0,\\ rtb_{21}+tb_{22}-rb_{11}-b_{12}&=0.\end{aligned} \tag{3.6}$$

If t and r are available, then the corresponding nonnormalized generalized Schur rotation is given by

$$Q=\frac{1}{\sqrt{1+|t|^2}}\begin{bmatrix} t & -1 \\ 1 & \bar{t}\end{bmatrix} \quad \text{and} \quad Z=\frac{1}{\sqrt{1+|r|^2}}\begin{bmatrix} r & -1 \\ 1 & \bar{r}\end{bmatrix}. \tag{3.7}$$

The inner generalized Schur rotation is the one with largest $|t|$. Analogous to the Jacobi method for the standard eigenvalue problem, we can give formulas for t and r of the inner generalized Schur rotation.

THEOREM 3.8. *Let B, $\left[\begin{smallmatrix} a_{21} & a_{22} \\ b_{21} & b_{22}\end{smallmatrix}\right]$, and $\left[\begin{smallmatrix} a_{11} & b_{11} \\ a_{21} & b_{21}\end{smallmatrix}\right]$ be nonsingular. Define*

$$\begin{aligned} f&:=a_{22}b_{21}-b_{22}a_{21}, \qquad e:=a_{12}b_{11}-b_{12}a_{11},\\ d&:=[(a_{22}b_{11}-b_{22}a_{11})+(a_{12}b_{21}-b_{12}a_{21})]/2,\end{aligned}$$

and let D be chosen from $\{d+\sqrt{d^2+ef},\ d-\sqrt{d^2+ef}\}$ such that its absolute value is maximal. Then the inner generalized Schur rotation for A, B is given by

$$t_I=\frac{D}{f} \quad \text{and} \quad r_{t_I}=\frac{b_{12}-t_Ib_{22}}{t_Ib_{21}-b_{11}}.$$

If $D\neq 0$, then the outer generalized Schur rotation is given by

$$t_O=-\frac{e}{D} \quad \text{and} \quad r_{t_O}=\frac{b_{12}-t_Ob_{22}}{t_Ob_{21}-b_{11}}.$$

Proof. Let $\varepsilon := t - r$. Then (3.6) implies

$$\begin{aligned} t^2 a_{21} + t(a_{22} - a_{11}) - a_{12} - \varepsilon(ta_{21} - a_{11}) &= 0, \\ t^2 b_{21} + t(b_{22} - b_{11}) - b_{12} - \varepsilon(tb_{21} - b_{11}) &= 0. \end{aligned} \tag{3.9}$$

We first prove that

$$tb_{21} - b_{11} \neq 0. \tag{3.10}$$

If $tb_{21} = b_{11}$, then $b_{21} \neq 0$, because otherwise $b_{11} = 0$ and therefore $\det B = 0$. Thus $t = b_{11}/b_{21}$ and substituting this in the second equation of (3.9) we get $0 = b_{11}^2 + b_{11}(b_{22} - b_{11}) - b_{12}b_{21} = \det B$, contradicting our assumption.

From the second equation in (3.9) we now derive

$$\varepsilon = \frac{t^2 b_{21} + t(b_{22} - b_{11}) - b_{12}}{tb_{21} - b_{11}}. \tag{3.11}$$

Substituting this into the first equation in (3.9) and rearranging gives, with our definitions,

$$ft^2 - 2dt - e = 0, \tag{3.12}$$

where $f = \det \left[\begin{smallmatrix} a_{21} & b_{21} \\ a_{22} & b_{22} \end{smallmatrix}\right] \neq 0$. The solutions of (3.12) are

$$t_{1/2} = (d \pm \sqrt{d^2 + ef})/f. \tag{3.13}$$

Therefore $t_I = D/f$ is a solution of (3.12).

If $D = 0$ then it is the only solution and the Schur rotation for A, B is given by $t = 0$ and $r = b_{12}/b_{11}$. If $D \neq 0$, then $D = d + s\sqrt{d^2 + ef}$ for a suitable $s \in \{1, -1\}$, and the other solution of (3.13) is

$$t_O = \frac{d - s\sqrt{d^2 + ef}}{f} = -\frac{e}{d + s\sqrt{d^2 + ef}} = -\frac{e}{D}.$$

It remains to show that $|t_O| \leqq |t_I|$. But $2|D|^2 \geqq |d + \sqrt{d^2 + ef}|^2 + |d - \sqrt{d^2 + ef}|^2 = 2(|d|^2 + |\sqrt{d^2 + ef}|^2)$. Therefore $|D|^2 \geqq |d|^2 + |d^2 + ef| \geqq |d|^2 + \|d|^2 - |ef\|$, which implies $|D|^2 \geqq |ef|$. Thus $|t_O/t_I| = |ef/D^2| \leqq 1$. □

Formulas for the computation of generalized Schur rotations, where Z is an inner or outer rotation are derived completely analogously.

THEOREM 3.14. *Let B, $\left[\begin{smallmatrix} a_{21} & a_{22} \\ b_{21} & b_{22} \end{smallmatrix}\right]$, and $\left[\begin{smallmatrix} a_{11} & b_{11} \\ a_{21} & b_{21} \end{smallmatrix}\right]$ be nonsingular. Define*

$$\begin{aligned} f' &:= a_{21}b_{11} - b_{21}a_{11}, \qquad e' := a_{12}b_{22} - b_{12}a_{22}, \\ d' &:= [(a_{11}b_{22} - b_{11}a_{22}) + (a_{12}b_{21} - b_{12}a_{21})]/2, \end{aligned}$$

and let D' be chosen from $\{d' + \sqrt{d'^2 + e'f'},\ d' - \sqrt{d'^2 + e'f'}\}$ such that its absolute value is maximal. Then the generalized Schur rotation for A, B with Z as inner rotation is given by

$$r_I = \frac{D'}{f'} \quad \text{and} \quad t_{r_I} = \frac{r_I b_{11} + b_{12}}{r_I b_{21} + b_{22}}.$$

If $D' \neq 0$, then the generalized Schur rotation with Z as outer rotation is given by

$$r_O = -\frac{e'}{D'} \quad \text{and} \quad t_{r_O} = \frac{r_O b_{11} + b_{12}}{r_O b_{21} + b_{22}}.$$

Proof. The proof is analogous to the proof of Theorem 3.8. □

Thus for each iteration step in Algorithms 3.1 or 3.4, t_I and r_{t_I} have to be computed. Recall that not necessarily $|r_{t_I}| > |r_{t_O}|$, i.e., the corresponding Z will not necessarily be an inner rotation, and likewise for t_{r_I}, t_{r_O}. The generalized Schur rotation can be constructed from these quantities. Note that the solution Q, Z of (3.5) can also be reconstructed as

$$
\begin{aligned}
Q &= \frac{1}{\sqrt{1+1/|t_I|^2}} \begin{bmatrix} 1 & -1/\bar{t}_I \\ 1/t_I & 1 \end{bmatrix}, \\
Z &= \frac{1}{\sqrt{1+1/|r_{t_I}|^2}} \begin{bmatrix} 1 & -1/\bar{r}_{t_I} \\ 1/r_{t_I} & 1 \end{bmatrix}
\end{aligned} \tag{3.15}
$$

rather than according to (3.7).

4. Numerical test results. To test and compare the numerical convergence of the algorithms described in §§ 2 and 3 FORTRAN 77 versions of the algorithms were run on an HP 850 in double precision with machine precision 2.7×10^{-17}, mainly on randomly generated matrix pairs.

Obviously, the amount of work for a sweep in the Charlier–Van Dooren method (forward or backward sweep) is the same as for a sweep in Algorithms 3.1 and 3.4, and likewise one sweep of the Stewart method [37] involves the same number of operations as one sweep of Eberlein's method [16]. The number of sweeps necessary to achieve an acceptable approximation to generalized Schur form is the quantity we use for comparison of the methods here.

After each sweep (for the Charlier–Van Dooren method, and for the Stewart method, after each double sweep) the relative error

$$
relerr(M) = \frac{\sum_{k>l} |m_{kl}|^2}{\sum_{k,l=1}^{n} |m_{kl}|^2}
$$

was computed for the current iterates A, B. The program was terminated if

$$
relerr(A) + relerr(B) < 10^{-9}
$$

or the upper bound of 400 sweeps was reached.

As a result of a first-order backward error analysis we can show the following. If *eps* is the relative precision in which we compute (here it will be 10^{-9}), then the computed upper triangular matrices $\tilde{A}$, $\tilde{B}$ are exactly unitarily equivalent to slight perturbations of the initial matrices A_0 and B_0, i.e., there exist unitary matrices G and H such that

$$
\tilde{A} = G^H(A_0 + \Delta A)H \quad \text{and} \quad \tilde{B} = G^H(B_0 + \Delta B)H,
$$

where $\|\Delta A\|_F \leqq 37.2n(n-1)k\,eps\|A\|_F$, $\|\Delta B\|_F = 37.2n(n-1)k\,eps\|B\|_F$. Here k is the number of sweeps necessary for the termination of the program, i.e., to achieve $relerr(A) + relerr(B) < eps$. For the actual computed transformation matrices $\tilde{G}$ and $\tilde{H}$ we can then prove that $\|\tilde{G} - G\|_F$ and $\|\tilde{H} - H\|_F$ are at most $17.1n^{3/2}(n-1)k\,eps$. The algorithms are numerically backward stable in this sense and comparing the computed eigenvalues with those computed by the QZ algorithm shows the expected correspondence in the results.

The random matrices M were created by the following procedure, where random (x) is a random number generator, creating real numbers ranging from 0 to 99.999.

```
      DO 10 i = 1, n
        DO 10 j = 1, n
          m(i, j) = random (x)
10    CONTINUE
      factor = random (x)/75.0
      DO 20 i = 1, n
        DO 20 j = 1, n
          m(i, j) = m(i, j) − factor * random (x)
20    CONTINUE.
```

Here we report on the observations of typical behavior of our experiments, which are illustrated in the appended graphics.

For dimensions 10, 20, 30, and 40 we tested 90 randomly generated matrix pairs, each with the Algorithm 3.1[1] in § 3 and the Charlier–Van Dooren method.

Figure 1 is typical for the performance. It shows the absolute frequency of the number of sweeps necessary to terminate the programs for 90 randomly generated 10×10 matrices, e.g., the bar with the cross pattern at point 132 on the horizontal axis of length four means that four of the 90 matrix pairs converged after 132 sweeps with the Charlier–Van Dooren method. For 20 of the 90 matrix pairs the relative error was not yet small enough after 400 sweeps using this method. Algorithm 3.1, however, shows a normal distribution-like behavior, which we would like to see for random matrices. For larger dimensions the number of sweeps necessary to achieve convergence increases for both methods. The corresponding tables remain qualitatively the same.

Figure 2 shows a corresponding table for Algorithm 3.1 on 20×20 randomly generated matrix pencils and on Algorithm 3 in which Algorithm 3.1 is modified by using the pivot ordering given by (5.12), (5.10), which is suitable for a parallel computation as described in [16]. The Charlier–Van Dooren method's performance was similar to the one in Fig. 1 with more examples not yet having reached generalized Schur form after 400 sweeps. (A figure including it for this case was very unreadable. Therefore we omitted it here.) For these examples Algorithm 3 showed a slightly better convergence than Algorithm 3.1 with the columnwise cyclic pivot ordering.

Figure 3 shows the corresponding graphic for the standard eigenvalue problem $Ax = \lambda x$ using Stewart's method [37] and the method of Eberlein [16] with cyclic columnwise pivot ordering. Convergence is achieved faster for the standard eigenvalue problem with both methods. In particular, all examples terminated with the Stewart method before 400 sweeps were reached, the slowest one needing 380 sweeps. For larger dimensions, however, this does not hold. To keep the table readable we only show the number of sweeps up to 200.

For the random matrices the Charlier–Van Dooren and the Stewart methods show no apparent relation between the number of sweeps necessary for convergence and the dimension of the matrices.

Figures 4 and 5 display the average number of sweeps necessary for convergence for different dimensions using Eberlein's method or Algorithm 3.1, where for each dimension 20 random matrices or matrix pencils were tested. Greenstadt [24] observed the linear behavior of the first part of the curve in Fig. 4 but could at that time not continue the tests for larger dimensions, for which the curve flattens. Figure 5 shows that the corresponding curve for matrix pencils is less favorable.

[1] We also show results for a modification of Algorithm 3.1, which we call Algorithm 3. Algorithm 3 modifies Algorithm 3.1 only by using the pivot ordering given by (5.12) and (5.10). This is one of the orderings suitable for parallel computation.

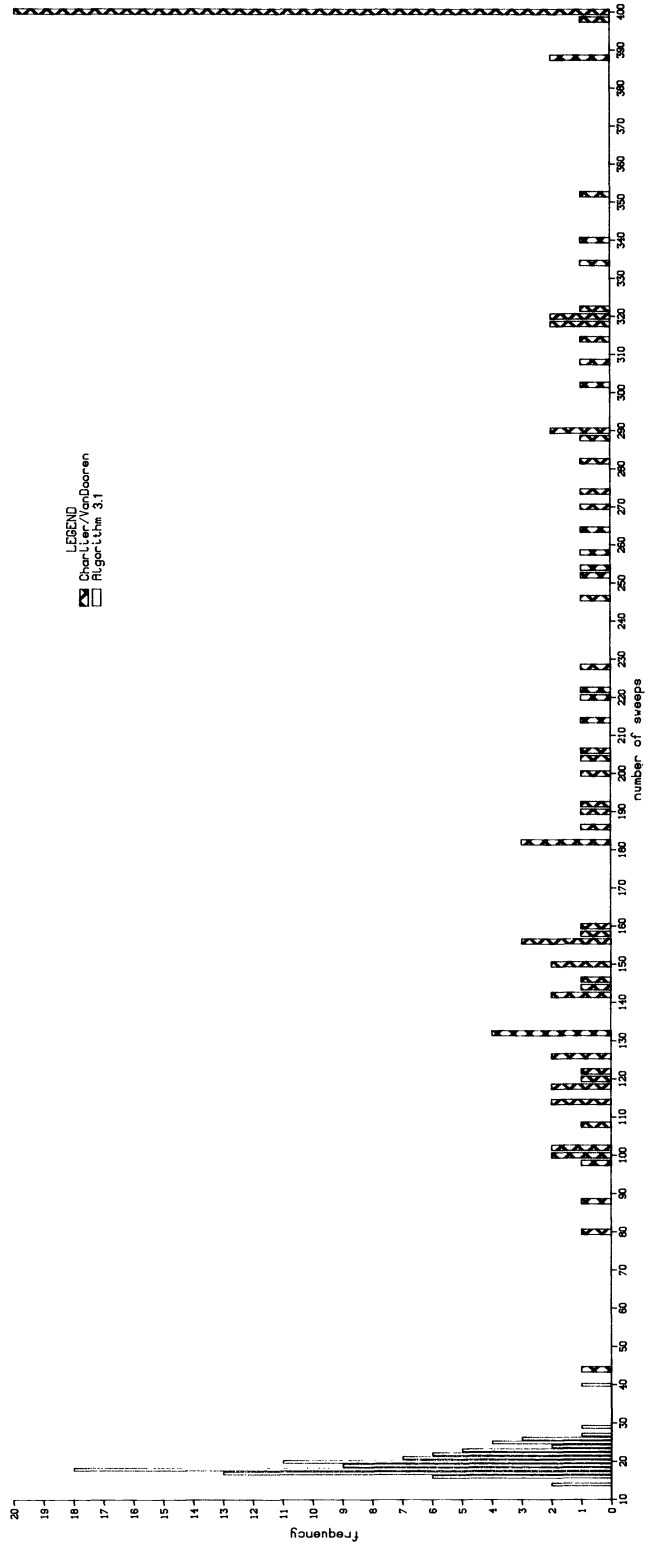


FIG. 1. *Ninety randomly generated* 10×10 *matrix pencils.*

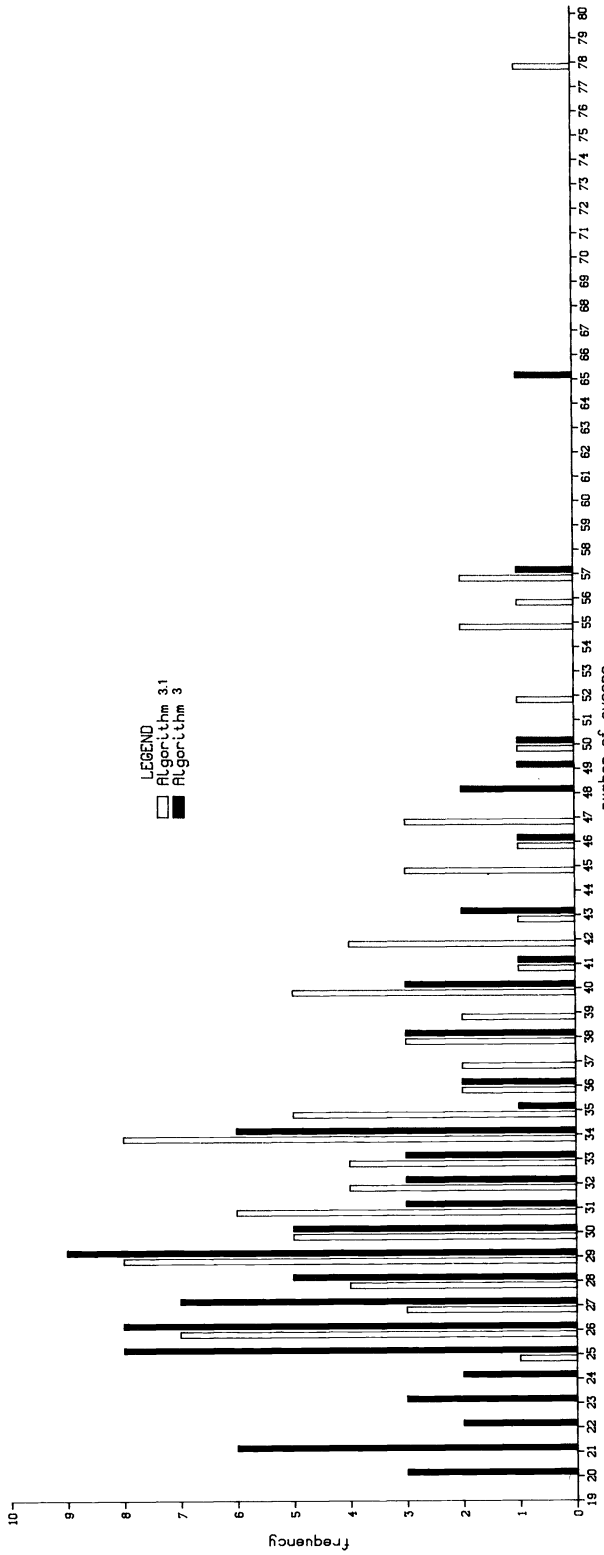


FIG. 2. *Ninety randomly generated* 20×20 *matrix pencils.*

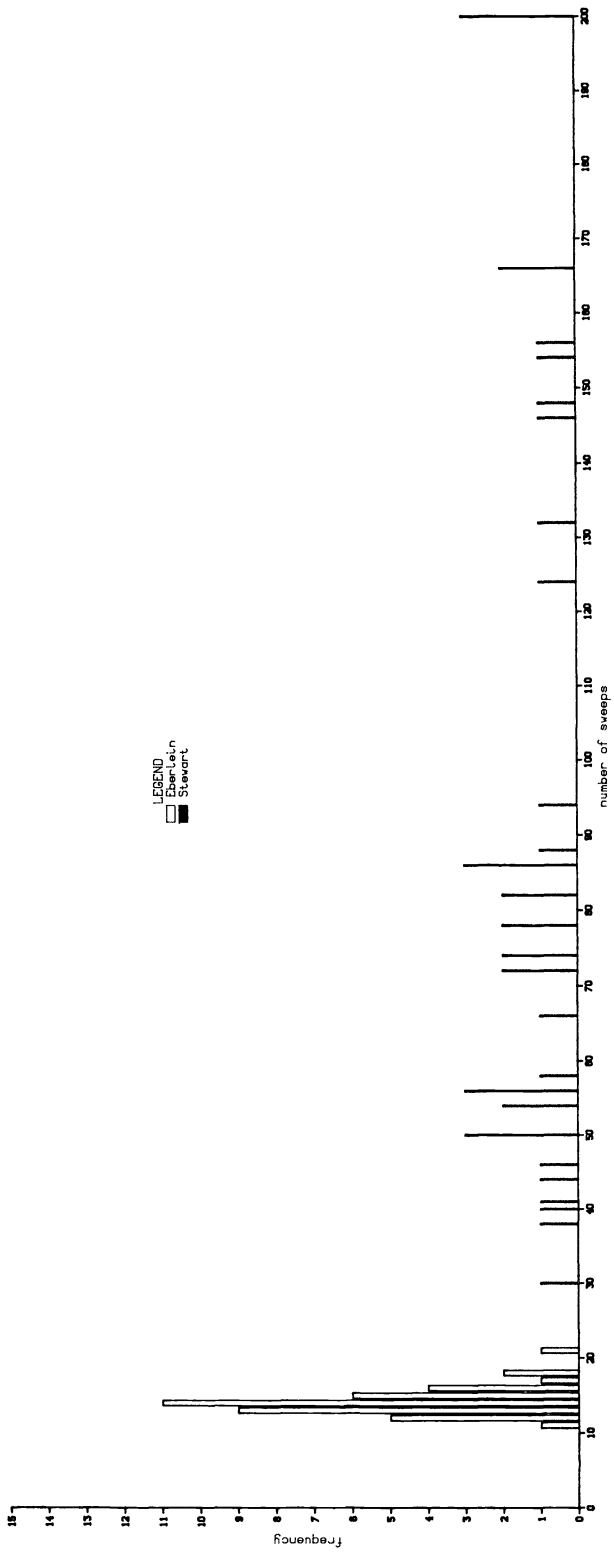


FIG. 3. *Forty randomly generated* 10×10 *matrices.*

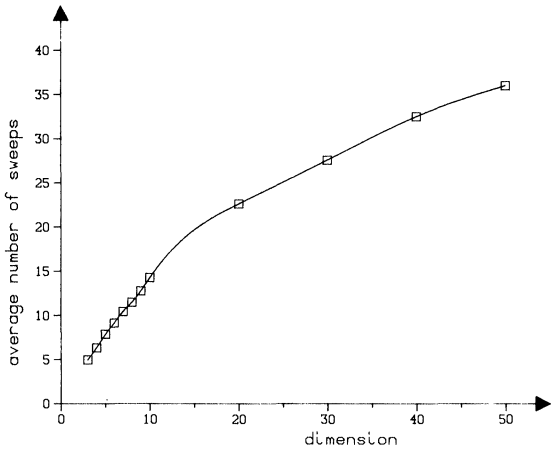


FIG. 4. *Algorithm* 3.1: *Standard eigenvalue problem.*

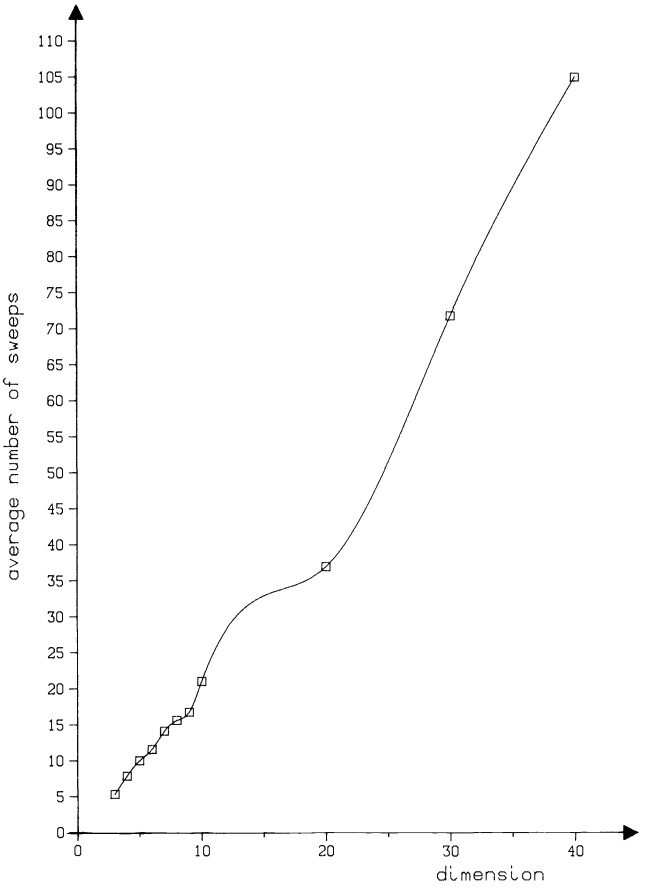


FIG. 5. *Algorithm* 3.1: *Generalized eigenvalue problem.*

If $relerr(A)+relerr(B)$ is chosen to be smaller than our bound 10^{-9}, then the number of sweeps necessary for convergence slightly increases, but the qualitative behavior remains the same.

Figure 6 again shows the absolute frequency of the number of sweeps necessary for convergence for 90 randomly generated matrix pencils using Algorithm 3.1, Algorithm 3.4, and Algorithm 3 in which Algorithm 3.1 is modified by using the pivot ordering given by (5.12), (5.10), suitable for a parallel computation as given in [16]. We see that there is no big essential difference in the behavior of the methods for these random matrices. Algorithm 3 shows a slightly better performance than the other two (see also Fig. 2).

A different set of random matrices was tested to investigate the dependence of the numerical performance on the problem's distance to normality.

The matrices A and B were constructed as

$$A = U(D_A + \alpha R_A)V, \qquad B = U(D_B + \alpha R_B)V,$$

where D_A, D_B and R_A, R_B are randomly generated diagonal and strictly upper triangular matrices, respectively. U and V result from orthogonalizing random matrices with the NAG routine F05AAF. For $\alpha = 0$ A, B can therefore simultaneously be diagonalized with the orthogonal matrices U and V. Note that this is a very restrictive assumption which will hardly ever be satisfied for practical problems. By enlarging α we depart from this normality.

Figures 7–9 show the performance for $\alpha = 0, 0.01, 0.1$ for 20 matrix pencils of dimension 10. In addition to the Charlier–Van Dooren method and Algorithm 3.1, we included here the modification of Algorithm 3.1 which instead of Q chooses Z always as inner rotation; this is called Algorithm Z in the figures. The method called Charlier–Van Dooren-Q is a modification of the Charlier–Van Dooren method which takes outer generalized Schur rotations throughout, i.e., Q is always outer. For $\alpha = 0.1$ with the Charlier–Van Dooren method, five examples did not converge within 400 sweeps. We only show the number of sweeps up to 200 to keep this table readable.

We see that for $\alpha = 0$ all methods perform equally well. Introducing a very small upper triangular part by setting $\alpha = 0.01$ causes an increase of the number of sweeps necessary for convergence which is relatively larger for the Charlier–Van Dooren methods. Setting $\alpha = 0.1$ already shows a picture like Fig. 1.

Figures 10–12 give the corresponding result for the standard eigenvalue problem treated with Stewart's and Eberlein's method. Here for $\alpha = 0.1$ two examples did not converge with Stewart's method within 400 sweeps. Again sweeps only up to 200 are displayed.

Finally, Figs. 13–16 show the development of the relative error $relerr(A)+relerr(B)$ during the processes depending on the parameter α for a typical 10×10 example. Note that in contrast to the Charlier–Van Dooren method for $\alpha = 0.1$, Algorithm 3.1 produces an error which for the first sweeps is not monotonically decreasing. In all examples, however, we observed that eventually the error decreases monotonically and slightly faster than linearly.

Remark 4.1. For ill-conditioned generalized eigenvalue problems convergence for Algorithms 3.1 and 3.4 is observed to slow down, as in the standard eigenvalue problem case (see [16], [24]).

5. Parallel implementation. Algorithms 3.1 and 3.4 from § 3 allow parallel computation following almost exactly the lines of those for the symmetric Jacobi method [3], [11]–[13], [16], [33], [41]. We will give a brief description of a possible implementation

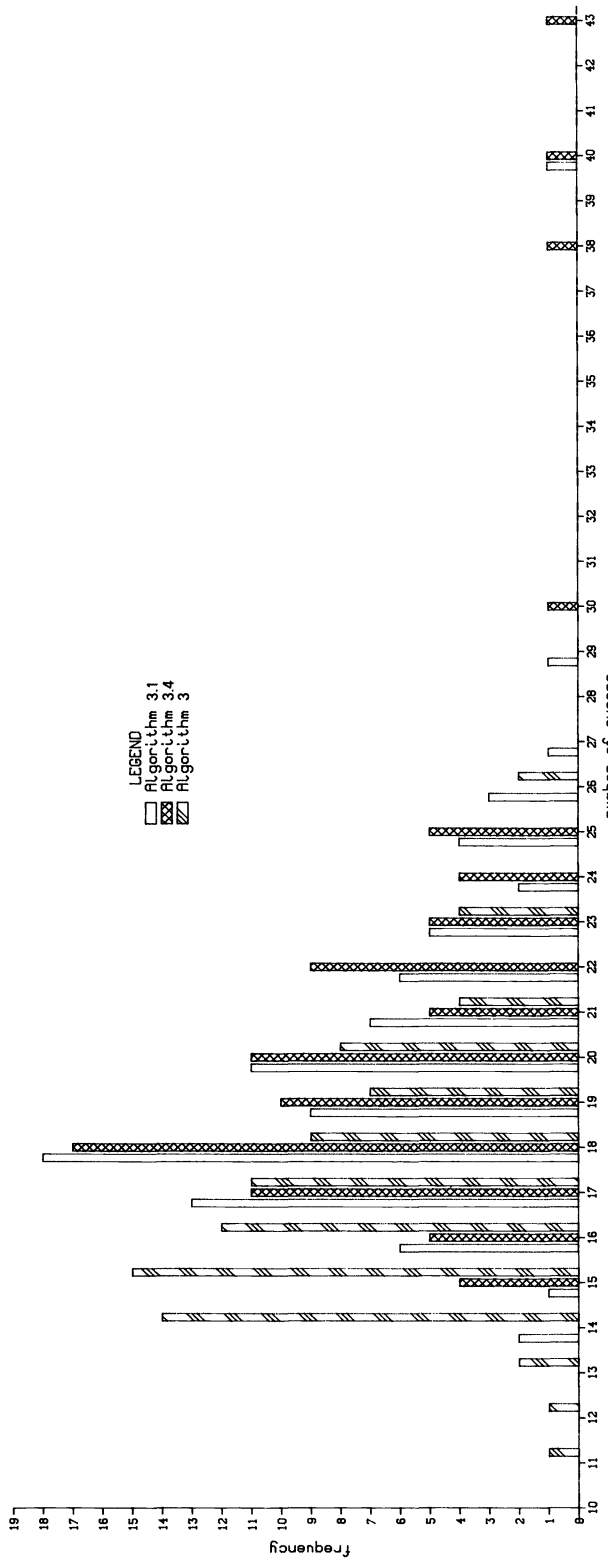


FIG. 6. *Ninety randomly generated* 10×10 *matrix pencils.*

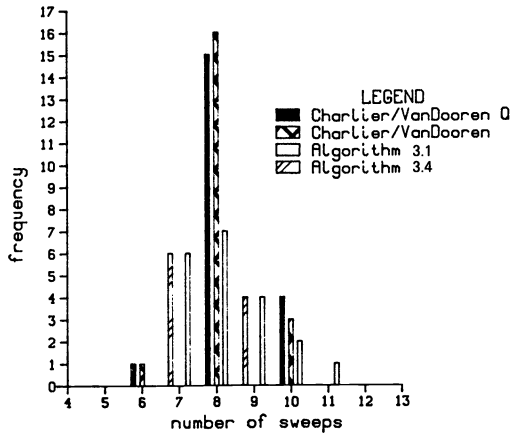


FIG. 7. *Twenty* 10×10 *matrix pencils alpha* $= 0$.

for a loop multiple processor system and for a mesh-connected grid of processors as introduced by Brent and Luk [3] for the symmetric Jacobi method.

As usual for an optimal parallel implementation (see [33]) the pivot index ordering must be altered such that in each step of the method $N = [n/2]$ $(= n/2$ for n even, $= (n-1)/2$ for n odd) generalized Schur rotations can be computed and performed simultaneously and each pivot index pair occurs exactly once in such a parallel sweep. Thus the pivot ordering has to be rearranged such that the pivot indices can be blocked into $n-1$ (n for n odd) index pairs

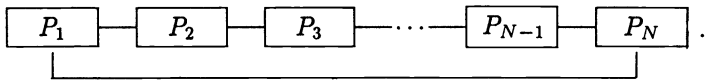
$$(k_1^j, l_1^j), \cdots, (k_N^j, l_N^j), \qquad j = 1, \cdots, n-1 \quad (n \text{ for } n \text{ odd}) \tag{5.1}$$

where $k_s^j < l_s^j$, each index occurs at most once in the jth block, and each pivot index pair (k, l), $k < l$, occurs exactly once in this sequence.

Then the generalized Schur transformation corresponding to (k_s^j, l_s^j) does not affect the 2×2 subproblem needed to compute the generalized Schur rotation corresponding to (k_r^j, l_r^j) if $r \neq s$; and the unitary matrices involved, $Q_s{}^j$, $Q_r{}^j$, and also $Z_s{}^j$, $Z_r{}^j$, commute. Thus the transformations can be carried out simultaneously. The pivot ordering should in addition be such that for the parallel architecture in question the communication costs are low.

We first consider a loop of N processors which can communicate with their direct neighbor

(5.2)



Such an architecture has been studied for symmetric Jacobi methods, for instance, in [41] and [13]. Here we follow [13] where an optimal implementation (in the sense above) is given with least communication between the processors.

For the following we assume that n is even. Otherwise we append one zero row and column to A and B and skip every generalized Schur rotation which would involve these zero rows and columns.

Let $(k_1^j, l_1^j), \cdots, (k_N^j, l_N^j)$, $j = 1, \cdots, n-1$ be a pivot ordering satisfying (5.1). If in the jth step we would assign columns k_s^j and l_s^j of the current iterates A, B and U, V to processor P_s then this processor can compute the generalized Schur rotation

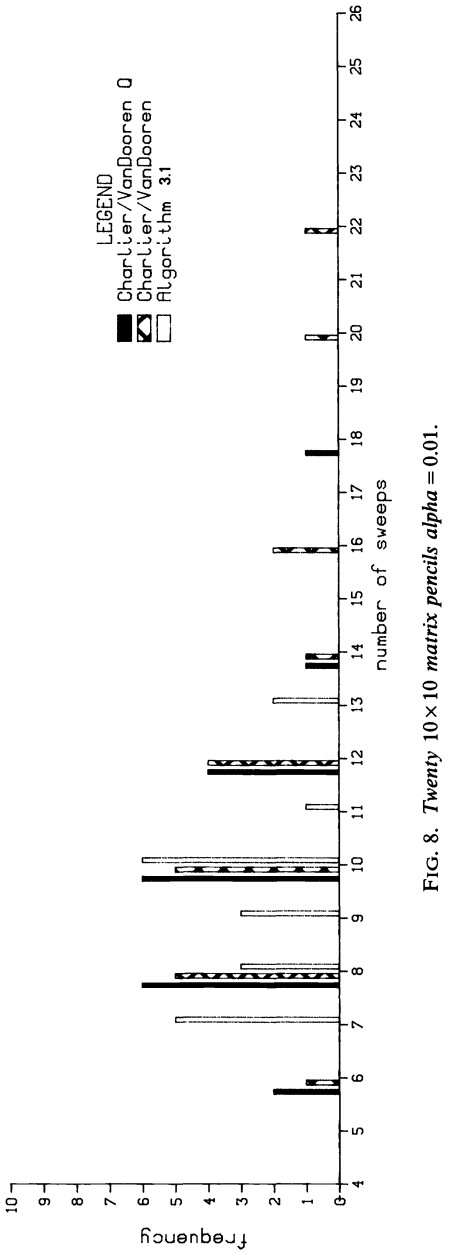


FIG. 8. *Twenty* 10×10 *matrix pencils alpha* $= 0.01$.

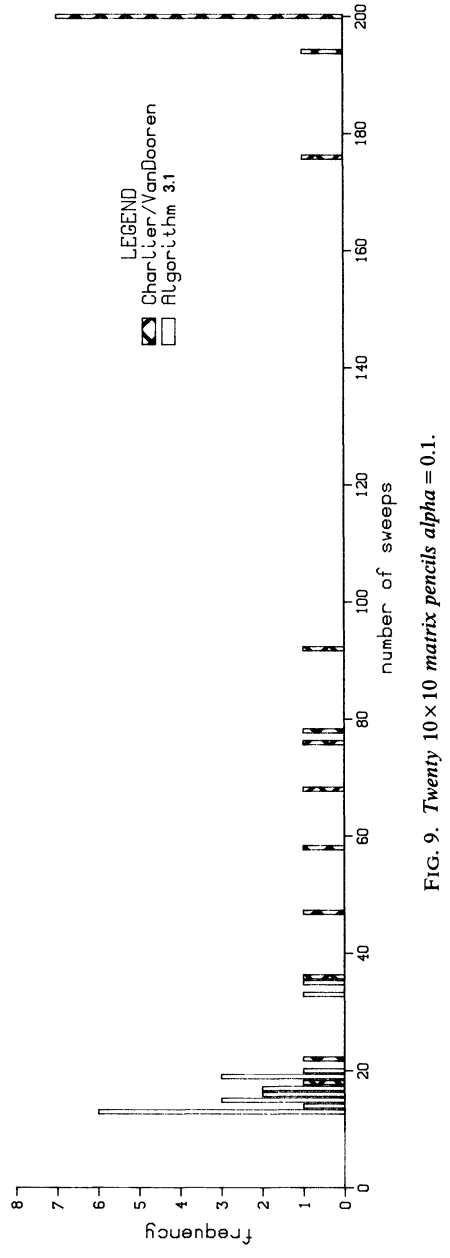


FIG. 9. *Twenty* 10×10 *matrix pencils alpha* $= 0.1$.

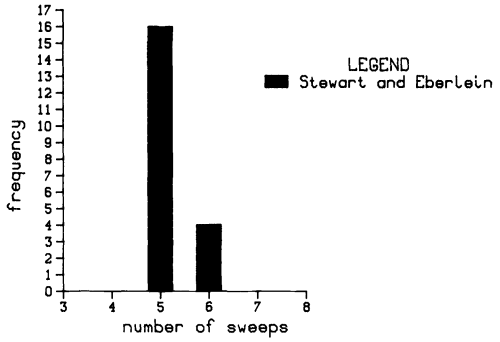


FIG. 10. *Twenty* 10×10 *matrices alpha* $= 0$.

$(Q_s{}^j, Z_s{}^j)$ corresponding to the pivot index pair (k_s^j, l_s^j). It also can perform the multiplications $AZ_s{}^j$, $BZ_s{}^j$, and $VZ_s{}^j$, $UQ_s{}^j$, because this only affects the columns kept in P_s. To complete this iteration step by the row modifications with $Q_s{}^j$, it would be necessary to send the cotangent parameter which determines $Q_s{}^j$ to each of the other processors. To avoid this, we confine ourselves to a *one-sided version* of our Jacobi-like method, i.e., we only carry out the column modification by multiplying from the right with $Z_s{}^j$. Thus Algorithms 3.1 or 3.4 will produce the sequences:

$$A_0 := A, \quad B_0 := B, \quad U_0 := I, \quad V_0 := I$$

and for $j = 1, \cdots, n-1$

$$\begin{aligned} A_j &= A_{j-1} Z_1{}^j \cdots Z_N{}^j, \qquad B_j = B_{j-1} Z_1{}^j \cdots Z_N{}^j, \\ U_j &= U_{j-1} Q_1{}^j \cdots Q_N{}^j, \qquad V_j = V_{j-1} Z_1{}^j \cdots Z_N{}^j. \end{aligned} \tag{5.3}$$

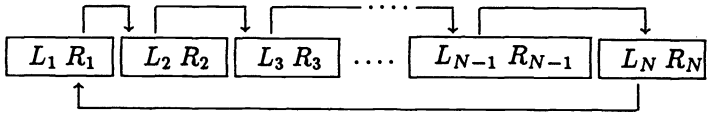
In the next step processor P_s, holding columns k_s^{j+1} and l_s^{j+1} of the current A, B, V, and U, can compute the 2×2 subproblem in the k_s^{j+1}, l_s^{j+1}-plane which determines the generalized Schur rotation, by forming the inner product of columns k_s^{j+1} and l_s^{j+1} of U with A and B. Then the generalized Schur rotation is computed and the columns in P_s are updated. No communication is necessary, once P_s has obtained its columns for the jth step. The number of operations in the jth step is exactly the same as for the original algorithm. But here we save sending the parameters for $Q_1{}^j, \cdots, Q_N{}^j$ through all processors.

A pivot ordering satisfying (5.1) can be created from the following rule of interchanging data between the processors after the jth step is completed.

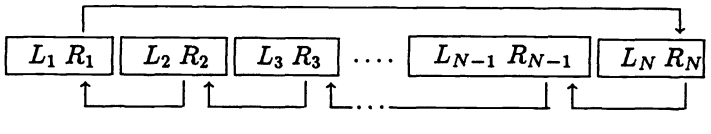
We assume that each processor P_s has registers, L_s and R_s. Initially, L_s will hold columns $2s-1$ and R_s columns $2s$ of A, B, U, V, i.e.,

$$(L_s, R_s) \leftarrow (2s-1, 2s) \quad \text{for } s = 1, \cdots, N. \tag{5.4}$$

After the completion of the $(j-1)$st step the columns are exchanged according to (5.5), (5.6):

(5.5) 

for j even,

(5.6) 

for j odd.

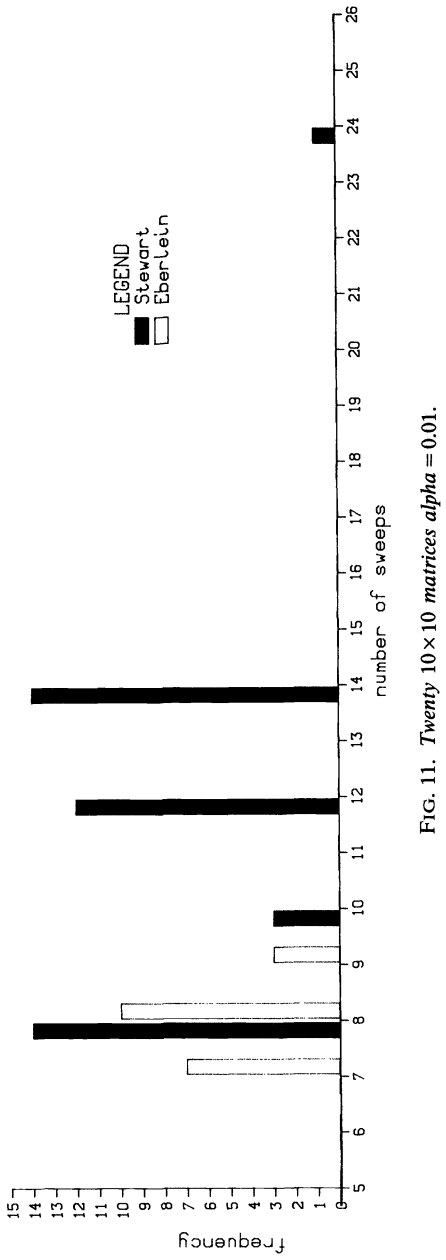


FIG. 11. *Twenty* 10×10 *matrices alpha* $= 0.01$.

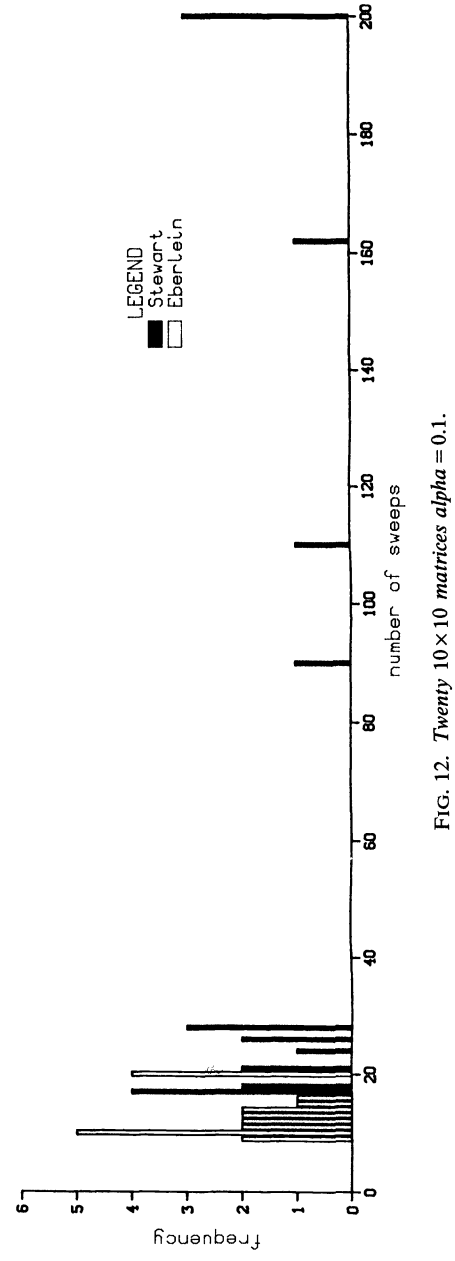


FIG. 12. *Twenty* 10×10 *matrices alpha* $= 0.1$.

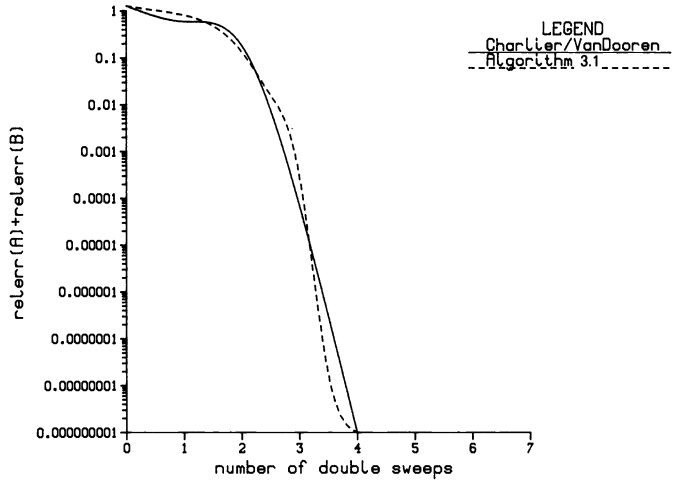


FIG. 13. 10×10 *matrix pencil alpha* $= 0$.

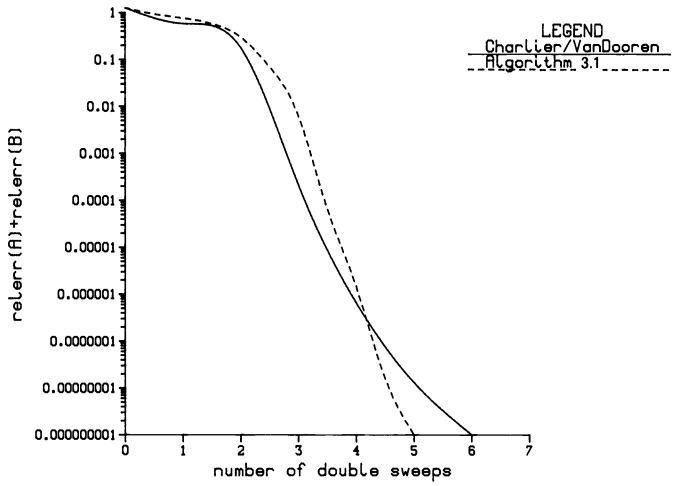


FIG. 14. 10×10 *matrix pencil alpha* $= 0.01$.

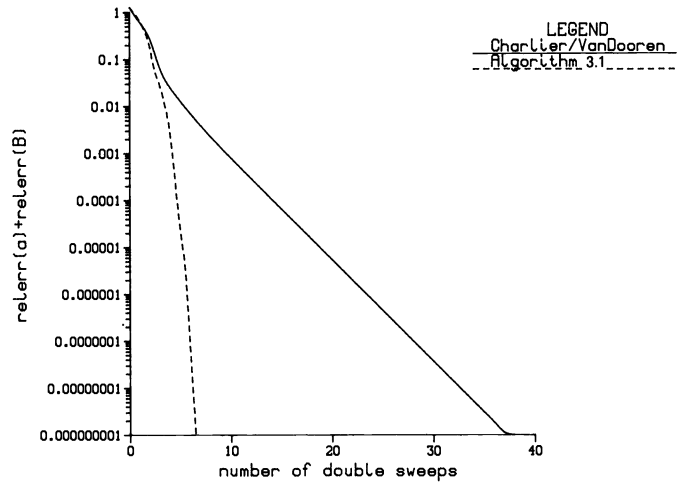


FIG. 15. 10×10 *matrix pencil alpha* $= 0.1$.

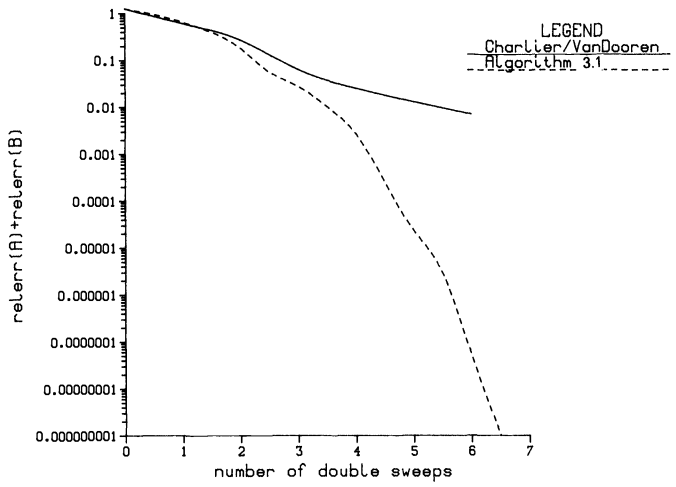


FIG. 16. 10×10 *matrix pencil alpha* $= 0.1$ (*blow up*).

Example 5.7. For $n = 8$ with the initial distribution (5.4) the interchange (5.5), (5.6) results in the following migration of pivot index pairs corresponding to the registers.

step				
1	1 2	3 4	5 6	7 8
2	1 7	2 4	3 6	5 8
3	1 4	2 6	3 8	5 7
4	1 5	4 6	2 8	3 7
5	1 6	4 8	2 7	3 5
6	1 3	6 8	4 7	2 5
7	1 8	6 7	4 5	2 3

step				
8	1 2	8 7	6 5	4 3
9	1 7	8 5	6 3	4 2
10	1 4	7 5	8 3	6 2
11	1 5	7 3	8 2	6 4
12	1 6	5 3	7 2	8 4
13	1 3	5 2	7 4	8 6
14	1 8	3 2	5 4	7 6

15	1 2	3 4	5 6	7 8

The example shows that the pivot index sequence does not quite satisfy (5.1). After $n-1$ parallel steps we are in step n back to the initial pairing of indices, but they correspond to different processors now. Moreover, except for the first pair, k_s is now assigned to an R and l_s to an L, i.e., their order is reversed. This holds true for the next steps, up to $2n-1$, where we again are back to the initial pairing, but now in the initial ordering of processors.

If an index pair (k_s^j, l_s^j), $k_s^j < l_s^j$, is assigned to a processor in reversed order, i.e., the columns l_s^j are contained in the left part and columns k_s^j in the right part, then the processor's task is slightly different. In this case it has to annihilate the l_s^jth entry of the vectors from A and B in the right part instead of the left. The processors therefore also need to get the index of the columns, such that they can distinguish between these two cases.

The distribution of $n/2$ index pairs on the processors in different order is irrelevant. It corresponds to a permutation of the matrices, but the same operations are performed. The program could for instance be terminated after an even number of sweeps only such that no final permutation of the matrices is necessary.

With these minor modifications the pivot index sequence created according to (5.5) and (5.6) can be shown to satisfy (5.1).

In each step of this parallel process $O(n)$ operations are necessary to alter the columns in each processor. Thus one sweep needs $O(n^2)$ timesteps and assuming that the number of sweeps necessary for convergence is $c \cdot n$ for some constant c (see Fig. 5 in § 4), on the whole $O(n^3)$ timesteps are needed.

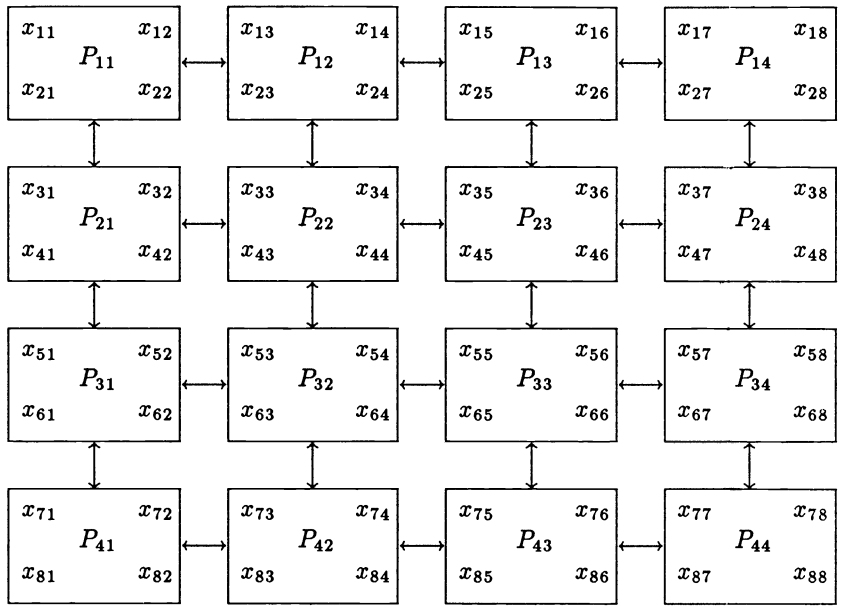
The time for altering the columns of the iterates can only be reduced by an order of magnitude, if $O(n^2)$ processors would be employed. Brent and Luk [3] suggested such a scheme for the symmetric Jacobi method. They considered a square array of $n/2 \times n/2$ processors P_{ij} with horizontal and vertical communication connections. The ideas used in [3] apply to Algorithms 3.1 and 3.4 as well (see also [16]).

Each processor P_{kl} initially holds four 2×2 submatrices

$$\begin{pmatrix} x_{2k-1,2l-1} & x_{2k-1,2l} \\ x_{2k,2l-1} & x_{2k,2l} \end{pmatrix},$$

where x stands for entries from A, B, U, V. The $x_{2k-1,2l-1}$ and $x_{2k,2l-1}$ are kept in registers L_{kl} and $x_{2k-1,2l}$ and $x_{2k,2l}$ in R_{kl}. As before, a zero row and column is added for odd dimension. Here it is advisable to add them in front. We illustrate the initial state for an $n = 8$ example:

(5.8)

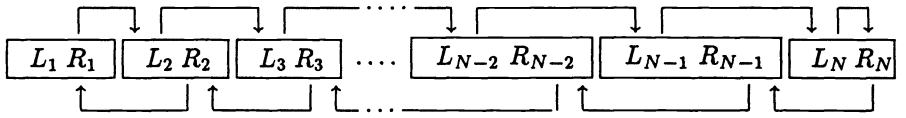


Each diagonal processor is assumed to compute the parameters for the generalized Schur rotation (Q, Z) by solving the 2×2 subproblems of A, B which they are holding. For the initial distribution this corresponds to the $n/2$ pivot index pairs

$$(1, 2), (3, 4), (5, 6) \cdots (n-1, n). \tag{5.9}$$

From each diagonal processor the computed parameters for Q and Z are passed through the processors in the same row and column, respectively. After having received its parameters each processor performs the corresponding operations on the 2×2 submatrices of A, B, U, and V, which it contains. After all operations are completed the processors interchange their data and proceed in the same way. As before, the interchanging creates a pivot index sequence which can be shown to satisfy (5.1) [3].

It corresponds to the following rule of interchange of indices, where we use the notation of (5.5), (5.6) (as before $N = n/2$)

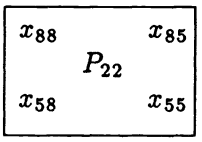
(5.10) 

The data is interchanged according to (5.10) among the processors in each row and among processors in each column, such that the N diagonal processors always contain the 2×2 subproblems corresponding to the pivot index pairs.

Example 5.11. For $n = 8$ with the initial distribution (5.8) the interchange (5.10) results in the following migration of pivot index pairs:

step				
0	1 2	3 4	5 6	7 8
1	1 4	2 6	3 8	5 7
2	1 6	4 8	2 7	3 5
3	1 8	6 7	4 5	2 3
4	1 7	8 5	6 3	4 2
5	1 5	7 3	8 2	6 4
6	1 3	5 2	7 4	8 6
7	1 2	3 4	5 6	7 8

Note that here again pivot index pairs occur in reversed order, e.g., in step 4 of the example above we get the pair (8, 5). For the 4×4 grid of processors that means that in step 4 processor P_{22} holds the 2×2 submatrix in the (5, 8) plane in reversed order, i.e.,



The task for processor P_{22} is now to compute a generalized Schur rotation which eliminates a_{85} and b_{85} which now are in registers R_{22}.

After $n-1$ steps the initial data distribution returns with this interchanging of data. One such parallel step of a sweep now only needs $O(1)$ timesteps. Even if communication costs are also considered, one sweep will only need $O(n)$ timesteps if the steps are nested as described in [3]. Thus, on the whole, $O(n^2)$ timesteps are necessary to compute the generalized Schur decomposition, if we assume that $c \cdot n$ sweeps are needed.

Instead of (5.9) we could start with any other pairing of pivot indices satisfying the condition (5.1) for $j = 1$. This corresponds to permuting the initial matrices and running the method on this permuted problem. The special pivot index ordering we chose might influence the speed of convergence (see [18]). For some of the examples

in § 4 we tried the initial pairing

$$(k, l) \quad \text{for } k = 1, \cdots, N \quad \text{with } l = n+1-k \tag{5.12}$$

suggested in [16].

Example 5.13. For $n = 8$ we get the following pivot ordering

step				
0	1 8	2 7	3 6	4 5
1	1 7	8 6	2 5	3 4
2	1 6	7 5	8 4	2 3
3	1 5	6 4	7 3	8 2
4	1 4	5 3	6 2	7 8
5	1 3	4 2	5 8	6 7
6	1 2	3 8	4 7	5 6
7	1 8	2 7	3 6	4 5

i.e., the entries on the second diagonal are eliminated first.

For these examples this pivot ordering showed a slightly better convergence than the cyclic ordering (see Figs. 2 and 6 in § 4).

6. Conclusion. We presented a Jacobi-like method for the computation of the generalized Schur decomposition of a matrix pencil $A - \lambda B$, which generalizes the Jacobi-like method for unsymmetric matrices by Eberlein [16]. We ran numerical experiments comparing the performance of this method with a Jacobi-like method recently developed by Charlier and Van Dooren [8], which is a generalization of the unsymmetric Jacobi-like method by Stewart [37]. It turns out that we are in a curious situation:

For the Charlier–Van Dooren algorithm a convergence result can be proved, whereas for the method here, as for the Eberlein method in the case of single matrices, we have no convergence theory yet. However, the performance of the latter method on the experimental data was much superior to the Charlier–Van Dooren method for pencils which are not almost normal pencils. We briefly showed that our method allows a parallel implementation on a ring of $n/2$ processors and on an $(n/2) \times (n/2)$ square array of mesh-connected processors in complete analogy to the case of single matrices.

The convergence of the algorithm as it stands is still too slow to be competitive with an efficiently implemented serial QZ algorithm and there are still many open questions. However, the results given here clearly demonstrate the significantly different behavior of the two approaches to develop a Jacobi method for arbitrary matrix pencils. It might thus lead to a direction for further development and research on parallelizable methods to solve the generalized eigenvalue problem.

REFERENCES

[1] M. BERRY AND A. SAMEH, *Multiprocessor Jacobi algorithms for dense symmetric eigenvalue and singular value decompositions*, in Proc. 1986 Internat. Conference on Parallel Processing, August 1986.

[2] ———, *A parallel algorithm for the singular value and dense symmetric eigenvalue problem*, Tech. Report 761, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, March 1988.

[3] R. P. BRENT AND F. T. LUK, *The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 69–84.
[4] R. P. BRENT, F. T. LUK, AND C. F. VAN LOAN, *Computation of the singular value decomposition using mesh-connected processors*, J. VLSI Comput. Syst., 1 (1984), pp. 242–270.
[5] C. BISCHOF, *A parallel ordering for the block Jacobi method on a Hypercube architecture*, Tech. Report 96-740, Department of Computer Science, Cornell University, Ithaca, NY, 1986.
[6] R. BYERS, *A Hamiltonian Jacobi algorithm*, to appear.
[7] R. L. CAUSEY, *Computing eigenvalues of non-Hermitian matrices by methods of Jacobi type*, J. Soc. Indust. Appl. Math., 6 (1958), pp. 172–181.
[8] J.-P. CHARLIER AND P. VAN DOOREN, *A Jacobi-like algorithm for computing the generalized Schur form of a regular pencil*, to appear.
[9] K. CHEN AND K. IRANI, *A Jacobi algorithm and its implementation on parallel computers*, in Proc. 18th Annual Allerton Conference on Communication Control and Computing, 1980.
[10] R. O. DAVIES AND J. J. MODI, *A direct method for computing eigenproblem solutions on a parallel computer*, Linear Algebra Appl., 77 (1986), pp. 61–74.
[11] P. P. M. DE RIJK, *A one-sided Jacobi algorithm for computing the singular value decomposition on a vector computer*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 359–371.
[12] P. J. EBERLEIN, *On one-sided Jacobi methods for parallel computation*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 790–796.
[13] ———, *On using the Jacobi method on the hypercube*, Tech. Report 86-23, Department of Computer Science, State University of New York, Buffalo, NY, 1986.
[14] ———, *A Jacobi method for the automatic computation of eigenvalues and eigenvectors of an arbitrary matrix*, J. Soc. Indust. Appl. Math., 10 (1962), pp. 74–88.
[15] ———, *Solution to the complex eigenproblem by a norm-reducing Jacobi-type method*, Numer. Math., 14 (1970), pp. 232–245.
[16] ———, *On the Schur decomposition of a matrix for parallel computation*, IEEE Trans. Comput., 36 (1987), pp. 167–174.
[17] P. J. EBERLEIN AND J. BOOTHROYD, *Solution to the eigenproblem by a norm-reducing Jacobi-type method*, Numer. Math., 11 (1968), pp. 1–12.
[18] P. J. EBERLEIN AND M. MANTHARAM, *Empirical effects of various Jacobi set orderings on convergence properties of Jacobi-like methods*, to appear.
[19] P. J. EBERLEIN AND H. PARK, *Efficient implementation of Jacobi algorithms and Jacobi sets on distributed memory architectures*, J. Parallel Distrib. Comput., Special Issue on Algorithms for Hypercube Computers, to appear.
[20] H. H. GOLDSTINE, *A footnote to a recent paper*, J. Assoc. Comput. Mach., 7 (1960), pp. 78–79.
[21] H. H. GOLDSTINE AND L. P. HORWITZ, *A procedure for the diagonalization of normal matrices*, J. Assoc. Comput. Mach., 6 (1959), pp. 176–195.
[22] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, North Oxford Academic, Oxford, 1983.
[23] J. GREENSTADT, *A method for finding roots of arbitrary matrices*, Math. Tables Aids Comput., 9 (1955), pp. 47–52.
[24] ———, *Some numerical experiments in triangularizing matrices*, Numer. Math., 4 (1962), pp. 187–195.
[25] V. HARI, *A Jacobi-like eigenvalue algorithm for general real matrices*, Glas. Mat. Ser. III, 11 (1976), pp. 3–5.
[26] P. HENRICI, *On the speed of convergence of cyclic and quasi-cyclic Jacobi methods for computing eigenvalues of Hermitian matrices*, J. Soc. Indust. Appl. Math., 6 (1958), pp. 144–162.
[27] G. G. J. JACOBI, *Über ein leichtes Verfahren, die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen*, J. Reine Angew. Math., 30 (1846), pp. 51–95.
[28] F. T. LUK, *A triangular processor array for computing singular values*, Linear Algebra Appl., 77 (1986), pp. 259–273.
[29] F. T. LUK AND H. PARK, *A proof of convergence for two parallel Jacobi* SVD *algorithms*, IEEE Trans. Comput., 38 (1989), pp. 806–811.
[30] C. B. MOLER AND G. W. STEWART, *An algorithm for generalized matrix eigenvalue problems*, SIAM J. Numer. Anal., 10 (1973), pp. 241–256.
[31] M. H. C. PAARDEKOOPER, *An eigenvalue algorithm based on norm-reducing transformations*, Ph.D. thesis, Tech. Hoogesch. Eindhoven, Eindhoven, the Netherlands, 1969.
[32] A. RUHE, *On the quadratic convergence of a generalization of the Jacobi method for arbitrary matrices*, BIT, 8 (1968), pp. 210–231.
[33] A. H. SAMEH, *On Jacobi and Jacobi-like algorithms for a parallel computer*, Math. Comp., 25 (1971), pp. 579–590.

[34] A. SCHÖNHAGE, *Zur Konvergenz des Jacobi-Verfahrens*, Numer. Math., 3 (1961), pp. 374–380.
[35] ———, *Zur quadratischen Konvergenz des Jacobi-Verfahrens*, Numer. Math., 5 (1964), pp. 410–412.
[36] D. S. SCOTT, M. T. HEATH, AND R. C. WARD, *Parallel block Jacobi eigenvalue algorithms using systolic arrays*, Linear Algebra Appl., 77 (1986), pp. 345–355.
[37] G. W. STEWART, *A Jacobi-like algorithm for computing the Schur decomposition of a non-Hermitian matrix*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 853–864.
[38] H. P. M. VAN KEMPEN, *On the convergence of the classical Jacobi-method for real symmetric matrices with non-distinct eigenvalues*, Numer. Math., 9 (1966), pp. 11–18.
[39] ———, *On the quadratic convergence of the special cyclic Jacobi-method*, Numer. Math., 9 (1966), pp. 19–22.
[40] K. VESELIC, *A convergent Jacobi method for solving the eigenproblem of arbitrary real matrices*, Numer. Math., 25 (1976), pp. 179–184.
[41] R. A. WHITESIDE, N. S. OSTLUND, AND R. G. HIBBARD, *A parallel Jacobi diagonalization algorithm for a loop multiple processor system*, IEEE Trans. Comput., 33 (1984), pp. 409–413.
[42] J. H. WILKINSON, *Note on the quadratic convergence of the cyclic Jacobi process*, Numer. Math., 4 (1962), pp. 296–300.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 940-949, July 1991
© 1991 Society for Industrial and Applied Mathematics
013

MULTIGRID TREATMENT OF "THIN" DOMAINS*

V. MIKULINSKY†

Abstract. Several attempts have been made to apply multigrid methods efficiently to problems on "thin" domains. In [J. Ruge and A. Brandt, "A multigrid approach for elasticity problems on 'thin' domains," in *Multigrid Methods: Theory, Applications and Supercomputing*, S. F. McCormick, ed., Marcel Dekker, New York, 1988, pp. 541-555] a multigrid method is described that permits overcoming difficulties in the case of elasticity equations on a "thin" domain bounded by straight lines. Here a method is presented that can be used for a wide class of differential problems on "thin" domains with straight or curved boundaries. For this method applied to the equations of elasticity on a domain bounded by straight lines, the computer time and storage are lower, and the programming and extension to three-dimensional cases are easier, than for the method described by Ruge and Brandt.

Key words. multigrid, relaxation, fine grid, coarse grid, residuals

AMS(MOS) subject classifications. 65N05, 65N20

1. Introduction.

Consider, for example, the following two-dimensional problem:

$$\Delta u = F \quad \text{on } [0, 1]\times[0, w], \tag{1}$$

where the domain is "thin," i.e., $w \ll 1$. Denote the meshsize of the grid on which the problem is discretized by h. We will study the case where $h \leqq w \leqq 2h$ since, even if w is larger, such a relation will eventually arise on one of the coarser grids when the problem is solved by a multigrid algorithm.

At neighborhoods where Dirichlet boundary conditions are given, convergence of the Gauss-Seidel relaxation scheme is very fast. This can be proved, for example, by a local mode analysis as in [1] or [2], or by the maximum principle. However, at neighborhoods where Neumann boundary conditions are given, convergence is slow. The phenomenon can be explained in the following way. Consider a smooth error $v(x, y) = v(x)$, that depends only on x. In this case, Neumann boundary conditions on the horizontal boundaries are satisfied exactly. Now the residual norm associated with this error is small compared with the error norm (see [3]), hence, this type of error cannot be reduced efficiently by local relaxation. According to the multigrid philosophy, this type of error should be reduced by the coarse grid correction, but the usual coarsening is impossible.

In [5], a coarsening for the equations of elasticity in "thin" domains has been described. It is limited to those equations and to domains bounded by straight lines. In this article, we suggest a new approach in which a one-dimensional grid should be used in order to obtain the coarse grid correction. Using the fact that the domain is "thin" (in the vertical direction, for example) and using the boundary conditions, we can eliminate terms including derivatives in the horizontal direction and obtain ordinary differential equations. It will be shown that this approach is applicable to a wide class of problems, including perhaps all "stable" elliptic problems, and that it yields a high multigrid efficiency. Our concept of stability will be described in § 2. In § 3, we will

* Received by the editors July 5, 1989; accepted for publication (in revised form) June 5, 1990. This research was supported by the Air Force Office of Scientific Research, United States Air Force, under grants AFOSR-86-0126 and AFOSR-86-0127, and by the National Science Foundation under grant DMS-87041169.

† Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel.

present the approach for two-dimensional differential equations with Neumann boundary conditions in domains bounded by straight lines. The approach will be extended in § 4 to equations bounded by curved lines with general linear boundary conditions, in § 5 to nonscalar partial differential equation (PDE) systems, and in § 6 to three-dimensional cases. In fact, the main advantage of the methods worked out here is that they admit three-dimensional analogues: The two-dimensional problems in "thin" domains could by themselves also be solved effectively by Gauss elimination.

2. Stability assumptions.

2.1. *h*-ellipticity. Consider the following discrete problem:

$$L^h u^h = F^h \quad \text{on } [0,1]\times[0,2h], \tag{2a}$$

where the grid is uniform, L^h is a discrete form of a differential operator L, and

$$L^h u^h = a_0(x)u^h_{xx} + b_0(x)u^h_{yy} + c_0(x)u^h_{xy} + d_0(x)u^h_x + e_0(x)u^h_y + f_0(x)u^h, \tag{2b}$$

subject to

$$u^h_y = a_1(x)u^h_x + b_1(x)u^h + c_1(x) \quad \text{on the lower boundary} \tag{3a}$$

and

$$u^h_y = a_2(x)u^h_x + b_2(x)u^h + c_2(x) \quad \text{on the upper boundary.} \tag{3b}$$

Here, terms like u^h_y and u^h_x represent suitable divided differences; in this section we do not specify the discretization. For simplicity we consider the case of a scalar equation (that is, u^h approximates a scalar function) and Dirichlet boundary conditions on the right and left vertical boundaries. This assumption makes analysis easier, but it is not necessary. In the framework of this study we consider only well-posed elliptic problems. Of course, if (2) is not elliptic, boundary conditions (3) specify an ill-posed problem. The natural requirement that should be imposed is *h-ellipticity* of equation (2a), a measure of which is defined as follows:

$$E^h(L^h) = \min |L^h(\underline{\theta})| / \|L^h(\underline{\theta})\|. \tag{4}$$

Here the complex function $L^h(\theta)$ is the *symbol* of L^h, i.e., $L^h e^{i\underline{\theta}\underline{x}/h} = L^h(\theta)\, e^{i\underline{\theta}\underline{x}/h}$, where $\underline{\theta} = (\theta_1, \theta_2)$, $\underline{x} = (x, y)$, $|L^h| = \max |L^h(\theta)|$, and $\pi \geqq |\theta| \geqq \pi/2$. When $E^h \approx O(1)$, the problem is h-elliptic. More details and explanations about h-ellipticity are presented in [1, § 2.1].

2.2. Convergence of relaxation. Not all h-elliptic problems are well posed. Multigrid solvers for ill-posed problems of different types were presented in [4]. In the framework of this study we treat difficulties related to "thin" domains. In order to deal with well-posed problems we introduce an additional stability requirement, namely, that highly oscillating components of the error lead to big residuals and therefore, according to [3], they can be reduced effectively by local relaxation processes.

Specifically, let v^h be an error and $r^h = L^h v^h$ its corresponding residual. Denote

$$E = \sum_i |v_i|^2 \quad \text{and} \quad R = \sum_i \left(|r_i|^2 \Big/ \sum_j |a_{ij}|^2 \right).$$

These definitions are taken from [3]. According to [3] the Kaczmarz relaxation process converges quickly when $E \sim R$. For $R \ll E$ convergence can be very slow. We assume that $E \sim R$ for all highly oscillating components of the error. As a result, these components of the error can be reduced by the relaxation process.

2.3. Eigenvalues of the problem. Another type of difficulty is connected with small eigenvalues of the problem corresponding to smooth components of the error. The following example illustrates the difficulty. Consider the Laplace equation ($\Delta u = 0$) in the domain $D = [0, \pi] \times [0, w]$, subject to

$$u_y - mu = 0 \tag{5}$$

on the horizontal boundaries and $u(0, y) = u(\pi, y) = 0$, where m is an integer. The solution is not unique: $u(x, y) = A \sin(mx) \exp(-my)$ solves the equation for any value of A.

Let λ^h and λ^H be the smallest eigenvalues of the respective fine and coarse grid operators. According to [4] the convergence factor of the usual multigrid cycle is small if and only if

$$|1 - \lambda^h/\lambda^H| \ll 1. \tag{6}$$

Formula (6) defines a relationship between discrete eigenvalues that must be satisfied for proper multigrid performance. The phenomenon is explained in detail in [4]. In the framework of this study we assume that (6) is satisfied by the smallest eigenvalues corresponding to any two grids used in the solution process.

3. Rectangular domains with Neumann boundary conditions. Consider (2a) and (2b) subject to Neumann boundary conditions. h is a meshsize of the grid, a, b, and c are given constants, F^h is a given function, u^h_{xx}, u^h_{yy}, and u^h_{xy} are second-order central difference approximations to the corresponding second-order derivatives, and u^h_x and u^h_y are suitable first-order differences corresponding to u_x and u_y, respectively. The Neumann boundary conditions are discretized using the difference approximation:

$$u^h_y = (u_{i,1} - 4u_{i,2} + 3u_{i,3})/2h \quad \text{on the upper boundary,}$$

where i and j are the grid indices in the horizontal and the vertical directions, respectively. A similar definition is made for the lower boundary. Denote the error by v. When the boundary conditions are satisfied exactly, then $v_{i,1} = v_{i,2} = v_{i,3}$ for all i. As a result, v^h_{yy}, v^h_{xy}, and v^h_y vanish and the residuals from (2a) and (2b) satisfy the following formula:

$$a_0(x)v^h_{xx} + d_0(x)v^h_x + f_0(x)v^h = R^h, \tag{7}$$

where R^h is the residual in (2). Equation (2) can be solved by the following multigrid procedure:

(a) Perform one relaxation sweep: for each i change $u_{i,1}$ and $u_{i,3}$ simultaneously in order to satisfy the boundary conditions. After this relaxation sweep, all boundary conditions are satisfied exactly and the error satisfies (7).

(b) Solve for v in the coarse grid equation (7). We can solve this equation by the usual multigrid procedure described in [1].

(c) Interpolate the coarse grid correction v to the fine grid by the rule $v_{i,1} = v_{i,2} = v_{i,3} = v_i$. Add it to the current approximation u.

We can see that this is just the usual multigrid procedure. The only differences are: the coarse grid is one-dimensional and the coarse grid equation is of different form than the fine grid equation. After this cycle, an exact solution on the fine grid is obtained. Of course we do not have to solve (7) exactly. A V or W cycle can be performed. In this case we do not obtain an exact solution on the fine grid equation, but the error is nevertheless properly reduced.

This procedure can be used for $w<2h$ and for systems of equations subject to Neumann boundary conditions, as the following example shows. Techniques that permit dealing with systems with more complicated boundary conditions are presented in § 5.

3.1. Equations of elasticity. In this section we consider the equations of elasticity discretized on the domain $D=[0,1]\times[0,2h]$ on a uniform grid with meshsize h. The equations of elasticity (see [5]) may be written in the following form:

$$u_{xx}+0.5^*(1-\nu)u_{yy}+0.5^*(1+\nu)v_{xy}=g_1, \tag{8a}$$

$$v_{yy}+0.5^*(1-\nu)v_{xx}+0.5^*(1+\nu)u_{xy}=g_2, \tag{8b}$$

where ν is the Poisson ratio. Consider these equations subject to Neumann boundary conditions. According to the development in the beginning of this section, this system can be solved (almost) exactly by one two-level cycle. In this case, one-dimensional coarse grid equations obtain the following form:

$$(e_u)_{xx}=r_1, \tag{9a}$$

$$0.5^*(1-\nu)(e_v)_{xx}=r_2, \tag{9b}$$

where e_u and e_v are errors in u and v, respectively, and r_1 and r_2 are the respective residuals from (8a) and (8b).

4. General domains and boundary conditions.

4.1. General description. Let $Y_1(x)$ and $Y_2(x)$ be two continuous functions defined for $a<x<b$, and let $d(x)=Y_2(x)-Y_1(x)$. Define domain D as follows:

$$D=\{(x,y)|a<x<b,\ Y_1(x)<y<Y_2(x)\}. \tag{10}$$

To simplify the analysis and programming, we continue to assume that $h\leqq d(x)\leqq 2h$, although the techniques described here can be used in conjunction with local coarsening even in cases where $d=d(x)$ changes by an order of magnitude.

Consider (2a) and (2b) subject to boundary conditions (3a) and (3b) in the domain D given by (10). The following subsection treats the multigrid solver for problems with constant coefficients in the boundary conditions. In § 4.3 we consider variable coefficients.

4.2. Constant coefficients. Consider (2a) and (2b) subject to the boundary conditions (3a) and (3b). Assume that a_1, a_2, b_1, b_2 are constants, and $a_1=a_2$ and $b_1=b_2$. In this case, as before, the equations can be solved up to roundoff error by one two-level cycle according to the following approach:

Define the new variables $y_{\text{new}}=(a_1x_{\text{old}}+b_1y_{\text{old}})/S$ and $x_{\text{new}}=(a_1y_{\text{old}}-b_1x_{\text{old}})/S$, where $S=(a_1^2+b_1^2)^{1/2}$. Then the boundary conditions (3) simplify to the form

$$u_y=b_2u+c_2 \quad \text{on the upper and lower boundaries.} \tag{11}$$

By performing this transformation to the differential equations, then discretizing them, according to our assumptions there will be only three points in the vertical direction (otherwise an additional two-dimensional coarsening can be done). Thus, a procedure similar to that described in the previous section for Neumann boundary conditions can now be applied for the boundary conditions (11). The only difference is that the error satisfies the following formulae:

$$v_{i,1}=\alpha(x)v_{i,2}=\beta(x)v_{i,3}, \tag{12}$$

where α and β can be found as a function of b_2 (they also depend on the domain geometry). This replaces the formulae $v_{i,1} = v_{i,2} = v_{i,3}$ in the case of Neumann boundary conditions.

4.3. Variable coefficients. Here we extend the multigrid solver to the case of variable boundary coefficients. The method is no longer exact, but it does converge rapidly, as our numerical results will show. The boundary conditions (3) can be written as follows:

(13a) $u_y = (a_1 + a')u_x + (b_1 + b')u + c_1$ on the upper horizontal boundary of D

and

(13b) $u_y = a_1 u_x + b_1 u + c_2$ on the lower horizontal boundary of D.

Here $a' = a_1 - a_2$ and $b' = b_1 - b_2$. Assume that all given functions are smooth on scale h. In this case function f is called smooth if its second derivative exists everywhere and $hf' \ll f$. Using the relaxation process, we can satisfy the boundary conditions exactly (or "almost exactly") and smooth the error. Then, using simple calculus the following formulae can be derived:

$$v_{yy} = a_1 v_{xy} + b_1 v_y + a'/d v_x + b_2'/d v + O(h), \tag{14a}$$

$$v_{xy} = a_1 v_{xx} + b_1 v_x + a'/d_1 v_x + b'/d_1 v + O(h), \tag{14b}$$

where d is the distance between curves Y_1 and Y_2 in the vertical direction, and d_1 is the distance in the horizontal direction as is shown in Fig. 1.

Substituting equations (14a) and (14b) into (2a) and (2b) we obtain

$$a^* u_{xx} + b^* u_x + c^* u = F + O(h), \tag{15}$$

where a^*, b^*, and c^* are known. The discrete form of this equation should be used to obtain the coarse grid correction. The multigrid cycle should be performed as usual. Equation (15) approximates (2a) and (2b) up to $O(h)$, which is usually enough to gain high multigrid efficiency.

4.4. Full multigrid algorithm. In the full multigrid algorithm (FMG), which is presented in [1] and [2], the first approximation is obtained by interpolation of the solution from the next coarser level. When the coefficients are constant, we can start FMG from the coarsest *two-dimensional* grid. By one two-level cycle the problem is solved almost exactly (as was shown in the beginning of this section). Then the solution can be interpolated to the next finer grid.

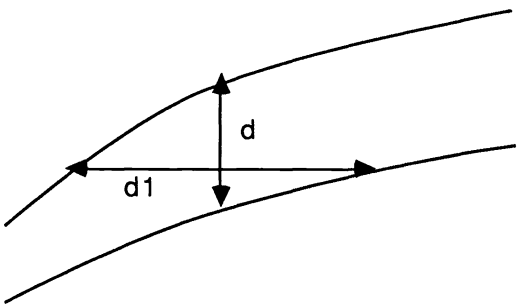


FIG. 1

When the coefficients are variable, we must start FMG from the one-dimensional grid. The first approximation on the coarsest two-dimensional grid should be obtained from the solution on the finest one-dimensional grid. These one-dimensional equations are slightly more complicated than equations presented in the previous section. We obtained (14a) and (14b) using the fact that the error satisfies homogeneous boundary conditions. The original function satisfies boundary conditions (3a) and (3b), which are not necessarily homogeneous. In this case the procedure described in the previous section can be used; (14a) and (14b) have the following form:

$$u_{yy} = a_1 u_{xy} + b_1 u_y + a'/d u_x + b_2'/d u + (c_2 - c_1)/d + O(h),$$
$$u_{xy} = a_1 u_{xx} + b_1 u_x + a'/d_1 u_x + b'/d_1 u + (c_2 - c_1)/d + O(h),$$

where c_1 and c_2 are the right-hand sides of the differential boundary conditions.

4.5. Examples. Consider the Poisson equation $\Delta u = F$, subject to

$$\frac{\partial u}{\partial y} = \frac{\partial u}{\partial x} \quad \text{on the upper and lower horizontal boundaries.}$$

We do not specify D, because this algorithm can be applied to any domain. First we must perform a transformation of coordinates:

$$x_{\text{new}} = (x+y)/\sqrt{2}, \qquad y_{\text{new}} = (x-y)/\sqrt{2}.$$

The Laplacian is invariant under rotation. Boundary conditions in the new coordinates are $u_y = 0$. Now the equation should be discretized—the Laplacian by a centered five-point formula and the boundary conditions by a second-order difference scheme. (Since we continue to assume that there are three points in vertical direction, there is only one second-order difference scheme that approximates u_y.) One boundary relaxation sweep now ensures that the boundary conditions are satisfied exactly. As a result $v^h_{yy} = 0$. The coarse grid equation is $v^h_{xx} = R^h$. By the procedure presented in § 3.3, the Poisson equation with these boundary conditions can be solved almost exactly.

In the second example, boundary conditions on the upper and the lower boundaries are different. Consider the Poisson equation subject to

$$u_y + u_x = 0 \quad \text{on the upper boundary}$$

and

$$u_y = 0 \quad \text{on the lower boundary.}$$

According to § 3.4, the coarse grid system of difference equations should be

$$u^h_{xx} - u^h_x/d = R^h.$$

In Table 1 results of the numerical experiments are presented. The parameters of the

TABLE 1
Dynamic residuals before each V-cycle on Poisson equation.

Cycle number	Interior residuals	Upper boundary residuals	Lower boundary residuals
1	0.353	1.12	0.231
2	0.0385	0.591	0.013
3	0.454E−2	0.705E−2	0.282E−3
4	0.860E−3	0.709E−3	0.280E−4
5	0.871E−4	0.662E−4	0.341E−5
6	0.104E−4	0.986E−5	0.437E−6
7	0.175E−5	0.148E−5	0.62E−7

experiment are $D=[0,1]\times[0,0.02]$; $h=0.01$. Additional boundary conditions are $u(0,y)=u(1,y)=0$. The Laplacian and u_x are discretized by central differences, u_y by the stencil $(3,-4,1)/2h$. Each full relaxation sweep consists of one interior Gauss-Seidel relaxation sweep over the entire domain, two boundary Gauss-Seidel relaxation sweeps, and two additional sweeps near zero and one (for x: $0\leqq x\leqq 2h$ and $1-2h\leqq x\leqq 1$). The table presents residuals computed before each of several V-cycles.

The next example presents the convergence rates of multigrid for the Poisson equation subject to Neumann boundary conditions. The domain is bounded by curves

$$0\leqq x\leqq 1, \quad Y_1=\sin 5x, \quad Y_2=\sin 5x+0.015.$$

Additional boundary conditions are

$$u(0,y)=u(1,y)=0.$$

The problem is discretized by the use of ghost points: gridpoints located in the domain D will be called interior gridpoints; gridpoints which are located outside the domain at the distance of h from the interior points will be called ghost points. See Fig. 2. At interior points the Poisson equation is discretized by the usual five-point scheme. For each ghost gridpoint we impose the boundary condition at the nearest point of the boundary (not necessarily a gridpoint). We impose this condition by forming a discrete approximation involving the ghost point and five of its neighbors, as indicated in Fig. 3.

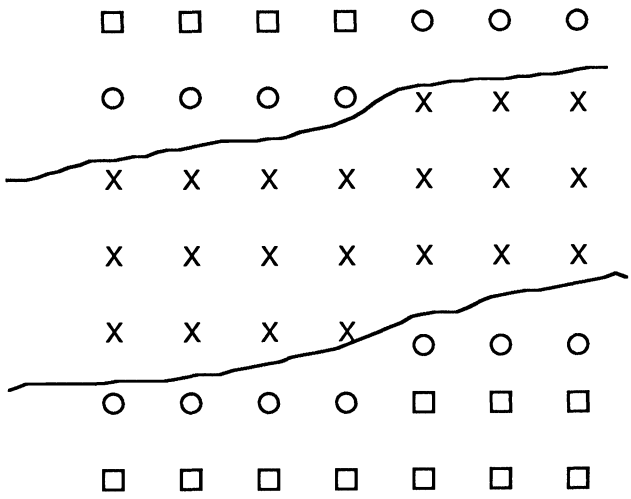


FIG. 2. *Interior points* (×), *ghost points* (○), *and other points* (□).

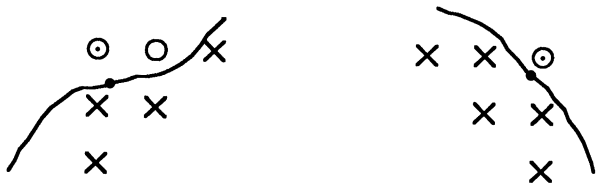


FIG. 3. *Illustration of two possible point configurations for implementing boundary conditions: the discrete conditions at the ghost point indicated by* ⊙ *involves itself and five neighbors, either four interior points* (×) *and one other ghost point* (○), *or five interior points. The discrete condition represents an approximation to the actual boundary condition on the point* (•) *nearest to* (⊙) *on the boundary.*

To determine the coarse grid equations, write the Neumann boundary conditions as $v_y = Y'v_x$. Substituting this formula into equations (14a) and (14b) we obtain

$$v_{yy} = Y'v_{xy}, \qquad v_{xy} = Y'v_{xx}.$$

Substituting this in turn into the Poisson equation, we get $v_{xx} = F/(1+Y'^2)$. This is the formula used on the coarse grid.

For our experiments we used the usual Gauss-Seidel relaxation method and a meshsize of $h = 0.01$. Table 2 presents the convergence behavior of $V(2, 1)$ cycles, where each full relaxation sweep includes one interior relaxation sweep, two boundary relaxation sweeps on upper and lower boundaries, and two relaxation sweeps near the ends ($0 \leqq x \leqq 2h$ and $1-2h \leqq x \leqq 1$).

5. System of equations. Consider the following problem:

$$Au_{xx} + Bu_{xy} + Cu_{yy} = F \tag{16}$$

subject to

$$u_y = \alpha u_x + \beta u + g_2 \quad \text{on the upper boundary} \tag{17a}$$

and

$$u_y = \alpha u_x + \beta u + g_1 \quad \text{on the lower boundary.} \tag{17b}$$

Here A, B, C, α, and β are $n \times n$ matrices and u, F, and g are vectors. There are n equations with n unknown functions and n boundary conditions at each boundary point.

TABLE 2
Dynamic residuals before each V-cycle on Poisson equation.

Cycle number	Interior residuals	Upper boundary residuals	Lower boundary residuals
1	7.95	0.83	0.76
2	1.05	0.074	0.069
3	0.161	0.011	0.009
4	0.028	2.42E−3	2.12E−3
5	6.7E−3	5.8E−4	5.0E−4
6	1.6E−3	1.4E−4	1.2E−4
7	3.9E−4	3.4E−5	2.9E−5

We assume that $D = [0, 1] \times [0, w]$ and all coefficients are constant in order to make calculations easier, although these assumptions are not essential: the method can just as well be applied to the case of variable coefficients and general domains.

Equation (16) is not the most general form of a system of differential equations. For example, Stokes or Navier-Stokes equations cannot be written in this form. By simple calculus we can prove that Stokes equations subject to Neumann boundary conditions are not stable. However, we believe that the basic principles presented in this paper can be applied to any stable h-elliptic system, even if the formulae cannot be applied directly.

Consider a discrete analogue of (16) subject to boundary conditions (17a) and (17b). After a few relaxation sweeps on the two-dimensional fine grid, we get the following residual problem:

$$Av^h_{xx} + Bv^h_{xy} + Cv^h_{yy} = R^h \tag{18}$$

subject to

$$v_y^h = \alpha v_x^h + \beta v^h \quad \text{on the upper and lower boundary.} \tag{19}$$

A corresponding one-dimensional coarse grid correction system can now be constructed as follows:

(a) Using (19) we obtain

$$u_{yy}^i = \sum_{j=1}^{n} \alpha_{ij} u_{xy}^i + \beta_{ij} u^i \tag{20a}$$

and

$$u_{xy}^i = \sum_{j=1}^{n} \alpha_{ij} u_{xx}^j + \beta_{ij} u_x^j \tag{20b}$$

for $i = 1, n$. Since (19) is satisfied exactly on the boundaries, (20a) and (20b) are satisfied up to $O(h^2)$ in the domain.

(b) Substitute formulae derived in (a) and (b) into the original system.

(c) Use a partial Gauss elimination process to eliminate derivatives in the y direction.

This produces n coarse grid equations. This system approximates the original one up to $O(h)$, so in the framework of the multigrid process it can be used in order to obtain a suitable coarse grid correction.

6. Three-dimensional cases. The main advantage of the methods developed here is that they admit three-dimensional analogues. To see this, consider a three-dimensional domain bounded by surfaces $z = Z_1(x, y)$ and $z = Z_2(x, y)$, where $Z_1(x, y) < Z_2(x, y)$ and $Z_2 - Z_1 = O(h)$. Consider the following discrete problem:

$$a_1 u_{xx} + a_2 u_{xy} + a_3 u_{yy} + a_4 u_{xz} + a_5 u_{yz} + a_6 u_{zz} + a_7 u_x + a_8 u_y + a_9 u_z + a_{10} u = F \quad \text{on } D$$

subject to

$$u_z = g \quad \text{on } \partial D.$$

By one relaxation sweep, we can satisfy the boundary conditions exactly. In this case the error can be written $v(x, y, z) = v(x, y)$ (i.e., the error is a function of x and y only); therefore, it satisfies the following two-dimensional equation:

$$a_1 v_{xx} + a_2 v_{xy} + a_3 v_{yy} + a_7 v_x + a_8 v_y + a_{10} v = F \quad \text{on } D.$$

As in the two-dimensional case, when boundary conditions with constant coefficients are given we can solve the equation by one two-level cycle. The formulae for the three-dimensional case are similar to those presented in § 2.2. When we solve an equation with variable coefficients, we can use formulae analogous to those presented in § 2.3. Three-dimensional systems can be solved by an approach similar to that presented in § 3.

Acknowledgment. The author is indebted to Professor A. Brandt, The Weizmann Institute of Science, Rehovot, Israel, and to Professor S. McCormick, University of Colorado, Denver, for helpful discussions.

REFERENCES

[1] A. BRANDT, *Guide to multigrid development*, 1984 Report, Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel, 1984.

[2] A. BRANDT, *Multigrid techniques*: 1984 *guide with applications to fluid dynamics*, Report, Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel, 1984.

[3] ———, *Algebraic multigrid theory*: *The symmetric case*, Appl. Math. Comput., 19 (1986), pp. 23–56.

[4] A. BRANDT AND S. TA'ASAN, *Multigrid methods for nearly singular and slightly indefinite problems*, in Proc. 2nd European Conference of Multigrid Methods, October 1985, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, New York, 1988.

[5] J. RUGE AND A. BRANDT, *A multigrid approach for elasticity problems on "thin" domains*, in Multigrid Methods: Theory, Applications and Supercomputing, S. F. McCormick, ed., Marcel Dekker, New York, 1988, pp. 541–555.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 950–970, July 1991
©1991 Society for Industrial and Applied Mathematics
014

AUTOMATIC DOMAIN PARTITIONING IN THREE DIMENSIONS*

STEPHEN A. VAVASIS†

Abstract. The problem of automatically decomposing a mesh contained in a three-dimensional domain into p pieces is considered. This problem arises in the context of domain-decomposition solvers for partial differential equations (PDEs). Finite-difference meshes are considered, and a class of graphs that would naturally arise from a finite-element mesh in three dimensions is introduced. A general separation algorithm that matches a lower bound is given for this class. An algorithm satisfying better bounds in the case that the domain is a union of boxes is also given. The proposed algorithms are intended to subdivide the region efficiently as a preprocessing step for the parallel numerical solution of the PDE.

Key words. partitioning, domain decomposition, separators, meshes, grids, three-dimensional meshes

AMS(MOS) subject classifications. 65F50, 68R10, 65V05

1. Domain decomposition.

We consider the problem of automatically partitioning a three-dimensional mesh into p subdomains. In order to motivate the problem, we need to provide some background on the relevant numerical methods. We start with background on discretizing elliptic boundary value problems and domain decomposition. Two types of discretization are considered, namely, finite elements and finite differences. These give rise to two kinds of meshes, namely, local graphs and grid graphs (see below). An example of a partial differential equation that can be solved in this manner is:

$$\begin{aligned} \triangle u &= f \quad \text{on } \Omega \quad \text{and} \\ u &= 0 \quad \text{on } \partial\Omega \end{aligned} \tag{1}$$

where Ω is an open, bounded, connected region of $\mathbf{R}^3$, u is an unknown function on $\Omega \cup \partial\Omega$, $\triangle$ is the Laplacian operator, and f is a given real-valued function on Ω.

One version of the *method of finite elements*, first discussed by Courant [7], is based on a triangulation of Ω. This method produces a positive definite sparse system of linear equations $Au = b$, where u is now a vector of unknowns, one unknown for each of the N node points. The matrix A has the important property (assuming that the *standard nodal basis* is used) that its (i, j) entry is nonzero only if nodes i and j are adjacent in the triangulation. See Strang and Fix [27] for more information.

Another approach to solving (1) is the *method of finite differences*. In this approach, the region is discretized by the *lattice points*, that is, points of the form (r, s, t), where r, s, and t are integers. This lattice might actually be embedded in a region Ω' that results from transforming the coordinates of the original region Ω. See Thompson, Warsi, and Mastin [28] for more information. The matrix equation $Au = b$ in this case has similar properties to the finite-element case, namely, it is sparse, has one variable for each node, and has a nonzero entry at position (i, j) only if nodes i and j are *adjacent*, that is, distance 1 in $\mathbf{R}^3$.

One technique to solve the matrix equation is *domain decomposition*. An example of domain decomposition applied to a finite-element discretization with the standard

* Received by the editors June 5, 1989; accepted for publication (in revised form) May 29, 1990. This work was supported by a Fannie and John Hertz Foundation graduate fellowship while the author was at Stanford University, and by the U. S. Army Research Office through the Mathematical Sciences Institute at Cornell.

† Department of Computer Science, Cornell University, Ithaca, New York 14853.

nodal basis is the method of Bramble, Pasciak, and Schatz [3] or Bjørstad and Widlund [2]. Domain decomposition can be thought of as a means to take advantage of the sparsity of A. It has regained popularity in recent years because of its applicability to parallel computing: see, for example, the conference proceedings edited by Glowinski, Golub, Meurant, et al. [14] and Chan, Glowinkski, and Périaux [4].

The method of domain decomposition divides the original region $\Omega \cup \partial\Omega$ into p subdomains. It is useful to think of p as the number of processors in the parallel computer, although this need not be the case in general. The interface between all the subdomains may be called the *boundary*. There are two possible ways to take advantage of this partitioning. First, if the values of the solution at the boundary nodes are known, then the original problem is decoupled into independent problems for each subregion. Second and more important, iterative methods exist for determining the values of the boundary nodes that rely on the solution of decoupled problems on the subregions.

We go into a bit more detail following Chan and Resasco [5]. The N gridpoints of the domain are partitioned into p *subdomains*. The variables of u are reordered so that the interior nodes of subdomain 1 are listed first, then subdomain 2, and so on, with the boundary variables (denoted $u_\sim$) at the end. We partition $u = (u_1, u_2, \cdots, u_p, u_\sim)^T$, and the original system $Au = b$ takes the form:

$$\begin{bmatrix} A_{11} & & & & & A_{1,\sim} \\ & A_{22} & & & & A_{2,\sim} \\ & & A_{33} & & & A_{3,\sim} \\ & & & \ddots & & \vdots \\ & & & & A_{pp} & A_{p,\sim} \\ A_{\sim,1} & A_{\sim,2} & A_{\sim,3} & \cdots & A_{\sim,p} & A_{\sim,\sim} \end{bmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_p \\ u_\sim \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_p \\ b_\sim \end{pmatrix}.$$

We can solve for the variables $u_\sim$. Define the matrix

$$C = A_{\sim,\sim} - A_{\sim,1}A_{11}^{-1}A_{1,\sim} - \cdots - A_{\sim,p}A_{pp}^{-1}A_{p,\sim}. \tag{2}$$

Note that C, which is called the *capacitance matrix*, is the result of eliminating all the variables except those in $u_\sim$ from the last row in the above block decomposition. Solving the original system $Au = b$ now reduces to solving systems whose coefficient matrices are given by C and by the matrices A_{ii}.

The system $Cu_\sim = \hat{b}_\sim$ can be solved iteratively using, for example, preconditioned conjugate gradients. In such a scheme, it is necessary in the iteration loop to form the product Cx for a vector x (see Golub and Van Loan [16]). This product can be formed without computing C; examining (2), we see that Cx can be computed by matrix-vector multiplication and by solving systems involving A_{ii}.

This method is attractive for parallel computation because the processes of solving the systems involving matrices A_{ii}, $i = 1, \cdots, p$, are independent from one another and hence can be done in parallel on separate processors.

Solving the system whose coefficients are given by C is the most difficult step computationally. Therefore, it is advantageous for the number of variables in $u_\sim$ to be small. In addition, it is advantageous for the number of variables in u_1 to be roughly equal to the number of variables in u_2, and so on up to u_p, because this balances the load of work among the processors in the steps involving the matrices A_{ii}, $i = 1, \cdots, p$.

We introduce some graph notation. For the finite-element case, let G be the graph that has one vertex for each node in the triangulation, and an edge between two vertices if and only if they share an edge of the triangulation.

For the finite-difference case, let G be a graph whose vertices are the lattice points in Ω, and whose edges indicate adjacency. Such a graph will be called a *three-dimensional grid graph*; one of the properties of such a graph is that no vertex has more than six neighbors. Adjacency in graph G corresponds exactly to the nonzero structure of A (as in the finite-element case).

In this paper we take the point of view that G (i.e., the triangulation or finite-difference grid) is given as input data, and the problem is to compute a partition into subregions, which corresponds to partitioning G into subgraphs. An alternative approach would be to decompose $\Omega \cup \partial\Omega$ into subregions and then create a triangulation that respects the subdomain boundaries—we do not pursue this approach here.

Accordingly, the problem is to decompose G into subgraphs. We make this notion precise with the following definition. A *partition* of G into p subdomains is a partition of the nodes of G into $p+1$ subsets, say, $G_1, G_2, \cdots, G_p, G_\sim$, with the following three properties:

(a) $G_1 \cup \cdots \cup G_p \cup G_\sim$ includes all the nodes of G,
(b) $G_1, G_2, \cdots, G_p, G_\sim$ are pairwise disjoint, and
(c) There is no edge of G between G_i and G_j if $1 \leq i < j \leq p$ (edges from G_i to $G_\sim$ are permitted).

The *subdomains* of the partition are the sets $G_1, \cdots, G_p$; the *boundary* is $G_\sim$. Condition (c) enforces the rule that the subdomains are decoupled.

Our main result is a method for automatically partitioning a mesh of nodes (lying in $\Omega \subset \mathbf{R}^3$) into p subdomains. As discussed above, the two goals for such a partition are:

1. The p subdomains all have a roughly equal number of points, and
2. The total boundary size is small.

These two goals assure maximum efficiency when the numerical computations are done in a parallel environment. In the finite-element case it is not possible to produce such a partition in general, even assuming that the mesh is a triangulation of a convex region of $\mathbf{R}^3$, as we shall see in the next section. It is therefore necessary to describe a property that characterizes "good" meshes. The next section is devoted to describing this property ("locality") and proving a separator theorem for graphs that have it. Finite-difference meshes always have this property.

The result in §2 is that a three-dimensional grid graph or local mesh can be *weakly* partitioned into two subdomains with an upper bound $c \cdot N^{2/3}$ on the size of the boundary, where N is the total number of nodes in the graph (the number of variables in the system) and c depends on parameters of the problem. The term *weak partition* refers to a partition in which each subdomain is no more than a constant multiple larger than the others. The existence theorem for the partition will lead to a simple algorithm described in §3. This treatment is extended to cover strong partitions and p subdomains in §4—in a *strong* partition the subdomains are equal in size, except for low-order terms. Parallel aspects of the partitioning algorithm are briefly discussed in §5. In §6 it is proved that for a standard class of graphs our algorithm is optimal up to a constant factor. In §7 we show that better bounds on the size of the boundary are possible in the special case that the graph is a three-dimensional grid contained in the union of a small number of boxes. Our computational experience is described in §8.

Occasionally one has an additional goal in mind for domain partitioning, namely, subdomains with nice shapes. In general, our algorithms will not provide such a partitioning; in fact, a particular subdomain may have more than one connected com-

ponent, even if the original domain is connected. We remark, however, that our general-purpose algorithm of §4 has the property that if the initial domain is a finite-difference grid in a *box* (the Cartesian product of three intervals), then all of the subdomains will also be boxes. (See [31] for a further discussion of this fact.)

The problem of partitioning a finite-element mesh in two dimensions is considered by Lipton and Tarjan [19], [20]. Their techniques were extended to graphs of fixed genus by Gilbert, Hutchinson, and Tarjan [13]. These methods do not generalize to three dimensions, because a three-dimensional triangulation does not necessarily have a small separator as explained in the next section. Therefore, it is necessary to restrict attention to the subset of triangulations with locality.

Previous research on separators has concentrated on applications to *nested dissection*, a sparse matrix technique introduced by George [10]; see also George and Liu [11]. For example, a variant of Theorem 2.1 was known to Lipton, Rose, and Tarjan [18], and a theorem related to Theorem 6.4 was known to Eisenstat, Schultz, and Sherman [8]. These techniques as applied to VLSI layout problems are discussed in Ullman [30].

There has also been work on partitioning meshes for the purposes of load balancing in which boundary size is not analyzed. See for example, Morison and Otto [23] and Berger and Bokhari [1]. Finally, there is a great deal of literature on partitioning techniques for general undirected graphs; see, for example, Pothen, Simon, and Liu [25] for one such technique and for a bibliography on earlier work.

We remark that the method of finite elements can be applied to problems more general than (1). Our results apply equally well to these situations provided that the matrix A has a sparsity pattern based on the geometric locations of vertices in $\mathbf{R}^3$. The important property of the system for this paper is that the (i, j) matrix entry is nonzero only if nodes i and j are nearby points in $\mathbf{R}^3$.

2. Local graphs and weak partitioning. To obtain an algorithm for automatic decomposition of a triangulated domain, we need to exploit the sparsity properties of the matrix A associated with such a triangulation. Unfortunately, the matrix associated with a general triangulation of a three-dimensional region need not be sparse. We examine the following example due to Preparata and Shamos [26]. Consider the following set of $2n + 1$ points in $\mathbf{R}^3$:

$$X = \{v_0, v_1, \cdots, v_n, w_1, \cdots, w_n\}$$

where $v_i = (0, 0, i)$ and $w_j = (\cos(2\pi j/n), \sin(2\pi j/n), 0)$. Let C be the convex hull of X. It can be verified that C is triangulated by all tetrahedra whose vertices are at $(v_i, v_{i+1}, w_j, w_{j+1})$ as i runs from 0 to $n - 1$ and j runs (independently) from 1 to n (where w_{n+1} is identified with w_1). The matrix associated with the standard nodal basis for the points of X would have a dense block relating the elements associated with v_i, $i = 0, \cdots, n$ to those associated with w_j, $j = 1, \cdots, n$. This dense block (and the corresponding symmetric dense block) would have $\Omega(n^2)$ nonzero entries, and hence the matrix is not sparse.

This triangulation of C is pathological in some sense because the tetrahedra get stretched arbitrarily as $n \to \infty$. In order to exclude cases like this, we propose a class of graphs that capture an interesting subset of three-dimensional triangulations, namely, the class of *local* graphs. This definition also applies to meshes other than triangulations.

DEFINITION. Let $\| \cdot \|$ denote some l_p norm on $\mathbf{R}^3$. Let graph G be embedded in $\mathbf{R}^3$. Let ρ be the minimum distance between any two nodes of G (with respect to

$\|\cdot\|$). Let σ be the maximum distance between any pair of nodes that are connected by an edge of G. Then we say that G is **r-local** if $\sigma/\rho \leq r$.

Note that this definition is a property of G and its embedding. Since we are interested in finite-element graphs, such a graph typically comes with a natural embedding in $\mathbf{R}^3$. It can be proved that any N-node graph has an embedding to make it $N^{1/3}$-local (see [31]). The interesting case is when a family of graphs is r-local for a value r that does not depend on the size of the graph. For a regular triangularization of a three-dimensional region, we would expect r to be a constant independent of the number of gridpoints or the level of refinement.

For example, in any l_p norm a three-dimensional grid graph defined in §1 would be 1-local, because the minimum separation between any two nodes is 1, and the maximum distance traversed by any edge is also 1. Notice that $r = 1$ is the smallest possible value for r in the above definition (assuming that the graph has at least one edge).

As a second example, consider triangulating a cube by dividing it into small subcubes, and then triangulating each subcube with Kuhn's K_1 triangulation (see Todd [29]). Each node of the resulting graph would have an edge to 14 of its neighbors, all of which are of distance 1 with respect to the l_∞. Thus, this graph is 1-local in the ∞-norm.

If a graph is r-local, then it can be proved that the maximum degree Δ of any node of the graph satisfies

$$\Delta \leq (2r+1)^3 - 1.$$

(The proof is given in [31]; this bound is sharp in some cases.) Note that $\Delta + 1$ is the maximum number of nonzeros in any row of A. In general, however, there may be a better bound known for Δ based on the discretization at hand. For example, the 7-point Laplacian would have $\Delta = 6$, but the above estimate asserts only that $\Delta \leq 26$.

The following two theorems are the basis for most of the paper. Note that the first theorem considers three-dimensional grid graphs, whereas the second theorem considers the more general case of r-local graphs. The bounds in the first theorem are better, however, so we state the theorems separately.

THEOREM 2.1. *Suppose G is an N-node three-dimensional grid. Then the nodes of G can be partitioned into $G_1, G_2, G_\sim$ such that $|G_1|, |G_2| \leq 6N/7$ and $|G_\sim| \leq \beta N^{2/3}$, where $\beta = 1.71$. Furthermore, this partition has the property that the nodes of $G_\sim$ all lie on the same plane.*

Remarks.

(a) Lipton, Rose, and Tarjan [18] state without proof a variant of this theorem in the context of nested dissection. Specifically, they observe that subgraphs of d-dimensional grids can be separated into two pieces, each no more than a fixed percentage of the original subgraph, with separators of size $O(N^{(d-1)/d})$. For three dimensions this means that the separator size is $O(N^{2/3})$. Theorem 2.1 is a strengthening of their $d = 3$ result because we are able to give all of the relevant constants. We are also able to stipulate that the separator can be a plane, which leads to a linear-time algorithm for finding the separator. (Lipton, Rose, and Tarjan do not provide an algorithm.)

(b) Since G_2 can be six times larger than G_1 (if $|G_2| = 6N/7$ and $|G_1| = N/7 - |G_\sim|$) or six times smaller, then we refer to this partion as a *weak partition.* In §4 we show how to make the partition more balanced at the expense of a larger value of β.

For the purpose of the following theorem, let V be the volume of the unit ball with respect to $\|\cdot\|$. Note that $4/3 \leq V \leq 8$ for l_p norms on $\mathbf{R}^3$ (see [31]).

THEOREM 2.2. *Suppose G is an N-node graph embedded in $\mathbf{R}^3$ so that it is r-local. Assume that $N \geq 336$. Then the nodes of G can be partitioned into $G_1, G_2, G_\sim$ such that $|G_1|, |G_2| \leq 6N/7$ and such that $|G_\sim| \leq \beta N^{2/3}$, where $\beta = r(448/V)^{1/3}$.*

Remark. Gilbert [12] states a theorem similar to this for two-dimensional graphs; he omits the details and proof. He states that the constants associated with his theorem would be too large to be practical, but our constants seem to be reasonable.

The top-level structures of the proofs of these two theorems are identical. The technical details diverge, however, because we can make stronger estimates with simpler arguments in the case of Theorem 2.1. Accordingly, we must give a separate proof of each theorem.

Proof of Theorem 2.1. We start with an embedding of G in $\mathbf{R}^3$ so that all the nodes are at lattice points and all the edges are between neighbors—such an embedding exists by the assumption that G is a three-dimensional grid. Say that a plane specified by $\{(x_1, x_2, x_3) : x_1 = a\}$ is a *separating x_1 plane* if a is an integer and

$$|\{(x_1, x_2, x_3) \in G : x_1 \leq a\}| \geq N/7, \quad \text{and}$$
$$|\{(x_1, x_2, x_3) \in G : x_1 \geq a\}| \geq N/7.$$

Define a *separating x_2 plane* and *separating x_3 plane* similarly.

Note that if $\{x_1 = a\}$ and $\{x_1 = c\}$ are both separating x_1 planes for two integers a and c, then $\{x_1 = b\}$ is also a separating x_1 plane for every integer b between a and c.

Now, we consider three cases based on the number of separating planes in each dimension. The case analysis shows that if there is not a separating x_1 plane or x_2 plane with the right properties, then there must be a good separating x_3 plane.

We will let θ be a real parameter. The exact numerical value of θ will be provided after the three cases have been analyzed.

Case 1. *There are at least $\theta N^{1/3}$ separating x_1 planes.* Since there are at least $\theta N^{1/3}$ separating x_1 planes (which are disjoint), then there must be a separating x_1 plane, say, $\{x_1 = a^*\}$, containing no more than $N/(\theta N^{1/3}) = (1/\theta)N^{2/3}$ graph nodes on it. This follows because there are no more than N points total on all the planes. Then let

$$G_1 = \{(x_1, x_2, x_3) \in G : x_1 \leq a^* - 1\},$$
$$G_2 = \{(x_1, x_2, x_3) \in G : x_1 \geq a^* + 1\} \quad \text{and}$$
$$G_\sim = \{(x_1, x_2, x_3) \in G : x_1 = a^*\}.$$

The triple $(G_1, G_2, G_\sim)$ given by these equations will be called the partition *induced* by $\{x_1 = a\}$. It is clear that this partition of G satisfies the conditions of the theorem. The bound on the size of $G_\sim$ will follow below when we select a value for θ.

Case 2. *There are at least $\theta N^{1/3}$ separating x_2 planes.* This is handled in the same way as Case 1. In particular, we draw the conclusion that there is a separating x_2 plane with no more than $(1/\theta)N^{2/3}$ nodes on it.

Case 3. *There are fewer than $\theta N^{1/3}$ separating x_1 planes and fewer than $\theta N^{1/3}$ separating x_2 planes.* Let s_1 be the smallest integer such that $\{x_1 = s_1\}$ is a separating x_1 plane, and let t_1 be the largest such integer. Define s_2, t_2 similarly for the second dimension. The argument above shows that if a is any integer between s_1 and t_1, then $\{x_1 = a\}$ is a separating x_1 plane. Thus, we can use the hypothesis of Case 3 to

conclude that

$$t_1 - s_1 + 1 < \theta N^{1/3}$$

and a similar inequality holds for s_2, t_2.

Define W to be the tube in $\mathbf{R}^3$ given by the Cartesian product

$$W = [s_1, t_1] \times [s_2, t_2] \times \mathbf{R}.$$

Then we claim that $|W \cap G| \geq 3N/7$. A node of $\overline{W} \cap G$ (where $\overline{W}$ is the complement of W with respect to $\mathbf{R}^3$) satisfies one (or more) of the four inequalities $x_1 < s_1$, $x_1 > t_1$, $x_2 < s_1$, or $x_2 > t_2$. We claim that the number of nodes that could satisfy any one of these four inequalities is less than $N/7$. This will imply that the number of nodes not in W is less than $4N/7$, which establishes the minimum size of $W \cap G$.

In this regard, let us consider the inequality $x_1 < s_1$. Since $\{x_1 = s_1\}$ is the smallest separating x_1 plane, then $\{x_1 = s_1 - 1\}$ was not a separating x_1 plane, i.e., there are more than $6N/7$ nodes that satisfy $x_1 > s_1 - 1$, i.e., there are more than $6N/7$ nodes that satisfy $x_1 \geq s_1$. Thus, there are fewer than $N/7$ nodes that satisfy $x_1 < s_1$. This holds for the other cases, establishing the claim that $|W \cap G| \geq 3N/7$.

Define a *separating W plane* to be a plane $\{x_3 = a\}$ such that a is an integer, at least $N/7$ nodes of $W \cap G$ satisfy $x_3 \leq a$, and at least $N/7$ nodes of $W \cap G$ satisfy $x_3 \geq a$. Let s_3 be the smallest integer a such that $\{x_3 = a\}$ is a separating W plane, and let t_3 be the largest such integer. Note that a separating W plane is also a separating x_3 plane.

We claim that there are at least $N/7$ nodes of G in the box

$$W_1 = [s_1, t_1] \times [s_2, t_2] \times [s_3, t_3].$$

The argument is similar to the argument we used to bound the size of $W \cap G$. To prove the claim, observe that there are at least $3N/7$ nodes in W, and at most $N/7$ are excluded by requiring $x_3 \geq s_3$, and a further $N/7$ could be excluded by requiring $x_3 \leq t_3$.

We conclude that the number of graph nodes in this box is at least $N/7$. The maximum number of nodes that could lie in such a box is $(t_1 - s_1 + 1) \cdot (t_2 - s_2 + 1) \cdot (t_3 - s_3 + 1)$, since each of the points is a lattice point. Thus,

$$(t_1 - s_1 + 1)(t_2 - s_2 + 1)(t_3 - s_3 + 1) \geq N/7.$$

We have hypothesized upper bounds on the first two terms.

$$\theta^2 N^{2/3}(t_3 - s_3 + 1) \geq N/7$$

so

$$t_3 - s_3 + 1 \geq N^{1/3}/(7\theta^2).$$

Recall that there are at most $4N/7$ points of G not in W. Hence there is an integer a^* in the range $[s_3, t_3]$ such that $\{x_3 = a^*\}$ intersects at most

$$\frac{4N/7}{N^{1/3}/(7\theta^2)}$$

nodes of $\overline{W} \cap G$, i.e., at most $4\theta^2 N^{2/3}$ nodes. The number of nodes of $W \cap G$ that this plane could intersect is bounded by $(t_1 - s_1 + 1)(t_2 - s_2 + 1)$ by definition of W. This number is bounded by $\theta^2 N^{2/3}$. Hence the total number of points on this separator (counting the points in W and the points in $\overline{W}$) is at most $5\theta^2 N^{2/3}$. Let $(G_1, G_2, G_\sim)$ be the partition induced by this separating plane $\{x_3 = a^*\}$.

Thus, the bound on the separator size in Cases 1 and 2 was $(1/\theta)N^{2/3}$. In the third case the bound was $5\theta^2 N^{2/3}$. If we set $\theta = (1/5)^{1/3}$, then both coefficients are $\sqrt[3]{5}$. In this case, the separator size is no more than $1.71N^{2/3}$. □

Remark. The technique of this proof can be generalized to d dimensions for any integer $d > 0$. In particular, we can show that a d-dimensional grid graph G with N nodes can be partitioned into two pieces, neither of which has more than the fraction $1 - 1/(2d+1)$ of the original graph, and such that the boundary size is $(2d-1)^{1/d}N^{(d-1)/d}$.

Proof of Theorem 2.2. Assume without loss of generality that the embedding in $\mathbf{R}^3$ has that $\rho = 1$ and $\sigma = r$ in the definition of locality.

We have assumed that $N \geq 336$; the theorem holds for smaller values of N, but the upper bounds used in this proof are too crude to work for smaller N.

We define a *separating* x_1 *plane* to be a plane of the form $P = \{(x_1, x_2, x_3) : x_1 = a\}$ for a real number a such that at least $N/7$ nodes (x_1, x_2, x_3) of G satisfy $x_1 \geq a$ and at least $N/7$ satisfy $x_1 \leq a$. Define *separating* x_2 *plane* and *separating* x_3 *plane* analogously. In the proof of Theorem 2.1, "separating plane" had a similar definition except we limited consideration to integer values of a.

Allowing real values for a necessitates the following observation: Suppose that $\{x_1 = a\}$ is a separating x_1 plane for all a in some set of real numbers Y. Then $\{x_1 = l\}$ and $\{x_1 = L\}$ are both separating x_1 planes if we define $l = \inf Y$ and $L = \sup Y$. (See [31] for this argument.) Accordingly, we conclude that the set of a such that $\{x_1 = a\}$ is a separating x_1 plane is a closed interval of $\mathbf{R}$.

Let s_1 be the smallest real number such that $\{x_1 = s_1\}$ is a separating x_1 plane. Let t_1 be the largest such real number. Define s_2, t_2 for the second dimension analogously.

Now, we consider three cases based on these quantities. Let θ be a constant depending on V but not N. The exact formula for θ is derived during the course of the proof.

Case 1. $t_1 - s_1 > \theta N^{1/3}$. Let $J = \lfloor (t_1 - s_1)/r \rfloor$. Consider the sequence of planes given by

$$\begin{aligned} P_0 &= \{s_1\} \times \mathbf{R} \times \mathbf{R}, \\ P_1 &= \{s_1 + r\} \times \mathbf{R} \times \mathbf{R}, \\ &\vdots \\ P_J &= \{s_1 + Jr\} \times \mathbf{R} \times \mathbf{R}. \end{aligned}$$

By choice of J, $s_1 + Jr \leq t_1$. Thus, each of these planes is a separating x_1 plane.

We will say that a node $Q = (x_1, x_2, x_3)$ is a *gateway* across plane P_i if either (a) $x_1 = s_1 + ir$, or (b) $x_1 > s_1 + ir$ and Q is adjacent (in the graph) to a point (y_1, y_2, y_3) such that $y_1 < s_1 + ir$. Case (b) implies that $x_1 < s_1 + (i+1)r$ by our assumption of r-locality. Hence if Q is a gateway across P_i, then it cannot be a gateway across P_j for any $j \neq i$.

Let U_i be the set of gateway nodes for plane P_i. We can conclude that the sets U_i are pairwise disjoint for $i = 0, \cdots, J$.

Thus, the U_i are pairwise disjoint, and there are $J+1$ such sets; hence there is one set U_{i^*} such that

$$\begin{aligned} |U_{i^*}| &\leq N/(J+1) \\ &\leq \frac{N}{(t_1 - s_1)/r} \end{aligned}$$

$$\leq \frac{N}{(\theta N^{1/3})/r}$$
$$\leq (r/\theta)N^{2/3}.$$

Define

$$G_1 = \{(x_1, x_2, x_3) \in G : x_1 < s_1 + i^* r\},$$
$$G_\sim = U_{i^*},$$
$$G_2 = G - (G_1 \cup G_\sim).$$

We will call this the partition *induced* by plane $\{x_1 = s_1 + i^* r\}$. It is a generalization of the definition of "induced" in the previous proof. Since each P_i is a separating plane, it is easy to verify that G_1 and G_2 both have no more than $6N/7$ elements—in particular, note that every point of G_2 satisfies $x_1 > s_1$. Also, we know that the size of $G_\sim$ is no more than $(r/\theta)N^{2/3}$. Finally, we have to check that there is no edge from G_1 to G_2. This follows from our choice of the gateway nodes; no node in G_2 can be connected to a point in the halfspace $\{x_1 < s_1 + i^* r\}$.

Case 2. $t_2 - s_2 > \theta N^{1/3}$. This is handled in the same way as Case 1.

Case 3. $t_1 - s_1 \leq \theta N^{1/3}$ *and* $t_2 - s_2 \leq \theta N^{1/3}$. Define W to be the following "tube" in $\mathbf{R}^3$:

$$W = [s_1, t_1] \times [s_2, t_2] \times \mathbf{R}.$$

Then we claim that $|W \cap G| \geq 3N/7$. The same argument as in the previous proof is used.

We call a plane $\{x_3 = v\}$ a *separating* W *plane* if

$$|W \cap G \cap \{(x_1, x_2, x_3) : x_3 \leq v\}| \geq N/7 \quad \text{and}$$
$$|W \cap G \cap \{(x_1, x_2, x_3) : x_3 \geq v\}| \geq N/7.$$

Let s_3 be the smallest value of v such that $\{x_3 = v\}$ is a separating W plane, and let t_3 be the largest such value of v. Define

$$W_1 = [s_1, t_1] \times [s_2, t_2] \times [s_3, t_3].$$

As in the previous proof, W_1 is a box containing at least $N/7$ nodes of G.

In general, suppose we have an $a \times b \times c$ box in $\mathbf{R}^3$. We claim that it can have at most $8(a+1)(b+1)(c+1)/V$ points of G inside of it. Surround each point of G inside of the box with a $\|\cdot\|$-sphere of radius $\frac{1}{2}$. All of these spheres must be disjoint by our assumption that $\rho = 1$, and they all have volume $V/8$. The total volume of these spheres is contained in a box that is obtained by moving all the faces of the original $a \times b \times c$ box outward by $\frac{1}{2}$. The volume of this larger box is $(a+1)(b+1)(c+1)$; dividing gives the bound.

In our case, W_1 is a rectangle containing at least $N/7$ nodes of G. Therefore,

$$N/7 \leq 8(t_1 - s_1 + 1)(t_2 - s_2 + 1)(t_3 - s_3 + 1)/V.$$

By the hypothesis for Case 3, we have bounds on the first two factors on the right-hand side; substituting these yields

$$N/7 \leq 8(\theta N^{1/3} + 1)(\theta N^{1/3} + 1)(t_3 - s_3 + 1)/V.$$

If we assume that $\theta N^{1/3} \geq 1$ (this assumption is justified below), then we can we can put an upper bound of $2\theta N^{1/3}$ on the first two factors of the right-hand side:

$$N/7 \leq 32\theta^2 N^{2/3}(t_3 - s_3 + 1)/V.$$

So, solving for $t_3 - s_3$, we get

$$t_3 - s_3 \geq \frac{VN^{1/3}}{224\theta^2} - 1.$$

Now if we assume that $(VN^{1/3})/(224\theta^2) \geq 2$ (this assumption justified below), then we conclude that

$$t_3 - s_3 \geq \frac{VN^{1/3}}{448\theta^2}.$$

Now we observe that any plane of the form $\{x_3 = v\}$, where $s_3 \leq v \leq t_3$, is a separating x_3 plane because it has at least $N/7$ nodes on either side. Thus, we have an interval in which every plane is a separating x_3 plane.

We consider separating planes within this region of the form $\{x_3 = s_3 + ir\}$. As in Case 1, there are at least $(t_3 - s_3)/r$ such separating planes, and their gateway nodes are disjoint. There are a total of N nodes of G that could be gateway nodes for these planes, so there is one such separating plane with no more than

$$\frac{N}{(VN^{1/3})/(448\theta^2 r)}$$

gateway nodes, i.e., no more than

$$\frac{448\theta^2 r}{V} N^{2/3}$$

nodes. Denote this plane by $\{x_3 = s_3 + i^* r\}$ and let $(G_1, G_2, G_\sim)$ be its induced partition.

The bound on the size of $G_\sim$ for Cases 1 and 2 was $(r/\theta)N^{2/3}$ and for Case 3 it was $(448\theta^2 r/V)N^{2/3}$. If we set $\theta = (V/448)^{1/3}$, these bounds both become

$$|G_\sim| \leq r(448/V)^{1/3} N^{2/3}$$

which proves the theorem.

We now have to justify our assumptions made earlier that $\theta N^{1/3} \geq 1$ and that $(VN^{1/3})/(448\theta^2) \geq 1$. By the choice of θ in the previous paragraph, the left-hand sides of these two inequalities are exactly the same, so we need only concentrate on the first inequality. We want to have $(V/448)^{1/3} N^{1/3} \geq 1$, so cubing both sides, we want

$$\frac{VN}{448} \geq 1.$$

Since V is always at least $4/3$, we require that $N \geq 336$. □

To compare Theorem 2.1 and Theorem 2.2 in the case of a grid graph, note that the grid graph is 1-local in the ∞-norm on $\mathbf{R}^3$; hence we can take $V = 8$ and $r = 1$ in the above theorem. Theorem 2.2 above gives a bound of $3.83N^{2/3}$ on the size of the separator, assuming $N \geq 336$. This bound is of the same form as the bound of Theorem 2.1 but has an inferior constant and demands a lower bound on N.

3. An algorithm for a weak partition. The next task is to describe algorithms to find the separator specified by Theorem 2.1 or Theorem 2.2. We start by describing the $O(N)$ algorithm for the three-dimensional grid graph case, which we call **Weak-Split-FD**. This algorithm is then generalized to Algorithm **Weak-Split-FE** for the local mesh case.

The separator specified by Theorem 2.1 can be found in $O(N)$ steps by trying out every possible plane; for the next few paragraphs we expand on this idea. Assume

that we are given the embedding of the nodes G as lattice points in $\mathbf{R}^3$. Without loss of generality, the graph is embedded so that all three coordinates of every node lie between 1 and N. Algorithm **Weak-Split-FD** makes tallies for each i of how many nodes have their x_1 coordinate exactly equal to i. This takes N steps. The same is done for the x_2 and x_3 tallies.

Then **Weak-Split-FD** computes running totals of the x_1 tallies—the ith running total will be equal to the number of nodes whose x_1 coordinate is at most i; the same is done for the second and third coordinates. From the tallies and the running totals it is possible to compute for each plane of the form $\{x_d = i\}$ $(d = 1, 2, 3)$ exactly how many nodes are in each set of the induced partition $(G_1, G_2, G_\sim)$. The algorithm scans all of these quantities to find a plane that fits the theorem.

In general, this scanning procedure might encounter several planes that satisfy the conditions of Theorem 2.1. Choosing among these planes will often mean trading off a smaller boundary against a more even partition. All other things being equal, we propose the following heuristic for **Weak-Split-FD**. Suppose we find a plane that separates the graph into two pieces, the larger of which contains αN points, and suppose the number of points on the boundary is $\beta N^{2/3}$. We reject the plane unless $\alpha \leq 6/7$ and $\beta \leq 1.71$, since we want to consider only those planes that satisfy the theorem. Among all such planes, it is reasonable to choose the one that minimizes

$$\frac{\beta}{1 - \alpha^{2/3}}.$$

The motivation for this formula is discussed in §4. Pseudocode for **Weak-Split-FD** is spelled out in [31].

For the r-local graph case, **Weak-Split-FE** assumes that the nodes of the graph are specified by coordinates in $\mathbf{R}^3$. Thus, the graph must already be embedded before the algorithm starts. The algorithm does not need to know the value of r, the locality parameter.

The algorithm is as follows. The outer loop is on $d = 1, 2, 3$ as we search for a separating plane in each of the three dimensions. Let us consider the $d = 1$ phase—the other two phases are similar. In this phase we search for the best separating x_1 plane.

In order to limit the number of separating planes that need to be considered, it should be observed that if a separating x_1 plane $\{x_1 = a\}$ does not contain any graph nodes, then there is no loss of generality in the proof of the theorem if we slide the plane rightward to $\{x_1 = a' - \epsilon\}$, where a' is the lowest value of an x_1 coordinate among graph nodes contained in $\{x_1 > a\}$, and $\epsilon > 0$ is infinitesimal. This means that we have to examine only $O(N)$ possible separating planes to find one satisfying the theorem.

In order to get an efficient algorithm, it is necessary to be able to compute quickly the sizes of $(G_1, G_2, G_\sim)$ in the induced partitions. As in the finite-difference case, this is done with running totals. It is first necessary to sort the N nodes according to their x_1 coordinates; this takes $O(N \log N)$ time. We also need to sort two tags for each node. The tags for a node x indicate the smallest and largest values of a such that $x \in G_\sim$ for the partition induced by the plane $\{x_1 = a\}$. These tags can be determined from graph adjacency information, and they also must be sorted. Once the nodes and the tags are sorted, then we can pass over the list of possible separating planes and update the sizes of $(G_1, G_2, G_\sim)$ in a cumulative fashion. The full listing of **Weak-Split-FE** is contained in [31]. Again, we search for a plane that has $|G_1|$ and $|G_2|$ bounded by $6N/7$ and that minimizes $\beta/(1 - \alpha^{2/3})$, where α and β are as above.

This algorithm takes $O(N \log N)$ steps; everything in the algorithm is linear except for the sorting. In many cases the properties of the graph will be well enough understood to make it possible to generate the nodes in order of x_i coordinate values. In this case, the algorithm would be linear time.

4. Strong partitions and p subdomains. Suppose we are given a three-dimensional grid G whose nodes are partitioned into $G_1, \cdots, G_p, G_\sim$. Suppose also that G_1 is the largest and G_p the smallest of the subdomains. We define the *balance ratio* to be $|G_1|/|G_p|$. Ideally, the balance ratio is 1. The algorithms of the last section guarantee a balance ratio of at most $6 + O(N^{-1/3})$ in the case that $p = 2$, because $|G_1|$ is at most $6N/7$ and $|G_2|$ is at least $N/7 - O(N^{2/3})$. This was an example of a *weak partition.*

In this section we show how to make the balance ratio exactly 1.0 plus low-order terms using a technique common in the literature; see, for example, [19] or [30]. We will call such a partition a *strong* partition. After that, we move on to the case of more than two subdomains.

The algorithm for a strong partition is called **Strong-Split**. Suppose we are given an N-node three-dimensional grid G and numbers t_1, t_2 such that $t_1 + t_2 = N$ and $0 < t_1, t_2 < N$. We claim by induction on the size of G that, given G, t_1, t_2, **Strong-Split** partitions G into G_1^*, G_2^*, and $G_\sim^*$ such that $|G_1^*| \le t_1$ and $|G_2^*| \le t_2$. The procedure for finding this partition is recursive.

The first step of **Strong-Split** is to apply the relevant **Weak-Split** algorithm of the previous section to split G into G_1, G_2, and $G_\sim$. Now we check whether $t_1 \ge |G_1|$ and $t_2 \ge |G_2|$. If so, then we are done—let $G_1^* = G_1$, $G_2^* = G_2$, and $G_\sim^* = G_\sim$.

Otherwise, if $t_1 < |G_1|$ then we recursively call **Strong-Split** to partition G_1 into $\hat{G}_1, \hat{G}_2, \hat{G}_\sim$ such that $\hat{G}_1$ is of size no more than t_1 and $\hat{G}_2$ is of size no more than $|G_1| - t_1$. Then we set $G_1^* = \hat{G}_1$, $G_\sim^* = G_\sim \cup \hat{G}_\sim$, and $G_2^* = \hat{G}_2 \cup G_2$. By construction $|G_1^*| \le t_1$; we verify that

$$\begin{aligned} |G_2^*| &= |\hat{G}_2| + |G_2| \\ &\le |G_1| - t_1 + |G_2| \\ &\le N - t_1 \\ &\le t_2. \end{aligned}$$

Similarly, if $t_2 < |G_2|$ we perform a recursive call of **Strong-Split** on G_2.

The running time to find these subsets of size no more than t_1 and t_2 is still linear in N in the grid graph case because each recursive call is for a graph of size at most $6/7$ the size of the graph at the previous level of recursion, so that the running times form a decreasing geometric series whose first term is $O(N)$. Accordingly, this series sums to a constant multiple of N. A similar argument shows that the running time for the local-mesh case is still $O(N \log N)$.

The size of the boundary from this recursive algorithm can also be analyzed as a geometric series. The size of $G_\sim^*$ is always bounded by the sum of the sizes of boundaries generated by **Weak-Split** at each recursive level. This boundary in the worst case has size $\beta N^{2/3}$ at the top level of the recursion, and then $\beta((6/7) \cdot N)^{2/3}$ at the first level down, $\beta((6/7)^2 \cdot N)^{2/3}$ at the second level, and so on, where $\beta = 1.71$ in the grid graph case and $\beta = r(448/V)^{1/3}$ in the case of r-local meshes. Summing the infinite geometric sum, we get an upper bound:

$$\text{total boundary size} \le \frac{\beta N^{2/3}}{1 - (6/7)^{2/3}}$$

$$\leq 10.24\beta N^{2/3}.$$

Thus, to partition G into G_1, G_2, and $G_\sim$ so that G_1 and G_2 are balanced, we would use this algorithm with t_1 and t_2 set to $N/2$. The sizes of G_1 and G_2 would be in the range $N/2 - O(N^{2/3})$ to $N/2$. The boundary size would be no more than $10.24\beta N^{2/3}$.

In general, if the expression $6N/7$ of Theorem 2.1 were replaced by αN, then the coefficient of 10.24β would be replaced by $\beta/(1-\alpha^{2/3})$. This fraction is the motivation for our heuristic proposed in §3.

Finally, we consider the case when $p > 2$. We assume in this case that p is a power of 2, although the following analysis and algorithm can be generalized to arbitrary positive integers. We define a recursive algorithm **Strong-Multi-Split** whose inputs are a domain D and an integer p; its function is to partition D into p subdomains. Assume D has N nodes. **Strong-Multi-Split** checks whether $p = 1$; if so, then it returns D as the one subdomain. If $p \geq 2$, then **Strong-Multi-Split** calls **Strong-Split** with $t_1 = t_2 = N/2$ to split the original region into two roughly equal subdomains, say, D_1 and D_2. Then **Strong-Multi-Split** calls itself twice recursively to split each of D_1 and D_2 into $p/2$ subdomains.

We can visualize the Algorithm **Strong-Multi-Split** as a tree of calls to **Strong-Split**; the tree has $\log_2 p$ levels. The top level separator has at most $10.24\beta N^{2/3}$ nodes. In general, there will be 2^i pieces at the ith level, each of size at most $N/2^i$, and to separate one of these pieces will take at most $10.24\beta N^{2/3}/2^{2i/3}$ nodes, hence to separate all of the pieces at the ith level will incur a boundary size of at most $10.24\beta N^{2/3}2^{i/3}$. We have to find all the separators up to the level when $i = (\log_2 p)-1$, so the last level will have total separator sizes of $10.24\beta N^{2/3}(p/2)^{1/3}$. The separator sizes on each level form a finite geometric series that can be summed:

$$\begin{aligned}\text{total boundary size} &\leq \frac{10.24\beta N^{2/3}p^{1/3} - 10.24\beta N^{2/3}}{2^{1/3}-1} \\ &\leq 39.4\beta N^{2/3}(p^{1/3}-1).\end{aligned}$$

We state this result as a theorem.

THEOREM 4.1. *Let G be an N-node graph as in Theorem 2.1 or Theorem 2.2. Then G can be partitioned into $G_1, G_2, \cdots, G_p, G_\sim$ such that $|G_i| \leq N/p$ for all $i = 1, \cdots, p$, and such that the size of the boundary $G_\sim$ is at most $39.4\beta N^{2/3}(p^{1/3}-1)$ when p is a power of 2.*

If we plug in β from Theorem 2.1, we come out with a bound of $67.4N^{2/3}(p^{1/3}-1)$. If we plug in β from Theorem 2.2 into this theorem, we come out with an upper bound of $301V^{-1/3}rN^{2/3}(p^{1/3}-1)$.

The running time for **Strong-Multi-Split** is $O(N \log p)$ in the grid-graph case, because the algorithm spends $O(N)$ time on each of the $\log p$ levels of the tree. For the local-mesh case, the running time is $O(N \log N)$ because the sorting need only be done in the very first call to **Weak-Split-FE** (see [31]).

5. Parallelizing the algorithm. As mentioned in the introduction, the main motivation for domain partitioning is as a preprocessing step for parallel solution of an elliptic problem. This leads to the following question: Can our partitioning algorithms themselves be parallelized? Algorithm **Strong-Multi-Split** parallelizes in a straightforward manner because the top-level call to **Strong-Split** yields two smaller domains of nearly equal size. These two could be assigned to independent processors. The algorithm continues to produce independent problems at each level of the recursion, ultimately using $p/2$ processors.

It might also be of some interest to parallelize each call to **Strong-Split**, which in turn requires a parallelization of the **Weak-Split** algorithm. In the case of local graphs this requires a parallelization of sorting. The literature on sorting is too vast to survey here, but a good starting point is Knuth [17].

6. A lower bound for partitioning. As mentioned at the end of §4 our best upper bound grows like $rN^{2/3}p^{1/3}$. In this section we derive a lower bound for the cube showing that this product is the best possible. Previous authors have deduced lower bounds for separating grid graphs, including [8], [13], and [18]. The techniques of this section have been used before, but the main result (Theorem 6.4) is new.

We assume that r is an integer, N is a perfect cube, $p \geq 8$, $N/p \geq 16$, and $N^{1/3} \geq 4r$ to make our arguments work out.

The graph G under consideration for this section is an r-local domain under the ∞-norm on $\mathbf{R}^3$, where r is a positive integer. Assume N is a perfect cube, say $N = k^3$. The nodes of G are the N lattice points lying in $[1,k]\times[1,k]\times[1,k]$. An edge is inserted between every pair of nodes (x,y) such that $\|x-y\|_\infty \leq r$.

We start with a few lemmas. Let $\pi_1 : \mathbf{R}^3 \to \mathbf{R}^2$ be the projection given by $\pi_1(x,y,z) = (y,z)$. Define $\pi_2(x,y,z) = (x,z)$, and $\pi_3(x,y,z) = (x,y)$. Given a finite set of points $S \subset \mathbf{R}^3$, let $\pi_i(S)$ denote the image of S under π_i (where $i = 1,2,3$), and let $|\pi_i(S)|$ denote the cardinality of this image.

LEMMA 6.1. *For any finite set $S \subset \mathbf{R}^3$, $|\pi_1(S)| \cdot |\pi_2(S)| \cdot |\pi_3(S)| \geq |S|^2$.*

This follows from the much more general "product theorem" of Chung, Graham, Frankl and Shearer [6]; it can also be proved by a direct combinatorial argument (see [31] for the direct argument).

DEFINITION. Let $(G_1, \cdots, G_p, G_\sim)$ be a partition of G and let $x \in G_\sim$. Then we say that x is **on the boundary** of G_i for some i if there is a point $y \in G_i$ such that $\|x-y\|_\infty \leq r$.

LEMMA 6.2. *A node $x \in G_\sim$ can be on the boundary of at most eight distinct subdomains.*

Proof. Suppose $x = (x_1, x_2, x_3)$ is on the boundary of G_i, and suppose $y = (y_1, y_2, y_3)$ is the node of G_i specified by the previous definition. Then $y_d = x_d + \sigma_d e_d$ for $d = 1,2,3$, where $\sigma_d = \pm 1$ and $e_d \in \{0,1,2,\cdots,r\}$. The triple $(\sigma_1,\sigma_2,\sigma_3)$ is called the *signature* of G_i with respect to x. (G_i may have more than one signature—choose one arbitrarily.) Let G_j be another subdomain that has x on its boundary. We claim that G_j cannot have the same signature as G_i; if G_j had a vertex $y' = (x_1 + \sigma_1 e_1', x_2 + \sigma_2 e_2', x_3 + \sigma_3 e_3')$, then y' and y would be distance no more than r from each other, contradicting the assumption that they are in different connected components. Therefore, every G_i for which x is on the boundary has a different signature, and there are at most eight possible signatures. □

The following lemma is geometrically obvious and its proof is omitted.

LEMMA 6.3. *Let $T = \{1,\cdots,k\}$. Let S be a subset of T of size no more than $k-r$. Then there are at least r points of $T - S$ that are within r units of S.*

The main theorem for this section is the following.

THEOREM 6.4. *Let G be the N-node graph described above. Then, in order to partition G into $(G_1, G_2, \cdots, G_p, G_\sim)$ such that $N/(2p) \leq |G_i| \leq N/p$ for $i = 1,\cdots,p$, the size of $G_\sim$ must be at least $0.026rN^{2/3}p^{1/3}$. As mentioned above, we assume that $N/p \geq 16$, $k \geq 4r$, and $p \geq 8$.*

Proof. Let s be the size of a G_i; by assumption $N/(2p) \leq s \leq N/p$. By Lemma 6.1 there is a projection π_d ($d \in \{1,2,3\}$) such that $|\pi_d(G_i)| \geq s^{2/3}$. Without loss of generality, $d = 1$. Let x be a point in $\pi_1(G_i)$. We say that x is *fully loaded* if more

than $k-r$ points of G_i project to x. Let t be the number of fully loaded image points. The first claim is that $t \leq s^{1/3}$. To see this, note that for each fully loaded image point there are at least $k-r$ points of G_i associated with that image point. A computation gives a bound on $k-r$:

$$\begin{aligned} k-r &\geq 3k/4 \\ &\geq 3N^{1/3}/4 \\ &\geq 3s^{1/3}p^{1/3}/4 \\ &\geq 3s^{1/3}/2. \end{aligned}$$

Thus, each fully loaded image point accounts for at least $1.5s^{1/3}$ points of G_i, which means that there can be no more than $2s^{2/3}/3$ fully loaded image points since G_i has at most s points. Therefore, there are at least $s^{2/3}/3$ image points of G_i that are not fully loaded, i.e., at least $N^{2/3}/(3 \cdot 2^{2/3}p^{2/3})$ image points. For each image point not fully loaded, there are r points of $G-G_i$ projecting to the image point such that all r points are within distance r of G_i (this follows from Lemma 6.3). These points must be in $G_\sim$ and must be on the boundary of G_i. Accordingly, each image point not fully loaded contributes r points to the boundary of G_i.

This is true for all p subdomains. Therefore, the aggregate sum of the boundary sizes for each subdomain is pr times the quantity in the previous paragraph, that is, $rN^{2/3}p^{1/3}/(3 \cdot 2^{2/3})$. This overcounts the size of $G_\sim$ because a particular node could be on the boundary of more than one G_i. But by Lemma 6.2 each point of $G_\sim$ lies on the boundary of no more than eight of the G_i's. Thus, the aggregate overcounts $|G_\sim|$ by at most a factor of eight. Dividing by eight gives the theorem. □

7. Partitioning a small union of boxes. In §4 we saw that an N-node three-dimensional grid graph can be separated into two roughly equal pieces with a boundary of no more than $10.24\beta N^{2/3}$ nodes. In this section we consider the special case that the original domain is the union of a small number of nonoverlapping boxes. The goal for this section is to get a better bound on the separator size in the case that the number of boxes is small, and to get a simpler (nonrecursive) algorithm for computing the separators.

For this section a *box* is the graph that results from taking the finite-difference mesh induced by lattice points lying in a Cartesian product of three intervals in $\mathbf{R}^3$. Two boxes are *disjoint* if they do not have any point in common. Two disjoint boxes B_1 and B_2 are *adjacent* if there is a point $x \in B_1$ and a point $y \in B_2$ such that x and y are neighbors as lattice points—x and y are called *contact points.*

The kind of domain we consider is as follows. We start with k disjoint boxes, say, $B_1, \cdots, B_k$, embedded in $\mathbf{R}^3$. The nodes within each box are already interconnected. Then we interconnect the boxes by inserting an edge between all the pairs of contact points on all the pairs of adjacent boxes. Such a graph will be called a *joining* of k boxes.

This type of domain comes up often in the literature: for example, Golub and Mayers [15] used a region that was a joining of two boxes (in two dimensions) as their main example. Our main theorem toward constructing a separator is below, but we need two preliminary lemmas.

LEMMA 7.1. *Let G be an N-node box. Choose any integer a. Consider partitioning G into three sets given by:*

$$G_1 = \{(x,y,z) \in G : \ x+y+z < a\},$$
$$G_2 = \{(x,y,z) \in G : \ x+y+z > a\},$$

$$G_\sim = \{(x,y,z) \in G : \ x+y+z = a\}.$$

Then $|G_\sim| \leq N^{2/3}$.

Proof. Suppose the box is of size $s \times t \times u$ so that $N = stu$. Without loss of generality, $s \leq t \leq u$, so that $st \leq N^{2/3}$. We claim that the number of points in G that satisfy $x+y+z = a$ is bounded by st. This is because there are a total of st ways to choose x and y so that (x,y,z) lies in G. Once x and y are chosen, there is a unique z that satisfies $x+y+z = a$. Thus, the number of points of $G_\sim$ is at most $N^{2/3}$. □

LEMMA 7.2. *Let G be an N-node graph that is the joining of k boxes. Let a, G_1, G_2, and $G_\sim$ be as above. Then $|G_\sim| \leq k^{1/3}N^{2/3}$.*

Proof. Let the boxes be $B_1, \cdots, B_k$, and suppose their sizes are $b_1, \cdots, b_k$. Since G is a disjoint union of nodes, then $b_1 + \cdots + b_k = N$. We apply the preceding lemma to each box individually and then sum the results to get a bound on the size of $G_\sim$:

$$|G_\sim| \leq b_1^{2/3} + \cdots + b_k^{2/3}.$$

Given the constraint that $b_1 + \cdots + b_k = N$, this sum on the right-hand side is maximized when all of the b_i are equal, i.e., when they all contain N/k nodes. This means that

$$\begin{aligned} |G_\sim| &\leq k \cdot \left(\frac{N}{k}\right)^{2/3} \\ &\leq k^{1/3}N^{2/3}. \end{aligned}$$

□

Using this second lemma, it is easy to find a partition of a joining of boxes.

THEOREM 7.3. *Let G be an N-node graph that is a joining of k boxes. Then G can be partitioned into p subdomains $G_1, G_2, \cdots, G_p, G_\sim$ such that the boundary $G_\sim$ is the union of $p-1$ planes of total size no more than $(p-1)k^{1/3}N^{2/3}$, and such that $|G_i| = N/p \pm O(k^{1/3}N^{2/3})$ for $i = 1, \cdots, p$.*

Remark. When $p = 2$ in this theorem, $|G_\sim| \leq k^{1/3}N^{2/3}$. This bound should be compared to the bound given for Algorithm **Strong-Split**. When $p > 2$ the separator size for this theorem grows linearly with p, whereas the separator size for **Strong-Multi-Split** grows only like $p^{1/3}$. Accordingly, the performance of this theorem relative to **Strong-Multi-Split** decays rapidly for larger p.

Proof. For each integer a, let $l(a)$ be the number of nodes of G that satisfy $x+y+z \leq a$, that is,

$$l(a) = \left|\{(x,y,z) \in G : \ x+y+z \leq a\}\right|.$$

Consider some i in the range $i = 1, \cdots, p-1$. Let a_i be the smallest integer such that $l(a_i) \geq i \cdot (N/p)$. Observe that, since $l(a_i - 1) < i \cdot (N/p)$, then by the preceding lemma, $l(a_i) = i \cdot (N/p) + O(k^{1/3}N^{2/3})$, because the $l(a_i) - l(a_i - 1)$ is equal to the number of nodes of G on the plane $\{x+y+z = a_i\}$.

We compute a_i for each i, $1 \leq i \leq p-1$. Set $a_0 = -\infty$ and $a_p = +\infty$. Finally, we set

$$G_i = \{(x,y,z) \in G : \ a_{i-1} < x+y+z < a_i\}$$

for $i = 1, \cdots, p$ and we set

$$G_\sim = \{(x,y,z) \in G : \ x+y+z = a_i \text{ for some } i,\ 1 \leq i \leq p-1\}.$$

We must show the bound on the size of $G_\sim$. Notice that by the preceding lemma, each plane of $G_\sim$ intersects at most $k^{1/3}N^{2/3}$ nodes of G. Since there are $p-1$ planes

TABLE 1
Results for Algorithm **Strong-Multi-Split**.

Domain	N	$\lvert G_\sim \rvert$	Ratio	Max. $\lvert G_\sim \rvert$	Min. $\lvert G_\sim \rvert$
D_1	26,970	2,610	1.07	60,616	46.7
D_2	24,540	1,948	1.01	56,918	43.9
D_3	16,800	1,459	1.13	44,212	34.1
D_4	13,955	1,699	1.10	39,068	30.1

in $G_\sim$, this proves the bound on the separator. It is clear from the arguments in the preceding paragraphs that the number of nodes enclosed between any two successive planes of the form $x + y + z = a_i$ is $N/p \pm O(k^{1/3}N^{2/3})$. □

There is a simple $O(N + p)$ algorithm to compute the separator given by this theorem, namely, for each point (x, y, z) in G, compute the sum $x+y+z$. Then make a tally for each a of the number of nodes (x, y, z) in G such that $x + y + z = a$. Make a pass through these tallies to compute partial sums (the partial sums correspond to the quantities $l(a)$ in the proof of Theorem 7.3), and from the partial sums, determine the $p-1$ planes. Note that this algorithm does not use the fact that G is a joining of k boxes; only the bounds on the resulting partition depend on this fact. This algorithm will be named **Diag-Split**.

Finally, we should remark that Algorithms **Weak-Split** and **Strong-Split** have the property that they generate a collection of subdomains on which they could then be reapplied. This is not true for the methods of this section because the subdomains given by Theorem 7.3 are no longer unions of a small number of boxes. In particular, this means that **Diag-Split** is not meant to be applied recursively.

8. Computational experience. We tried out the Algorithms **Strong-Multi-Split** and **Diag-Split** on four domains with a finite-difference grid graph embedded. In all four cases we took $p = 8$. Domain D_1 is a $29 \times 30 \times 31$ box; domain D_2 is a $29 \times 30 \times 31$ box with an $18 \times 5 \times 27$ hole removed from the center; domain D_3 is an L-shaped region with one arm $40 \times 10 \times 20$ and the other arm $11 \times 50 \times 20$ (the overlap being an $11 \times 10 \times 20$ box). Finally, domain D_4 is the set of lattice points of $\mathbf{R}^3$ in the nonconvex region given by

$$\sqrt{|x_1|} + \sqrt{|x_2|} + \sqrt{|x_3|} < \sqrt{35}.$$

The results for Algorithm **Strong-Multi-Split** are summarized in Table 1. The first column is the domain, the second is the number of nodes. The third column is the total boundary size for each of the regions, and the fourth column is the ratio of size of the largest subdomain to the smallest subdomain (ideally 1.0). The third and fourth column together are a measure of how well the algorithm performed. The fifth and sixth column are the upper and lower bounds on the boundary size given by Theorems 4.1 and 6.4. Notice that the actual boundary sizes in the third column are not close to either bound.

The results for Algorithm **Diag-Split** are summarized in Table 2. The first column is the domain, the second is the number of nodes, and the third is the number of boxes needed to express the domain as a disjoint union of boxes. The fourth, fifth, and sixth columns are analogous to those in Table 1. Again, the theoretical upper bound (given by Theorem 7.3 this time) is an overestimate, but it is not as far off as the upper bounds of the previous table. There is no meaningful theoretical upper bound in the case of D_4 because it cannot be expressed as a disjoint union of a small number of

TABLE 2
Results for Algorithm **Diag-Split**.

Domain	N	k	$\lvert G_\sim \rvert$	Ratio	Max. $\lvert G_\sim \rvert$
D_1	26,970	1	3,847	1.37	6,295
D_2	24,540	6	3,359	1.37	10,742
D_3	16,800	2	2,473	1.29	5,785
D_4	13,955		1,322	1.26	

TABLE 3
Comparing **Strong-Split** *and* **Diag-Split** *when* $p = 2$.

Domain	**Strong-Split**		**Diag-Split**	
	$\lvert G_\sim \rvert$	ratio	$\lvert G_\sim \rvert$	ratio
D_1	870	1.00	675	1.05
D_2	413	1.01	585	1.03
D_3	220	1.02	421	1.00
D_4	1021	1.03	255	1.02

boxes.

For these test problems **Strong-Multi-Split** outperformed **Diag-Split** with respect to the balance ratio for all four domains, and it outperformed **Diag-Split** in all but the last case with respect to the boundary size.

As mentioned in §7, **Diag-Split** performs poorly for large p. We should compare **Strong-Split** to **Diag-Split** in the case that $p = 2$ to see how **Diag-Split** performs in its best case. These results are summarized in Table 3. Notice that the two algorithms are comparable except in the case of D_4, for which **Diag-Split** was much better. Domain D_4 is a bad case for **Strong-Split** because orthogonal planes through its center happen to hit a large cross section. We observed during computation that **Strong-Split** running on this domain called itself recursively down to 14 levels.

We conclude that **Strong-Multi-Split** works well in most cases as a general-purpose method for partitioning a domain into p subdomains for larger values of p. Algorithm **Diag-Split** seems to be preferable mainly because it is implemented easily without recursion and because it sometimes works better when $p = 2$.

9. Finding a good embedding. The upper bound on the boundary sizes in the r-local case all stem from Theorem 2.2, and for this theorem the upper bound depends on r. This means that **Strong-Split** and **Strong-Multi-Split** will probably work best when the graph is embedded in $\mathbf{R}^3$ so that the ratio of the longest edge to the minimum node separation is small.

This raises the following question: Given a graph G, can we embed it (or re-embed it) in $\mathbf{R}^3$ to make this ratio as small as possible? Consider the graph embedded in $\mathbf{R}^2$ as illustrated in Fig. 1. This embedding has a poor locality measure; in particular, one can verify with a ruler that the maximum edge length divided by the minimum node separation is about 24. On the other hand, if the same graph is embedded differently in $\mathbf{R}^2$ as illustrated in Fig. 2, then the graph is 3.8-local. In fact, the problem of determining the minimum r such that a graph is r-local is NP-hard. (See Garey and Johnson [9] for the definition of NP-hard.) This suggests that no efficient algorithm exists for the general problem. If we restate the definition of r-local for the case of embedding the graph in $\mathbf{R}^1$ (rather than $\mathbf{R}^3$), then an r-local graph is exactly the same

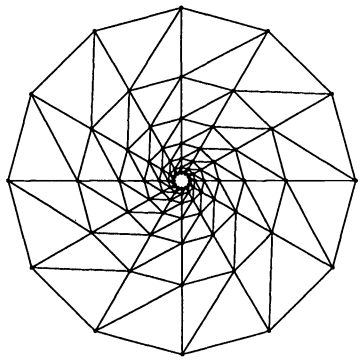


FIG. 1. *An embedding with poor locality.*

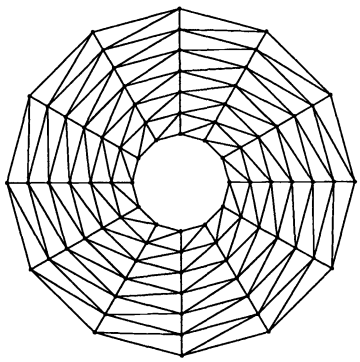


FIG. 2. *An embedding with good locality.*

as a graph of *bandwidth* r. A graph is said to have bandwidth r if the vertices can be embedded at the integer points of the number line such that all the edges have length no more than r. Papadimitriou [24] showed that finding the minimum bandwidth of a graph is NP-hard.

We are able to show that the bandwidth problem can be reduced to the problem of determining whether a graph is r-local with respect to the ∞-norm in three dimensions. This establishes the NP-hardness of minimum r-locality (the proof is contained in [31]).

10. Open questions and further research. It would be interesting to see how all the algorithms of this report behave on test problems generated by a mesh generator for real test domains.

The lower bound of Theorem 6.4 shows the same dependence on r, N, and p as the upper bound given by Theorem 4.1 except that the constants are off from each other by a factor of about 10^4. This gap should be closed. The computational results indicate that both bounds are off.

It was observed in the previous section that it is NP-hard to find the minimum r such that a graph is r-local. Are there approximation algorithms or other heuristics for this problem?

Examining Figs. 1 and 2 again, we see that Fig. 1 is a bad embedding because of varying densities of the nodes. Essentially, the embedding is regular except that

the nodes are laid out with increasing density near the center of the figure. In fact, this kind of unevenly dense triangulation arises in practice when the problem is to simulate phenomena with areas of local interest (flow around the wing of an airplane, for instance).

Is it possible to come up with a more general definition of r-local that takes into account varying densities of node points? The idea would be, of course, to come up with a theorem and algorithm like Theorem 2.2 and Algorithm **Strong-Split** that would apply for the new generalized definition.

We remark that, after our paper was submitted, Miller and Thurston [21] proposed an algorithm that in fact is able to handle varying densities of nodes and still have growth like $N^{2/3}$ in the separator size. Their algorithm, however, imposes restrictions on the graph not necessary to our approach. In particular, their graphs ("stable graphs") must be triangulations and must have a well-behaved exterior boundary.

Miller and Vavasis [22] have a class of graphs ("density graphs") that generalizes the local graphs of this paper and generalizes most features of the class of stable graphs.

Finally, another approach to domain decomposition and finite elements assumes that the domain is partitioned first, and then the triangulation is generated to respect the domain partition. Our assumption throughout this paper is that these two steps are done in the opposite order. An interesting future research topic is to address the partitioning problem before the grid is inserted.

Acknowledgments. Thanks to Jack Snoeyink of Stanford for making me aware of the Preparata and Shamos example mentioned in §2. Thanks to Alfred Schatz of Cornell for explaining some of the features of the method of finite elements. Thanks to Nicholas Higham of Cornell and Manchester and Mark Hanisch of Cornell for making valuable comments about the presentation.

REFERENCES

[1] M. J. Berger and S. Bokhari, *A partitioning strategy for nonuniform problems on multiprocessors,* IEEE Trans. Comput., C-36 (1987), pp. 570–580.

[2] P. E. Bjørstad and O. B. Widlund, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures,* SIAM J. Numer. Anal., 23 (1986), pp. 1097–1120.

[3] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, *An iterative method for elliptic problems on regions partitioned into substructures,* Math. Comp., 46 (1986), pp. 361–369.

[4] T. F. Chan, R. Glowinski, and J. Périaux, *Domain Decomposition Methods,* Proceedings of the Second International Conference on Domain Decomposition Methods, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[5] T. F. Chan and D. C. Resasco, *A framework for the analysis and construction of domain decomposition preconditioners,* Tech. Report CAM-87-09, Department of Mathematics, University of California, Los Angeles, CA, 1987.

[6] F. R. K. Chung, R. L. Graham, P. Frankl, and J. B. Shearer, *Some intersection theorems for ordered sets and graphs,* J. Combin. Theory Ser. A, 43 (1986), pp. 23–37.

[7] R. Courant, *Variational methods for the solution of problems of equilibrium and vibrations,* Bull. Amer. Math. Soc. (N.S.), 49 (1943), pp. 1–23.

[8] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman, *Applications of an element model for Gaussian elimination,* in *Sparse Matrix Computations,* J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness,* W. H. Freeman and Company, San Francisco, 1979.

[10] J. A. George, *Nested dissection of a regular finite element mesh,* SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[11] A. GEORGE AND J. W. H. LIU, *An automatic nested dissection algorithm,* SIAM J. Numer. Anal., 15 (1978), pp. 1053–1069.

[12] J. R. GILBERT, *Graph Separator Theorems and Sparse Gaussian Elimination,* Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA, 1980.

[13] J. R. GILBERT, J. P. HUTCHINSON, AND R. E. TARJAN, *A separator theorem for graphs of bounded genus,* J. Algorithms, 5 (1984), pp. 391–407.

[14] R. GLOWINSKI, G. H. GOLUB, G. A. MEURANT, AND J. PÉRIAUX, *Domain Decomposition Methods,* Proceedings of the First International Conference on Domain Decomposition Methods, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[15] G. H. GOLUB AND D. MAYERS, *The use of pre-conditioning over irregular regions,* in Computing Methods in Applied Sciences and Engineering VI, R. Glowinski and J.-L. Lions, eds., North–Holland, Amsterdam, 1984.

[16] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations,* Second Edition, Johns Hopkins University Press, Baltimore, 1989.

[17] D. E. KNUTH, *The Art of Computer Programming Volume 3, Searching and Sorting,* Addison-Wesley, Reading, MA, 1973.

[18] R. J. LIPTON, D. J. ROSE, AND R. E. TARJAN, *Generalized nested dissection,* SIAM J. Numer. Anal., 16 (1979), pp. 346–358.

[19] R. J. LIPTON AND R. E. TARJAN, *A separator theorem for planar graphs,* SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[20] ———, *Applications of a planar separator theorem,* SIAM J. Computing, 9 (1980), pp. 615–627.

[21] G. L. MILLER AND W. THURSTON, *Separators in two and three dimensions,* preprint, 1989.

[22] G. L. MILLER AND S. A. VAVASIS, *Density graphs and separators,* Tech. Report 90-1169, Department of Computer Science, Cornell University, Ithaca, NY, 1990.

[23] R. MORISON AND S. OTTO, *The scatter decomposition method for finite elements,* J. Sci. Comput., 2 (1987), pp. 59–76.

[24] C. H. PAPADIMITRIOU, *The NP-completeness of the bandwidth minimization problem,* Computing, 16 (1976), pp. 263–270.

[25] A. POTHEN, H. D. SIMON, AND K.-P. LIU, *Partitioning sparse matrices with eigenvectors of graphs,* Tech. Report RNR-89-009, NASA Ames Research Center, Moffett Field, CA, 1989.

[26] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry: An Introduction,* Springer–Verlag, New York, 1985.

[27] G. STRANG AND G. J. FIX, *An Analysis of the Finite Element Method,* Prentice–Hall, Englewood Cliffs, NJ, 1973.

[28] J. F. THOMPSON, Z. U. A. WARSI, AND C. W. MASTIN, *Numerical Grid Generation,* North–Holland, New York, 1985.

[29] M. J. TODD, *The Computation of Fixed Points and Applications,* Springer–Verlag, Berlin, 1976.

[30] J. D. ULLMAN, *Computational Aspects of* VLSI, Computer Science Press, Rockville, MD, 1984.

[31] S. A. VAVASIS, *Automatic domain partitioning in three dimensions,* Tech. Report 90-1082, Department of Computer Science, Cornell University, Ithaca, NY, 1990.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 4, pp. 971–989, July 1991
© 1991 Society for Industrial and Applied Mathematics
015

A DEFERRED CORRECTION METHOD FOR NONLINEAR TWO-POINT BOUNDARY VALUE PROBLEMS: IMPLEMENTATION AND NUMERICAL EVALUATION*

J. R. CASH† AND M. H. WRIGHT‡

Abstract. A deferred correction method for the numerical solution of nonlinear two-point boundary value problems has been derived and analyzed in two recent papers by the first author. The method is based on mono-implicit Runge–Kutta formulas and is specially designed to deal efficiently with problems whose solutions contain nonsmooth parts—in particular, singular perturbation problems of boundary layer or turning point type. This paper briefly describes an implementation of the method and gives the results of extensive numerical testing on a set of nonlinear problems that includes both smooth and increasingly stiff (and difficult) problems. Results on the test set are also given using the available codes COLSYS and COLNEW. Although the intent is not to make a formal comparison, the code described appears to be competitive in speed and storage requirements on these problems.

Key words. deferred correction, two-point boundary value problems, software evaluation

AMS(MOS) subject classifications. 65L10, 65L20

1. Introduction.

In [CASH86], [CASH88], a deferred correction method is described and analyzed that is suitable for the numerical solution of the general first-order, nonlinear, two-point boundary value problem

$$\frac{dy}{dx} = f(x,y), \quad a \le x \le b, \quad g\big(y(a), y(b)\big) = 0, \quad y \in \Re^s. \tag{1.1}$$

The motivating idea behind this approach is to combine the desirable properties of two other widely used approaches to the solution of (1.1): collocation as typified by COLSYS [ACR79a], [ACR79b] or its more recent variant COLNEW [BA87]; and iterated deferred correction as typified by the PASVA series of codes (see, e.g., [PER68]), based on the approach of Lentini and Pereyra [LP74]. The unifying theme connecting collocation and deferred correction for first-order systems is the use of *implicit Runge–Kutta methods.*

The COLSYS codes are based on an expensive high-order fully implicit Runge–Kutta method defined on Gauss quadrature points. (We shall use the term "COLSYS codes" when discussing features common to both COLSYS and COLNEW.) The PASVA codes are based on the trapezium rule, which is a second-order Runge–Kutta method defined on Lobatto points; increased accuracy is obtained through deferred corrections. A typical PASVA implementation allows up to ten deferred corrections, giving a method with overall order about 20.

Intermediate between these two approaches is one that uses a different class of Runge–Kutta method—one that is more expensive than the trapezium rule of PASVA, but less expensive than the high-order method of the COLSYS codes. Increased order with such a method is obtained using deferred correction; however, only a few such corrections (say, a maximum of three) are allowed.

* Received by the editors September 1989; accepted for publication (in revised form) September 23, 1990.
† Department of Mathematics, Imperial College, London SW7 2BZ, United Kingdom.
‡ AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

With this approach in mind, the new class of mono-implicit Runge–Kutta methods was derived [CS82a], [CS82b]. These formulae are implemented in a deferred correction framework in a radically different way from that used by Lentini and Pereyra. Suppose that we have available mono-implicit Runge–Kutta formulae φ_4, φ_6, and φ_8 of orders four, six, and eight, respectively. In particular, the kth component of φ_4 is given by

$$h\varphi_4^k(\eta) = \eta_{k+1} - \eta_k - \tfrac{1}{6}h\Big(f_{k+1} + f_k + 4f\big(x_{k+1/2}, \tfrac{1}{2}(\eta_k + \eta_{k+1}) - \tfrac{1}{8}h(f_{k+1} - f_k)\big)\Big),$$

where $f_k = f(x_k, \eta_k)$ and $f_{k+1} = f(x_{k+1}, \eta_{k+1})$. What is particularly interesting about φ_4 is that it has a single value for each subinterval $[x_k, x_{k+1}]$, and this value can be computed explicitly in terms of η_k and η_{k+1}. The same property holds for φ_6 and φ_8.

Solutions η, $\bar{\eta}$, and $\bar{\bar{\eta}}$ of order four, six, and eight can be computed using

$$\varphi_4(\eta) = 0 \tag{1.2a}$$

$$\varphi_4(\bar{\eta}) = -\varphi_6(\eta) \tag{1.2b}$$

$$\varphi_4(\bar{\bar{\eta}}) = \varphi_4(\bar{\eta}) - \varphi_8(\bar{\eta}). \tag{1.2c}$$

This procedure has a significantly smaller linear algebra cost than that of COLSYS, but stability and accuracy are normally degraded.

Compared with PASVA, the approach (1.2) has the advantage that the deferred corrections are *local*, i.e., if we consider the nth component of (1.2b) or (1.2c), the deferred correction term uses information only from the interval $[x_n, x_{n+1}]$. Locality of the deferred correction is expected to improve performance on singular perturbation problems compared with using high-order corrections based on many adjacent grid points. The approach of Lentini and Pereyra seems to require some smoothness in the distribution of grid points, and the PASVA code never removes points in going from one grid to another. The result is that PASVA works extremely well on smooth problems, but can be inefficient for difficult singular perturbation problems.

Although theoretical analysis indicates that a method based on (1.2a)–(1.2c) should be effective for solving difficult problems of the form (1.1), the final proof of an implementation is its performance in practice. Our code HAGRON is a type-insensitive implementation of the deferred correction scheme (1.2) with the order chosen automatically. In §4, we give details of numerical experiments for HAGRON on a set of difficult nonlinear test problems. Computational results are also given for the COLSYS codes. However, we emphasize that it is *not* a major aim of this paper to compare codes for solving (1.1).

Any attempt to compare complex pieces of software is inherently extremely difficult and contentious. Nevertheless, we believe that numerical experiments involving different codes are essential to advance understanding of methods for boundary value problems.

It is worth noting the contrast to the state of the art in software for initial value problems. Both Adams and Runge–Kutta methods have been programmed in so many forms during the past 20 years that any dramatic improvements in performance now seem unlikely. With the possible exception of problems with singular solutions, most initial value problems can be solved by existing codes with reasonable efficiency and reliability. By comparison, codes for boundary value problems, although in wide

use, are still in an embryonic state. This situation is not surprising, given that the underlying mathematical theory for initial value problems is much better understood than for the boundary value case, and that the problem class is much smaller.

Our numerical experiments have therefore been guided by the following three considerations:

- It seems useful to establish a set of test problems that can be enlarged over time, and to begin to develop a set of benchmarks for comparisons among codes.
- We believe that boundary value codes are still *evolving*, and are likely to continue to do so for some time. This view is supported by the multiple versions of the PASVA codes, and by the development of COLNEW, an updated version of COLSYS. Boundary value methods are composed of several major numerical subproblems—for example, factorization of highly structured linear systems, solution of nonlinear algebraic equations, and mesh refinement. Our results indicate that the performance of boundary value methods is affected in complicated and interconnected ways by the strategies used in solving these subproblems. Numerical experiments should help not only to identify the important relationships among these computational tasks, but also to suggest improved techniques for some or all of the subproblems.
- Certain problems of the general class (1.1) can cause special difficulties for boundary value solvers, including:
  (i) problems defined on infinite intervals;
  (ii) problems with singularities, e.g.,

$$y'' + \frac{y'}{x} + \frac{y}{(1+x)^2} = 1, \quad y(1) = \alpha, \quad y'(0) = 0;$$

  (iii) eigenvalue and other parameter estimation problems;
  (iv) bifurcation problems.
  It is hoped that a detailed understanding of the performance of an algorithm on problem (1.1) will lead to the development of better methods for the solution of (i)–(iv).

Much of the mathematical theory associated with the method of HAGRON has been described in detail elsewhere [CASH86], [CASH88]. The procedures used to solve the nonlinear equations are discussed in [CW90]. The mesh refinement procedure is explained in the next section.

2. Mesh refinement. In this section we describe in some detail the algorithm used to choose the mesh. The important features of the strategy for linear systems have been described in [CASH86], [CASH88], and we therefore concentrate on the extension of the strategy to nonlinear problems.

Each problem in our test set contains a parameter that controls the degree of difficulty encountered by a numerical method. Usually, a "small" parameter value leads to a singular perturbation problem with rapidly varying parts in its solution trajectory. The aim of a grid-selection strategy is to add grid points to these difficult regions, and to retain as few points as possible in the smoother regions. Two main guidelines (which are by no means infallible) used to detect that a problem includes regions of rapid variation are:

1. The error estimate in a converged solution is very large at some points unless the grid spacing is sufficiently small;

2. The Newton iteration scheme for the nonlinear equations exhibits slow convergence or fails to converge.

As explained in §1, the algorithm tries to compute three separate solutions η, $\bar{\eta}$, and $\bar{\bar{\eta}}$, which are of orders four, six, and eight, respectively. For purposes of this discussion, we assume the following: the current grid contains the $N+1$ points x_0, $x_1, \ldots, x_N$; the solution of the differential equation has dimension s; tolerances are imposed on components $j_1, j_2, \ldots, j_{ntol}$, where $1 \le ntol \le s$ and $J_s = \{j_1, \ldots, j_{ntol}\}$; and tolerance tol_j is imposed on the jth component.

If the nonlinear iteration procedures for η and $\bar{\eta}$ converge, we compute the error estimates

$$err_{i,j} = \frac{|\eta_{i,j} - \bar{\eta}_{i,j}|}{\max(1, |\eta_{i,j}|)}, \qquad 0 \le i \le N, \quad j \in J_s,$$

where $\eta_{i,j}$ denotes the jth component of the solution vector at x_i. If

$$\max_i \; err_{i,j} \le tol_j, \quad j \in J_s, \tag{2.1}$$

the solution $\bar{\eta}$ is accepted, and the algorithm stops.

If the error test (2.1) is not satisfied and the iteration scheme for $\bar{\bar{\eta}}$ converges, we compute

$$\overline{err}_{i,j} = \frac{|\bar{\eta}_{i,j} - \bar{\bar{\eta}}_{i,j}|}{\max(1, |\bar{\eta}_{i,j}|)} \quad \text{and test whether} \quad \max_i \overline{err}_{i,j} \le tol_j, \;\; j \in J_s. \tag{2.2}$$

If (2.2) is satisfied, the algorithm stops and accepts $\bar{\bar{\eta}}$; if not, the mesh is redistributed. The method is thus able to accept either $\bar{\eta}$ or $\bar{\bar{\eta}}$, providing that an appropriate stopping criterion is satisfied.

When neither (2.1) nor (2.2) is satisfied, the logic of the mesh-selection strategy is divided into the following mutually exclusive and exhaustive cases.

Case 1. The nonlinear iterations for η, $\bar{\eta}$, and $\bar{\bar{\eta}}$ all converge.

When all three Newton iterations converge, the next step is to re-grid so as to equidistribute an error estimate. Since it is difficult in general to find a valid estimate for the error in $\bar{\bar{\eta}}$, we have to be content with using a Fehlberg embedding approach. Our equidistribution algorithm is based on the estimate $\varphi_4(\bar{\bar{\eta}}) - \varphi_4(\bar{\eta})$ $(\equiv -\varphi_8(\bar{\eta}))$, which in our numerical experience leads to a more computationally efficient algorithm than the estimate $\bar{\bar{\eta}} - \bar{\eta}$.

The general idea of equidistribution is well known (see, for example, [AMR88, pp. 362–363]) and widely used. Our version of equidistribution uses Φ_i, the principal error function of the Runge–Kutta method at the point x_i, in deciding whether to add points in the interval $[x_{i-1}, x_i]$.

Let U_i^j denote the jth component of the current numerical solution U at the grid point x_i, with an analogous notation for other vectors. The quantity E_i^j is defined as

$$E_i^j = \frac{\varphi_{8,i}^j(\bar{\eta})}{\max(1, |U_i^j|)}, \qquad 0 \le i \le N, \quad j \in J_s,$$

and $\tilde{j}$ is the index of the component for which the maximum of $|E_i^j|$ is achieved.

Letting $h_i = x_i - x_{i-1}$, we have

$$h_i^6 \, \Phi_i^{\tilde{j}} = E_i^{\tilde{j}} + O(h_i^8). \tag{2.3}$$

We wish to have a uniform mesh spacing $\bar{h}_i$ in $[x_{i-1}, x_i]$ satisfying

$$\bar{h}_i^6 \, \Phi_i^{\tilde{\jmath}} = tol_{\tilde{\jmath}} + O(\bar{h}_i^8). \tag{2.4}$$

From (2.3) and (2.4) it follows that $\bar{h}_i$ should be chosen so that

$$\frac{\bar{h}_i}{h_i} = \left(\frac{tol_{\tilde{\jmath}}}{E_i^{\tilde{\jmath}}} \right)^{1/6},$$

which in turn means that there should be $\lfloor h_i/\bar{h}_i \rfloor + 1$ equally spaced points in the interval $[x_{i-1}, x_i]$, where $\lfloor \cdot \rfloor$ denotes the integer part of its argument. This is the (standard) strategy for *adding* points.

The above strategy relies on the existence of a value of i for which $tol_{\tilde{\jmath}} < E_i^{\tilde{\jmath}}$. If no such value exists, we simply base our equidistribution on the error estimate $10^p E_i^{\tilde{\jmath}}$, where p is the minimum integer for which $tol_{\tilde{\jmath}} < 10^p E_i^{\tilde{\jmath}}$ for at least one value of i. The approach of basing an equidistribution algorithm on the error estimate $10^p E_i^{\tilde{\jmath}}$ has the desirable property of adding points to places where the error estimate is relatively large.

The algorithm for *removing* points from the grid involves additional heuristics. For purposes of explanation, assume that we are considering whether to remove points from the left-hand end of the grid; the general strategy follows in a similar way.

In the intervals $[x_0, x_1]$ and $[x_1, x_2]$, we have relations of the following form:

$$\begin{aligned} (x_1 - x_0)^6 \, \Phi_0^{\tilde{\jmath}} &= E_0^{\tilde{\jmath}} + O(x_1 - x_0)^8; \\ (x_2 - x_1)^6 \, \Phi_1^{\tilde{\jmath}} &= E_1^{\tilde{\jmath}} + O(x_2 - x_1)^8 \end{aligned}$$

(cf. (2.3)). We wish to examine the possibility of removing x_1. Let

$$\hat{\Phi} = \max\left(\frac{|E_0^{\tilde{\jmath}}|}{(x_1 - x_0)^6}, \frac{|E_1^{\tilde{\jmath}}|}{(x_2 - x_1)^6} \right).$$

If $(x_2 - x_0)^6 \, \hat{\Phi} \leq 0.1 \, tol_{\tilde{\jmath}}$, then x_1 is removed. Otherwise, the same test is applied to error estimates in the ranges $[x_1, x_2]$ and $[x_2, x_3]$.

If x_1 is removed, we consider the further possibility of removing x_2 as well, using the following strategy. Define

$$\tilde{\Phi} = \max\left(\frac{|E_2^{\tilde{\jmath}}|}{(x_3 - x_2)^6}, \hat{\Phi} \right).$$

If $(x_3 - x_0)^6 \, \tilde{\Phi} \leq 0.1 \, tol_{\tilde{\jmath}}$, then x_2 is removed.

This procedure is continued in an obvious fashion, with the heuristic restriction that no more than four consecutive points may be removed—i.e., if x_1, x_2, x_3, and x_4 have been removed, x_5 is retained and we then examine the possibility of removing x_6, shifting the "origin" to x_5. In general, if x_{n+1}, x_{n+2}, x_{n+3}, and x_{n+4} are removed, we always retain x_{n+5}.

Case 2. The nonlinear iterations converge for η and $\bar{\eta}$, but not for $\bar{\bar{\eta}}$.

The same steps are repeated as in Case 1, except that φ_8 is replaced by φ_6, and powers of 6 and 8 become 4 and 6, respectively.

Case 3. The nonlinear iteration converges for η, but not for $\bar{\eta}$.

The deferred correction $\varphi_6(\eta)$ is formed. The combination of convergence of the scheme for solving $\varphi_4(\eta) = 0$ and failure to solve $\varphi_4(\bar{\eta}) = -\varphi_6(\eta)$ probably signifies that $\varphi_6(\eta)$ is in some sense "large," i.e., η is a poor approximation to $\bar{\eta}$. Our procedure is first to examine the vector $\varphi_6(\eta)$ to determine whether any of its components are "very large" relative to the others.

Case 3a.

The strategy for choosing the "right place" to add mesh points is based on a simple analysis of the quantities $\{|E_i^j|\}$, defined with φ_6 rather than φ_8. If one component is "much larger" than the others, we try a certain number of times to add points only in the relevant interval.

The counter "*jcount*" is initialized to zero at the start of execution. The mesh refinement strategy is summarized by the pseudocode given below.

Calculate r_1 (the largest value of $|E_i^j|$) and r_2 (the second largest value of $|E_i^j|$), for $0 \le i \le N$, $j \in J_s$;
Let $\bar{\imath}$ be the index of the mesh point for which the maximum of $|E_i^j|$ is achieved;
if $r_1 < 10^{-2}$ **or** $r_1 < 10(\sum_{i,j} |E_i^j| - r_1)/N$ **then**
    halve the mesh and exit
end if;
jcount $\leftarrow$ *jcount* $+ 1$;
if *jcount* ≤ 2 **or** $\{r_1 > 2r_2$ **and** *jcount* $\le 4\}$ **then**
    if $\bar{\imath} = 0$ **then** add 15 points uniformly in $[x_0, x_1]$
    else if $\bar{\imath} = N$ **then** add 15 points uniformly in $[x_{N-1}, x_N]$
    else add 10 points uniformly in $[x_{\bar{\imath}-1}, x_{\bar{\imath}}]$ and $[x_{\bar{\imath}}, x_{\bar{\imath}+1}]$
    end if
else *jcount* $\leftarrow 0$; halve the mesh
end if

Case 3b.

In some cases, the situation is encountered where the iteration for $\varphi_4(\eta)$ converges but the iteration for $\varphi_4(\bar{\eta}) = -\varphi_6(\eta)$ does not, and the grid is consequently doubled in size. The same sequence may then be repeated on the doubled grid, that is, we have a solution η of $\varphi_4(\eta) = 0$ on a grid of $N + 1$ points, and a solution $\hat{\eta}$ of $\hat{\varphi}_4(\hat{\eta}) = 0$ on a grid of $2N + 1$ points, where in both cases the iteration for the sixth-order solution fails to converge. In this case, error estimates for the solution η can be derived from the well known procedure of *Richardson extrapolation*, and the grid is chosen using equidistribution based on these error estimates.

Case 4. The nonlinear iteration for η does not converge.

In this final case (the most worrying!), the iteration scheme applied to solve $\varphi_4(\eta) = 0$ fails to converge, and no converged solutions are available. Let $\tilde{\eta}$ denote the "partially converged solution" at which the nonlinear iteration scheme was terminated. The strategy of Case 3a is applied with $\varphi_6(\eta)$ replaced by $\varphi_4(\tilde{\eta})$.

The above four cases cover all eventualities. The only additional restriction is that the number of grid points is not allowed to increase too rapidly. If the grid-choosing strategy as described calls for the number of grid points to increase by more than a factor of three, we simply halve the mesh.

A novel feature of this approach is that, if the grid-selection strategy indicates that the solution is very bad (either because of a large deferred correction or a large residual in the nonlinear equations), we first try to add grid points to just one or two

mesh intervals. Points can be added to several intervals only if adding them to one or two does not appear to produce an improvement.

This mesh-selection strategy, which tries to use information from nonconverged Newton iterations as well as from error estimates associated with converged solutions, appears to be crucial to the efficiency of the present implementation. However, our strategy contains many heuristics, and the performance of the code on difficult problems can be sensitive to the choice of parameters. We believe that a better understanding of the relationship between the positioning of the mesh points and convergence properties of the Newton iteration will lead to a significant advance in the production of high-quality codes for two-point boundary value problems.

3. Test problems. Only problems of the form (1.1) with separated boundary conditions will be considered. COLNEW and COLSYS can solve more general problems, namely, higher-order equations, problems with nonseparated boundary conditions, and problems with certain singularities at the ends of the range of integration. PASVA deals with nonseparated boundary conditions, discontinuous data, and extra parameters appearing in the differential equation.

All of the test problems are expressed mathematically in the form of a single high-order equation. In order to apply the codes, the equations are converted to a first-order system in the usual way. Problems 1–12 are therefore solved as a system of two first-order equations, and Problems 13 and 14 are solved as a system of four first-order equations.

We point out that this transformation does not make use of the flexibility of the collocation codes, which may well perform better by solving these problems directly, as high-order equations (a facility not available for HAGRON or PASVA), or by using fewer collocation points for low tolerances. We have not investigated these options but have instead used the default parameters in the COLSYS codes when solving the test problems as first-order systems.

Each problem is associated with a parameter that controls the difficulty of numerical solution. Unless otherwise indicated, this parameter is ϵ, with increasing difficulty for smaller values.

An inevitable issue in testing software is the composition of the test set. Twelve of the fourteen problems in our test set are only of second order; and six of these twelve vary only in the choice of boundary condition (although the solutions differ radically in behavior). The conclusions drawn from our numerical experience must therefore be tempered by these limitations.

Problem 1. This problem is given in Hemker [HEM77, p. 154]:

$$\epsilon y'' + \exp(y)y' - \frac{\pi}{2}\sin\left(\frac{\pi x}{2}\right)\exp(2y) = 0, \quad 0 \le x \le 1, \quad y(0) = y(1) = 0,$$

and has a simple boundary layer at $x = 0$. The asymptotic solution for $\epsilon \to 0$ of the problem is given in [OMAL74, p. 123]:

$$y(x) = -\ln\left(\left(1 + \cos\left(\frac{\pi x}{2}\right)\right)\left(1 - \tfrac{1}{2}\exp\left(\frac{-x}{2\epsilon}\right)\right)\right) + O(\epsilon).$$

Problem 2. This problem appears in [PRN68], [WASOW70], and [HEM77, p. 158]:

$$\epsilon y'' + (y')^2 = 1, \quad 0 \le x \le 1. \tag{3.1}$$

The analytic solution is

$$y(x) = \alpha + \epsilon \ln \cosh\Big(\frac{x-\beta}{\epsilon}\Big),$$

where α and β are determined by the boundary conditions. As in [HEM77], we have chosen $\alpha = 1$ and $\beta = 0.745$, which give as boundary conditions

$$y(0) = 1 + \epsilon \ln \cosh\Big(\frac{-0.745}{\epsilon}\Big), \quad y(1) = 1 + \epsilon \ln \cosh\Big(\frac{0.255}{\epsilon}\Big).$$

As $\epsilon \to 0$, $y(x) \to \alpha + |x - \beta|$, which shows that for small ϵ the solution has a corner layer at $x = \beta$.

Problem 3.

$$\epsilon^2 y'' = y + y^2 - \exp\Big(\frac{-2x}{\epsilon}\Big),$$
$$y(0) = 1, \quad y(1) = \exp\Big(\frac{-1}{\epsilon}\Big).$$

The analytic solution of this problem is

$$y(x) = \exp\Big(\frac{-x}{\epsilon}\Big),$$

which gives a boundary layer near $x = 0$ for small ϵ. This problem is included in the test set as the "simple" example.

Problem 4. This problem is given in Maier [MAI85, p. 213]:

$$\epsilon y'' + y' + y^2 = 0, \quad y(0) = 0, \quad y(1) = \tfrac{1}{2},$$

and has a boundary layer near $x = 0$.

Problem 5. (Troesch's equation. Parameter μ; difficulty increases with μ.)

$$y'' = \mu \sinh(\mu y), \quad y(0) = 0, \quad y(1) = 1.$$

This problem has been discussed by many authors (see, for example, [SCWA77], [DLOPRS77], [KHH79]). It is fairly difficult for $\mu = 10$ and becomes very difficult as μ increases to about 40. For $\mu = 40$ the solution is zero almost everywhere except near $x = 1$, where a sharp boundary layer causes the solution to rise from 0 to 1. An added practical difficulty is that the sinh function will overflow if any approximation to y is too large.

Problem 6. This problem is discussed in [AMR88, p. 21] and [PRN68, p. 162]:

$$\epsilon A(x) y y'' - \Big(\frac{1+\gamma}{2} - \epsilon A'(x)\Big) y y' + \frac{y'}{y} + \frac{A'(x)}{A(x)}\Big(1 - \frac{(\gamma-1)}{2} y^2\Big) = 0, \quad 0 \le x \le 1.$$

Problem 6 is the steady-state Navier–Stokes equation, which represents a shock wave in a one-dimensional nozzle flow. The parameter x corresponds to the normalized

downstream distance from the throat of the nozzle, y is a normalized velocity, and $A(x)$ is the area of the nozzle at x. In our tests,

$$A(x) = 1 + x^2, \quad \gamma = 1.4,$$
$$y(0) = 0.9129 \text{ (supersonic flow in the throat)}; \quad y(1) = 0.375.$$

The value of ϵ is the inverse of the Reynolds number.

Problems 7–12.

$$\epsilon y'' + yy' - y = 0, \quad 0 \le x \le 1, \quad y(0) = \alpha, \quad y(1) = \beta.$$

Problems 7–12 correspond to the well known Lagerstrom–Cole equation, which has been studied by many authors (see, for example, [COLE68, pp. 29–38], [OMAL68, pp. 389–390]). The solution for small ϵ has a markedly different behavior depending on the choice of the boundary values α and β. For a sketch of this behavior in the α-β plane, see Hemker [HEM77, p. 160].

Problem 7. $\alpha = -\frac{1}{3}$, $\beta = \frac{1}{3}$. Corner layers at $x = \frac{1}{3}$ and $x = \frac{2}{3}$.

Problem 8. $\alpha = 1$, $\beta = -\frac{1}{3}$. Boundary layers at $x = 0$ and $x = 1$.

Problem 9. $\alpha = 1$, $\beta = \frac{1}{3}$. Boundary layer at $x = 0$, corner layer at $x = \frac{2}{3}$.

Problem 10. $\alpha = 1$, $\beta = \frac{3}{2}$. Boundary layer at $x = 0$.

Problem 11. $\alpha = 0$, $\beta = \frac{3}{2}$. Boundary layer at $x = 0$.

Problem 12. $\alpha = -\frac{7}{6}$, $\beta = \frac{3}{2}$. Shock layer at $x = \frac{1}{3}$.

Problem 13. This problem is given in [AMR88, p. 18] and involves nonlinear elastic beams (deformation of a beam under the action of axial and transverse loading resting on a nonlinear foundation).

$$y' = \sin\theta, \qquad \theta' = M,$$
$$\epsilon M' = -Q, \qquad \epsilon Q' = (y-1)\cos\theta - MT, \qquad T = \sec\theta + \epsilon Q \tan\theta.$$

The simple support boundary conditions are

$$y(0) = y(1) = 0, \quad M(0) = M(1) = 0.$$

Problem 14. (Parameter R; difficulty increases with R.)

This fourth-order problem arises from fluid injection through one side of a long vertical channel [AMR88, p. 8]. After manipulation of the Navier–Stokes and heat transfer equations, we obtain

$$y'''' = R(y'y'' - yy'''),$$

where R is the Reynolds number. The boundary conditions are

$$y(0) = y'(0) = 0, \quad y(1) = 1, \quad y'(1) = 0.$$

4. Numerical testing.

4.1. The COLSYS codes. The COLSYS codes [ACR79a], [ACR79b], [BA87] are based on finding a piecewise polynomial function satisfying the boundary conditions and the ordinary differential equation at a set of collocation points. A particular collocation method is defined by the basis functions used in constructing the piecewise polynomial and by the collocation points. In both COLSYS and COLNEW the collocation points are chosen to be the Gauss points. If $k > m$, the COLSYS codes give $O(h^{k+m})$ global convergence and $O(h^{2k})$ superconvergence at the mesh points.

COLSYS and COLNEW differ in the choice of basis functions. In COLSYS, B-splines are the basis for the piecewise polynomial solutions [ACR79a]. COLNEW uses a monomial basis that was shown in [APR83] to be superior to the B-spline basis for the particular case of a single linear high-order ordinary differential equation. Further theoretical properties are identified in [BA87] that may give COLNEW an advantage. In addition, the different linear equation solvers in COLSYS and COLNEW are likely to affect the relative efficiency of the codes.

The COLSYS codes have several advantages over other codes. In particular, they can accept multipoint mixed-order systems; they produce a continuous rather than a discrete solution; and they can deal with problems with certain kinds of singularities such as (ii) in §1. The ability to obtain a solution at "nongrid" points is important because of the difficulties with high-order interpolation discussed by Pruess [PR86].

Providing that the problem is sufficiently smooth, the behavior of the error for $x \in [x_i, x_{i+1})$ is

$$\|y^{(l)}(x) - v^{(l)}(x)\|_{(i)} = c_l \, |y^{(k+m)}(x_i)| \, h_i^{k+m-l} + \text{higher-order terms}, \quad l = 0, \ldots, m-1, \tag{4.1}$$

where $v^{(l)}$ is the lth derivative of the approximating polynomial and $\|\cdot\|_{(i)}$ is the max norm in $[x_i, x_{i+1})$. For a more precise definition of these quantities, see [ACR81, p. 210]. Expression (4.1) is used both for error estimation (by mesh halving) to check whether the user-specified tolerance is satisfied, and for grid refinement.

4.2. Details of the testing. To aid in giving a reasonable indication of relative performance, it was decided to give all three codes the same fixed maximum amount of floating-point workspace—namely, 250,000 words. This restriction in turn imposes a limit on the maximum number of grid points that each code can use. For problems with two components (Problems 1–12), the limits were 6250 for HAGRON, 1700 for COLSYS, and 1644 for COLNEW; for problems with four components (Problems 13 and 14), the limits were 2426 for HAGRON, 536 for COLSYS, and 516 for COLNEW.

Our tests were run on a Cray XMP without multiprocessing, using the *cft* compiler and the *unicos* operating system. Machine precision (single) is approximately 3.6×10^{-15}. All times shown are in CPU seconds. In repeating many runs, we observed that the measured time for the same problem may vary by as much as 10 percent between runs, even when the elapsed time is more than 10 seconds. We therefore give each time with two decimal digits, but caution that the second figure is somewhat unreliable.

The initial guess was always taken as $y(x) = 0$, except for Problem 6, where the initial guess was set to $\frac{1}{2}$ because the expression for y'' involves a division by y. The value of zero is not in any sense an optimal guess, and does not in general satisfy the boundary conditions, but it is convenient for numerical comparison.

All problems were run for four tolerances (10^{-2}, 10^{-4}, 10^{-6}, and 10^{-8}) that specify the accuracy requested in the solution (see §§2 and 4.1). Each of the COLSYS codes was run twice for all examples, with the parameter `IPAR(10)` set to zero and to one. The significance of this parameter is discussed in [ACR81]. As mentioned in §3, each problem is associated with a parameter that controls the difficulty of numerical solution. For each problem, the limiting parameter value in our numerical testing was chosen either because all three codes failed for the "next" value, or because the execution times for one or all of the codes became prohibitively long.

We illustrate this process with Problem 3, which is convenient because it can still be solved for the "small" parameter value $\epsilon = 10^{-6}$. Table 1 shows the run times in CPU seconds obtained for a tolerance of 10^{-4} and four values for ϵ. The notation "†" following a time means that the maximum storage allocation was reached. Since all three codes failed for $\epsilon = 10^{-7}$, the value of ϵ for Problem 3 ranges from 10^{-1} to 10^{-6}. (An interesting observation from our results is that, when codes failed, they most often did so "gracefully" by reaching the storage limit.)

TABLE 1
Results for Problem 3.

ϵ	10^{-4}	10^{-5}	10^{-6}	10^{-7}
Colsys, ipar(10)=0	13.0†	30.0†	58.0†	99.0†
Colsys, ipar(10)=1	3.1	9.9	44.0	93.0†
Colnew, ipar(10)=0	11.0†	22.0†	51.0†	81.0†
Colnew, ipar(10)=1	2.5	8.6	36.0	71.0†
Hagron	0.26	0.84	3.8	27.0†

A parenthetical note concerning this problem is that, when all three codes failed for $\epsilon = 10^{-7}$, the run time until termination for HAGRON is less than for COLSYS and COLNEW. This behavior is *not* typical. There is no clear pattern as to which code runs longest before declaring a failure, but on average we found that HAGRON took the longest time. A possible explanation for this phenomenon is that for difficult problems the COLSYS codes consistently double the mesh before running out of storage, whereas HAGRON works hard trying to refine the mesh in the correct places before giving up with a reasonable mesh but inadequate storage.

Our original hope was to include the PASVA codes in our experiments. The version of PASVA available to us was the subroutine D02RAF from the NAG library. After several tries, we found that it was not possible to solve the full set of problems without considerable user intervention. In particular, for many of the harder problems the error message "`IFLAG = 3`" was produced, which means that the Newton iteration did not converge, and that the user must provide either a better grid or a better initial guess. Because we did not wish to become involved in setting up a continuation procedure, PASVA has been omitted altogether from the experiments reported here. Our feeling is that PASVA will work well for mildly stiff problems, but is harder to use for difficult singular perturbation problems.

5. Numerical results.

5.1. Summary tables. This section summarizes the main results from runs of the COLSYS codes and HAGRON on 12 of the 14 test problems defined in §3. (Problems 4 and 11 are omitted here because their results are qualitatively similar to those of other problems.) Results for the COLSYS codes are shown only for the runs in which `IPAR(10)` was set to zero. Details of all test runs are given in [CW89].

Every table corresponds to a problem. The columns are associated with tolerances and the rows with parameter values. Each "entry" in the table consists of a stacked list of three numbers, which are execution times (in seconds on the Cray; see §4.2) for HAGRON (at the top), COLSYS with `IPAR(10)` set to zero, and COLNEW with `IPAR(10)` set to zero.

The notation "†" following a time means that the code failed because the maximum storage allocation was reached; when a time is followed by "⋆," the code terminated normally, but the solution was wrong; if no time is shown for a code, the problem was not run because the estimated running time was excessive (see §4.2).

The final column of each table (labeled "Normalized error") gives a measure of the accuracy achieved by the three codes. Since the exact solution is unknown except for Problems 2 and 3, the following procedure was used to estimate the error. An "exact" solution was obtained by running a collocation code with a requested tolerance of 10^{-10} and a storage allocation of 500,000 words. (This solution was computed by COLNEW in testing HAGRON and COLNEW, and by COLSYS in testing COLSYS.) Results are given here only for instances in which the collocation code with tolerance 10^{-10} was successful.

For each code, the "error" at every grid point in the final mesh is defined as the difference between the computed solution and the "exact" solution obtained as just described. The normalized error measure shown is the maximum error over all mesh points *divided by the tolerance.* If the normalized error measure is less than or equal to one, the desired accuracy was achieved at all mesh points.

PROBLEM 1

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
10^{-1}	.0064 .044 .032	.016 .051 .037	.022 .089 .076	.040 .20 .16	3.4×10^{-2} 4.8×10^{-5} 4.8×10^{-5}
10^{-2}	.057 .066 .038	.074 .13 .11	.11 .25 .20	.13 .40 .34	4.4×10^{-2} 2.8×10^{-1} 3.5×10^{-1}
10^{-3}	.39 9.6† 8.1†	.44 6.6† 8.1†	.48 6.6† 8.1†	.53 6.6† 8.1†	
10^{-4}	1.2 – –	1.5 – –	1.6 – –	1.9 – –	

PROBLEM 2

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
$\frac{1}{6}$	.0083 .052 .040	.0091 .059 .045	.019 .066 .050	.039 .18 .14	9.0×10^{-2} 3.3×10^{-3} 3.3×10^{-3}
$\frac{1}{36}$	.020 .12 .081	.036 .13 .092	.060 .19 .13	.11 .43 .32	3.1×10^{-1} 7.4 5.3
$\frac{1}{216}$	.34 5.0† 4.0†	.20 5.0† 4.0†	.14 5.0† 4.0†	.18 5.0† 4.0†	

PROBLEM 3

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
10^{-1}	.0055	.013	.020	.040	5.1×10^{-2}
	.036	.043	.090	.33	3.2×10^{-4}
	.040	.044	.078	.26	3.2×10^{-4}
10^{-2}	.031	.035	.062	.083	5.1×10^{-2}
	.051	.13	.15	.32	2.2
	.042	.10	.13	.25	2.2
10^{-3}	.092	.15	.14	.19	2.0×10^{-1}
	.61 ⋆	19.0†	21.0†	15.0†	1.0×10^{2}
	.071⋆	11.0†	23.0†	9.7†	1.7×10^{1}
10^{-4}	.16	.27	.50	.59	8.7×10^{-2}
	3.2 ⋆	13.0†	11.0†	15.0†	
	.076⋆	11.0†	12.0†	16.0†	
10^{-5}	.67	.84	.90	2.0	2.0×10^{-2}
	23.0 †	30.0†	23.0†	143.0†	
	.077⋆	22.0†	15.0†	16.0†	
10^{-6}	3.5	3.8	4.2	4.4	4.3×10^{-1}
	51.0†	58.0†	46.0†	48.0†	
	.077⋆	51.0†	40.0†	49.0†	

PROBLEM 5

$\mu \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
10	.062	.077	.10	.17	2.8×10^{-2}
	.12	.22	.43	.57	4.6×10^{-1}
	.099	.18	.36	.49	4.6×10^{-1}
20	.32	.36	.41	.54	4.7×10^{-3}
	.36	1.2	1.3	1.8	4.5
	.17	.64	1.2	1.3	5.0
30	.80	.86	.94	1.0	
	.57	3.8	3.9	5.3	
	.38	3.4	3.4	4.5	
40	.88	.99	1.1	1.2	
	.79	8.6	6.7	11.0	
	1.3	7.4	5.9	27.0†	

PROBLEM 6

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
$\frac{1}{3}$	.0051	.0064	.0081	.017	7.6×10^{-3}
	.051	.057	.057	.10	2.0×10^{-5}
	.051	.058	.058	.094	5.6×10^{-5}
$\frac{1}{9}$	.0084	.0096	.032	.034	2.3×10^{-2}
	.067	.074	.074	.20	6.6×10^{-3}
	.060	.066	.072	.16	6.6×10^{-3}
$\frac{1}{27}$	.021	.038	.070	.13	3.0×10^{-2}
	.19	.18	.26	.32	1.7×10^{-2}
	.080	.085	.18	.24	8.5×10^{-2}
$\frac{1}{81}$	.64	.64	.70	.79	4.7×10^{-2}
	.30	.37	.49	.89	1.4×10^{-1}
	.27	.31	.41	.75	1.4×10^{-1}

PROBLEM 7

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
10^{-1}	.0039 .036 .033	.0048 .044 .035	.0056 .042 .041	.012 .049 .041	1.6×10^{-3} 2.7×10^{-5} 2.8×10^{-5}
10^{-2}	.0060 .050 .046	.0071 .057 .052	.017 .064 .058	.033 .19 .15	3.8×10^{-2} 2.2×10^{-2} 2.2×10^{-2}
10^{-3}	.014 .099 .092	.028 .11 .098	.048 .23 .20	.099 .69 .60	3.3×10^{-1} 5.3×10^{-2} 5.3×10^{-2}
10^{-4}	.76 .21⋆ .16⋆	1.8 8.6† .19⋆	1.9 6.4† .36⋆	2.0 6.6† .57⋆	
10^{-5}	9.3 – –	7.6 – –	3.5 – –	3.8 – –	

PROBLEM 8

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
10^{-1}	.0059 .043 .045	.013 .050 .045	.019 .15 .14	.036 .21 .19	1.4×10^{-2} 6.9×10^{-3} 6.9×10^{-3}
10^{-2}	.048 .091 .073	.055 .18 .15	.081 .38 .33	.11 .80 .70	8.2×10^{-3} 3.5×10^{-1} 3.5×10^{-1}
10^{-3}	.29 .43⋆ 14.0 †	.35 14.0† 11.0†	.44 1.2 11.0†	.54 1.6 11.0†	
10^{-4}	.69 5.3† 5.0†	.79 5.4† 5.0†	1.1 5.4† 5.0†	1.1 5.3† 5.0†	
10^{-5}	1.1 – –	1.2 – –	1.3 – –	1.3 – –	

PROBLEM 9

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
10^{-1}	.0058 .045 .046	.013 .050 .045	.018 .092 .087	.033 .19 .17	1.2×10^{-2} 5.1×10^{-5} 5.1×10^{-5}
10^{-2}	.022 .089 .074	.027 .17 .15	.045 .25 .22	.10 .49 .41	8.2×10^{-2} 3.5×10^{-1} 3.5×10^{-1}
10^{-3}	.076 11.0 8.5†	.13 11.0 8.5†	.24 11.0 8.5†	.27 12.0 8.5†	
10^{-4}	9.5 – –	9.8 – –	10.0 – –	10.0 – –	

PROBLEM 10

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
10^{-1}	.0062	.013	.018	.033	3.1×10^{-2}
	.053	.059	.098	.18	1.5×10^{-2}
	.045	.051	.085	.16	1.2×10^{-3}
10^{-2}	.040	.043	.083	.11	4.1×10^{-1}
	.13	.20	.25	.37	3.0×10^{-1}
	.085	.16	.21	.31	3.5×10^{-1}
10^{-3}	.42	.47	.49	.56	
	6.4†	14.0†	14.0†	8.1†	
	5.2†	4.8†	4.8†	4.8†	
10^{-4}	1.3	1.4	1.5	1.9	
	—	—	—	—	
	—	—	—	—	

PROBLEM 12

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
$\frac{1}{5}$	.0048	.0068	.025	.033	1.6×10^{-3}
	.051	.051	.058	.18	4.2×10^{-4}
	.044	.050	.050	.14	4.2×10^{-4}
$\frac{1}{25}$	.021	.038	.044	.13	2.1×10^{-2}
	.082	.13	.21	.42	5.7×10^{-2}
	.061	.10	.17	.36	5.7×10^{-2}
$\frac{1}{125}$	.14	.16	.21	.26	6.1×10^{-2}
	.18	10.0†	.81	1.5	2.0
	.13	8.5	.68	.87	1.7
$\frac{1}{625}$	20.0	20.0	20.0	21.0	
	54.0	54.0	54.0	54.0	
	12.0†	12.0†	12.0†	12.0†	

PROBLEM 13

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
10^{-1}	.0093	.0097	.031	.041	2.4×10^{-2}
	.088	.11	.11	.20	1.4×10^{-4}
	.077	.077	.093	.17	1.4×10^{-4}
10^{-2}	.0098	.031	.083	.14	2.1×10^{-1}
	.089	.11	.11	.36	1.9
	.079	.095	.095	.31	1.9
10^{-3}	.022	.024	.20	.36	2.4×10^{-1}
	.11	.11	.11	.15	5.1×10^{1}
	.078	.095	.096	.11	5.1×10^{1}
10^{-4}	.022	.023	.086	.79	2.1
	.088	.11	.12	.12	1.8×10^{-1}
	.078	.095	.096	.11	1.8×10^{-1}
10^{-5}	.022	.024	.086	1.1	2.2
	.088	.11	.12	1.4	6.4×10^{-3}
	.080	.097	.097	.11	2.5×10^{-5}
10^{-6}	.022	.024	.086	1.1	
	.088	.11	.26	7.4	
	.080	.097	.097	.11	

PROBLEM 14

$\epsilon \backslash tol$	10^{-2}	10^{-4}	10^{-6}	10^{-8}	Normalized error
	.013	.013	.030	.076	4.8×10^{-2}
10	.13	.13	.15	.23	4.9×10^{-5}
	.11	.11	.13	.20	4.9×10^{-5}
	.038	.063	.086	.15	5.9×10^{-2}
10^2	.18	.18	.27	.35	3.7×10^{-3}
	.15	.17	.25	.31	3.7×10^{-3}
	.064	.070	.20	.18	1.3×10^{-1}
10^3	.37	.38	.49	.73	1.1
	.30	.31	.39	.63	1.1
	.11	.11	.24	.33	1.1×10^{-1}
10^4	.58	.71	.89	1.0	3.2×10^{-1}
	.78	.81	.97	1.1	3.2×10^{-1}
	.22	.23	.40	.70	1.2×10^{-2}
10^5	1.8	1.9	2.2	2.4	1.7×10^{-1}
	5.6†	7.5†	1.7	1.6	1.3×10^{-3}
	1.7	1.8	1.9	2.2	
10^6	.43	8.4†	9.0†	7.0†	
	.45	6.7†	7.8†	7.3†	

5.2. Discussion of numerical results. Before discussing the numerical results, we wish to emphasize again, because we feel it to be a crucial point, that our aim is not to compare or rank the codes. A major problem inherent in any numerical comparison is the relative importance to be attached to different measures of goodness. For example, a user may be interested in computer time, storage space, time required to formulate and program the problem, portability, robustness, ease of use, and so on. We take the view that numerical testing should be used to eliminate obviously poor codes and to point to codes that appear to be competitive with others on a large class of important problems. We regard efficient codes as being *complementary* to each other, and a major aim of this paper has been to show that the deferred correction code HAGRON is worth further consideration.

Some interesting features of the COLSYS codes as well as of HAGRON are shown by our results. We begin with a few remarks concerning the COLSYS codes. Firstly, COLNEW was faster than COLSYS by about 20 percent on average, although it was sometimes slower or less reliable. A second observation is that the COLSYS codes were sometimes unreliable at low tolerances (a not unexpected situation). For example, on Problem 3 with $\epsilon = 10^{-3}$ and tolerance 10^{-2}, both COLSYS and COLNEW obtained bad solutions that did not satisfy the error criterion.

Thirdly, for the "easiest" versions of some problems, the collocation codes produced approximate solutions several orders of magnitude more accurate than the requested tolerance (see the entries for Problems 1, 3, 7, 9, and 13 with $\epsilon = 10^{-1}$, for Problem 6 with $\epsilon = \frac{1}{3}$, and for Problem 14 with $\epsilon = 10$). This property probably explains why the execution times for the collocation codes for these values of ϵ are considerably larger than the corresponding times for HAGRON.

Finally, the final and maximum numbers of grid points used by the COLSYS codes were exactly the same for all problems, in marked contrast to HAGRON, which tended to add many grid points to identify places where bad behavior occurred, and then remove points from the smooth regions once the bad behavior had been identified.

As mentioned earlier, a major difficulty in evaluating the performance of the

codes is that the true solution is unknown for most of our test problems. However, the general behavior of the solution is known for all problems and can be checked. In addition, the regions of bad behavior are known (see §3), and study of the final mesh allowed us to determine whether the grid-choosing algorithm put the grid points in the correct place(s). Although proper placement of the mesh points does not guarantee the accuracy of the final solution, it does provide (reassuring) verification that the correct solution profiles are being obtained.

As described in §5.1, an error estimate was computed at each point in the final grid, and the maximum of these errors is given in the summary tables. The results show that the requested accuracy is typically achieved by all codes, and no code consistently produces a more accurate solution than the others.

Our overall results show that HAGRON performed well on the test problems, which were chosen to exhibit most kinds of difficult behavior—boundary layers, turning points, corner layers, and shock layers. It was found that over all problems, HAGRON was normally faster than both COLSYS and COLNEW, and it often achieved solutions where the COLSYS codes failed. ("Failure" of the COLSYS codes corresponds to running out of storage space, i.e., exceeding the maximum number of grid points.)

Perhaps the only problems where timings were comparable were Problem 6 with $\epsilon = 1/81$ (an interior layer problem) and Problem 13. Problem 13 is of some interest because it has a smooth solution. Our results show that the time required to achieve various degrees of accuracy for this problem is almost constant as ϵ decreases. Thus, Problem 13 is an example where a rapidly varying solution is possible, but the boundary conditions enforce a smooth solution. The phenomenon of COLSYS's superiority to HAGRON on such problems has previously been noted in the linear case [CASH89].

In Tables 2 and 3, we give the time ratios for COLSYS and COLNEW to HAGRON on Problems 5 and 14, where the column headings under each comparison contain the tolerance.

TABLE 2

Time ratios for Problem 5.

Parameter	Time Ratio (Colsys/Hagron)				Time Ratio (Colnew/Hagron)			
	10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-2}	10^{-4}	10^{-6}	10^{-8}
10	1.9	2.9	4.3	3.4	1.6	2.3	3.6	2.9
20	1.1	3.3	3.2	3.3	0.5	1.8	2.9	2.4
30	0.7	4.4	4.1	5.3	0.5	4.0	3.6	4.5
40	0.9	8.7	6.1	9.2	1.5	7.5	5.4	*

TABLE 3

Time ratios for Problem 14.

Parameter	Time Ratio (Colsys/Hagron)				Time Ratio (Colnew/Hagron)			
	10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-2}	10^{-4}	10^{-6}	10^{-8}
10	10.0	10.0	5.0	3.0	8.5	8.5	4.3	2.6
10^2	4.7	2.9	3.1	2.3	3.9	2.7	2.9	2.1
10^3	5.8	5.4	2.5	4.1	4.7	4.4	1.9	3.5
10^4	5.3	6.5	3.7	3.0	7.1	7.4	4.0	3.3
10^5	8.2	8.3	5.5	3.4	*	*	4.3	2.3
10^6	0.3	*	*	*	0.3	*	*	*

Although it cannot be seen in the tables, an important feature of the performance of HAGRON was the efficiency of the linesearch method GETPTQ [GM74] (see

[CW90]). The greatest difficulty for any linesearch algorithm occurs in early Newton iterations far from the solution, yet GETPTQ consistently computed a satisfactory steplength with few trials (rarely more than three), even when the nonlinear system was badly behaved. An unplanned by-product of our experiments is the observation that GETPTQ "struggles" when the Jacobian matrices are programmed incorrectly; see [GMW81] for a discussion of the effects on the linesearch of errors in programming derivatives.

Based on all these results, we believe that HAGRON is a useful tool for solving two-point boundary value problems. As discussed in the introduction, boundary value codes are less well developed than initial value codes, and much work remains to be done before boundary value codes can achieve a consistently high level of reliability. We hope that experiments with codes such as HAGRON will lead to better understanding of the nature of boundary value algorithms as well as software.

Acknowledgment. The authors thank the referees and the associate editor for detailed, careful, and helpful suggestions.

REFERENCES

[ACR79a] U. ASCHER, J. CHRISTIANSEN, AND R. D. RUSSELL, *A collocation solver for mixed order systems of boundary value problems*, Math. Comp., 33 (1979), pp. 659–679.

[ACR79b] ———, COLSYS–*A collocation code for boundary value problems*, in Codes for Boundary Value Problems in Ordinary Differential Equations, Lecture Notes in Computer Science 76, B. Childs, M. Scott, J. W. Daniel, E. Denman, and P. Nelson, eds., Springer-Verlag, New York, 1979, pp. 164–185.

[ACR81] ———, *Collocation software for boundary value ordinary differential equations*, ACM Trans. Math. Software, 7 (1981), pp. 209-222.

[AMR88] U. ASCHER, R. M. M. MATTHEIJ, AND R. D. RUSSELL, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[APR83] U. ASCHER, S. PRUESS, AND R. D. RUSSELL, *On spline basis selection for solving differential equations*, SIAM J. Numer. Anal., 20 (1983), pp. 121–142.

[BA87] G. BADER AND U. ASCHER, *A new basis implementation for a mixed order boundary value* ODE *solver*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 483–500.

[CASH86] J. R. CASH, *On the numerical integration of nonlinear two-point boundary value problems using iterated deferred corrections, Part 1: A survey and comparison of some one-step formulae*, Comput. Math. Appl., 12a (1986), pp. 1029–1048.

[CASH88] ———, *On the numerical integration of nonlinear two-point boundary value problems using iterated deferred corrections, Part 2: The development and analysis of highly stable deferred correction formulae*, SIAM J. Numer. Anal., 25 (1988), pp. 862–882.

[CASH89] ———, *A comparison of some global methods for solving two-point boundary value problems*, Appl. Math. Comput., 31 (1989), pp. 449–462.

[CS82a] J. R. CASH AND A. SINGHAL, *Mono-implicit Runge–Kutta formulae for the numerical integration of stiff differential equations*, IMA J. Numer. Anal., 2 (1982), pp. 211–227.

[CS82b] ———, *High-order methods for the numerical solution of two-point boundary value problems*, BIT, 22 (1982), pp. 184–199.

[CW89] J. R. CASH AND M. H. WRIGHT, *A deferred correction method for nonlinear two-point boundary problems: Implementation and numerical evaluation*, Computing Science Tech. Report 146, AT&T Bell Laboratories, Murray Hill, NJ, 1989.

[CW90] ———, *Implementation issues in solving nonlinear equations for two-point boundary value problems*, Computing, 45 (1990), pp. 17–37.

[COLE68] J. D. COLE, *Perturbation Methods in Applied Mathematics*, Blaisdell, Waltham, MA, 1968.

[DLOPRS77] H. J. DIEKOFF, P. LORY, H. J. OBERLE, H. J. PESCH, P. RENTROP, AND R. SEYDEL, *Comparing routines for the numerical solution of initial value problems of ordinary differential equations in multiple shooting*, Numer. Math., 27 (1977), pp. 449–469.

[GM74] P. E. GILL AND W. MURRAY, *Safeguarded steplength algorithms for optimization using descent methods*, Report NAC 37, National Physical Laboratory, Teddington, United Kingdom, 1974.

[GMW81] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, London, United Kingdom, 1981.

[HEM77] P. W. HEMKER, *A numerical study of stiff two-point boundary value problems*, Math. Centrum, Amsterdam, the Netherlands, 1977.

[KHH79] M. KUBÍČEK, V. HLAVÁČEK, AND M. HOLODNIOK, *Test examples for comparison of codes for nonlinear boundary value problems in ordinary differential equations*, in Codes for Boundary Value Problems in Ordinary Differential Equations, Lecture Notes in Computer Science 76, B. Childs, M. Scott, J. W. Daniel, E. Denman, and P. Nelson, eds., Springer-Verlag, New York, NY, 1979, pp. 325–346.

[LP74] M. LENTINI AND V. PEREYRA, *A variable order finite difference method for nonlinear multipoint boundary value problems*, Math. Comp., 28 (1974), pp. 981–1004.

[MAI85] M. R. MAIER, *Numerical solution of singular perturbed boundary value problems using a collocation method with tension splines*, in Numerical Boundary Value Ordinary Differential Equations, U. M. Ascher and R. D. Russell, eds., Birkhäuser, Boston, MA, 1985, pp. 207–225.

[OMAL68] R. E. O'MALLEY, *Topics in singular perturbations*, Adv. Math., 2 (1968), pp. 365–470.

[OMAL74] ———, *Introduction to Singular Perturbations*, Academic Press, New York, 1974.

[PRN68] C. E. PEARSON, *On nonlinear ordinary differential equations of boundary layer type*, J. Math. Phys., 47 (1968), pp. 351–358.

[PER68] V. PEREYRA, *Iterated deferred corrections for nonlinear boundary value problems*, Numer. Math., 11 (1968), pp. 111–125.

[PR86] S. PRUESS, *Interpolation schemes for collocation solutions of two point boundary value problems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 322–333.

[SCWA77] M. R. SCOTT AND H. A. WATTS, *Computational solution of linear two point boundary value problems via orthonormalization*, SIAM J. Numer. Anal., 14 (1977), pp. 40–70.

[WASOW70] W. WASOW, *The capriciousness of singular perturbations*, Nieuw Arch. Wisk., 18 (1970), pp. 190–210.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 991-999, September 1991
© 1991 Society for Industrial and Applied Mathematics

RUNGE–KUTTA DEFECT CONTROL USING HERMITE–BIRKHOFF INTERPOLATION*

DESMOND J. HIGHAM†

Abstract. Two techniques for reliably controlling the defect (residual) in the numerical solution of nonstiff initial value problems were given in [D. J. Higham, *SIAM J. Numer. Anal.*, 26 (1989), pp. 1175-1183]. This work describes an alternative approach based on Hermite-Birkhoff interpolation. The new approach has two main advantages—it is applicable to Runge-Kutta schemes of any order, and it gives rise to a defect of the optimum asymptotic order of accuracy. For a particular Runge-Kutta formula the asymptotic analysis is verified numerically.

Key words. Runge-Kutta, defect, residual, backward error, Hermite-Birkhoff interpolation

AMS(MOS) subject classification. 65L05

1. Introduction. This work deals with the control of errors in the numerical solution of the nonstiff initial value problem

$$y'(x) = f(x, y(x)), \qquad y(a) = y_a \in \mathbb{R}^N, \qquad a \leqq x \leqq b,$$

using an explicit Runge-Kutta method. These methods produce discrete approximations $y_n \approx y(x_n)$ by proceeding in a stepwise fashion; a typical step involves advancing the numerical approximation from x_n to $x_{n+1} := x_n + h_n$. To complement the approximation at the meshpoints $\{x_n\}$, many authors have derived interpolants $p(x)$ which provide approximations $p(x) \approx y(x)$ for other values of x (see, for example, [1], [7], [9], [12], [13]). It is desirable for $p(x)$ to provide efficient, accurate approximations, and to have at least global C^1 continuity. The corresponding defect (residual),

$$\delta(x) := p'(x) - f(x, p(x)),$$

may then be used to measure the error in the numerical solution. As a means of error-control, Enright [5] suggests that the defect be sampled at one or more points on each step. By retaking the step with a smaller stepsize if necessary, we could ensure that the sampled values were sufficiently small on every step. The idea of controlling the defect is intuitively reasonable—if the solution has a small defect then it solves a nearby system of differential equations and has a small "backward error." For a discussion of the defect control philosophy, and its relation to the more traditional local error control, see [5] and [6].

When standard Runge-Kutta interpolation schemes are used, the shape of the defect over each step cannot be determined a priori, since it depends on f. In [10] two special classes of interpolant were derived which have the property that asymptotically (as $h_n \to 0$) each component of the defect behaves like a multiple of a known polynomial over each step, and hence the maximum or root-mean-square integral value of the defect can be approximated from one sample. (We mention that it has been shown that in the case of Adams PECE multistep formulas, there are natural interpolation

* Received by the editors October 16, 1989; accepted for publication (in revised form) May 1, 1990. This research was supported by the Information Technology Research Centre of Ontario, and the Natural Sciences and Engineering Research Council of Canada. Part of this manuscript was prepared while the author was visiting the Computer Science Department, Cornell University, Ithaca, New York.

† Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4. Present address, Department of Mathematics and Computer Sciences, University of Dundee, DD1 4HN, Scotland, U.K.

schemes which automatically satisfy this property [11].) Two main drawbacks of the schemes in [10] are:

- They are only viable for low-order Runge-Kutta formulas (say $\leqq 5$); and
- They produce a defect with an asymptotic order of one less than the optimal value.

The purpose of this work is to show that both difficulties can be overcome by the use of Hermite-Birkhoff interpolants.

In the next section we set out some basic definitions and define the particular type of Hermite-Birkhoff interpolant that we need. We then show that the corresponding defect has the desired asymptotic properties. In the final two sections we specialize to fifth-order Runge-Kutta formulas and give some numerical verification of the theoretical results.

2. The Hermite–Birkhoff interpolant. An s-stage explicit Runge-Kutta formula for advancing the numerical solution over a step of length $h(=h_n)$ can be written

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i,$$

where

$$k_1 = f(x_n, y_n),$$
$$k_i = f\left(x_n + c_i h, y_n + h \sum_{j=1}^{i-1} a_{ij} k_j\right), \qquad 2 \leqq i \leqq s.$$

The local solution for the step $u(x)$ is the solution curve which passes through y_n at x_n; that is, $u'(x) = f(x, u(x))$ and $u(x_n) = y_n$. A Runge-Kutta formula is said to be of *order* p if p is the largest integer such that the local error satisfies $y_{n+1} - u(x_n + h) = O(h^{p+1})$. (Here, and in the following analysis, we assume that f is sufficiently differentiable.) We say that a corresponding local interpolant $p(x)$ has *local order* $q+1$ if q is the largest integer such that $p(x_n + \tau h) - u(x_n + \tau h) = O(h^{q+1})$ for any fixed $\tau \in [0, 1]$. The interpolation condition $p(x_n + h) = y_{n+1}$ ensures that $q + 1 \leqq p + 1$.

In deriving the interpolant $p(x)$, a natural approach is to use a polynomial interpolant to solution and derivative approximations. The data $y_n, f(x_n, y_n), y_{n+1}$ and $f(x_{n+1}, y_{n+1})$ is available free of charge, but normally extra data is needed to achieve the desired order, and more stages must be added to the Runge-Kutta process. An automatic bootstrapping technique for generating polynomial interpolants of local order up to $p+1$ was developed by Enright et al. [7]. The technique makes use of a special kind of Hermite-Birkhoff interpolant. We show below that these interpolants can also be used to derive robust defect control schemes.

The Hermite-Birkhoff interpolants that we consider are chosen to match the meshpoint data plus some extra derivative data at intermediate points in the step. If f is Lipschitzian, then we have the result

$$y_{n+1} - u(x_{n+1}) = O(h^{p+1}) \Rightarrow f(x_{n+1}, y_{n+1}) - u'(x_{n+1}) = O(h^{p+1}).$$

We write $\gamma_1 = 0$, $\gamma_2 = 1$, $u_1' := f(x_n, y_n)$ and $u_2' := f(x_{n+1}, y_{n+1})$, and we suppose that there are $r-2$ further derivative approximations $\{u_i'\}_{i=3}^{r}$ available which also satisfy

$$u_i' - u'(x_n + \gamma_i h) = O(h^{p+1}),$$

where $\{\gamma_i\}_{i=3}^{r}$ are distinct points in $(0, 1)$. This data could be generated, for example, by constructing one of the standard locally ordered $p+1$ interpolants, $p(x)$, and fixing $u_i' := f(x_n + \gamma_i h, p(x_n + \gamma_i h))$. We may then consider the Hermite-Birkhoff polynomial interpolant $H(x)$ of degree $\leqq r+1$ which satisfies

$$H(x_n) = y_n, \qquad H(x_{n+1}) = y_{n+1}, \qquad H'(x_n + \gamma_i h) = u_i', \qquad 1 \leqq i \leqq r. \tag{2.1}$$

Note that $H(x)$ does not satisfy the usual Hermite interpolatory conditions, since it matches a derivative value, but not a solution value at $x_n+\gamma_i h$, $i=3,\cdots,r$.

We point out that the existence and uniqueness of $H(x)$ cannot be guaranteed in general. However, for any particular r it is always possible to choose $\{\gamma_i\}_{i=3}^r$ so that $H(x)$ exists uniquely [7, p. 197]. In the remainder of this section we assume that this has been done.

Using the normalized variable, $\tau=(x-x_n)/h$, we may write the interpolant in the form

$$H(x_n+\tau h)=d_1(\tau)y_n+d_2(\tau)y_{n+1}+h\sum_{i=1}^{r}e_i(\tau)u_i', \tag{2.2}$$

where $d_1(\tau)$, $d_2(\tau)$, and $\{e_i(\tau)\}_{i=1}^r$ are scalar polynomials in τ of degree $\leqq r+1$. To examine the local error and the defect in $H(x)$, we adopt the usual strategy of isolating the interpolation and data errors. To this end, we let $Q(x)$ denote the Hermite-Birkhoff interpolant that matches the exact local solution values:

$$Q(x_n+\tau h)=d_1(\tau)u(x_n)+d_2(\tau)u(x_{n+1})+h\sum_{i=1}^{r}e_i(\tau)u'(x_n+\gamma_i h). \tag{2.3}$$

First, we consider the interpolation error $Q(x)-u(x)$. To find the asymptotic order of this expression we follow the approach of Dormand et al. [4, pp. 5-6], which relies only on the uniqueness of the interpolant. (Incidentally, although there is a well-known error expression in the case of Hermite interpolation, the author is not aware of any general expression for the error in $Q(x)$.)

Suppose that the local solution $u(x)$ has at least $r+2$ continuous derivatives. Using the subscript t to denote the tth component of a vector-valued function, we may substitute the truncated Taylor expansions

$$u_t(x_n+h)=\sum_{k=0}^{r+1}\frac{h^k}{k!}u_t^{(k)}(x_n)+\frac{h^{r+2}}{(r+2)!}u_t^{(r+2)}(x_n+\theta_{t,0}h),$$

$$u_t'(x_n+\gamma_i h)=\sum_{k=0}^{r}\frac{(\gamma_i h)^k}{k!}u_t^{(k+1)}(x_n)+\frac{(\gamma_i h)^{r+1}}{(r+1)!}u_t^{(r+2)}(x_n+\theta_{t,i}h),$$

where $0<\theta_{t,0},\theta_{t,1},\cdots,\theta_{t,r}<1$, into (2.3) to give

$$\begin{aligned}Q_t(x_n+\tau h)=d_1(\tau)u_t(x_n)+d_2(\tau)\left\{\sum_{k=0}^{r+1}\frac{h^k}{k!}u_t^{(k)}(x_n)+\frac{h^{r+2}}{(r+2)!}u_t^{(r+2)}(x_n+\theta_{t,0}h)\right\}\\ +h\sum_{i=1}^{r}e_i(\tau)\left\{\sum_{k=0}^{r}\frac{(\gamma_i h)^k}{k!}u_t^{(k+1)}(x_n)+\frac{(\gamma_i h)^{r+1}}{(r+1)!}u_t^{(r+2)}(x_n+\theta_{t,i}h)\right\}.\end{aligned}$$

This may be rearranged as

$$\begin{aligned}Q_t(x_n+\tau h)=u_t(x_n)\{d_1(\tau)+d_2(\tau)\}\\ +\sum_{k=1}^{r+1}\frac{h^k}{k!}u_t^{(k)}(x_n)\left\{d_2(\tau)+\sum_{i=1}^{r}e_i(\tau)k\gamma_i^{k-1}\right\}\\ +h^{r+2}\left\{\frac{d_2(\tau)}{(r+2)!}u_t^{(r+2)}(x_n+\theta_{t,0}h)\right.\\ \left.+\sum_{i=1}^{r}e_i(\tau)\frac{\gamma_i^{r+1}}{(r+1)!}u_t^{(r+2)}(x_n+\theta_{t,i}h)\right\}.\end{aligned} \tag{2.4}$$

Since $d_2(\tau)$, $\{e_i(\tau)\}_{i=1}^r$ and $u^{(r+2)}(x_n+\tau h)$ are bounded on $[0, 1]$, this may be written as

$$Q_t(x_n+\tau h) = u_t(x_n)\{d_1(\tau)+d_2(\tau)\}+\sum_{k=1}^{r+1}\frac{h^k}{k!}u_t^{(k)}(x_n) \cdot\left\{d_2(\tau)+\sum_{i=1}^{r} e_i(\tau)k\gamma_i^{k-1}\right\}+O(h^{r+2}). \tag{2.5}$$

Now, by uniqueness, $Q_t(x_n+\tau h)$ will be exact for $u_t(x_n+\tau h)\equiv(\tau h)^k$, for $0\leqq k\leqq r+1$. Hence, in (2.3),

$$d_1(\tau)+d_2(\tau)\equiv 1, \tag{2.6}$$

$$d_2(\tau)+\sum_{i=1}^{r} e_i(\tau)k\gamma_i^{k-1}\equiv\tau^k, \qquad 1\leqq k\leqq r+1. \tag{2.7}$$

Substituting into (2.5) this gives

$$\begin{aligned} Q_t(x_n+\tau h) &= u_t(x_n)+\sum_{k=1}^{r+1}\frac{(\tau h)^k}{k!}u_t^{(k)}(x_n)+O(h^{r+2}) \\ &= u_t(x_n+\tau h)+O(h^{r+2}). \end{aligned} \tag{2.8}$$

Similarly, by differentiating (2.4), (2.6), and (2.7) with respect to τ, we can show that

$$\frac{1}{h}\frac{d}{d\tau}Q_t(x_n+\tau h) = u_t'(x_n+\tau h)+O(h^{r+1}). \tag{2.9}$$

Using (2.2) and (2.3), the data error $H(x)-Q(x)$ may be written

$$\begin{aligned} H(x_n+\tau h)-Q(x_n+\tau h) &= d_2(\tau)[y_{n+1}-u(x_{n+1})] \\ &\quad +h\sum_{i=1}^{r} e_i(\tau)[u_i'-u'(x_n+\gamma_i h)] \\ &= d_2(\tau)[y_{n+1}-u(x_{n+1})]+O(h^{p+2}), \end{aligned} \tag{2.10}$$

since the derivative approximations are accurate to $O(h^{p+1})$. Similarly,

$$\begin{aligned} &\frac{1}{h}\frac{d}{d\tau}\{H(x_n+\tau h)-Q(x_n+\tau h)\} \\ &\quad =\frac{1}{h}d_2'(\tau)[y_{n+1}-u(x_{n+1})]+O(h^{p+1}). \end{aligned} \tag{2.11}$$

If we have $r\geqq p$, that is, if we use a sufficient amount of derivative data, then, combining (2.8) and (2.10) we have

$$H(x_n+\tau h)-u(x_n+\tau h) = d_2(\tau)[y_{n+1}-u(x_{n+1})]+O(h^{p+2}).$$

We thus have an interpolant of local order $p+1$. Furthermore, $H(x)$ has the desirable property that the local error at any point in the step can be directly related to the local error at the next meshpoint $y_{n+1}-u(x_{n+1})$. Some low-order interpolation schemes with this property have been discussed in [9]. The defect in $H(x)$ satisfies

$$\begin{aligned} \delta(x) &= H'(x)-f(x, H(x)) \\ &= H'(x)-u'(x)+f(x, u(x))-f(x, H(x)) \\ &= H'(x)-u'(x)+O(h^{p+1}), \end{aligned}$$

which, using (2.9) and (2.11), has the form

$$\delta(x_n+\tau h)=\frac{1}{h}\,d_2'(\tau)[y_{n+1}-u(x_{n+1})]+O(h^{p+1}). \tag{2.12}$$

Hence, for sufficiently small stepsizes each component of the defect behaves like a multiple of a known polynomial over each step.

As we mentioned earlier, the extra derivative data $\{u_i'\}_{i=3}^r$ could be obtained by first constructing a standard interpolant $p(x)$ of local order $p+1$, and then setting $u_i'=f(x_n+\gamma_i h, p(x_n+\gamma_i h))$. With $r=p$, this would mean that the new interpolant requires $p-2$ more f evaluations per step than $p(x)$. However, in the case where $p(x)$ is a Hermite interpolating polynomial it is possible that some $u_i'=f(x_n+\gamma_i h, p(x_n+\gamma_i h))$ data was used in the formation of $p(x)$ and hence will be available "free of charge." An example of this will be seen in the next section.

From the expansion (2.12) we see that in order to have a small defect, it is desirable that the local error per unit step $[y_{n+1}-u(x_{n+1})]/h$ be small. By examining the truncation coefficients in the asymptotic local error expansions, it is possible to derive Runge-Kutta formulas with "minimal" local errors (see [3] for an overview). Such formulas would clearly be useful in our context. The polynomial $d_2'(\tau)$ appearing in (2.12) is determined solely by the choice of $\{\gamma_i\}_{i=3}^r$. (From (2.1) and (2.2), $d_2(\tau)$ is determined by the conditions $d_2(0)=d_2'(\gamma_i)=0$, $1\leqq i\leqq r$, and $d_2(1)=1$.) Hence it is sensible to choose these points so that the relevant measure of $d_2'(\tau)$ is small. In particular, if we are concerned with controlling $\max_{\tau\in[0,1]}\|\delta(x_n+\tau h)\|$ over each step, for some vector norm $\|\cdot\|$, then the most efficient scheme asymptotically is given by choosing $\{\gamma_i\}_{i=3}^r$ to solve the minimax problem

$$\min_{\{\{\gamma_i\}_{i=3}^r\in(0,1)\text{ distinct}\}}\ \max_{\tau\in[0,1]}|d_2'(\tau)|. \tag{2.13}$$

The author does not know whether this problem can be solved analytically. In any case it may be necessary to place other constraints on $\{\gamma_i\}_{i=3}^r$. For example, some γ_i values may be fixed a priori, and the γ_i should be reasonably well spaced out across the step (see the next section).

The following lemma gives a little insight into the problem.

LEMMA 2.1. *Suppose there exists a unique polynomial $d_2(\tau)$ of degree $\leqq r+1$ satisfying*

$$d_2(0)=0, \qquad d_2(1)=1,$$
$$d_2'(\gamma_i)=0, \qquad 1\leqq i\leqq r,$$

where $\gamma_1=0$, $\gamma_2=1$ and $\{\gamma_i\}_{i=3}^r\in(0,1)$ are distinct. Let $d_2^*(\tau)$ be the corresponding polynomial which satisfies

$$d_2^*(0)=0, \qquad d_2^*(1)=1,$$
$$d_2^{*\prime}(1-\gamma_i)=0, \qquad 1\leqq i\leqq r.$$

Then $1-d_2(\tau)\equiv d_2^*(1-\tau)$ and hence $d_2'(\tau)\equiv d_2^{*\prime}(1-\tau)$, so that

$$\max_{[0,1]}|d_2'(\tau)|=\max_{[0,1]}|d_2^{*\prime}(\tau)|.$$

Proof. Given $d_2(\tau)$ as in the lemma, $g(\tau):=1-d_2(\tau)$ is the unique polynomial of degree $\leqq r+1$ which satisfies

$$g(0)=1, \qquad g(1)=0,$$
$$g'(\gamma_i)=0, \qquad 1\leqq i\leqq r.$$

These conditions are precisely those which $d_2^*(1-\tau)$ must satisfy. □

The lemma shows that solutions to the minimax problem (2.13) generally arise in pairs $\{\gamma_i\}_{i=3}^r$ and $\{1-\gamma_i\}_{i=3}^r$. If the problem is to be solved by performing a grid search, then the lemma allows us to reduce the amount of searching. For example, if $r=5$, then rather than considering $\gamma_3, \gamma_4, \gamma_5 \in (0,1)$, we may restrict attention to $\gamma_5 \in (0,1)$ and $\gamma_3, \gamma_4 \in (0, \frac{1}{2}]$. (We may also assume, without loss of generality, the ordering $\gamma_3 < \gamma_4 < \gamma_5$.)

In the remaining sections we focus on the case $p=5$. First we derive the conditions which guarantee the existence of a unique Hermite-Birkhoff interpolant. We then compute the optimal points $\{\gamma_i\}_{i=3}^5$ and test the resulting defect control scheme numerically.

3. The case $p=5$. With a fifth-order Runge-Kutta formula ($p=5$) we must choose $r \geqq 5$ in order to construct a Hermite-Birkhoff interpolation scheme of the required form. Taking $r=5$, we have three free parameters γ_3, γ_4, and γ_5, which must be distinct in $(0,1)$. The interpolation conditions that $H(x)$ must satisfy can be cast as a system of linear equations. The determinant of the system can be shown to be a constant multiple of

$$\det := 3(\gamma_3+\gamma_4+\gamma_5) - 5(\gamma_3\gamma_4+\gamma_3\gamma_5+\gamma_4\gamma_5) + 10\gamma_3\gamma_4\gamma_5 - 2.$$

Hence $H(x)$ exists uniquely if and only if $\{\gamma_3, \gamma_4, \gamma_5\}$ are chosen so that $\det \neq 0$. For example, $\{0.4, 0.5, 0.6\}$, $\{0.1, 0.5, 0.9\}$, and $\{0.28, 0.75, 0.8\}$ are invalid parameters. Further straightforward but tedious algebra shows that $d_2(\tau)$ has the form

$$d_2(\tau) = \tau^2(a\tau^4 + b\tau^3 + c\tau^2 + d\tau + e),$$

where

$$a = \frac{10}{\det}, \qquad b = \frac{-12}{\det}(\gamma_3+\gamma_4+\gamma_5+1),$$

$$c = \frac{15}{\det}(\gamma_3+\gamma_4+\gamma_5+\gamma_3\gamma_4+\gamma_3\gamma_5+\gamma_4\gamma_5),$$

$$d = -2-2c-3b-4a, \qquad e = 3+3a+2b+c.$$

As discussed earlier, one reasonable way to choose $\{\gamma_3, \gamma_4, \gamma_5\}$ is to minimize the quantity $\text{d2max} := \max_{[0,1]} |d_2'(\tau)|$. Using a simple three-dimensional grid search, we found that as the grid spacing decreased, the optimal γ_5 value seemed to approach one. To avoid approaching this pathological case, we chose to restrict the parameters to $[0.1, 0.9]$. Varying the parameters in steps of 0.1 gave optimal values of $\{0.3, 0.4, 0.9\}$ for which d2max = 2.35. Although finer grid spacing slightly reduced d2max, it appeared that γ_3 and γ_4 were converging to the same value.

In the numerical experiments described in the next section we base our Hermite-Birkhoff interpolant on the six-stage, fifth-order formula from the RK5(4)7FM pair derived by Dormand and Prince [2]. For this formula, Shampine [13] showed that by using $f(x_{n+1}, y_{n+1})$ and adding one extra stage, it is possible to construct an approximation $y_{n+1/2}$ which satisfies $y_{n+1/2} - u(x_n + h/2) = O(h^6)$. It follows that the Hermite interpolant to $y_n, f(x_n, y_n), y_{n+1/2}, f(x_n + h/2, y_{n+1/2}), y_{n+1}$, and $f(x_{n+1}, y_{n+1})$ is locally sixth order. While the general Hermite-Birkhoff interpolant derived above requires three extra f evaluations per step, if we set $\gamma_3 = \frac{1}{2}$ then we can save one evaluation by using $f(x_n + h/2, y_{n+1/2})$ as derivative data. With $\gamma_3 = \frac{1}{2}$ and $\{\gamma_4, \gamma_5\}$ varying in $[0.1, 0.9]$, a grid search with spacing of 0.0001 gave $\{0.1, 0.7051\}$ as the optimal values. The corresponding d2max value is 3.46, and the optimal sample point is $\tau^* = 0.89994049343102$.

4. Numerical results. We now describe some numerical testing of the scheme outlined above. On each step we sampled the defect in $H(x)$ at the asymptotically optimal point and accepted the step if and only if $\|\delta(x_n+\tau^*h)\|_\infty < \mathrm{TOL}$, where TOL is an absolute error tolerance. Stepsize selection was based on the usual asymptotically motivated formula

$$\frac{h_{\text{new}}}{h_{\text{old}}} = 0.9\left(\frac{\mathrm{TOL}}{\|\delta(x_n+\tau^*h)\|_\infty}\right)^{1/5}.$$

For comparison, we also implemented a defect control scheme using the locally sixth-order Hermite interpolant based on $y_n, f(x_n, y_n), y_{n+1/2}, f(x_n+h/2, y_{n+1/2}), y_{n+1}$, and $f(x_{n+1}, y_{n+1})$. In this case the shape of the defect is not known a priori—it is problem dependent and varies from step to step. The construction of this interpolant requires two fewer f evaluations than that of the Hermite-Birkhoff interpolant; hence we chose to sample the defect at three points on each step in order to give schemes with the same overall cost per step. The maximum observed defect at the three sample points was used as the defect estimate. Noting that the defect is zero at the midpoint of the step, we used $\tau = \frac{1}{4}, \frac{2}{3}$, and $\frac{5}{6}$ as the three sample points.

To measure the accuracy of the defect sample in the Hermite-Birkhoff scheme we formed

$$D := \max_n \left\{\frac{\max_{0\le j\le 100}\|\delta(x_n+0.01jh)\|_\infty}{\|\delta(x_n+\tau^*h)\|_\infty}\right\}, \tag{4.1}$$

which is the worst case of the sample value underestimating the "true" maximum defect over all steps. Similarly, we measured the accuracy of the underlying interpolant by computing

$$G := \max_n \left\{\frac{\max_{0\le j\le 100}\|H(x_n+0.01jh)-y(x_n+0.01jh)\|_\infty}{\|y_{n+1}-y(x_{n+1})\|_\infty}\right\}. \tag{4.2}$$

The ratio G compares the global error in $H(x)$ with that of the basic Runge-Kutta formula. Corresponding results were obtained for the Hermite scheme, with $\max_{\tau\in\{1/4,2/3,5/6\}}\|\delta(x_n+\tau h)\|_\infty$ in the denominator of (4.1).

We used the following nonstiff systems:

(i) A problem due to Fehlberg [see 11]:

$$y_1' = 2xy_1\log(\max(y_2, 10^{-3})), \qquad y_1(1) = \exp(\sin(1)),$$
$$y_2' = -2xy_2\log(\max(y_1, 10^{-3})), \qquad y_2(1) = \exp(\cos(1)), \qquad 1\le x\le 5.$$

(ii)-(iv) The orbit equations [8, Class D]:

$$y_1' = y_3, \qquad y_1(0) = 1-\varepsilon,$$
$$y_2' = y_4, \qquad y_2(0) = 0,$$
$$y_3' = \frac{-y_1}{(y_1^2+y_2^2)^{3/2}}, \qquad y_3(0) = 0,$$
$$y_4' = \frac{-y_2}{(y_1^2+y_2^2)^{3/2}}, \qquad y_4(0) = \left(\frac{1+\varepsilon}{1-\varepsilon}\right)^{1/2}, \qquad 0\le x\le 20,$$

with values of 0.1, 0.5, and 0.9, respectively, for the eccentricity parameter ε.

The results can be found in Tables 1 and 2.

We see that for the Hermite-Birkhoff scheme, the defect sample is generally very reliable, more so than for the original robust schemes in [10]. It is noticeable that the

TABLE 1
Defect ratios D for the Hermite-Birkhoff (HB) *and Hermite* (H) *schemes.*

		TOL 10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
(i)	HB	1.002	1.002	1.000	1.001	1.071
	H	1.946	2.570	2.417	2.413	2.405
(ii)	HB	1.000	1.000	1.000	1.000	1.004
	H	1.001	1.001	1.003	1.198	1.383
(iii)	HB	1.000	1.001	1.000	1.000	1.012
	H	1.076	1.115	1.746	2.125	2.078
(iv)	HB	1.025	1.032	1.706	1.032	1.463
	H	1.887	2.718	3.242	2.565	2.500

TABLE 2
Global error ratios G for the Hermite-Birkhoff (HB) *and Hermite* (H) *schemes.*

		TOL 10^{-2}	10^{-4}	10^{-6}	10^{-8}	10^{-10}
(i)	HB	1.070	1.001	1.000	1.000	1.000
	H	1.037	1.000	1.000	1.000	1.000
(ii)	HB	1.033	1.010	1.000	1.000	1.000
	H	1.003	1.004	1.001	1.000	1.000
(iii)	HB	1.023	1.011	1.004	1.001	1.000
	H	1.000	1.025	1.000	1.000	1.000
(iv)	HB	1.038	1.004	1.002	1.000	1.000
	H	1.018	1.007	1.001	1.000	1.000

ratio D worsens as TOL decreases from 10^{-8} to 10^{-10}. This behavior can be attributed to rounding errors—the defect is formed as a numerical difference which can involve a significant amount of cancellation. (For these computations, the unit roundoff was $\approx 2 \times 10^{-16}$.) Such rounding errors are especially prone to occur on the first few steps of an integration. Here a code will typically take conservatively small steps until it finds the scale of the problem. On these steps the defect may be considerably less than TOL. Overall the scheme performed worst on the $\varepsilon = 0.9$ orbit problem. Here the solution is known to change rapidly in certain regions, and hence it is likely that there are some steps where the stepsize-selection scheme has not adequately taken account of these rapid changes and the higher-order terms in the defect expansion cause the "optimum" sample to be slightly less accurate than normal. The Hermite scheme performs less reliably than the Hermite-Birkhoff scheme on all four problems. It is liable to underestimate the defect by a factor of around two. This behavior is to be expected given the problem-dependent nature of the defect. In fact it is perhaps surprising that the defect ratio D remains reasonably small (in theory it can be arbitrarily large). The original experiments of Enright [5] with such "nonrobust" defect control schemes gave similar results. In terms of global errors, Table 2 shows that both interpolants deliver almost exactly the same accuracy as the Runge-Kutta formula on each test problem.

In summary, we have given a general technique for constructing robust defect control schemes, and the particular case that we implemented performed reliably in practice. Two important questions, which lie beyond the scope of this paper, are the following:

- How to compare numerically two different defect control schemes, using such criteria as efficiency, reliability, and the proportionality of the global error to the tolerance.
- How to compare numerically schemes which use different types of error control, such as defect control and the various types of local error control (see [6] for more details).

It is hoped that these issues will be addressed in the near future using a modified version of the DETEST package [8] which is currently under development at the University of Toronto.

Acknowledgments. I am grateful to Wayne Enright, Nick Higham, and Ken Jackson for helpful comments.

REFERENCES

[1] J. R. CASH, *A block 6(4) Runge-Kutta formula for nonstiff initial value problems*, ACM Trans. Math. Software, 15 (1989), pp. 15-28.

[2] J. R. DORMAND AND P. J. PRINCE, *A family of embedded Runge-Kutta formulae*, J. Comput. Appl. Math., 6 (1980), pp. 19-26.

[3] ———, *Practical Runge-Kutta processes*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 977-989.

[4] J. R. DORMAND, M. A. LOCKYER, N. E. MCGORRIGAN, AND P. J. PRINCE, *Global error estimation with Runge-Kutta triples*, Tech. Report TP-CS-88-01, Department of Computer Science, Teesside Polytechnic, Middlesbrough, England, 1988.

[5] W. H. ENRIGHT, *A new error-control for initial value solvers*, Appl. Math. Comput., 31 (1989), pp. 288-301.

[6] ———, *Analysis of error control strategies for continuous Runge-Kutta methods*, SIAM J. Numer. Anal., 26 (1989), pp. 588-599.

[7] W. H. ENRIGHT, K. R. JACKSON, S. P. NØRSETT, AND P. G. THOMSEN, *Interpolants for Runge-Kutta formulas*, ACM Trans. Math. Software, 12 (1986), pp. 193-218.

[8] W. H. ENRIGHT AND J. D. PRYCE, *Two* FORTRAN *packages for assessing initial value methods*, ACM Trans. Math. Software, 13 (1987), pp. 1-27.

[9] I. GLADWELL, L. F. SHAMPINE, L. S. BACA, AND R. W. BRANKIN, *Practical aspects of interpolation in Runge-Kutta codes*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 322-341.

[10] D. J. HIGHAM, *Robust defect control with Runge-Kutta schemes*, SIAM J. Numer. Anal., 26 (1989), pp. 1175-1183.

[11] ———, *Defect estimation in Adams* PECE *codes*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 964-976.

[12] ———, *Highly continuous Runge-Kutta interpolants*, Tech. Report No. 220/89, Department of Computer Science, University of Toronto, Canada, 1989; ACM Trans. Math. Software, to appear.

[13] L. F. SHAMPINE, *Some practical Runge-Kutta formulas*, Math. Comp., 46 (1986), pp. 135-150.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1000–1028, September 1991
© 1991 Society for Industrial and Applied Mathematics

ITERATED RUNGE–KUTTA METHODS ON PARALLEL COMPUTERS*

P. J. VAN DER HOUWEN† AND B. P. SOMMEIJER†

Abstract. This paper examines diagonally implicit iteration methods for solving implicit Runge–Kutta methods with high stage order on parallel computers. These iteration methods are such that after a finite number of m iterations, the iterated Runge–Kutta method belongs to the class of diagonally implicit Runge–Kutta methods (DIRK methods) using mk implicit stages where k is the number of stages of the generating implicit Runge–Kutta method (corrector method). However, a large number of the stages of this DIRK method can be computed in parallel, so that the number of stages that have to be computed sequentially is only m. The iteration parameters of the method are tuned in such a way that fast convergence to the stability characteristics of the corrector method is achieved. By means of numerical experiments it is also shown that the solution produced by the resulting iteration method converges rapidly to the corrector solution so that both stability and accuracy characteristics are comparable with those of the corrector. This implies that the reduced accuracy often shown when integrating stiff problems by means of DIRK methods already available in the literature (which is caused by a low stage order) is not shown by the DIRK methods developed in this paper, provided that the corrector method has a sufficiently high stage order.

Key words. diagonally implicit Runge–Kutta methods, parallelism, stability

AMS(MOS) subject classifications. 65M10, 65M20

C.R. classification. 5.17

1. Introduction.

1.1. Runge–Kutta methods. Suppose that we want to solve stiff initial-value problems for systems of first-order, ordinary differential equations (ODEs), i.e.,

$$\frac{d\mathbf{y}(t)}{dt}=\mathbf{f}(t,\mathbf{y}(t)),\qquad \mathbf{y}(t_0)=\mathbf{y}_0,\qquad \mathbf{y}:\mathbb{R}\to\mathbb{R}^d,\qquad \mathbf{f}:\mathbb{R}\times\mathbb{R}^d\to\mathbb{R}^d, \tag{1.1}$$

by means of a Runge–Kutta (RK) method. Then the stiffness of the problem requires that the RK method should be sufficiently stable, preferably A-stable, and therefore implicit. This leads us to fully implicit RK methods (IRK methods) in which the Butcher array,

$$\begin{array}{c|c} \mathbf{c} & A \\ \hline & \mathbf{b}^T \end{array}, \tag{1.2}$$

has a full A matrix. Most widely used are the IRK methods based on Gaussian quadrature formulas (such as Gauss–Legendre, Lobatto, and Radau methods), which are known to be A-stable for any order of accuracy. However, the high degree of implicitness of these methods implies that solving the implicit relations is rather costly. In general, a k-stage IRK method (that is, $\mathbf{b}$ and $\mathbf{c}$ are k-dimensional vectors and A is a k-by-k matrix) requires in each step the solution of a system of dimension kd, so that the computational complexity is of order $(kd)^3$. This compares unfavourably with implicit linear multistep methods which require in each step the solution of a system of dimension d.

In order to reduce the computational labour involved when using implicit RK methods, various people have considered diagonally implicit RK methods (DIRK methods) possessing a lower triangular A matrix and therefore requiring (in general)

* Received by the editors January 16, 1990; accepted for publication (in revised form) September 12, 1990.
† Centre for Mathematics and Computer Science, Post Box 4079, 1009 AB Amsterdam, the Netherlands.

in each step the solution of k systems of dimension d. Hence, the computational complexity is now of order kd^3 instead of order $(kd)^3$. Unfortunately, the price we have to pay for the less expensive DIRK methods is a considerable drop in accuracy in many stiff problems. This is caused by the phenomenon of order reduction (cf., e.g., [21], [9], [11]) which reduces the observed order of RK methods to their stage order (or their stage order plus one). Most DIRK methods are particularly sensitive to order reduction because their stage order is only one or two, which is much smaller than for k-stage Gauss-Legendre, Lobatto IIIA and Radau IIA methods which have all stage order k.

An alternative for the DIRK methods are the singly implicit RK methods (SIRK methods) of Burrage [2] which possess a high stage order. By means of a transformation technique due to Butcher (see [5], [6]), these SIRK methods can be transformed into methods that are, like DIRK methods, only diagonally implicit. However, the additional transformations required in each step cause that the total costs per step are considerably higher than for DIRK methods.

Yet another possibility is the use of parallel processors. In this paper, we shall show that on parallel computers the fully implicit relations associated with IRK methods can be solved efficiently by using the highly parallelizable iteration methods of diagonally implicit type proposed in van der Houwen, Sommeijer, and Couzy [13]. This brings us back to using IRK methods as corrector method instead of using DIRK or SIRK methods. In particular, we shall concentrate on iterating IRK methods possessing high *stage* orders.

1.2. IRK methods with high stage orders. Most IRK methods are designed in such a way that they have a high order at the step points. However, as already remarked above, a high order at step points is often spoiled by order reduction, so that it seems more natural to look for IRK methods with as high a stage order as possible. In order to achieve this, we shall consider $(k+1)$-stage IRK methods of the type

$$\begin{array}{c|cc} 0 & 0 & \mathbf{0}^T \\ \mathbf{c} & \mathbf{a} & A \\ \hline & b_0 & \mathbf{b}^T \end{array}, \tag{1.3}$$

where b_0 is a scalar, $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ are k-dimensional vectors, and A is again a k-by-k matrix. IRK methods of this type have roughly the same computational complexity as the IRK methods of type (1.2), but they possess the additional parameter vector $\mathbf{a}$ which can be used for increasing the stage order. To see that (1.2) and (1.3) are (almost) equally expensive, let us assume (for simplicity of notation) that (1.1) is a scalar problem (i.e., $d=1$), and let us introduce the vectors

$$\mathbf{Y}_{n+1} := (y_{n,1}, \cdots, y_{n,k})^T, \qquad \mathbf{c} := (c_1, \cdots, c_k)^T,$$

where $y_{n,i}$ denotes a numerical approximation to the exact solution value $y(t_n + c_i h)$, h being the stepsize. Then we can write (1.3) in the form

$$\begin{aligned} &\mathbf{Y}_{n+1} - hAf(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}_{n+1}) = \mathbf{e}y_n + h\mathbf{a}f(t_n, y_n), \\ &y_{n+1} = y_n + hb_0 f(t_n, y_n) + h\mathbf{b}^T f(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}_{n+1}). \end{aligned} \tag{1.3'}$$

Here, $\mathbf{e}$ is the vector with unit entries, and we used the convention that for any given vectors $\mathbf{v} = (v_j)$ and $\mathbf{t} = (t_j)$, $f(\mathbf{t}, \mathbf{v})$ denotes the vector with entries $f(t_j, v_j)$. If $b_0 = 0$ and $\mathbf{a} = 0$, then it follows from (1.3′) that (1.3) reduces to (1.2), so that in each step the computational complexity of (1.2) and (1.3) differ by the evaluation of $f(t_n, y_n)$, but

both methods require the solution of a system of dimension *kd*. Since the bulk of the computational effort goes into solving this system, the methods (1.2) and (1.3) may be considered as equally expensive.

The vectors $\mathbf{Y}_{n+1}$ and $\mathbf{c}$ will, respectively, be called the *stage vector* and the *block point vector*, and the points t_n and $t_n + c_j h$ will, respectively, be called *step points* and *block points*. The minimal order achieved at the block points and step points are, respectively, the *stage order* and *step point order*.

If the method parameters are chosen in such a way that the stage order is as large as possible with $\mathbf{c}$ arbitrary, then (1.3) is equivalent to the IRK method derived from Lagrange quadrature formulas and will be called a *Lagrange method*. If $c_j = j/k$, then Lagrange methods reduce to the *Newton-Cotes methods* studied in Watts and Shampine [23], and if the components of $\mathbf{c}$ equal the Lobatto quadrature points, then they reduce to the Lobatto IIIA methods. However, Newton-Cotes and Lobatto IIIA methods are only *weakly A*-stable (i.e., the method hardly damps the highly stiff components in the numerical error). It is our aim to construct Lagrange methods with better stability properties than Newton-Cotes and Lobatto IIIA methods, i.e., methods which damp both nonstiff and stiff components occurring in the numerical error (*strongly A*-stable methods).

An important family of IRK methods are the so-called *stiffly accurate methods* (cf. Alexander [1]). If the IRK method is of the form (1.3), then this family is obtained by setting

$$b_0 = \mathbf{e}_k^T \mathbf{a}, \qquad \mathbf{b}^T = \mathbf{e}_k^T A, \qquad c_k = 1, \tag{1.4}$$

where $\mathbf{e}_k$ is the kth unit vector. Notice that, when represented by their Butcher array (1.3), the last row in (1.3) equals the preceding one. It was shown by Hairer, Lubich, and Roche [11] that this property implies that for certain classes of stiff problems the method does not suffer the effect of order reduction. Examples of stiffly accurate IRK methods are the Lobatto IIIA, Radau IIA and Newton-Cotes methods.

1.3. Diagonally implicit iteration of IRK methods. After a finite number of m iterations of the implicit relation for $\mathbf{Y}_{n+1}$ given in (1.3′) by the aforementioned diagonally implicit iteration process (or briefly *diagonal iteration*) (see also § 3), the resulting scheme actually is an $(mk+1)$-stage DIRK method. One of these stages is explicit and the other mk stages are of diagonally implicit form. However, a large number of these mk implicit stages can be computed in parallel, resulting in a process where only m stages have to be computed sequentially.

The iteration parameters of the method can be tuned in such a way that we get fast convergence to the stability characteristics of the corrector method, provided that the corrector is stiffly accurate (in § 3.3.1, we will show that the diagonal iteration of the type employed in this paper is not suitable for iterating *nonstiffly accurate* correctors).

Second, it has been demonstrated that the iterated methods based on strongly *A*-stable correctors (such as the Radau IIA correctors and the Lagrange correctors derived in § 4) are within a few iterations strongly *A*-stable themselves. It is highly unlikely that this nice property is shared by the methods based on (weakly) *A*-stable IRK correctors because the stability function of the iterated methods should converge to a (weakly) *A*-acceptable function. In fact, for a number of Newton-Cotes and Lobatto IIIA correctors it was checked that the stability function becomes *A*-acceptable only after an infinite number of iterations.

Finally, numerical experiments reveal that the drop in accuracy, exhibited in many stiff problems by the conventionally constructed DIRK methods, is not shown by the

DIRK methods constructed by the diagonal iteration process of this paper. In a forthcoming paper [4] it is intended to present a theoretical analysis of this phenomenon using the error analysis proposed in Burrage [3].

2. Accuracy and stability of the corrector. In the following two subsections, we discuss the stage order, step point order, and stability of the corrector equation (1.3′).

2.1. Stage order. Let $\mathbf{Y}(t_{n+1})$ denote the vector with components $y(t_n + c_i h)$ where y is the locally exact solution of (1.1) satisfying $y(t_n) = y_n$, then, following Butcher [7], (1.3′) is said to have stage order r if the residual left upon substitution of $\mathbf{Y}(t_{n+1})$ into the formula for $\mathbf{Y}_{n+1}$ is of order $r+1$ in h, i.e.,

$$\mathbf{Y}(t_{n+1}) - hAf(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}(t_{n+1})) - \mathbf{e}y_n - h\mathbf{a}f(t_n, y_n) = O(h^{r+1}). \tag{2.1}$$

The stage-order conditions for (1.3′) are straightforwardly derived (cf. [22]) and are given by

$$\mathbf{C}_j = \mathbf{0}, \quad j = 1, \cdots, r, \quad \mathbf{C}_1 := \mathbf{a} + A\mathbf{e} - \mathbf{c}, \quad \mathbf{C}_j := jA\mathbf{c}^{j-1} - \mathbf{c}^j, \quad j = 2, 3, \cdots, \tag{2.2}$$

where $\mathbf{c}^j$ denotes the vector with components $(c_i)^j$. Thus, to achieve stage order r for a given block point vector $\mathbf{c}$, we have to solve rk linear equations in $k^2 + k$ unknowns, so that the maximal stage order equals $k+1$. The corresponding methods will be called *Lagrange methods.*

2.2. Step point order. Consider the formula for y_{n+1} given in (1.3′):

$$y_{n+1} = y_n + hb_0 f(t_n, y_n) + h\mathbf{b}^T f(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}_{n+1}). \tag{2.3}$$

Since $\mathbf{Y}_{n+1}$ approximates $\mathbf{Y}(t_{n+1})$ with (local) order $r+1$, r being the stage order (cf. (2.1)), we can derive that y_{n+1} has (at least) order $p = \min\{r+1, q\}$ if the conditions

$$D_j = 0, \quad j = 1, \cdots, q, \quad D_1 := b_0 + \mathbf{b}^T\mathbf{e} - 1, \quad D_j := j\mathbf{b}^T\mathbf{c}^{j-1} - 1, \quad j = 2, 3, \cdots \tag{2.4}$$

are satisfied. We remark that p may be larger than $\min\{r+1, q\}$ if the methods possess the property of so-called "superconvergence" which for example is the case in Gauss, Radau, and Lobatto methods. The error constant of (2.3) is given by

$$E_{q+1} := \frac{D_{q+1}}{(q+1)!} = \frac{(q+1)\mathbf{b}^T\mathbf{c}^q - 1}{(q+1)!}. \tag{2.5}$$

Assuming that $\mathbf{c}$ is given, the conditions (2.4) present a linear system of q equations in $k+1$ unknowns, so that by setting $q = k+1$ we achieve at least step point order $p = \min\{r+1, k+1\}$ for any block point vector $\mathbf{c}$.

As already observed in the introduction, the usual approach in exploiting the vector $\mathbf{c}$ is the maximization of the step point order (to obtain "superconvergence"). Alternatively, we may use $\mathbf{c}$ for improving the stability of the method or for the minimization of error constants. In this paper, we shall use $\mathbf{c}$ for achieving strong A-stability.

In the special case of stiffly accurate methods satisfying condition (1.4), y_{n+1} equals the last component of $\mathbf{Y}_{n+1}$ so that the step point order p is also at least the stage order r, but is sometimes higher. For instance, the Newton-Cotes methods have stage order $k+1$ and, if k is even, step point order $k+2$.

2.3. Stability. By applying (1.3′) to the test equation $y' = \lambda y$, we are led to recursions of the form

$$\mathbf{Y}_{n+1} = [I - zA]^{-1}[\mathbf{e} + z\mathbf{a}]y_n, \quad y_{n+1} = (1 + b_0 z)y_n + z\mathbf{b}^T\mathbf{Y}_{n+1}, \quad z := \lambda h. \tag{2.6}$$

Hence,

$$y_{n+1} = R(z)y_n, \qquad R(z) := 1 + b_0 z + z\mathbf{b}^T[I - zA]^{-1}[\mathbf{e} + z\mathbf{a}]. \tag{2.7}$$

$R(z)$ is called the stability function of the one-step method. In the special case of stiffly accurate methods where (1.4) is satisfied, (2.7) reduces to

$$y_{n+1} = R(z)y_n, \qquad R(z) := \mathbf{e}_k^T[I - zA]^{-1}[\mathbf{e} + z\mathbf{a}]. \tag{2.8}$$

The stability region of the method is defined by the region where R is bounded by 1. In the case of the Newton-Cotes methods where the components of $\mathbf{c}$ are equally spaced, it was shown in Watts and Shampine [23] that they are A-stable for $k \leqq 8$ (but they are not for $k = 9$ and $k = 10$).

We conclude this section by summarizing in Table 2.1 the characteristics of a number of correctors available in the literature. In this table, it is assumed that the IRK method is presented in the form (1.3′), so that for all methods listed the dimension of the implicit relation to be solved equals kd, d being the dimension of the system of ODEs.

3. Diagonal iteration. We shall use a diagonal iteration method to solve the stage vector $\mathbf{Y}_{n+1}$ from the fully implicit (corrector) equation defined in (1.3′). For scalar differential equations, the iteration method reads

$$\begin{aligned} &\mathbf{Y}^{(1)} - hDf(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}^{(1)}) = y_n\mathbf{e} + h\mathbf{a}f(t_n, y_n) + h[A - D]f(\mathbf{t}^{(0)}, \mathbf{Y}^{(0)}), \\ &\mathbf{Y}^{(j)} - hDf(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}^{(j)}) = y_n\mathbf{e} + h\mathbf{a}f(t_n, y_n) \\ &\qquad\qquad + h[A - D]f(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}^{(j-1)}), \qquad j = 2, 3, \cdots, \end{aligned} \tag{3.1a}$$

where $(\mathbf{t}^{(0)}, \mathbf{Y}^{(0)})$ is an initial approximation to $(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}_{n+1})$ and D is an arbitrary diagonal matrix. If m iterations are performed, then y_{n+1} is defined by

$$y_{n+1} = y_n + hb_0 f(t_n, y_n) + h\mathbf{b}^T f(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}^{(m)}) \quad \text{or} \quad y_{n+1} = \mathbf{e}_k^T\mathbf{Y}^{(m)}, \tag{3.1b}$$

respectively, for nonstiffly and stiffly accurate correctors (cf. (1.4)).

By virtue of the diagonal structure of D, the iterated method (3.1) is suitable for use on parallel processors because in each iteration the components of $\mathbf{Y}^{(j)}$ can be computed in parallel.

TABLE 2.1

Summary of characteristics of IRK *methods.*

Method	Stages	Order p	Stage order r	Stability	Stiffly accurate	Reference
Gauss-Legendre	k	$2k$	k	A-stable for all k	no	Butcher [7]
Lobatto IIIA	$k+1$	$2k$	$k+1$	A-stable for all k	yes	Dekker and Verwer [9]
Radau IIA	k	$2k-1$	k	L-stable for all k	yes	Butcher [7]
Newton-Cotes	$k+1$	$2[(k+2)/2]$	$k+1$	A-stable for $k \leqq 8$	yes	Watts and Shampine [23]
Lagrange	$k+1$	$k+1$	$k+1$	Strongly A-stable	yes	For $k \leqq 4$ see § 4

There are many possibilities for choosing the matrix D which we summarize below:

(i) $D = O$: this is the most simple choice and yields an *explicit* iteration method (fixed point or functional iteration). This approach was followed in Nørsett and Simonsen [20], Lie [18], van der Houwen and Sommeijer [12], and Burrage [3]. These papers deal with the iteration of implicit methods for solving *nonstiff* ODEs. In the case of *stiff* ODEs, we should use matrices $D \neq O$.

(ii) D is such that for a *prescribed* number of iterations the method has favourable stability characteristics like A-stability or L-stability. This approach was followed in van der Houwen, Sommeijer, and Couzy [13], where the corrector only serves for providing its order of accuracy. In fact, it was shown that one may even use *explicit* correctors and still can obtain A- and L-stability after the particular number of iterations and a suitable choice of the matrix D.

(iii) $D = \text{diag}(A\mathbf{e})$ or $D = \text{diag}(A)$: this choice leads to nonlinear Jacobi-type iteration. The few experiments we performed revealed that the convergence is rather poor, so that we dropped this option.

(iv) D is such that the *nonstiff* components in the iteration error are strongly damped. This type of diagonal iteration will be called *nonstiff iteration.* Nonstiff iteration can be achieved by minimizing the spectral radius of the matrix $A - D$ (see § 3.2). A large number of experiments showed that this is not the way to proceed, at least not in the case of the one-step initial approximations to $\mathbf{Y}_{n+1}$ used in this paper.

(v) D is such that the stability function $R_m(z)$ of the iterated method rapidly converges to the stability function $R_{\text{corr}}(z)$ of the corrector. Hence, the corrector not only serves for providing its order of accuracy as in [13], but the iterated method also reflects the (assumed) nice stability properties of the corrector. Within this "stability function approach" there are various approaches:

- $D^{-1}\mathbf{c} = A^{-1}\mathbf{c}$: this relation uniquely defines D provided that A is nonsingular. As observed by Hundsdorfer [15], such matrices D imply that the stability functions of the corrector and of the iterated method are identical at infinity. Although a few first experiments did not yet show satisfactory results, this option should be investigated more closely (see [4]).
- Minimization of the spectral radius of the matrix $I - D^{-1}A$. This choice implies that R_m converges fast to R_{corr} at infinity, but, at the same time, it also strongly damps the stiff components of the iteration error. This type of diagonal iteration will be called *stiff iteration.* It is the approach adopted in the present paper (see § 3.3). Our experiments in § 5 reveal that stiff iteration is suited for suppressing the phenomenon of order reduction within a few iterations, and in this respect, the methods of this paper perform much better than the methods proposed in [13].
- Other options as suggested by one of the referees, where some norm of $I - D^{-1}A$ is minimized rather than the spectral radius, or where $R_m - R_{\text{corr}}$ is minimized along the negative z-axis (or larger portions of the left halfplane), has not yet been tested and may turn out to be still more effective.

(vi) D is such that the lower order error terms in the truncation error are minimized. Since after a finite number of iterations the iterated method (3.1) formally is still a DIRK method and therefore suffers from order reduction, such an approach directly attacks the source for order reduction. This topic will also be considered in [4].

The approach of stiff iteration followed in this paper seems to be rather effective. However, by no means we do claim that this is the best way to proceed. In [4] we shall present more firm theoretical and experimental evidence of the merits of the various approaches for choosing the matrix D.

3.1. Computational costs. Each step of the (outer) iteration method (3.1a) requires the solution of a diagonally implicit relation. In order to solve this relation, we apply Newton iteration (inner iteration). There are various possibilities for starting the iteration method (3.1a) and the Newton iteration method, and for choosing the Jacobian matrix $J := \partial f/\partial y$ needed in the Newton iteration process. Obvious choices are listed in Table 3.1.

All possible combinations are equally expensive because the values of $f(t_n, y_n)$, $f(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}^{(j-1)})$ and diag (J) are anyhow needed. The first-order approximations will reduce the magnitude of the smooth error components (low frequencies) more than the zero-order approximations do, but, unlike the zero-order approximations, they will also introduce stiff error components in the case of stiff differential equations. This particularly applies to the Jacobian matrix and the initial inner iterate because these approximations are needed in each outer iteration. Therefore, we shall only consider zero-order approximations to the Jacobian matrix and to the initial inner iterate (notice that in the case of systems of equations, the matrix J becomes a block-diagonal matrix). Furthermore, our experiments revealed that using zero-order approximations for the initial outer iterate is more robust than the above first-order approximations, and yields comparable accuracies. However, it should be observed that the topic of choosing suitable initial approximations to the stage vector (including multistep approximations in order to reduce the number of iterations) is extremely important and needs further research. Burrage [3] discussed this topic in the case of a general class of *explicit* predictor-corrector methods for nonstiff problems. His approach may be used to study initial approximations in the case of diagonally implicit predictor-corrector methods for stiff problems.

By performing m iterations, the method (3.1) may be considered as a DIRK method with $mk+1$ stages, of which one stage is explicit and the other mk stages are diagonally implicit. In fact, we may represent the method by the Butcher array:

(3.1′)

$j=0$	0								
$j=1$	$\mathbf{c}-D\mathbf{e}$	D							
$j=2$	$\mathbf{a}$	$A-D$	D						
$j=3$	$\mathbf{a}$	O	$A-D$	D					
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$			
$j=m$	$\mathbf{a}$	O		$\cdots$		O	$A-D$	D	
	b_0	$\mathbf{0}^T$		$\cdots$		$\mathbf{0}^T$	$\mathbf{0}^T$	$\mathbf{b}^T$	(nonstiffly accurate correctors)
	$\mathbf{e}_k^T\mathbf{a}$	$\mathbf{0}^T$		$\cdots$		$\mathbf{0}^T$	$\mathbf{e}_k^T(A-D)$	$\mathbf{e}_k^T D$	(stiffly accurate correctors)

Since each iteration step in (3.1a) essentially requires the "wall clock time" involved in evaluating one component of $f(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}^{(j-1)})$ and solving one system of dimension d, we conclude that, effectively, the work involved in performing one step by the DIRK

TABLE 3.1

Starting the inner and outer iteration processes.

Order of approximation	0	1
Jacobian matrix	diag $[J(\mathbf{e}t_n, \mathbf{e}y_n)]$	diag $[J(\mathbf{e}t_n + \mathbf{c}h, y_n\mathbf{e} + h\mathbf{c}f(t_n, y_n))]$
Initial iterate in (3.1a)	$\mathbf{Y}^{(0)} = y_n\mathbf{e}$, $\mathbf{t}^{(0)} = \mathbf{e}t_n$	$\mathbf{Y}^{(0)} = y_n\mathbf{e} + h\mathbf{c}f(t_n, y_n)$, $\mathbf{t}^{(0)} = \mathbf{e}t_n + \mathbf{c}h$
Initial Newton iterate	$\mathbf{Y}^{(j-1)}$	$y_n\mathbf{e} + h\mathbf{a}f(t_n, y_n) + hAf(\mathbf{e}t_n + \mathbf{c}h, \mathbf{Y}^{(j-1)})$

method (3.1′) consists of

$$\begin{aligned}&(\text{evaluation of } f \text{ and } J)+(\text{LU decomposition of } I-d_jhJ)\\&\quad+m[\text{evaluation of } f+N(\text{forward/backward substitution}+\text{evaluation of } f)].\end{aligned}\tag{3.2}$$

In this expression N is defined by

$$N:=\frac{N_1+N_2+\cdots+N_m}{m},\tag{3.3}$$

with N_j denoting the number of Newton iterations for computing that component of $\mathbf{Y}^{(j)}$ which requires the largest number of Newton iterations. Usually, the m iterations are the most expensive part of the total effort per step, and therefore we shall say that a DIRK method has m *effective* or *sequential* stages if there are m diagonally implicit systems to be solved.

3.1.1. Comparison with conventional DIRK methods. In the experiments reported in this paper, we used the stopping criterion that the Newton correction should be about the machine precision which is for our computer 10^{-14}. It turned out that N_j rapidly decreases with j which can be explained by observing that the initial iterate for starting the next inner iteration becomes more accurate when j increases. This is an advantage when compared with conventionally constructed DIRK methods already available in the literature (such DIRK methods will be indicated by "conventional" DIRK methods), because, for conventional DIRK methods, the number of Newton iterations for solving the implicit relations in the successive stages do, in general, not decrease.

In order to appreciate the computational costs of DIRK methods of type (3.1′), we should compare m with the number of sequential stages of conventional DIRK methods. In Table 3.2, the characteristics of such DIRK methods are listed together with the PARK and PDIRK methods derived in [16] and [13].

3.2. Order of accuracy. In order to analyse the order of accuracy of the iterated method (3.1), let $\mathbf{Y}(t_{n+1})$ denote the vector with components $y(t_n+c_ih)$ where y is the locally exact solution of (1.1). Then, in first approximation, we obtain

$$\begin{aligned}\mathbf{Y}(t_{n+1})-\mathbf{Y}^{(j)}&=[\mathbf{Y}(t_{n+1})-\mathbf{Y}_{n+1}]+[\mathbf{Y}_{n+1}-\mathbf{Y}^{(j)}]\\&=[\mathbf{Y}(t_{n+1})-\mathbf{Y}_{n+1}]+Z[\mathbf{Y}_{n+1}-\mathbf{Y}^{(j-1)}]\\&=[\mathbf{Y}(t_{n+1})-\mathbf{Y}_{n+1}]+Z^j[\mathbf{Y}_{n+1}-\mathbf{Y}^{(0)}],\qquad j=1,2,\cdots,\end{aligned}\tag{3.4a}$$

TABLE 3.2

Summary of characteristics of DIRK, PARK, *and* PDIRK *methods of order* $p\geqq 3$.

Order	Stage order	Sequential stages	Processors	Stability	Reference
$p=3$	1	$p-1$	1	A-stable	Nørsett [19]
$p=3$	2	$p-1$	1	Strongly A-stable	Crouzeix [8]
$p=4$	1	$p-1$	1	A-stable	Crouzeix [8], Alexander [1]
$p=4$	1	$p-2$	2	L-stable	Iserles and Nørsett [16]
$p=3,4,5$	1	$p-1$	$[(p+1)/2]$	Strongly A-stable	van der Houwen et al. [13]
$p=6,7$	1	$p-1$	$[(p+1)/2]$	Strongly $A(\alpha)$-stable	ibid
$p\leqq 6, p=8$	1	p	$[(p+1)/2]$	L-stable	ibid
$p=7,8,10$	1	$p+1$	$[(p+1)/2]$	L-stable	ibid

where Z is the iteration matrix defined by

$$Z = Z(hDJ) := [I - hDJ]^{-1}[AD^{-1} - I]hDJ, \tag{3.4b}$$

with J again denoting the Jacobian matrix of f.

Let r be the stage order of the corrector (1.3), then (cf. (2.1))

$$\mathbf{Y}(t_{n+1}) - \mathbf{Y}_{n+1} = O(h^{r+1}).$$

Since $Z = O(h)$ and $\mathbf{Y}_{n+1} - \mathbf{Y}^{(0)} = O(h)$, the local error of the stage vectors satisfy the order relation

$$\mathbf{Y}(t_{n+1}) - \mathbf{Y}^{(j)} = O(h^{r+1}) + O(h^{j+1}), \tag{3.5}$$

so that, after m iterations, (3.1) defines a method in which $\mathbf{Y}^{(m)}$ approximates $\mathbf{Y}(t_{n+1})$ with order $r^* = \min\{r, m\}$. We shall say that (3.1) has *stage order* r^* (although formally, when (3.1) is considered as a DIRK method, its stage order is only 1). Thus, the optimal stage-order methods, that is the methods based on the Lagrange methods as defined above, have stage order $r^* = k+1$ provided that at least $m = k+1$ iterations are performed.

In order to get more insight into the rate of convergence of the iteration process (3.1), we consider the test equation

$$\frac{dy(t)}{dt} = \lambda y(t), \tag{3.6}$$

where λ runs through the spectrum $\Lambda(J)$ of J. The matrix Z assumes the form

$$Z = zD[I - zD]^{-1}[D^{-1}A - I] = z[I - zD]^{-1}[A - D], \qquad z := \lambda h. \tag{3.7}$$

Suppose that J has a complete eigensystem, and let us call the eigenvectors of hJ corresponding to the eigenvalues of large and small modulus, respectively *stiff* and *nonstiff* components. From (3.7) we see that for the nonstiff components (i.e., corresponding to small values of $|z|$) the matrix Z behaves approximately as $z[A - D]$. Hence, these components in the iteration error are strongly damped if the matrix $A - D$ has eigenvalues of small magnitude. Thus, rapid convergence of the nonstiff components is obtained by minimizing the spectral radius of $A - D$. However, as already remarked above, such a *nonstiff* iteration process gives a poor overall convergence. Alternatively, for the stiff components (i.e., corresponding to large values of $|z|$), the matrix Z behaves as $-D^{-1}[A - D]$. Hence, a strong damping of these components requires the minimization of the spectral radius of $I - D^{-1}A$, leading to *stiff* iteration. In the following section, we shall see that this condition also plays a role in the stability of the iterated method.

3.3. Stability. One may argue that there is no reason to continue the iteration process after $m = r$ iterations, because the stage errors of the corrector and of the iterated method have become of the same order in h and may therefore be expected to be of comparable magnitude. However, there is no guarantee that after $m = r$ iterations the *stability properties* of (1.3′) are also comparable with those of the corrector. This brings us to consider the stability of the DIRK method (3.1′). In order to see how the stability depends on the number of iterations m, we apply the method to the test equation (3.6), so that (3.1a) reduces to

$$\mathbf{Y}^{(m)} = \left(Z^m \mathbf{e} + [I - Z]^{-1}[I - Z^m][I - zD]^{-1}(\mathbf{e} + z\mathbf{a})\right) y_n.$$

We shall discuss the stability of iterating a nonstiffly accurate and a stiffly accurate corrector separately.

3.3.1. Nonstiffly accurate correctors. If y_{n+1} is computed by means of the formula

$$y_{n+1} = [1 + zb_0]y_n + z\mathbf{b}^T\mathbf{Y}^{(m)},$$

then it can be expressed as

$$y_{n+1} = \left[1 + zb_0 + z\mathbf{b}^T\left(Z^m\mathbf{e} + [I - Z]^{-1}[I - Z^m][I - zD]^{-1}(\mathbf{e} + z\mathbf{a})\right)\right]y_n, \tag{3.8}$$

so that the stability function is given by

$$R_m(z) := 1 + zb_0 + z\mathbf{b}^T\left(Z^m\mathbf{e} + [I - Z]^{-1}[I - Z^m][I - zD]^{-1}(\mathbf{e} + z\mathbf{a})\right). \tag{3.9}$$

It is easily verified that this function can be written in the form

$$R_m(z) := 1 + zb_0 + z\mathbf{b}^T[I - zA]^{-1}(\mathbf{e} + z\mathbf{a}) - z^2\mathbf{b}^TZ^m[I - zA]^{-1}(A\mathbf{e} + \mathbf{a}).$$

Assuming that the stage order of the corrector is at least one, we may set $A\mathbf{e} + \mathbf{a} = \mathbf{c}$ (see (2.2)), so that

$$R_m(z) := R_{\text{corr}}(z) - z^2\mathbf{b}^T[Z(zD)]^m[I - zA]^{-1}\mathbf{c}, \tag{3.10}$$

where R_{corr} denotes the stability function of the corrector given by (2.7). Finally, on substitution of (3.7) into (3.10) we obtain

$$R_m(z) = R_{\text{corr}}(z) - z^{m+2}\mathbf{b}^T\left([I - zD]^{-1}[A - D]\right)^m[I - zA]^{-1}\mathbf{c}. \tag{3.11}$$

From this expression we can derive the convergence behaviour of R_m to R_{corr} for large values of $|z|$:

$$R_m(z) = R_{\text{corr}}(z) + z\mathbf{b}^T[I - D^{-1}A]^mA^{-1}\mathbf{c} \quad \text{as } |z| \to \infty,$$

showing that for any fixed m the stability function becomes unbounded as $|z|$ tends to infinity, unless the matrix D is such that

$$\mathbf{b}^T[I - D^{-1}A]^mA^{-1}\mathbf{c} = 0.$$

Writing this equation as

$$\mathbf{b}^T[I - D^{-1}A]^{m-1}[I - D^{-1}A]A^{-1}\mathbf{c} = \mathbf{b}^T[I - D^{-1}A]^{m-1}[A^{-1}\mathbf{c} - D^{-1}\mathbf{c}] = 0,$$

we see that it can be satisfied for all m if we choose D such that [15]

$$D^{-1}\mathbf{c} = A^{-1}\mathbf{c}. \tag{3.12}$$

Unfortunately, a few first experiments showed that the performance of the corresponding method (3.1′) is not satisfactory (see § 5.3). Therefore, we conclude that diagonal iteration as defined by (3.1′) is in general not suitable for iterating nonstiffly accurate correctors and excludes the Gauss-Legendre formulas as suitable corrector methods. However, it should be remarked that by defining the initial iterate $\mathbf{Y}^{(0)}$ implicitly, rather than just setting $\mathbf{Y}^{(0)} = y_n\mathbf{e}$, the above stability problem can be avoided (cf. [13]), so that the matrix D remains available for improving the performance of the iteration process. As observed in § 3.1, the topic of finding suitable initial approximations to the stage vector in diagonally iterated RK methods deserves further research, but will not be an issue in this paper.

3.3.2. Stiffly accurate correctors. In the stiffly accurate case where y_{n+1} is computed by means of the formula

$$y_{n+1} = \mathbf{e}_k^T\mathbf{Y}^{(m)},$$

we arrive at the stability function

$$R_m(z) = R_{\text{corr}}(z) - z\mathbf{e}_k^T[Z(zD)]^m[I - zA]^{-1}\mathbf{c}, \tag{3.13}$$

where R_{corr} is defined by (2.8). We may express this function in the form

$$R_m(z) = R_{\text{corr}}(z) - [\sigma_m(z)]^m, \tag{3.13'}$$

where

$$\sigma_m(z) := \left[z\mathbf{e}_k^T[Z(zD)]^m[I - zA]^{-1}\mathbf{c}\right]^{1/m}$$
$$= \left[z^{m+1}\mathbf{e}_k^T([I - zD]^{-1}[A - D])^m[I - zA]^{-1}\mathbf{c}\right]^{1/m}.$$

For fixed values of m and assuming that D has positive diagonal elements, the function $\sigma_m(z)$ is bounded for all z in the closed left halfplane. This suggests to characterize the rate of convergence of R_m to R_{corr} by means of $\sigma_m(z)$. We shall call $\sigma_m(z)$ the *convergence factor associated with z.* For example, we have

$$\sigma_m(0) = 0, \quad \sigma_m(\infty) := \left[-\mathbf{e}_k^T[Z(-\infty)]^m A^{-1}\mathbf{c}\right]^{1/m} = \left[-\mathbf{e}_k^T[I - D^{-1}A]^m A^{-1}\mathbf{c}\right]^{1/m}. \tag{3.14}$$

Ideally, in order to get fast convergence of the stability function $R_m(z)$ to that of the corrector, we should try to minimize $\sigma_m(z)$ in the closed left halfplane. However, since in actual computation m is determined by some error criterion, we do not know m in advance, so that such an approach may be unattractive, particularly for larger values of k where more values of m have to be considered. Nevertheless, in a future paper [4], this possibility will be studied more closely in order to get further insight into how crucial the choice of D really is.

Another possibility is the minimization of $\sigma_m(z)$ for the highly stiff components (large values of $|z|$), because (3.14) shows that $\sigma_m(z)$ is already small for the nonstiff components. The most simple way to achieve this determines D according to (3.12), so that $\sigma_m(\infty)$ vanishes for all m [15]. In the experiments done so far, the convergence of the corresponding iteration process (3.1) is not satisfactory.

However, by choosing the matrix D, for a given corrector, such that the spectral radius of $Z(-\infty) = I - D^{-1}A$ is minimized over all possible diagonal matrices D with positive entries, we obtained a satisfactory convergence behaviour in a large number of experiments (see § 5, and the Appendix to [14]). The better convergence may be explained by observing that in this way, not only the value of $\sigma_m(\infty)$ is expected to be small (cf. (3.14)), but as already shown in § 3.2, at the same time the stiff components in the iteration error are strongly damped.

Together with the computation of the matrix D (cf. § 4), we computed, as a posteriori test, for a few values of m the "worst" convergence factor defined by

$$\sigma_m := \max_{\text{Re } z \le 0} |\sigma_m(z)|. \tag{3.15}$$

Because $\sigma_m(z)$ is an analytic function in the closed left halfplane, its maximum is assumed on the boundary, i.e., on the imaginary axis.

In calculating σ_m it turned out that this quantity is larger than 1 for small values of m but rather quickly decreases to a moderate size as m increases. The values of σ_m show by what factor the (maximal) difference between the two stability functions is reduced in each iteration if we continue to iterate when the stage order of the corrector has been reached. Due to the fact that $\sigma_m > 1$ for small m, it is likely that the corresponding iterated method is not A-stable. On the other hand, assuming that the iteration process (3.1) is convergent, we know that $[\sigma_m(z)]^m \to 0$ for $m \to \infty$, i.e., $R_m(z)$ converges to the A-acceptable stability function $R_{\text{corr}}(z)$. Therefore, it is of interest to know the minimal value of m such that $R_m(z)$ is A-acceptable for all m equal to or larger than this minimal value. This for the iteration process critical number of iterations will be denoted by m_{crit}. Evidently, the value of m_{crit} is expected to be large if the

corrector is not strongly A-stable. In order to illustrate this, we considered the methods using weakly A-stable Newton-Cotes and Lobatto IIIA correctors (cf. Table 2.1) with minimized spectral radius of $I-D^{-1}A$. We verified that (for z in the closed left halfplane) the value of $\max|R_m(z)|\downarrow 1$ as $m\to\infty$, so that A-stability is only obtained in the limit. Hence, the Lobatto IIIA and the Newton-Cotes formulas seem to be less suitable as corrector methods. For the strongly A-stable Lagrange correctors and the L-stable Radau IIA correctors however, we found modest values of m_{crit}, so that after a few iterations the resulting method is already A-stable (see § 4).

4. Construction of methods. In this section, we consider a number of stiffly accurate correctors and we will construct the corresponding matrices D for use on two-, three- or four-processor computers (i.e., methods of dimension $k=2, 3, 4$).

For $k=2$, we shall give a rather detailed derivation, because in this case, it is still possible to construct suitable matrices D analytically. We derive matrices D for correctors of Newton-Cotes, Lobatto IIIA, strongly A-stable Lagrange, Radau IIA, and Gauss-Legendre type. The Gauss-Legendre method is not stiffly accurate, and therefore not suitable for diagonal iteration of type (3.1′), but it is included to demonstrate its unstable performance. For $k>2$, we resort to numerical search methods for finding suitable matrices D. Here, we refrained from looking for D matrices for the Gauss-Legendre method because of the rather poor two-processor results. In § 4.4 a summary of the main properties of the various methods is given.

It may be of interest to note that in our numerical search for strongly A-stable correctors we encountered strong numerical evidence for the following conjecture.

Conjecture. A necessary condition for a stiffly accurate Lagrange method as defined in § 1.2 to be strongly A-stable is

$$\sum_{j=1}^{k} c_j > \frac{k+1}{2}. \qquad \square$$

In order to save space, the correctors are presented by means of the matrix A and the vectors $\mathbf{a}$ and $\mathbf{c}$, and the iterated versions by only giving the matrix D, because, together with the corrector, D completely defines the iterated method. In the following, we only consider *stiff* iteration, that is, the construction of D will always be based on the minimization of the spectral radius $\rho(I-D^{-1}A)$ of the matrix $I-D^{-1}A$. If the entries of D are not exact (i.e., for $k\geqq 3$), then they are approximated by rational expressions. In addition to D, we present the values of $\rho(I-D^{-1}A)$, the range for σ_m with $r\leqq m\leqq 10$, the corresponding interval I_σ on the imaginary axis where the maxima are assumed, and the value of m_{crit} are given (cf. § 3.3.2). Finally, the stage and step point orders of the method are denoted by r and p, respectively.

4.1. Two-processor methods.

4.1.1. Lagrange methods. Let us first consider two-dimensional Lagrange methods ($k=2$) satisfying the condition (1.4). The stage-order conditions (2.2) can be solved for $r=3$ and yield the stiffly accurate Lagrange method

$$A=\frac{1}{6(1-c)}\begin{pmatrix} c(3-2c) & -c^3 \\ c^{-1} & 2-3c \end{pmatrix}, \qquad \mathbf{a}=\frac{1}{6(1-c)}\begin{pmatrix} 3c-4c^2+c^3 \\ -c^{-1}+4-3c \end{pmatrix},$$

$$\mathbf{c}=\begin{pmatrix} c \\ 1 \end{pmatrix}, \qquad p=r=3, \tag{4.1a}$$

where c is a free parameter (recall that $p=4$ if $c=\frac{1}{2}$). An elementary calculation shows

that the stability function of (4.1a) is given by

$$R(z)=\frac{6+2(2-c)z+(1-c)z^2}{6-2(c+1)z+cz^2}. \tag{4.2}$$

This function is A-acceptable for $c\geqq\frac{1}{2}$ and strongly A-acceptable for $c>\frac{1}{2}$.

Next, we determine the matrix D in (3.1). It is convenient to write

$$D=\frac{1}{6(1-c)}\begin{pmatrix}1/\delta_1 & 0\\ 0 & 1/\delta_2\end{pmatrix},$$

so that

$$I-D^{-1}A=\begin{pmatrix}1-c(3-2c)\delta_1 & c^3\delta_2\\ -c^{-1}\delta_1 & 1-(2-3c)\delta_2\end{pmatrix}.$$

The eigenvalues of $I-D^{-1}A$ satisfy the equation

$$\mu^2-S\mu+P=0, \qquad S:=2-c(3-2c)\delta_1-(2-3c)\delta_2,$$

$$P:=[1-c(3-2c)\delta_1][1-(2-3c)\delta_2]+c^2\delta_1\delta_2.$$

By setting $S=P=0$ we achieve that $\rho(I-D^{-1}A)$ vanishes. The parameters δ_1 and δ_2 then satisfy the equations

$$c(3-2c)\delta_1+(2-3c)\delta_2=2, \qquad [1-c(3-2c)\delta_1]^2-c^2\delta_1\delta_2=0,$$

leading to

$$\delta_1=\frac{1+Q}{c(3-2c)}, \qquad \delta_2=\frac{1-Q}{2-3c}, \qquad Q:=\frac{\pm\sqrt{6c}}{6(1-c)},$$

so that the matrix D is given by

$$D=\frac{1}{6(1-c)}\begin{pmatrix}c(3-2c)/(1+Q) & 0\\ 0 & (2-3c)/(1-Q)\end{pmatrix}, \quad \rho(I-D^{-1}A)=0. \tag{4.1b}$$

The iterated Lagrange method with zero convergence factor at infinity is completely determined by the corrector (4.1a) and the matrix (4.1b).

For $c=\frac{1}{2}$ we derive from (4.1a) the Newton-Cotes corrector (with $p=4$ and $r=3$)

$$A=\frac{1}{24}\begin{pmatrix}8 & -1\\ 16 & 4\end{pmatrix}, \qquad \mathbf{a}=\frac{1}{24}\begin{pmatrix}5\\ 4\end{pmatrix}, \qquad \mathbf{c}=\begin{pmatrix}\frac{1}{2}\\ 1\end{pmatrix}. \tag{4.3a}$$

We observe that this corrector coincides with the three-stage Lobatto IIIA method. The stability function R of (4.3a) reduces to the $(2,2)$ Padé approximation to the exponential function. Recall that R is A-acceptable but not *strongly* A-acceptable. From (4.1b) we obtain the matrix

$$D=\frac{1}{2}\begin{pmatrix}2/(3+\sqrt{3}) & 0\\ 0 & 1/(3-\sqrt{3})\end{pmatrix}, \qquad \rho(I-D^{-1}A)=0,$$

$$\sigma_m\in[0.21,0.36], \qquad I_\sigma=[3.9i,5.1i], \qquad m_{\text{crit}}=\infty. \tag{4.3b}$$

A natural question now is, whether it is possible to choose c such that the stability is improved. Unfortunately, (4.1a) shows that it is not possible to achieve L-stability

(which would require $c=1$), but strong A-stability is obtained for $c>\frac{1}{2}$. For example, by choosing $c=\frac{3}{4}$ we have $R(\infty)=\frac{1}{3}$. The corresponding Lagrange method is defined by

$$A=\frac{1}{288}\begin{pmatrix}216 & -81\\ 256 & -48\end{pmatrix},\qquad \mathbf{a}=\frac{1}{288}\begin{pmatrix}81\\ 80\end{pmatrix},\qquad \mathbf{c}=\begin{pmatrix}\frac{3}{4}\\ 1\end{pmatrix} \tag{4.4a}$$

for which $p=r=3$. The iterated version is defined by

$$D=\frac{1}{12}\begin{pmatrix}9/(\sqrt{2}+1) & 0\\ 0 & 2/(\sqrt{2}-1)\end{pmatrix},\qquad \rho(I-D^{-1}A)=0,$$
$$\sigma_m\in[0.21,\,0.33],\qquad I_\sigma=[3.2i,\,4.1i],\qquad m_{\text{crit}}=2. \tag{4.4b}$$

4.1.2. Gauss and Radau methods. As reference methods for our numerical experiments, we take the conventional two-stage Gauss–Legendre and Radau IIA methods. The Gauss–Legendre corrector, and its iterated version is defined by

$$A=\frac{1}{12}\begin{pmatrix}3 & 3-2\sqrt{3}\\ 3+2\sqrt{3} & 3\end{pmatrix},\qquad \mathbf{a}=\mathbf{0},\qquad b_0=0,$$
$$\mathbf{b}=\frac{1}{2}\mathbf{e},\qquad \mathbf{c}=\frac{1}{12}\begin{pmatrix}6-2\sqrt{3}\\ 6+2\sqrt{3}\end{pmatrix},\qquad p=4,\qquad r=2, \tag{4.5a}$$

$$D=\frac{1}{6}\begin{pmatrix}1 & 0\\ 0 & 3\end{pmatrix},\qquad \rho(I-D^{-1}A)=0. \tag{4.5b}$$

The Radau IIA-based method is given by

$$A=\frac{1}{12}\begin{pmatrix}5 & -1\\ 9 & 3\end{pmatrix},\qquad \mathbf{a}=\mathbf{0},\qquad b_0=0,\qquad \mathbf{b}^T=\mathbf{e}_2^TA,$$
$$\mathbf{c}=\begin{pmatrix}\frac{1}{3}\\ 1\end{pmatrix},\qquad p=3,\qquad r=2, \tag{4.6a}$$

$$D=\frac{1}{30}\begin{pmatrix}20-5\sqrt{6} & 0\\ 0 & 12+3\sqrt{6}\end{pmatrix},\qquad \rho(I-D^{-1}A)=0,$$
$$\sigma_m\in[0.27,\,0.35],\qquad I_\sigma=[2.6i,\,3.7i],\qquad m_{\text{crit}}=1. \tag{4.6b}$$

4.2. Three-processor methods.

4.2.1. Newton–Cotes method. For $k=3$ and equidistant abscissas the corrector is given by

$$A=\frac{1}{72}\begin{pmatrix}19 & -5 & 1\\ 32 & 8 & 0\\ 27 & 27 & 9\end{pmatrix},\qquad \mathbf{a}=\frac{1}{72}\begin{pmatrix}9\\ 8\\ 9\end{pmatrix},$$
$$b_0=\mathbf{e}_3^T\mathbf{a},\qquad \mathbf{b}^T=\mathbf{e}_3^TA,\qquad \mathbf{c}=\frac{1}{3}\begin{pmatrix}1\\ 2\\ 3\end{pmatrix} \tag{4.7a}$$

with $p = r = 4$, and with A-acceptable stability function (see Watts and Shampine [23]). By a numerical search we found the matrix

$$D = \begin{pmatrix} \frac{897}{7303} & 0 & 0 \\ 0 & \frac{2485}{10968} & 0 \\ 0 & 0 & \frac{8980}{27627} \end{pmatrix}, \qquad \rho(I - D^{-1}A) \approx 0.01,$$

(4.7b)

$$\sigma_m \in [0.49, 0.77], \qquad I_\sigma = [7.1i, 8.4i], \qquad m_{\text{crit}} = \infty.$$

4.2.2. Lobatto IIIA method. For $k=3$ and Lobatto abscissas the corrector is given by

$$A = \frac{1}{120}\begin{pmatrix} 25-\sqrt{5} & 25-13\sqrt{5} & -1+\sqrt{5} \\ 25+13\sqrt{5} & 25+\sqrt{5} & -1-\sqrt{5} \\ 50 & 50 & 10 \end{pmatrix}, \qquad \mathbf{a} = \frac{1}{120}\begin{pmatrix} 11+\sqrt{5} \\ 11-\sqrt{5} \\ 10 \end{pmatrix},$$

(4.8a)

$$b_0 = \mathbf{e}_3^T\mathbf{a}, \qquad \mathbf{b}^T = \mathbf{e}_3^T A, \qquad \mathbf{c} = \frac{1}{10}\begin{pmatrix} 5-\sqrt{5} \\ 5+\sqrt{5} \\ 10 \end{pmatrix},$$

with $p = 6$ and $r = 4$, and with A-acceptable stability function (see Dekker and Verwer [9]). The iterated version is generated by

$$D = \begin{pmatrix} \frac{2661}{5542} & 0 & 0 \\ 0 & \frac{754}{6891} & 0 \\ 0 & 0 & \frac{1567}{9771} \end{pmatrix}, \qquad \rho(I - D^{-1}A) \approx 0.0043,$$

(4.8b)

$$\sigma_m \in [0.52, 0.88], \qquad I_\sigma = [8.9i, 10i], \qquad m_{\text{crit}} = \infty.$$

4.2.3. Lagrange method. By keeping c_1 and c_2 free, we can construct *strongly* A-stable methods with stage order four. It can be shown that the stability function is A-acceptable for $c_1 + c_2 = 1$ and strongly A-acceptable for $c_1 + c_2 > 1$. A numerical search produced the block point vector $\mathbf{c} = (\frac{7}{12}, \frac{5}{6}, 1)^T$ for which parameter values of acceptable magnitude and a damping factor $|R(\infty)| \approx 0.143$ are obtained. The corresponding corrector reads

$$A = \frac{1}{120960}\begin{pmatrix} 98392 & -81634 & 31213 \\ 112000 & -61600 & 28000 \\ 110592 & -48384 & 36288 \end{pmatrix}, \qquad \mathbf{a} = \frac{1}{120960}\begin{pmatrix} 22589 \\ 22400 \\ 22464 \end{pmatrix},$$

(4.9a)

$$b_0 = \mathbf{e}_3^T\mathbf{a}, \qquad \mathbf{b}^T = \mathbf{e}_3^T A, \qquad \mathbf{c} = \frac{1}{12}\begin{pmatrix} 7 \\ 10 \\ 12 \end{pmatrix}$$

with $p = r = 4$. The iterated method is generated by

$$D = \begin{pmatrix} \frac{2246}{10669} & 0 & 0 \\ 0 & \frac{2537}{8794} & 0 \\ 0 & 0 & \frac{3026}{8923} \end{pmatrix}, \qquad \rho(I - D^{-1}A) \approx 0.011,$$

(4.9b)

$$\sigma_m \in [0.49, 0.69], \qquad I_\sigma = [5.1i, 6.2i], \qquad m_{\text{crit}} = 3.$$

4.2.4. Radau method. The 3-stage Radau IIA corrector is defined by [7]

$$A=\begin{pmatrix} \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{pmatrix}, \tag{4.10a}$$

$$\mathbf{a}=\mathbf{0}, \quad \mathbf{b}_0=0, \quad \mathbf{b}^T=\mathbf{e}_3^T A, \quad \mathbf{c}=A\mathbf{e}$$

with $p=5$, $r=3$, and L-acceptable stability function. The matrix D is given by

$$D=\begin{pmatrix} \frac{4365}{13624} & 0 & 0 \\ 0 & \frac{1032}{7373} & 0 \\ 0 & 0 & \frac{1887}{5077} \end{pmatrix}, \qquad \rho(I-D^{-1}A)\approx 0.0047, \tag{4.10b}$$

$$\sigma_m\in[0.52, 1.0], \qquad I_\sigma=[6.6i, 9.3i], \qquad m_{\text{crit}}=5.$$

4.3. Four-processor methods.

4.3.1. Newton–Cotes method. For $k=4$ and equidistant abscissas the corrector is given by

$$A=\frac{1}{2880}\begin{pmatrix} 646 & -264 & 106 & -19 \\ 992 & 192 & 32 & -8 \\ 918 & 648 & 378 & -27 \\ 1024 & 384 & 1024 & 224 \end{pmatrix}, \qquad \mathbf{a}=\frac{1}{2880}\begin{pmatrix} 251 \\ 232 \\ 243 \\ 224 \end{pmatrix}, \tag{4.11a}$$

$$b_0=\mathbf{e}_4^T\mathbf{a}, \qquad \mathbf{b}^T=\mathbf{e}_4^T A, \qquad \mathbf{c}=\frac{1}{4}\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

with $p=6$, $r=5$, and with A-acceptable stability function. A numerical search did not produce a better matrix D than

$$D=\begin{pmatrix} \frac{992}{10759} & 0 & 0 & 0 \\ 0 & \frac{1365}{8107} & 0 & 0 \\ 0 & 0 & \frac{2709}{11281} & 0 \\ 0 & 0 & 0 & \frac{1717}{5549} \end{pmatrix}, \qquad \rho(I-D^{-1}A)\approx 0.1, \tag{4.11b}$$

$$\sigma_m\in[0.76, 1.04], \qquad I_\sigma=[8.7i, 11.8i], \qquad m_{\text{crit}}=\infty.$$

4.3.2. Lobatto IIA method. For $k=4$ and Lobatto abscissas the corrector is given by

$$A=\begin{pmatrix} \frac{343-9\sqrt{21}}{2520} & \frac{392-96\sqrt{21}}{2205} & \frac{343-69\sqrt{21}}{2520} & \frac{-21+3\sqrt{21}}{1960} \\ \frac{392+105\sqrt{21}}{2880} & \frac{8}{45} & \frac{392-105\sqrt{21}}{2880} & \frac{3}{320} \\ \frac{343+69\sqrt{21}}{2520} & \frac{392+96\sqrt{21}}{2205} & \frac{343+9\sqrt{21}}{2520} & \frac{-21-3\sqrt{21}}{1960} \\ \frac{49}{180} & \frac{16}{45} & \frac{49}{180} & \frac{1}{20} \end{pmatrix}, \tag{4.12a}$$

$$\mathbf{a}=\begin{pmatrix}\frac{119+3\sqrt{21}}{1960}\\ \frac{13}{320}\\ \frac{119-3\sqrt{21}}{1960}\\ \frac{1}{20}\end{pmatrix},\qquad b_0=\mathbf{e}_4^T\mathbf{a},\qquad \mathbf{b}^T=\mathbf{e}_4^TA,\qquad \mathbf{c}=\begin{pmatrix}\frac{7-\sqrt{21}}{14}\\ \frac{1}{2}\\ \frac{7+\sqrt{21}}{14}\\ 1\end{pmatrix}$$

with $p=8$, $r=5$, and with A-acceptable stability function. A numerical search produced the matrix

(4.12b)
$$D=\begin{pmatrix}\frac{2964}{9943} & 0 & 0 & 0\\ 0 & \frac{1875}{10334} & 0 & 0\\ 0 & 0 & \frac{608}{9403} & 0\\ 0 & 0 & 0 & \frac{3799}{23419}\end{pmatrix},\qquad \rho(I-D^{-1}A)\approx 0.021,$$

$$\sigma_m\in[0.87,\,1.32],\qquad I_\sigma=[15.4i,\,19i],\qquad m_{\text{crit}}=\infty.$$

4.3.3. Lagrange method. Numerically, we found that the stability function is A-acceptable for $c_1+c_2+c_3=\frac{3}{2}$ and strongly A-acceptable for $c_1+c_2+c_3>\frac{3}{2}$. For $\mathbf{c}=(\frac{1}{6},\frac{7}{12},\frac{11}{12},1)^T$ we obtained parameter values of acceptable magnitude and a damping factor $|R(\infty)|\approx 0.325$. The corresponding corrector with $p=r=5$ reads

(4.13a)
$$A=\frac{1}{49896000}\begin{pmatrix}5452832 & -872784 & 926800 & -556248\\ 17484082 & 13296591 & -6182575 & 3486252\\ 16192946 & 22005423 & 7263025 & -1229844\\ 16232832 & 21897216 & 9676800 & 598752\end{pmatrix},$$

$$\mathbf{a}=\frac{1}{332640}\begin{pmatrix}22436\\ 6811\\ 10043\\ 9936\end{pmatrix},\qquad b_0=\mathbf{e}_4^T\mathbf{a},\qquad \mathbf{b}^T=\mathbf{e}_4^TA,\qquad \mathbf{c}=\frac{1}{12}\begin{pmatrix}2\\ 7\\ 11\\ 12\end{pmatrix}.$$

The iterated method is generated by

(4.13b)
$$D=\begin{pmatrix}\frac{5147}{38467} & 0 & 0 & 0\\ 0 & \frac{1983}{17459} & 0 & 0\\ 0 & 0 & \frac{3197}{14090} & 0\\ 0 & 0 & 0 & \frac{3086}{12339}\end{pmatrix},\qquad \rho(I-D^{-1}A)\approx 0.045,$$

$$\sigma_m\in[0.59,\,0.93],\qquad I_\sigma=[8.2i,\,11.8i],\qquad m_{\text{crit}}=6.$$

4.3.4. Radau method. The four-stage Radau IIA corrector reads

(4.14a)
$$A=\begin{pmatrix}.11299947932316 & -.04030922072352 & .02580237742034 & -.0099046765073\\ .23438399574740 & .20689257393536 & -.04785712804854 & .01604742280652\\ .21668178462325 & .40612326386737 & .18903651817006 & -.02418210489983\\ .22046221117677 & .38819346884317 & .32884431998006 & \frac{1}{16}\end{pmatrix},$$

$$\mathbf{a}=\mathbf{0},\qquad b_0=0,\qquad \mathbf{b}^T=\mathbf{e}_4^TA,\qquad \mathbf{c}=A\mathbf{e}$$

with $p=7$, $r=4$, and with L-acceptable stability function. The iterated method is generated by

$$D=\begin{pmatrix} \frac{3055}{9532} & 0 & 0 & 0 \\ 0 & \frac{531}{5956} & 0 & 0 \\ 0 & 0 & \frac{1471}{8094} & 0 \\ 0 & 0 & 0 & \frac{1848}{7919} \end{pmatrix}, \qquad \rho(I-D^{-1}A)\approx 0.024, \tag{4.14b}$$

$$\sigma_m \in [0.74, 1.31], \qquad I_\sigma = [10.0i, 17.2i], \qquad m_{\text{crit}} = 7.$$

4.4. Survey of methods. In Table 4.1, we have summarized a few characteristics of the methods derived in the preceding sections. In this table, the value of the step point order p corresponds to values of m equal to or greater than p, and the value of the stage order r corresponds to that of the corrector. From a computational point of view, the Lagrange and Radau IIA methods are the most attractive ones, because m_{crit} is relatively small. Thus, if these methods are implemented with some local error strategy for automatically estimating the number of iterations m and the stepsize h needed to meet local error tolerance, then the value of the "computational efficiency" quantity mL/h for integrating an interval of length L will not be unnecessarily large because of the development of instabilities. This observation is confirmed by the numerical experiments in § 5.4.

5. Numerical experiments. In this paragraph, the (stiff) diagonal iteration method developed above will be tested by integrating a number of stiff test problems. Section 5.1 presents these test problems. Section 5.2 compares the effective orders of Gauss-Legendre, Newton-Cotes, Lobatto IIIA, Radau IIA and Lagrange correctors, and in § 5.3, the performance of the diagonal iteration process with respect to the number of iterations is tested for a few two-processor correctors. Finally, in § 5.4, we compare the efficiency of the iterated methods with a few DIRK methods from the literature.

We recall that we only used the zero-order approximations to the Jacobian matrix and to the initial inner and outer iterates. In the tables of results, the accuracy of the

TABLE 4.1

Main characteristics of diagonally iterated IRK *methods.*

Method	p	r	k	$\rho(I-D^{-1}A)$	σ_m-range ($r \leqq m \leqq 10$)	m_{crit}
Newton-Cotes (4.3)	4	3	2	0	[0.21, 0.36]	∞
Lagrange (4.4)	3	3	2	0	[0.21, 0.33]	2
Radau IIA (4.6)	3	2	2	0	[0.27, 0.35]	1
Gauss (4.5)	4	2	2	0	∞	∞
Newton-Cotes (4.7)	4	4	3	0.008	[0.49, 0.77]	∞
Lobatto IIIA (4.8)	6	4	3	0.0043	[0.52, 0.88]	∞
Lagrange (4.9)	4	4	3	0.01	[0.49, 0.69]	3
Radau IIA (4.10)	5	3	3	0.0047	[0.52, 1.0]	5
Newton-Cotes (4.11)	6	5	4	0.1	[0.76, 1.04]	∞
Lobatto IIIA (4.12)	8	5	4	0.021	[0.87, 1.32]	∞
Lagrange (4.13)	5	5	4	0.045	[0.59, 0.93]	6
Radau IIA (4.14)	7	4	4	0.024	[0.74, 1.31]	7

results is given by means of the number of correct digits Δ of the numerical solution at the endpoint T (i.e., we write the maximum norm of the error at $t=T$ in the form $10^{-\Delta}$). The computational costs are proportional to mL/h, where h is the fixed steplength, $L := T - t_0$ is the length of the integration interval, and m is the fixed number of outer iterations per step. In actual applications of these methods, some strategy is needed to select h and m. However, since our test problems are such that the exact solution is equally smooth in the whole integration interval, it is reasonable to use fixed h and m.

5.1. Test problems. We briefly discuss a few test problems partly taken from the literature and partly constructed in order to test some special aspect of the methods. All problems are defined on the interval $[t_0, T]$.

Our first problem is the stability test problem of Prothero and Robinson [21]

$$\text{(5.1a)} \qquad \frac{dy}{dt} = -\varepsilon^{-1}(y - g(t)) + g'(t), \qquad y(t_0) = g(t_0), \qquad t_0 = 0, \qquad T = 1,$$

where the exact solution equals $g(t)$ and ε is a small parameter. Prothero and Robinson used this problem to show the order reduction of RK methods when ε is small. In our experiments we set

$$\text{(5.1b)} \qquad g(t) = \cos(t), \qquad \varepsilon = 10^{-3}.$$

The second test problem is the nonlinearization of problem (5.1):

$$\text{(5.2a)} \qquad \frac{dy}{dt} = -\varepsilon^{-1}(y^3 - g(t)^3) + g'(t), \qquad y(t_0) = g(t_0), \qquad t_0 = 0, \qquad T = 1,$$

with exact solution $y(t) = g(t)$ for all values of the parameter ε. As in the preceding problem we set

$$\text{(5.2b)} \qquad g(t) = \cos(t), \qquad \varepsilon = 10^{-3}.$$

The third test problem is that of Kaps [17]:

$$\text{(5.3)} \qquad \frac{dy_1}{dt} = -(2+\varepsilon^{-1})y_1 + \varepsilon^{-1}(y_2)^2, \qquad \frac{dy_2}{dt} = y_1 - y_2(1+y_2),$$
$$y_1(t_0) = y_2(t_0) = 1, \qquad t_0 = 0, \qquad T = 1,$$

with the smooth exact solution $y_1 = \exp(-2t)$ and $y_2 = \exp(-t)$ for all values of the parameter ε. This problem belongs to the class of problems for which stiffly accurate RK methods do not suffer order reduction whatever small ε is (cf. Hairer, Lubich, and Roche [11]).

The test set of Enright, Hull, and Lindberg [10] contains the following system of ODEs describing a chemical reaction:

$$\text{(5.4a)} \qquad \frac{d\mathbf{y}}{dt} = -\begin{pmatrix} .013 + 1000y_3 & 0 & 0 \\ 0 & 2500y_3 & 0 \\ .013 & 0 & 1000y_1 + 2500y_2 \end{pmatrix}\mathbf{y},$$

with $\mathbf{y}(0) = (1, 1, 0)^T$. Since we use fixed stepsizes in our experiments, we avoided the initial phase by choosing the starting point at $t_0 = 1$ and we used the corresponding

initial values

$$\mathbf{y}(1) \approx \begin{pmatrix} 0.990731920827 \\ 1.009264413846 \\ -.366532612659 \times 10^{-5} \end{pmatrix}. \tag{5.4b}$$

At $t = T = 51$ we found the approximate solution

$$\mathbf{y}(51) = \begin{pmatrix} 0.591045966680 \\ 1.408952165382 \\ -.186793736719 \times 10^{-5} \end{pmatrix}.$$

In order to show the performance of the methods on PDEs we included the convection-diffusion problem

$$\frac{\partial u}{\partial t} = u\frac{\partial^2 u}{\partial x^2} - x\cos(t)\frac{\partial u}{\partial x} - x^2 \sin(t), \qquad 0 \leqq x \leqq 1, \qquad t_0 = 0, \qquad T = 1, \tag{5.5}$$

with Dirichlet boundary conditions and with exact solution $u(x, t) = x^2 \cos(t)$. Standard finite difference discretization of the spatial derivatives on a uniform grid with meshsize $1/40$ leads to a system of 39 ODEs whose exact solution is given by $(j/40)^2 \cos(t)$, $j = 1, \cdots, 39$.

5.2. Effective orders of the correctors. First of all, we want to show that in many stiff problems the property of superconvergence does not pay because of the phenomenon of order reduction, and that strong stability properties may improve the accuracy considerably.

The Tables 5.1(a) and 5.1(b) present Δ values for the various test problems obtained for $L/h = 1, 2, 4, 8, 16$ by iterating the corrector to convergence. From these results we can derive for each test problem the effective orders by computing $(\Delta(h) - \Delta(2h))/0.3$. For h we chose the smallest value for which results are available. The resulting effective

TABLE 5.1(a)

Problems (5.1), (5.2), *and* (5.3) *with* $\varepsilon = 10^{-3}$. *Values of* Δ *for* $L/h = 1, 2, 4, 8, 16$.

Corrector	p	r	k	(5.1)					(5.2)					(5.3) with $\varepsilon = 10^{-3}$				
(4.3a)	4	3	2	4.7	5.4	6.0	6.7	7.7	4.7	5.3	5.9	6.6	7.5	3.3	4.3	5.1	5.9	7.0
(4.4a)	3	3	2	5.1	5.9	6.8	7.8	8.8	5.0	5.8	6.7	7.7	8.7	2.7	3.6	4.4	5.3	6.2
(4.5a)	4	2	2	1.9	2.5	3.1	3.8	4.7	1.9	2.5	3.1	3.8	4.6	1.2	1.8	2.4	3.2	4.3
(4.6a)	3	2	2	4.2	4.7	5.3	5.9	6.5	4.2	4.7	5.2	5.8	6.4	2.4	3.2	4.1	5.0	5.9
(4.7a)	4	4	3	6.1	7.3	8.5	9.7	-	6.0	7.3	8.5	9.7	-	4.2	5.4	6.6	7.8	-
(4.8a)	6	4	3	6.1	7.3	8.6	9.8	-	6.1	7.3	8.5	9.7	-	4.7	6.0	7.3	9.3	-
(4.9a)	4	4	3	6.5	7.6	8.8	10.1	-	6.5	7.6	8.8	10.0	-	3.8	5.0	6.1	7.3	-
(4.10a)	5	3	3	5.0	6.0	6.9	7.9	-	4.9	5.9	6.9	7.8	-	4.0	5.3	6.3	7.3	-
(4.11a)	6	5	4	7.0	8.2	9.5	-	-	6.9	8.1	9.4	-	-	5.4	6.7	8.0	-	-
(4.12a)	8	5	4	7.1	8.4	9.6	-	-	7.0	8.3	9.5	-	-	5.6	6.8	8.2	-	-
(4.13a)	5	5	4	7.5	8.9	10.5	-	-	7.4	8.9	10.4	-	-	5.8	7.2	8.8	-	-
(4.14a)	7	4	4	6.3	7.4	8.6	-	-	6.3	7.3	8.5	-	-	5.0	6.4	7.8	-	-

TABLE 5.1(b)

Problems (5.3) *with* $\varepsilon = 10^{-8}$, (5.4) *and* (5.5). *Values of* Δ *for* $L/h = 1, 2, 4, 8, 16$.

Corrector	p	r	k	(5.3) with $\varepsilon = 10^{-8}$					(5.4)					(5.5)				
(4.3a)	4	3	2	3.3	4.5	5.7	6.9	8.1	4.5	5.7	6.9	8.2	9.4	3.2	4.2	5.4	6.5	7.7
(4.4a)	3	3	2	2.7	3.6	4.4	5.3	6.2	3.1	4.0	4.9	5.8	6.7	3.1	4.0	4.8	5.7	6.6
(4.5a)	4	2	2	1.2	1.8	2.4	3.0	3.6	5.0	6.1	7.3	8.5	9.7	1.9	2.6	3.2	3.9	4.8
(4.6a)	3	2	2	2.4	3.2	4.1	5.0	5.9	3.4	4.3	5.2	6.1	7.0	2.5	3.2	4.0	4.8	5.7
(4.7a)	4	4	3	4.2	5.4	6.7	7.9	-	4.7	5.9	7.1	8.3	-	4.6	5.9	7.2	8.4	-
(4.8a)	6	4	3	5.4	7.2	9.0	10.8	-	6.4	8.3	10.1	11.8	-	4.8	6.2	7.7	9.1	-
(4.9a)	4	4	3	3.9	5.0	6.2	7.3	-	4.2	5.4	6.6	7.8	-	4.5	5.6	6.8	7.9	-
(4.10a)	5	3	3	4.4	5.8	7.3	8.8	-	5.3	6.8	8.3	9.8	-	3.6	4.8	6.1	7.3	-
(4.11a)	6	5	4	5.9	7.7	9.6	-	-	6.7	8.5	10.3	-	-	5.7	7.4	9.2	-	-
(4.12a)	8	5	4	7.8	10.2	12.6	-	-	8.6	11.0	-	-	-	6.0	7.7	9.5	-	-
(4.13a)	5	5	4	6.0	7.4	8.8	-	-	6.9	8.2	9.7	-	-	6.4	7.8	9.3	-	-
(4.14a)	7	4	4	6.6	8.7	10.8	-	-	7.9	9.8	11.8	-	-	5.2	6.5	8.0	-	-

orders are listed in Table 5.2. For each problem, the result of the most accurate corrector is indicated in bold face.

The results for the first three problems clearly demonstrate that the various methods often do not show their step point order, so that the property of superconvergence is of limited value in the case of stiff problems.

5.3. Performance of the iteration process for two-processor correctors. In this section, we consider the performance of the iteration method for solving the two-processor corrector equations. Since the rate of convergence of a particular iteration method turned out to be comparable for the Newton-Cotes corrector and the Lagrange corrector, we only present results for the most accurate one. In the case of the Gauss and Radau corrector, the iteration methods behaved quite differently so that we include

TABLE 5.2

Effective orders shown by the correctors for problems (5.1)-(5.5).

Corrector	p	r	k	(5.1)	(5.2)	(5.3) $\varepsilon = 10^{-3}$	(5.3) $\varepsilon = 10^{-8}$	(5.4)	(5.5)
Newton-C. (4.3a)	4	3	2	3.3	3.0	**3.7**	**4.0**	4.0	**4.0**
Lagrange (4.4a)	3	3	2	**3.3**	**3.3**	3.0	3.0	3.0	3.0
Gauss (4.5a)	4	2	2	3.0	2.7	3.7	2.0	**4.0**	3.0
Radau IIA (4.6a)	3	2	2	2.0	2.0	3.0	3.0	3.0	3.0
Newton-C. (4.7a)	4	4	3	4.0	4.0	4.0	4.0	4.0	4.0
Lobatto IIIA (4.8a)	6	4	3	4.0	4.0	**6.7**	**6.0**	**5.7**	**4.7**
Lagrange (4.9a)	4	4	3	**4.3**	**4.0**	4.0	3.7	4.0	3.7
Radau IIA (4.10a)	5	3	3	3.3	3.0	3.3	5.0	5.0	4.0
Newton-C. (4.11a)	6	5	4	4.3	4.3	4.3	6.3	6.0	6.0
Lobatto IIIA (4.12a)	8	5	4	4.0	4.0	4.7	**8.0**	**8.0**	**6.0**
Lagrange (4.13a)	5	5	4	**5.3**	**5.0**	**5.3**	4.7	5.0	5.0
Radau IIA (4.14a)	7	4	4	4.0	4.0	4.7	7.0	6.7	5.0

results for both correctors. Moreover, the Gauss corrector was also iterated with a matrix D defined by the relation (3.12). Since for the two-processor Gauss corrector (4.5a) we have $A\mathbf{e}=\mathbf{c}$, i.e., $A^{-1}\mathbf{c}=\mathbf{e}$, it follows that $D^{-1}\mathbf{c}=\mathbf{e}$, so that

$$D=\operatorname{diag}(\mathbf{c})=\frac{1}{12}\begin{pmatrix}6-2\sqrt{3} & 0\\ 0 & 6+2\sqrt{3}\end{pmatrix}, \qquad \rho(I-D^{-1}A)=0.5.$$

In the Tables 5.3 and 5.4 we only present results for the problems (5.2) and (5.4) for which most methods, respectively, show their stage order and their step point order (additional results for the other test problems may be found in the Appendix to [14]). Divergence of the inner iteration is indicated by *, and values in bold indicate that the accuracy of the corrector is reached (and that Δ does not change anymore). For several values of L/h the accuracies corresponding to the correctors of Lagrange type (first column), of Gauss-Legendre with D defined by (4.5b) (second column), of Gauss-Legendre with D defined above (third column), and of Radau IIA (fourth column) are listed. These results confirm that, in general, the Gauss corrector is not suited to be iterated by diagonal iteration methods when started with an explicit predictor.

5.4. Efficiency of diagonally iterated IRK correctors. In this final section, we compare the efficiency of the diagonally iterated IRK correctors with three fourth-order

TABLE 5.3

Values of Δ *for problem* (5.2) *obtained by iterating the Lagrange corrector* (4.4a), *Gauss corrector* (4.5a), *and Radau* IIA *corrector* (4.6a).

m	$L/h=1$	$L/h=2$	$L/h=4$	$L/h=8$	$L/h=16$
1	3.5 0.3 0.3 3.8	4.1 −2.2 −2.2 5.3	4.0 * * 4.8	3.6 * * 5.0	2.7 * * 5.3
2	**5.0** 1.0 0.7 **4.2**	**5.8** −1.1 1.1 **4.7**	6.5 * 1.1 **5.2**	6.7 * 0.6 5.9	6.7 * * 6.7
3	**1.9** 1.0	2.4 2.2	**6.7** 2.9 2.6	**7.7** 3.9 3.2 **5.8**	8.4 1.9 3.8 **6.4**
4	1.4	**2.5** 1.9	**3.1** 2.5	**3.8** 3.1	**8.7** **4.6** 3.8
5	1.8	2.1	2.6	3.2	3.9
⋮	⋮	⋮	⋮	⋮	⋮
10	**1.9**	2.6	3.2	4.1	5.3
20		**2.5**	**3.1**	**3.8**	**4.6**

TABLE 5.4

Values of Δ *for Problem* (5.4) *obtained by iterating the Lagrange corrector* (4.4a), *Gauss corrector* (4.5a), *and Radau* IIA *corrector* (4.6a).

m	$L/h=1$	$L/h=2$	$L/h=4$	$L/h=8$	$L/h=16$
1	2.1 1.2 1.2 1.7	2.3 1.5 1.5 2.1	2.6 * * 2.4	2.8 * * 2.7	3.1 * * 3.0
2	3.4 2.6 2.2 2.9	3.9 2.9 2.8 3.5	4.5 * 3.4 4.1	5.2 * 4.0 4.7	5.8 * 4.5 5.3
3	4.3 3.8 3.0 3.6	5.4 4.8 3.8 4.5	6.4 5.7 4.7 5.4	7.4 6.6 5.6 6.3	8.3 6.9 6.5 7.2
4	**4.5** 4.7 3.8 **3.4**	**5.7** 5.9 4.9 **4.3**	**6.9** 7.1 6.1 **5.2**	8.1 8.3 7.2 **6.1**	9.3 9.5 8.4 **7.0**
5	**5.0** 4.4	**6.1** 5.7	**7.3** 7.1	**8.2** **8.5** 8.3	**9.4** **9.7** 9.5
6	4.8	**6.1**	**7.3**	**8.5**	9.5
7	**5.0**				**9.7**

DIRK methods from the literature, viz. the three-stage method generated by the Butcher array:

$$
\begin{array}{c|ccc}
\frac{1}{2}(1+\xi) & \frac{1}{2}(1+\xi) & & \\
\frac{1}{2} & -\frac{1}{2}\xi & \frac{1}{2}(1+\xi) & \\
\frac{1}{2}(1-\xi) & (1+\xi) & -(1+2\xi) & \frac{1}{2}(1+\xi) \\
\hline
 & \frac{1}{6\xi^2} & 1-\frac{1}{3\xi^2} & \frac{1}{6\xi^2}
\end{array}, \qquad \xi=\frac{2}{3}\sqrt{3}\cos\left(\frac{\pi}{18}\right) \tag{5.6}
$$

(cf. Crouzeix [8] and Alexander [1]), and the four-stage, parallel DIRK methods of Iserles and Nørsett [16]:

$$
\begin{array}{c|cccc}
\frac{1}{2} & \frac{1}{2} & & & \\
1 & 0 & 1 & & \\
\frac{1}{2} & \frac{3}{2} & -\frac{3}{2} & \frac{1}{2} & \\
0 & -3 & 2 & 0 & 1 \\
\hline
 & \frac{1}{3} & \frac{1}{6} & \frac{1}{3} & \frac{1}{6}
\end{array}, \tag{5.7}
$$

$$
\begin{array}{c|cccc}
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{2}{3} & 0 & \frac{2}{3} & & \\
\frac{1}{2} & -\frac{5}{2} & \frac{5}{2} & \frac{1}{2} & \\
\frac{1}{3} & -\frac{5}{3} & \frac{4}{3} & 0 & \frac{2}{3} \\
\hline
 & -1 & \frac{3}{2} & -1 & \frac{3}{2}
\end{array}. \tag{5.8}
$$

The method (5.6) is A-stable and requires three sequential stages per step. The methods (5.7) and (5.8) are A-stable and L-stable, respectively, and require only two sequential stages per step (when run on a two-processor computer).

We restrict our considerations to the above three DIRK methods and to the Newton–Cotes, Lobatto IIIA, Lagrange, and Radau IIA correctors where each method uses a fixed number of m iterations per step. Recalling that iterating an IRK corrector by means of m diagonal iterations in each step yields a method that is in fact a DIRK method with m sequential stages, we conclude that all methods have in common that they belong to the class of DIRK methods. However, in the case of the "genuine" DIRK methods (5.6), (5.7), and (5.8), the number of sequential stages per step is

known in advance, whereas in the case of the DIRK methods based on iteration the number of sequential stages m that yields acceptable accuracies, is not known in advance and, in actual computation, it should be determined on the basis of some local error strategy. On the other hand, as we shall see, the accuracy of the iterated methods is less sensitive to the phenomenon of order reduction.

In the Tables 5.5 and 5.6, m always denotes the number of sequential stages per step. Hence, all results in one column of these tables correspond to DIRK methods that use m sequential stages per step, so that all results corresponding to the same value of mL/h required roughly the same computational effort. In the tables, the highest value of Δ corresponding to the same mL/h value, that is, the "most efficient"

TABLE 5.5(a)

Problem (5.2): *results for diagonally iterated correctors and for the methods* (5.6), (5.7), *and* (5.8).

Method	k	L/h	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$	$m=7$	$m=8$	$m=9$	$m=10$	$\cdots$ $m=\infty$
Crouzeix-Alex. (5.6)	1	1	-	-	1.0								-
Iserles-Nørsett (5.7)	2		-	1.5									-
Iserles-Nørsett (5.8)			-	2.1									-
Newton-C. (4.3)			3.4	4.7									4.7
Lagrange (4.4)			3.5	**5.0**									5.0
Radau IIA (4.6)			**3.8**	4.2									4.2
Newton-C. (4.7)	3		3.2	3.7	**5.6**	6.1	6.0						6.0
Lobatto IIIA (4.8)			3.0	2.7	4.7	6.0	6.0	6.1					6.1
Lagrange (4.9)			3.2	3.9	5.5	**6.7**	**6.5**						6.5
Radau IIA (4.10)			3.4	3.1	5.0	4.9							4.9
Newton-C. (4.11)	4		3.1	3.6	4.9	4.7	5.2	6.0	**7.2**	7.0	6.9		6.9
Lobatto IIIA (4.12)			2.7	2.2	2.3	3.9	4.6	5.4	6.8	6.9	7.0		7.0
Lagrange (4.13)			3.0	2.8	3.1	3.9	5.0	6.4	7.1	**7.3**	**7.4**		**7.4**
Radau IIA (4.14)			2.9	2.8	3.0	4.7	5.6	**6.8**	6.3				6.3
Crouzeix-Alex. (5.6)	1	2	-	-	2.5								-
Iserles-Nørsett (5.7)	2		-	2.4									-
Iserles-Nørsett (5.8)			-	2.7									-
Newton-C. (4.3)			4.0	5.3									5.3
Lagrange (4.4)			4.1	**5.8**									5.8
Radau IIA (4.6)			**5.3**	4.7									4.7
Newton-C. (4.7)	3		3.4	3.5	**6.4**	**8.1**	7.2	7.3					7.3
Lobatto IIIA (4.8)			3.0	2.2	5.3	6.0	7.3						7.3
Lagrange (4.9)			3.5	3.8	5.9	7.5	**7.6**						7.6
Radau IIA (4.10)			3.8	2.8	5.9	5.7	5.9						5.9
Newton-C. (4.11)	4		3.3	3.3	5.2	5.2	5.3	5.9	6.7	7.8	8.3	8.1	8.1
Lobatto IIIA (4.12)			2.3	1.1	1.4	4.0	4.5	5.5	6.9	7.3	8.4	8.3	8.3
Lagrange (4.13)			2.9	2.3	2.7	4.9	5.2	6.5	**8.3**	**8.9**			**8.9**
Radau IIA (4.14)			2.8	2.2	2.6	5.0	6.0	7.0	7.5	7.3			7.3
Crouzeix-Alex. (5.6)	1	4	-	-	2.8								-
Iserles-Nørsett (5.7)	2		-	3.0									-
Iserles-Nørsett (5.8)			-	3.2									-
Newton-C. (4.3)			3.9	5.8	5.9								5.9
Lagrange (4.4)			4.0	**6.5**	**6.7**								6.7
Radau IIA (4.6)			**4.8**	5.2									5.2
Newton-C. (4.7)	3		3.1	3.0	6.6	**7.7**	8.4	8.5					8.5
Lobatto IIIA (4.8)			2.3	0.7	5.5	6.2	7.7	8.1	8.5				8.5
Lagrange (4.9)			3.2	3.5	6.2	**7.7**	**9.9**	**8.8**					8.8
Radau IIA (4.10)			3.6	2.0	5.6	6.2	6.8	6.9					6.9
Newton-C. (4.11)	4		2.9	2.5	5.0	5.5	5.5	6.0	6.8	7.7	8.7	9.8	9.4
Lobatto IIIA (4.12)			1.1	*	*	5.0	4.3	5.6	6.4	7.2	8.3	9.0	9.5
Lagrange (4.13)			2.3	0.8	1.5	5.1	5.6	6.8	7.9	**8.8**	**9.7**	**10.8**	**10.4**
Radau IIA (4.14)			2.1	0.6	1.2	5.2	6.3	7.9	8.4	8.5			8.5

TABLE 5.5(b)

Problem (5.2): *efficiency test of fourth-order methods.*

Method	p	m	k	$mL/h=4$	$mL/h=8$	$mL/h=16$
Iserles-Nørsett (5.7)	4	2	2	2.4	3.0	3.6
Iserles-Nørsett (5.8)	4	2	2	2.7	3.2	3.8
Newton-C. (4.3)	4	4	2	4.7	5.3	5.9
Newton-C. (4.7)	4	4	3	6.1	**8.1**	**7.7**
Lobatto IIIA (4.8)	4	4	3	6.0	6.0	6.2
Lagrange (4.9)	4	4	3	**6.7**	7.5	**7.7**
Radau IIA (4.10)	4	4	3	4.9	5.7	6.2
Newton-C. (4.11)	4	4	4	4.7	5.2	5.5
Lobatto IIIA (4.12)	4	4	4	3.9	4.0	5.0
Lagrange (4.13)	4	4	4	3.9	4.9	5.1
Radau IIA (4.14)	4	4	4	4.7	5.0	5.2

integration result, is indicated in bold. As in the preceding section, we only present results for the problems (5.2) and (5.4). Results for the additional test problems may be found in the Appendix to [14].

In the case of the nonlinear Prothero-Robinson problem, Table 5.5(a) shows that the number of iterations needed by the iterated methods to "reach" the accuracy of the corrector solution increases with k, that is, the higher-order methods need more iterations to solve the corrector; moreover, they have a "slow start": after two iterations the accuracy is still rather modest, whereas the lower-order methods have already converged, showing full corrector-precision. This can be explained by observing that we used a zero-order predictor for $\mathbf{Y}^{(0)}$ for all k, so that the "distance" between predictor and corrector solution increases with k. Thus, for this problem, the lower-order methods are more efficient than the higher-order ones, unless very high accuracies are requested. Furthermore, when we compare the various types of iterated methods (Newton-Cotes, Lobatto, Lagrange, or Radau), then the Lobatto IIIA methods perform not as well whereas the strongly A-stable Lagrange methods are slightly superior to the others. In the case of the "genuine" DIRK methods (5.6), (5.7) and (5.8), the Iserles-Nørsett methods are more accurate than the Crouzeix-Alexander method, which is presumably due to the L-stability property of the Iserles-Nørsett method.

It is of particular interest to see how the iterated methods compare with the "genuine" DIRK methods. For example, Table 5.5(a) shows that the Newton-Cotes, Lobatto IIIA, Lagrange and Radau IIA based methods, respectively, produce 5, 0, 21 and 4 "most efficient" results, whereas the "genuine" DIRK methods none. A further indication of the superiority of the iterated methods is given by Table 5.5(b) where we list results for the iterated methods with $m=4$ and for the parallel DIRK methods (5.7) and (5.8). All these methods have step point order $p=4$, but the accuracies obtained for the same computational-costs value of mL/h differ largely, which is caused by the order reduction exhibited by the "genuine" DIRK methods.

For the more innocent chemical reaction problem (5.4) the order reduction is not shown. Table 5.6(a) shows that the high-order iterated methods again require more iterations to obtain the corrector precision than the lower-order methods, however, here for low values of m, all iterated methods are roughly equally efficient. Furthermore, the scores of "most efficient" results for the Newton-Cotes, Lobatto IIIA, Lagrange, and Radau IIA based methods are, respectively, 8, 5, 6, and 7, and among the DIRK methods only (5.7) scores twice. The analogue of Table 5.5(b) is given by Table 5.6(b).

TABLE 5.6(a)

Problem (5.4): *results for diagonally iterated correctors and for the methods* (5.6), (5.7), *and* (5.8).

Method	k	L/h	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$	$m=7$	$m=8$	$m=9$	$m=10$	$\cdots$ $m=\infty$
Crouzeix-Alex. (5.6)	1	1	-	-	3.4								-
Iserles-Nørsett (5.7)	2		-	3.4									-
Iserles-Nørsett (5.8)			-	3.3									-
Newton-C. (4.3)			**2.1**	3.4	4.3	4.5							4.5
Lagrange (4.4)			**2.1**	3.5	3.1								3.1
Radau IIA (4.6)			1.7	2.9	3.6	3.4							3.4
Newton-C. (4.7)	3		1.8	3.5	5.1	4.7							4.7
Lobatto IIIA (4.8)			1.6	3.1	4.3	5.6	6.3	6.4					6.4
Lagrange (4.9)			1.8	3.5	4.3	4.2							4.2
Radau IIA (4.10)			2.0	3.2	4.3	5.9	5.3						5.3
Newton-C. (4.11)	4		1.7	**3.6**	5.2	6.5	6.7						6.7
Lobatto IIIA (4.12)			1.4	2.7	4.6	6.0	7.1	**8.3**	**8.6**				**8.6**
Lagrange (4.13)			1.6	3.1	**5.8**	6.6	7.0	6.9					6.9
Radau IIA (4.14)			1.5	3.2	4.8	**7.4**	**7.8**	7.9					7.9
Crouzeix-Alex. (5.6)	1	2	-	-	4.4								-
Iserles-Nørsett (5.7)	2		-	**4.5**									-
Iserles-Nørsett (5.8)			-	4.4									-
Newton-C. (4.3)			**2.3**	3.9	5.4	5.7							5.7
Lagrange (4.4)			**2.3**	**4.5**	4.0								4.0
Radau IIA (4.6)			2.1	3.5	4.5	4.3							4.3
Newton-C. (4.7)	3		2.0	4.2	6.2	5.9							5.9
Lobatto IIIA (4.8)			1.9	3.8	5.1	6.8	8.1	8.3					8.3
Lagrange (4.9)			2.1	4.1	5.5	5.4							5.4
Radau IIA (4.10)			2.2	3.8	5.1	6.9	6.8						6.8
Newton-C. (4.11)	4		2.0	**4.5**	**6.7**	7.9	8.5						8.5
Lobatto IIIA (4.12)			1.7	3.3	5.4	7.2	8.5	10.0	**10.9**	**11.0**			**11.0**
Lagrange (4.13)			1.9	3.7	6.3	7.5	8.3	8.2					8.2
Radau IIA (4.14)			1.8	3.7	5.6	**8.0**	**8.8**	**10.1**	9.8				9.8
Crouzeix-Alex. (5.6)	1	4	-	-	5.5								-
Iserles-Nørsett (5.7)	2		-	**5.7**									-
Iserles-Nørsett (5.8)			-	5.6									-
Newton-C. (4.3)			**2.6**	4.5	6.4	6.9							6.9
Lagrange (4.4)			**2.6**	4.7	4.9								4.9
Radau IIA (4.6)			2.4	4.1	5.4	5.2							5.2
Newton-C. (4.7)	3		2.3	5.0	**7.2**	7.1							7.1
Lobatto IIIA (4.8)			2.2	4.4	6.0	7.9	9.7	10.1					10.1
Lagrange (4.9)			2.4	4.8	6.8	6.6							6.6
Radau IIA (4.10)			2.5	4.5	6.0	7.9	8.3						8.3
Newton-C. (4.11)	4		2.3	5.4	7.1	8.9	**10.6**	10.3					10.3
Lobatto IIIA (4.12)			2.0	4.0	6.1	8.4	10.1	11.9	**12.3**				**12.3**
Lagrange (4.13)			2.2	4.2	**7.2**	8.7	9.9	9.7					9.7
Radau IIA (4.14)			2.1	4.3	6.6	**9.1**	10.2	**12.2**	11.8				11.8

It reveals that the iterated methods are usually much more efficient than the parallel DIRK methods, and in any case they are at least competitive.

6. Concluding remarks. In this paper we have derived a diagonally implicit iteration scheme to solve a fully implicit Runge-Kutta method. The structure of this iteration process is such that a parallel computer can be fully exploited. Starting with an implicit RK method with k implicit stages (the corrector), each iteration requires the solution of k systems of equations of dimension equal to the number of ODEs. Since these systems can be solved completely independently, the *effective* computational work per iteration equals the solution of one such system, provided that k processors are available.

TABLE 5.6(b)

Problem (5.4): *efficiency test of fourth-order methods.*

Method	p	m	k	$mL/h=4$	$mL/h=8$	$mL/h=16$
Iserles-Nørsett (5.7)	4	2	2	4.5	5.7	6.9
Iserles-Nørsett (5.8)	4	2	2	4.4	5.6	6.7
Newton-C. (4.3)	4	4	2	4.5	5.7	6.9
Newton-C. (4.7)	4	4	3	4.7	5.9	7.1
Lobatto IIIA (4.8)	4	4	3	5.6	6.8	7.9
Lagrange (4.9)	4	4	3	4.2	5.4	6.6
Radau IIA (4.10)	4	4	3	5.9	6.9	7.9
Newton-C. (4.11)	4	4	4	6.5	7.9	8.9
Lobatto IIIA (4.12)	4	4	4	6.0	7.2	8.4
Lagrange (4.13)	4	4	4	6.6	7.5	8.7
Radau IIA (4.14)	4	4	4	**7.4**	**8.0**	**9.1**

The free parameters in the iteration scheme are chosen in such a way that the corresponding stability functions converge as quickly as possible to the stability function of the corrector, which is chosen to be (at least) A-acceptable. Although we have numerical evidence that this is not a bad choice, we do not claim that it is the best possible. In a forthcoming paper it is intended to give theoretical support for this choice.

A second aspect considered in this paper, is the choice of the particular corrector method. The well-known implicit RK methods of high classical order, such as the Gauss-Legendre, Radau, and Lobatto methods, seem to be suitable candidates. However, since it is the *stage order* which usually determines the order behaviour in integrating stiff differential equations, these methods are not necessarily optimal correctors. Because the stage order is significantly smaller than the classical order for these methods, we will encounter the phenomenon of order reduction. Therefore, we also considered Newton-Cotes and Lagrange correctors, which have—for the same number of implicit relations per iteration—a stage order which is one higher than for Gauss-Legendre and Radau methods and is equal to the stage order of Lobatto methods.

Apart from these order considerations, it turned out that the stability behaviour of the iterated scheme largely depends on the choice of the corrector. For example, it is shown that the Gauss-Legendre corrector is not suitable in this context, since it is not stiffly accurate. Consequently, only for very "innocent" stiff problems, where we have no order reduction, the Gauss-Legendre corrector is useful, but as a method for general stiff problems it is disadvantageous.

The other four types of correctors are all stiffly accurate, which has the effect that certain classes of stiff problems can be integrated without order reduction. For such problems the classical order should be a decisive factor, viz. in these cases the Lobatto IIIA corrector is superior and also the Newton-Cotes corrector is a good choice. However, these correctors are only A-stable and it is shown that the stability function of the iterated method is not A-acceptable unless the corrector is really solved. This means that the iteration process based on these correctors easily encounters stability problems. Hence, a corrector possessing better stability characteristics, such as the Radau IIA method (L-stable) and the Lagrange method (strongly A-stable), will be much more robust. We showed that after a few iterations the stability function of the iterated methods based on these correctors is A-acceptable.

Since the stage order of the Lagrange corrector is one larger than that of the Radau IIA corrector, we think that it is a good choice for integrating general stiff

equations; it combines adequate stability characteristics with a relatively high stage order. Our numerical experiments confirm this advice.

Furthermore, we have compared our methods with sequential and parallel DIRK methods from the literature. This comparison is rather obvious since the effective computational work per iteration equals the work per stage in a DIRK method. It turned out that the diagonally iterated RK methods are much more efficient than the "conventional" DIRKs. The reason is that only low order "conventional" DIRKs with good stability properties are available in the literature and, more importantly, these DIRKs have a stage order equal to 1. This property gives these methods a very poor performance in case of general stiff problems.

Finally, we remark that the construction of diagonally iterated methods of arbitrarily high order is straightforward, and we observed in our experiments that, especially the high order methods, showed remarkably high accuracies.

Acknowledgment. The authors are grateful to Dr. W. H. Hundsdorfer for carefully reading the manuscript and for many valuable suggestions during the investigations reported in this paper.

REFERENCES

[1] R. ALEXANDER, *Diagonally implicit Runge-Kutta methods for stiff* ODEs, SIAM J. Numer. Anal., 14 (1977), pp. 1006-1021.

[2] K. BURRAGE, *A special family of Runge-Kutta methods for solving stiff differential equations*, BIT, 18 (1978), pp. 22-41.

[3] ———, *The error behaviour of a general class of predictor-corrector methods*, CMSR Report, University of Liverpool, Liverpool, U.K., 1989.

[4] K. BURRAGE, P. J. VAN DER HOUWEN, W. H. HUNDSDORFER, AND B. P. SOMMEIJER, *Diagonal iteration of Runge-Kutta methods*, in preparation.

[5] J. C. BUTCHER, *On the implementation of implicit Runge-Kutta methods*, BIT, 16 (1976), pp. 237-240.

[6] ———, *A transformed implicit Runge-Kutta method*, J. Assoc. Comput. Mach., 26 (1979), pp. 731-738.

[7] ———, *The Numerical Analysis of Ordinary Differential Equations, Runge-Kutta and General Linear Methods*, John Wiley, New York, 1987.

[8] M. CROUZEIX, *Sur l'approximation des équations différentielles opérationnelles linéaires par des méthodes de Runge-Kutta*, Ph.D. thesis, Université de Paris, Paris, France, 1975.

[9] K. DEKKER AND J. G. VERWER, *Stability of Runge-Kutta Methods for Stiff Nonlinear Differential Equations*, CWI Monograph 2, North-Holland, Amsterdam, 1984.

[10] W. H. ENRIGHT, T. E. HULL, AND B. LINDBERG, *Comparing numerical methods for stiff systems of* ODEs, BIT, 15 (1975), pp. 10-48.

[11] E. HAIRER, CH. LUBICH, AND M. ROCHE, *Error of Runge-Kutta methods for stiff problems studied via differential algebraic equations*, BIT, 28 (1988), pp. 678-700.

[12] P. J. VAN DER HOUWEN AND B. P. SOMMEIJER, *Parallel iteration of high-order Runge-Kutta methods with stepsize control*, J. Comput. Appl. Math. 29 (1990), pp. 111-127.

[13] P. J. VAN DER HOUWEN, B. P. SOMMEIJER, AND W. COUZY, *Embedded diagonally implicit Runge-Kutta algorithms on parallel computers*, Report NM-R8912, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, Math. Comp., 1992, to appear.

[14] P. J. VAN DER HOUWEN AND B. P. SOMMEIJER, *Iterated Runge-Kutta methods on parallel computers*, Report NM-R9001, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, 1990.

[15] W. H. HUNDSDORFER, private communication, 1990.

[16] A. ISERLES AND S. P. NØRSETT, *On the theory of parallel Runge-Kutta methods*, IMA J. Numer. Anal., 10 (1990), pp. 463-488.

[17] P. KAPS, *Rosenbrock-type methods*, in Numerical Methods for Stiff Initial Value Problems, G. Dahlquist and R. Jeltsch, eds., Bericht nr. 9, Inst. für Geometrie und Praktische Mathematik der RWTH Aachen, Aachen, Germany, 1981.

[18] I. LIE, *Some aspects of parallel Runge-Kutta methods*, Report No. 3/87, Division Numerical Mathematics, University of Trondheim, Trondheim, Norway, 1987.

[19] S. P. NØRSETT, *Semi-explicit Runge-Kutta methods*, Report Mathematics and Computation No. 6/74, Department of Mathematics, University of Trondheim, Trondheim, Norway, 1974.

[20] S. P. NØRSETT AND H. H. SIMONSEN, *Aspects of parallel Runge-Kutta methods*, in Numerical Methods for Ordinary Differential Equations, A. Bellen, C. W. Gear, and E. Russo, eds., Proc. L'Aquila 1987, Lecture Notes in Mathematics, Vol. 1386, Springer-Verlag, Berlin, 1989.

[21] A. PROTHERO AND A. ROBINSON, *On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations*, Math. Comp., 28 (1974), pp. 145–162.

[22] B. P. SOMMEIJER, W. COUZY, AND P. J. VAN DER HOUWEN, *A-stable parallel block methods*, Report NM-R8918, Centre for Mathematics and Computer Science, Amsterdam, the Netherlands, Appl. Numer. Math., to appear.

[23] H. A. WATTS AND L. F. SHAMPINE, *A-stable block implicit one-step methods*, BIT, 12 (1972), pp. 252–266.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1029-1057, September 1991
© 1991 Society for Industrial and Applied Mathematics
003

A CONTROL VOLUME FINITE ELEMENT APPROACH TO NAPL GROUNDWATER CONTAMINATION*

PETER A. FORSYTH†

Abstract. Finite difference discretizations of nonaqueous phase liquid (NAPL) groundwater contamination problems ensure that the discrete flux direction is always in the physical direction. This is not always the case for finite element discretizations. If linear triangular elements are used, this condition imposes a constraint on the type of triangulation allowed. Use of this type of triangulation is demonstrated for various three-phase contamination problems in two dimensions.

Key words. control volume, finite element, positive transmissibilities

AMS(MOS) subject classifications. 65M05, 76S05

1. Introduction. Contamination of groundwater due to nonaqueous phase liquids (NAPL) is a problem of growing concern. Liquids such as PCBs, TCE, dioxins, and various petroleum products are known to be leaking from many toxic waste sites [1]-[10]. It is of interest to track the movement of contamination in both the saturated and unsaturated zones, so that a full three-phase solution is required [1], [3], [4], [10]. In many cases, the NAPL solubility in the air phase is small, so that air phase transport can be neglected [3]. With this assumption, the number of equations and unknowns can be reduced by means of the passive air-phase approximation [3], [11]. This approximation has been shown to have acceptable accuracy in many practical situations due to the low air pressures involved in near surface flow [11].

Both finite difference [1], [3], [4], [5], [9] and finite element [2], [7], [10], [12] methods have been used to solve this problem. This system of equations can be informally classified as of mixed hyperbolic-parabolic type. For example, if the flow is two-phase, incompressible, one-dimensional, and the capillary pressure is zero, then the system of equations reduces to the well-known Buckley-Leverett equation [13]. This equation is a nonlinear hyperbolic equation for the fluid saturation. On the other hand, if the entire domain is single phase, then the system of equations reduces to a parabolic equation for the fluid pressure. Consequently, we can regard the pressure as a "parabolic-like" variable, while the fluid saturations are "hyperbolic-like" variables. In order to ensure convergence to the physically correct solution satisfying the Entropy condition (in all physical situations), it is common practice to use some form of upstream weighting [13] for saturation dependent terms.

In the finite difference case, standard upstream techniques can be employed [1], [3]-[5], [9]. For finite element discretizations, upstream weighted test functions have been devised [2], [10], [12].

It can be shown that implicit finite difference techniques produce stable, monotone solutions for one-dimensional countercurrent flow situations [13], [14]. In the pure hyperbolic case, phase upstream weighting is known to converge to the physically correct solution [13]. In all cases, the implicit discrete finite difference equations ensure that saturations are in the physical range (positive and less than one). In the single phase limit, the finite difference pressure equation becomes an M-matrix [15]. This

* Received by the editors October 20, 1989; accepted for publication (in revised form) August 28, 1990.
† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

implies (at least in this limiting case) that nonphysical local maxima and minima in the pressure cannot occur. In the multiphase case, this means that the direction of the discrete flux always corresponds to the physical direction. This desirable property of a discretization is termed "positive transmissibilities" or positive flux linkages [16], [17]. Similar positive transmissibility requirements are common in other application areas, such as semiconductor device modelling [18].

However, finite difference methods have the well-known disadvantage of being inflexible when applied to problems with complex geometries. Of course, finite element methods are well suited for situations with nonrectangular geometries. Unfortunately, upstream weighted finite element discretizations [10], [12] will not, in general, yield an M-matrix pressure equation in the single phase limit. This implies (in the multiphase case) that the discrete flux direction will not always correspond to the physical direction. In addition, it is not clear that the upstream weighted test functions [10], [12] ensure that the saturations remain positive.

The positive transmissibility requirement is extremely important when dealing with gravity dominated flows involving fluids having differing densities. As an example of this, consider the following situation. Suppose that node A is located physically higher than node B (depth is increasing as we move from A to B). Initially, nodes A and B have equal (mobile) saturations of a dense fluid and a light fluid. Physically, the dense fluid must sink, and the light (less dense) fluid must rise. If the discretization produces a positive transmissibility between nodes A and B, then the discrete fluid motion between A and B will correspond to the physically correct motion. However, if the discrete equations are such that the transmissibility between A and B is negative, then the discrete fluid motion will be opposite to the physical fluid flow. In other words, the light fluid will sink, and the dense fluid will rise. This is physically absurd.

The objective of this paper is to develop a flexible discretization technique which avoids, as much as possible, negative transmissibilities which cause nonphysical discrete flow. In the following, we refer to this discretization technique as a control volume-finite element approach. Similar ideas have been suggested by Dalen [19] in the context of reservoir simulation. Finite volume type methods have also been suggested by Potempa [20], [21] and Rozen [22], but no attempt was made to ensure that the single phase pressure equations produced an M-matrix. In fact, the basis functions used in [20]–[22] rarely produce M-matrices for practical grids. Another possible approach for unstructured grids is based on the integrated finite difference method [23], but this requires that the nodes have a specific geometric placement. Control volume-finite element methods have also been used in Navier–Stokes problems and are related to the box method in semiconductor device modelling [18], [24].

In this article, we will develop an upstream weighting method which ensures that the implicit discrete equations always produce saturations in the physical range, regardless of timestep size, mesh size, or sign of the transmissibilities. If linear triangular basis functions are used for two-dimensional problems, then for the special case of a convex domain with a constant permeability tensor, it is possible to use a simple algorithm which will triangulate a given set of nodes so that all interior (nonboundary) transmissibilities are positive. No movement or addition of nodes is required.

The triangulation algorithm is known to converge since, in the special case described above, the condition for positive internal transmissibilities is equivalent to carrying out a Delaunay triangulation [25]–[28] in a suitably transformed domain. Computational experiments indicate that a slightly generalized algorithm, when applied to nonconvex grids with widely varying permeabilities, produces only a small number of negative transmissibilities.

Note that, in general, a Delaunay triangulation of the physical domain will not produce positive transmissibilities. In fact, in three dimensions, even if the permeability tensor is the identity, a Delaunay triangulation does not produce positive internal transmissibilities. Consequently, the positive transmissibility condition itself must be used as a criterion for defining a suitable triangulation, not the Delaunay condition. The fact that the positive transmissibility condition and a two-dimensional Delaunay triangulation are equivalent in certain special cases is merely an interesting coincidence. However, this correspondence does suggest techniques for converting a given triangulation into a triangulation having only a small number of negative transmissibilities.

A coarse grid computational example will be given which demonstrates the effect of having a large number (greater than 25 percent) of negative transmissibilities, compared to a computation (for the same domain) with all positive transmissibilities. Physically reasonable behaviour is observed for the discretization having all positive transmissibilities, while the simulation with a large fraction of negative transmissibilities shows a dense fluid moving "uphill" away from the injection point.

Results are also given for discretizations having all positive transmissibilities compared to discretizations having a small fraction (less than 1 percent) of negative transmissibilities. These tests indicate that macroscopic computed parameters are relatively unaffected by a small number of negative transmissibilities.

These simulations were repeated with discretizations having large numbers (greater than 20 percent) of negative transmissibilities. In all cases, the simulations with a large fraction of negative transmissibilities required a greatly increased number of Newton iterations, inner iterations, or both. This situation was particularly severe in the case of dense NAPL contaminants. A physical explanation is given for this behaviour.

2. Formulation. Three phase flow in a porous medium generally requires solution of conservation equations for water (w), air (a) and NAPL (n). However, in keeping with previous work [1], [3]–[5], [9], [10] we will assume that the air phase pressure is constant. Consequently, the liquid saturation can be obtained from the air-NAPL capillary pressure relation, and the air conservation equation can be eliminated.

The resulting equations are:

Water conservation:

$$\frac{\partial}{\partial t}(\phi\rho_w S_w) = q'_w + \nabla\cdot[\mathbf{K}\rho_w\lambda_w(\nabla P_w - \rho_w g\nabla D)]; \tag{1}$$

NAPL conservation:

$$\frac{\partial}{\partial t}(\phi\rho_n S_n) = q'_n + \nabla\cdot[\mathbf{K}\rho_n\lambda_n(\nabla P_n - \rho_n g\nabla D)]; \tag{2}$$

where

S_l = saturation of phase $l = n, w$,
ϕ = porosity,
$\lambda_l = K_{rl}/\mu_l$,
K_{rl} = relative permeability of phase l,
μ_l = viscosity of phase l,
$\mathbf{K}$ = absolute permeability tensor,
D = depth,
g = gravitational acceleration,
P_l = pressure of phase l,
ρ_l = density of phase l,
q'_l = source/sink term for phase l.

In addition to the conservation equations, there are the additional equations:

$$S_n + S_a + S_w = 1, \tag{3}$$

$$P_a = P_n + \alpha P_{can}(S_a) + \{1-\alpha\}[P_{caw}(S_a) - P_{cnw}(S_w = 1)], \tag{4}$$

$$P_n = P_w + \alpha P_{cnw}(S_w) + \{1-\alpha\} P_{cnw}(S_w = 1) \tag{5}$$

where

$$\alpha = \min(1, S_n / S_n^*) \tag{6}$$

and P_{can}, P_{cnw}, P_{caw} are experimentally determined capillary pressure curves [1]. Note that equations (4)-(6) have been modified in comparison to the usual capillary pressure relations in reservoir simulation [29]. This is because the usual relations ($\alpha = 1$) do not produce the correct limit as S_n goes to zero [3], [10]. The usual relations implicitly assume the existence of a continuous nonaqueous phase. This assumption is clearly invalid in groundwater contamination problems since S_n is usually zero initially. If S_n^* in (6) is equal to the critical oil saturation, then equations (4)-(6) are correct in both limits of vanishing S_n, and for a continuous NAPL phase.

Note that since P_a (air pressure) is assumed constant, then (4) can be solved for S_a, so that

$$S_a = (1 - S_n - S_w) = f(P_n, S_n), \tag{7}$$

which differs from the usual reservoir simulation case. The three phase relative permeability for NAPL (K_{rn}) is given by Stone's model [29].

3. Discretization. To avoid obscuring the basic ideas, we will first discretize the model equation:

$$\frac{\partial S}{\partial t} = \nabla \cdot [\mathbf{K} \lambda(S) \nabla P], \tag{8}$$

which can be regarded as prototypical of equations (1)-(2). Let N_i be the usual C^0 finite element polynomial basis functions [30] where

$$\begin{aligned} N_i &= 1 \quad \text{at node } i \\ &= 0 \quad \text{at all other nodes,} \\ \sum_i N_i &= 1 \quad \text{everywhere in the domain.} \end{aligned} \tag{9}$$

We have not, as yet, specified the element geometry, or the type of basis function. Let

$$P = \sum_i P_i N_i. \tag{10}$$

Using a lumped mass approach for the time derivative term of (8), and a Galerkin approximation for the divergence term gives

$$\frac{S_i^{N+1} - S_i^N}{\Delta t} \int_v N_i \, dv = -\int_v \mathbf{K}[\lambda(S) \nabla P \cdot \nabla N_i] \, dv \tag{11}$$

where v is the volume of the region of interest, superscript N is the time level, and Δt is the time increment. We will leave as unspecified the time level of evaluation of the right-hand side of (11) for the time being. It follows from (9) that

$$\nabla \sum_j P_j N_j = \sum_{j \neq i} (P_j - P_i) \nabla N_j \tag{12}$$

and consequently, the right-hand side of (11) is given by

$$-\int_v (\mathbf{K}\nabla P\cdot\nabla N_i)\lambda(S)\,dv = -\sum_{j\in\eta_i}(P_j-P_i)\int_v (\mathbf{K}\nabla N_j\cdot\nabla N_i)\lambda(S)\,dv \tag{13}$$

where η_i is the set of neighbour nodes of i such that the above integral is nonzero. Anticipating some form of upstream weighting, the integral in (13) is approximated by

$$\int_v \mathbf{K}\nabla N_i\cdot\nabla N_j\lambda(S)\,dv \approx \lambda(S^*)\int_v \mathbf{K}\nabla N_i\cdot\nabla N_j\,dv \tag{14}$$

where S^* is some as yet unspecified point in the intersection of the support of N_i and N_j. The above approximation is called the influence coefficient technique [31], [32]. If

$$\gamma_{ij} = -\int_v \mathbf{K}\nabla N_i\cdot\nabla N_j\,dv, \qquad V_i = \int_v N_i\,dv, \tag{15}$$

then an upstream weighted discretization of (8) can be defined by

$$\left(\frac{S_i^{N+1}-S_i^N}{\Delta t}\right)V_i = \sum_{j\in\eta_i}\lambda_{ij}^{ups}\gamma_{ij}(P_j-P_i) \tag{16}$$

where

$$\begin{aligned}\lambda_{ij}^{ups} &= \lambda_j \quad \text{if } \gamma_{ij}(P_j-P_i)>0\\ &= \lambda_i \quad \text{if } \gamma_{ij}(P_j-P_i)<0.\end{aligned} \tag{17}$$

Since γ_{ij} (eq. (14)) is symmetric, then (16) is mass conservative. Note that if $\gamma_{ij}>0$, then (17) has precisely the same form as the upstream weighting used in finite difference discretization of multiphase flow [14]. In fact, if a rectangular grid is used, $\mathbf{K}$ is diagonal, and if the rectangles are subdivided into triangles and the N_i are linear basis functions defined on triangles, then the right-hand side of (16) is identical to the usual finite difference expression [1]. For $\gamma_{ij}<0$, we will demonstrate that the choice of upstream weighting in (17) ensures that saturations remain in the physical range.

Dalen [19] avoided the problem of selecting the upstream point for $\gamma_{ij}<0$ by assuming that the element geometry was such as to have all $\gamma_{ij}>0$. No algorithm was given in [19] for ensuring this condition.

Using the same approach, (1)-(2) can be discretized in a similar manner:

Water equation:

$$\frac{V_i}{\Delta t}[(\phi\rho_w S_w)_i^{N+1}-(\phi\rho_w S_w)_i^N] = q_{w,i}^{N+1}+\sum_{j\in\eta_i}\gamma_{ij}(\lambda_w^M\rho_w^M)_{ij}^{ups}\psi_{w,ij} \tag{18}$$

where

$$\psi_{w,ij} = (P_{n,j}^{N+1}-P_{n,i}^{N+1})-(P'_{cnw,j}-P'_{cnw,i})^M-\rho_{w,ij+1/2}^M g(D_j-D_i);$$

$$P'_{cnw} = P_n - P_w \quad \text{(see eq. (5))};$$

NAPL equation:

$$\frac{V_i}{\Delta t}[(\phi\rho_n S_n)_i^{N+1}-(\phi\rho_n S_n)_i^N] = q_{n,i}^{N+1}+\sum_{j\in\eta_i}\gamma_{ij}(\lambda_n^M\rho_n^M)_{ij}^{ups}\psi_{n,ij} \tag{19}$$

where

$$\psi_{n,ij} = (P_{n,j}^{N+1}-P_{n,i}^{N+1})-\rho_{n,ij+1/2}^M g(D_j-D_i)$$

and

$$
\begin{aligned}
q_l &= q_l' V_i, \qquad \lambda_w^M = \lambda_w(P_n^{N+1}, S_n^M), \\
\lambda_n^M &= \lambda_n(P_n^{N+1}, S_n^M), \qquad \rho_{l,ij+1/2} = (\rho_{l,j} + \rho_{l,i})/2, \\
(\lambda_l \rho_l)_{ij}^{ups} &= (\lambda_l \rho_l)_i \quad \text{if } \gamma_{ij}\psi_{l,ij} < 0 \\
&= (\lambda_l \rho_l)_j \quad \text{if } \gamma_{ij}\psi_{l,ij} > 0.
\end{aligned} \tag{20}
$$

When $M = N+1$ in (18) and (19), a fully implicit time discretization is used, while if $M = N$, an IMPES type method is defined. Allowing M to vary from node to node defines an adaptive implicit scheme [14], [33], which will be used in the following. The nonlinear algebraic equations (18), (19) are solved using full Newton iteration. If $(P_{n,i}, S_{n,i})$ are the primary variables for each node (i.e., those variables which are regarded as independent when constructing the Jacobian), and if

$$
\left.\frac{\partial \lambda_n}{\partial S_n}\right|_{S_n=0} = 0 \tag{21}
$$

then there will be no timestep restriction (in terms of stability) for an IMPES discretization in those regions where $S_n \equiv 0$. This is because if (21) holds, then when $S_n \equiv 0$, a Newton iterative procedure for solving equations (18)-(19) is equivalent to Newton iterative solution of a discrete form of the Richards equation [34] for unsaturated flow. This is solved fully implicitly for P_n (recall that $P_n = P_w + \text{const.}$ when $S_n = 0$). Consequently, if an adaptive implicit method is used, then the computational work in regions with $S_n = 0$ is comparable to a fully implicit solution of the Richards equation. Of course, there is a practical limitation to the size of the timestep, since the Newton iteration may not converge for very large timesteps.

It is not obvious that Stone's second method [1] for three-phase NAPL relative permeability will satisfy (21). In practice, if the relative permeability given by Stone's method is denoted by k_{rn}^*, then the actual relative permeability used in the computation is given by

$$
k_{rn} = \max(0, k_{rn}^*)
$$

since for S_n small k_{rn}^* is often negative. This is the approach used in this work. For reasonable NAPL-water and NAPL-gas relative permeability curves, the above approach produces k_{rn} which are identically zero for small (but nonzero) S_n, and hence (21) is satisfied. Physically, S_n^* (eq. (6)) should be selected to be smaller than the minimum of the critical NAPL saturation in the NAPL-water and the NAPL-gas systems. In other words, for $S_n < S_n^*$, it is assumed that the NAPL does not form a continuously connected phase, and hence some air is directly in contact with the water phase. Hence, a reasonable model for three phase NAPL relative permeability should have

$$
k_{rn} = 0 \qquad \text{if } S_n < S_n^*.
$$

If this turns out not to be the case, then it will probably be necessary to modify Stone's method for $S_n < S_n^*$. However, all the relative permeabilities used in this work satisfied equation (21), since the NAPL relative permeability was constrained to be nonnegative.

Consider a fully implicit solution of (18)-(19). We will demonstrate that the choice of the upstream point (eq. (20)) implies that the solution of the discrete equations must have saturations in the physical range. For simplicity, we will ignore source/sink

terms in the following argument; their inclusion merely adds algebraic complication. The discrete water conservation equation (18) can be written (dropping the q_w term):

$$\frac{V_i}{\Delta t}[(\phi\rho_w S_w)_i^{N+1}-(\phi\rho_w S_w)_i^N]=\sum_{j\in\eta_i}\{\max[0,\gamma_{ij}\psi_{w,ij}\rho_{wj}^{N+1}\lambda_w(S_{wj}^{N+1})] + \min[0,\gamma_{ij}\psi_{w,ij}\rho_{wi}^{N+1}\lambda_w(S_{wi}^{N+1})]\}. \tag{22}$$

Without loss of generality, the critical saturations can be taken to be zero, so that

$$\lambda_n=0 \quad \text{if } S_n<0, \qquad \lambda_w=0 \quad \text{if } S_w<0. \tag{23}$$

We proceed by demonstrating a contradiction. Given that initially

$$0\le S_{wi}^N\le 1, \tag{24}$$

assume that

$$S_{wi}^{N+1}<0. \tag{25}$$

Consequently, (22) becomes

$$\frac{V_i}{\Delta t}[(\phi\rho_w S_w)_i^{N+1}-(\phi\rho_w S_w)_i^N]=\sum_{j\in\eta_i}\max[0,\gamma_{ij}\psi_{w,ij}\rho_{wj}^{N+1}\lambda_w(S_{wj}^{N+1})]. \tag{26}$$

Assuming ϕ and ρ_w are constant, then (24) and (25) imply that the left-hand side of (26) must be negative, while the right-hand side of (26) is nonnegative. Consequently, (25) must be false, so that

$$S_{wi}^{N+1}\ge 0 \tag{27}$$

regardless of timestep size. A similar argument shows that

$$S_{ni}^{N+1}\ge 0. \tag{28}$$

The inversion of the liquid-air capillary pressure curve defines S_a (eq. (7)). This inversion always uses a constraint so that [11]

$$S_{ai}^{N+1}\ge 0. \tag{29}$$

From the fact that the saturations sum to one, it follows that

$$0\le S_{ni}^{N+1}\le 1, \qquad 0\le S_{wi}^{N+1}\le 1, \qquad 0\le S_{ai}^{N+1}\le 1. \tag{30}$$

If a node is in the saturated state ($S_{ai}=0$), then the form of (18)-(19) is identical to the form of the equations used in [14]. The results of [14] can be briefly stated as follows: if certain conditions hold, then monotonicity criteria can be determined, and this criteria can be used to determine if a node should be in an implicit or explicit state. If the conditions do not hold, the node is set implicit. This guarantees a bounded solution in the physically meaningful range. The monotonicity conditions were derived solely by algebraic means. Precisely the same results apply here if due regard is taken of the sign of γ_{ij} in determining the upstream point.

If a node is unsaturated ($S_{ai}>0$), then an extremely cautious approach is used. Any node with $S_{ai}>0$, $S_{ni}>0$ is set implicit.

Taken together, the adaptive implicit approach described above ensures that all saturations are in the physical range [0, 1] regardless of timestep size. The above arguments are independent of mesh size.

These arguments can be extended to nodes having source/sink terms, provided the form of these terms is as discussed in [10]. Briefly, whenever a phase is withdrawn from a node, the phase withdrawal rate is a function of the mobility (λ_l) for that phase. Provided equation (23) holds, then phase withdrawal "shuts-off" as the phase saturation tends to zero, and hence the phase saturation cannot be less than zero.

4. Positive transmissibility condition. The condition for positive transmissibilities is (from (13) and (18)-(19)):

$$\gamma_{ij} \geq 0 \qquad \forall i, j, \quad i \neq j. \tag{31}$$

It is instructive at this point to consider some typical two-dimensional basis functions. One possible choice consists of C^0 piecewise bilinear functions defined on quadrilaterals. However, this is a poor choice if condition (31) is desired.

To see this, consider a problem where $\mathbf{K}$ is constant. The algebraic influence of $\mathbf{K}$ in equation (14) can be eliminated by defining an (x', y') coordinate system such that $\mathbf{K}' = \mathbf{I}$ in this new system. This transformation is similar to the transformation used by D'Azevedo and Simpson [28]. This is always possible since $\mathbf{K}$ is symmetric positive definite. If the elements are rectangles with

$$\Delta x' = x'_{i+1} - x'_i, \qquad \Delta y' = y'_{j+1} - y'_j, \tag{32}$$

then some straightforward algebra shows that (31) is violated unless

$$\max\left\{\frac{\Delta x'}{\Delta y'}, \frac{\Delta y'}{\Delta x'}\right\} \leq \sqrt{2}. \tag{33}$$

For definiteness, if we assume that

$$\mathbf{K} = \begin{bmatrix} K_x & 0 \\ 0 & K_y \end{bmatrix}$$

and that x is the horizontal direction, and y is the vertical direction (depth), then (33) implies (in the original (x, y) coordinate system):

$$\frac{\Delta x}{\Delta y} \leq \left(\frac{2K_x}{K_y}\right)^{1/2}. \tag{34}$$

In typical contamination problems, condition (34) is not met by a large margin, since horizontal distances are typically an order of magnitude larger than vertical distances.

Another possible choice for C^0 basis functions are piecewise linear functions defined on triangles. Again, consider the (x', y') system so that $\mathbf{K}' = \mathbf{I}$. An arbitrary arrangement of nodes may be used. For an interior edge, as shown in Fig. 1, some simple geometry shows that condition (31) is satisfied if and only if [35]:

$$\alpha + \beta \leq \pi. \tag{35}$$

For a boundary edge, (31) requires that the angle opposite the edge is less than $\pi/2$. (All these angles are measured in the (x', y') coordinate system.)

In practical problems, the internal nodes are often placed to correspond with layers of rock or soil having different properties. Therefore, it is inappropriate to attempt to satisfy (31) by movement of internal nodes. We would also like to avoid adding extra internal nodes.

However, (35) has an interesting relation to a Delaunay triangulation [25]-[28]. A Delaunay triangulation satisfies the empty circle criteria: the circumcircle of each

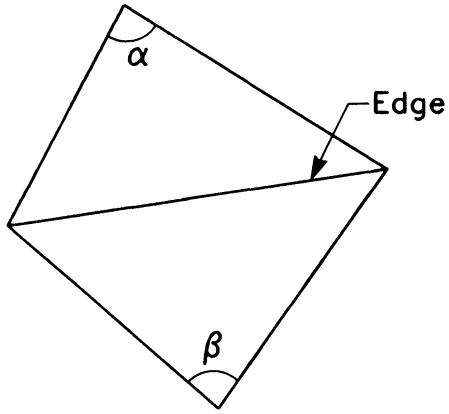


FIG. 1. *Quadrilateral formed by the two internal triangles which have a common edge.*

triangle contains no other nodes in its interior. Given a valid triangulation [36] (a triangulation that tiles the region without gaps or overlaps) in a convex region, then the triangulation can be converted to a Delaunay triangulation using a sequence of local edge swaps [26], [28]. Each interior edge of the triangulation is examined. If it forms part of a convex quadrilateral (Fig. 1), then the two circumcircles of the triangles are examined. If either of the two circumcircles contain the fourth vertex of the quadrilateral, then the diagonal of the quadrilateral is swapped (Fig. 2). This resulting configuration must then satisfy the local empty circle condition, or local optimality condition [26], [28]. This sequence of local edge swaps eventually converges [26], [28], so that each internal edge is locally optimal. Theorem 2.1 of [26] states that all internal edges of a triangulation are locally optimal if and only if it is a Delaunay triangulation. Hence this local edge swapping procedure converges to a Delaunay triangulation.

From Theorem 1 of [28], it follows that the local optimality condition is equivalent to (35) for interior edges. Consequently, the above edge swapping procedure can be stated in geometric terms: given a valid triangulation, an edge such as AC (Fig. 2) is examined. If the sum of the two angles opposite the edge is greater than π, then this edge is replaced by DB. This edge swap can, of course, only be carried out if ABCD (Fig. 2) is convex. If ABCD is not convex, then necessarily $(\alpha+\beta)$ (Fig. 1) must be

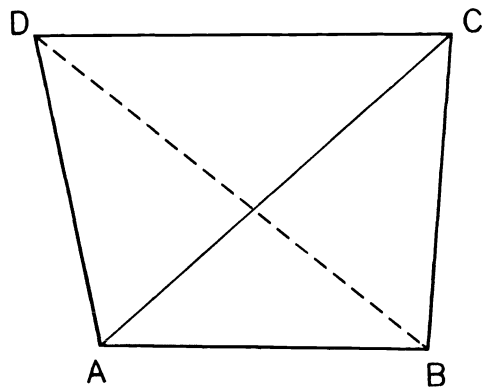


FIG. 2. *Local edge swap procedure: replace AC by DB.*

less than π. In general, this will disturb the edges of the quadrilateral ABCD, and these edges must be reexamined. However, each edge swap required by local optimality causes a strict advance through an ordering of all possible triangulations, and hence this process eventually terminates [26], [28].

Note that in a convex region, no addition or movement of nodes is required to convert an arbitrary valid triangulation into a Delaunay triangulation.

For our purposes, we generalize the edge swap criteria as follows: an edge AC is examined, and the transmissibility γ_{AC} is determined by carrying out the integration in (15). If γ_{AB} is negative, then replace this edge by DB, and continue. If the region is convex, and $\mathbf{K}$ is constant, this procedure is equivalent to finding the Delaunay triangulation in the (x', y') plane where $\mathbf{K}' = \mathbf{I}$, and hence must converge. (Note that a valid edge swap in the (x', y') plane is a valid edge swap in the (x, y) plane since convex polygons in the (x', y') plane are convex polygons in the (x, y) plane.)

In general, for nonconvex regions or nonconstant $\mathbf{K}$, this procedure may not produce a triangulation such that (31) is satisfied for all internal edges. However, since most regions which arise in practice can be thought of as a union of convex regions with constant $\mathbf{K}$ plus some interface regions, this local edge swap procedure should tend to minimize the number of internal edges having negative transmissibilities. In practice, to avoid any possible cycling, we limit the number of edge swaps allowed.

Note that this equivalence of (31) with a Delaunay triangulation is true only for interior edges in the transformed plane when the permeability tensor is constant. In general, a Delaunay triangulation of the physical plane will not ensure that (31) is satisfied, even for interior edges. The identification with a Delaunay triangulation is made only in the special case of convex regions with constant $\mathbf{K}$. In this case, we know the local edge swapping procedure, defined in terms of the positive transmissibility condition, will converge to a triangulation such that (31) is satisfied for all interior edges [26], [28], for a fixed placement of nodes. In fact, in three dimensions, the Delaunay empty sphere criteria is not equivalent to (31) [37]. Consequently, in the general case, equation (31) must be used as a criteria for triangulating, not the Delaunay empty circle condition [28].

Even for convex regions with constant $\mathbf{K}$, it is possible to end up with edges on the boundaries which have negative transmissibilities. (There is no possible edge swap for a boundary edge.) We can try to eliminate this problem by adding a boundary node as in Fig. 3. Suppose edge AB has $\gamma_{AB} < 0$. This implies that in the $(x'y')$ plane,

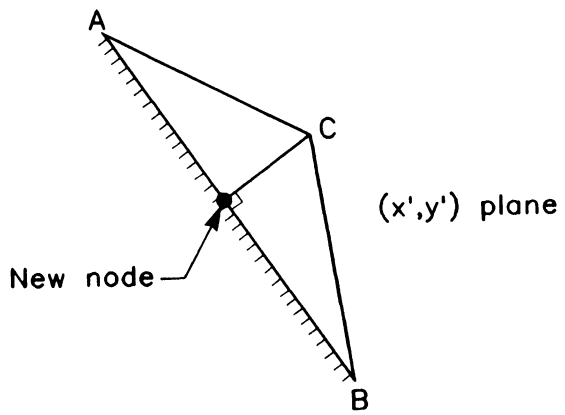


FIG. 3. *Boundary node addition procedure for a boundary edge.*

the angle opposite edge AB is greater than $\pi/2$. A new node is added at the intersection of AB with the perpendicular to AB drawn from C (this is all carried out in the (x', y') plane). This procedure will be described as boundary node addition.

It is interesting to observe that in two dimensions, a Delaunay triangulation has the following properties:

(1) Maximizes the minimum angle at all vertices;

(2) Has positive internal transmissibilities (eq. (31)) if $\mathbf{K} = \mathbf{I}$;

while in three dimensions, a Delaunay triangulation has *neither* of the properties [27], [37]:

(1) Maximizes the minimum solid angle at all vertices;

(2) Positive internal transmissibilities if $\mathbf{K} = \mathbf{I}$.

It seems extremely unlikely that any higher order basis functions will give rise to any reasonable criteria which will satisfy (23). Consequently, in the following, we will restrict attention to two-dimensional linear basis functions defined on triangles.

5. Nonphysical discrete flow example. In order to demonstrate that nonphysical fluid flow can result from negative transmissibilities, consider the coarse grid discretization shown in Fig. 4. This represents a cross-sectional problem, with boundary conditions selected so that the water flow is from upper left to lower right. The permeability tensor was taken to be the identity matrix. Initially, the region is completely filled with water (saturated). A coarse 4×4 grid is used.

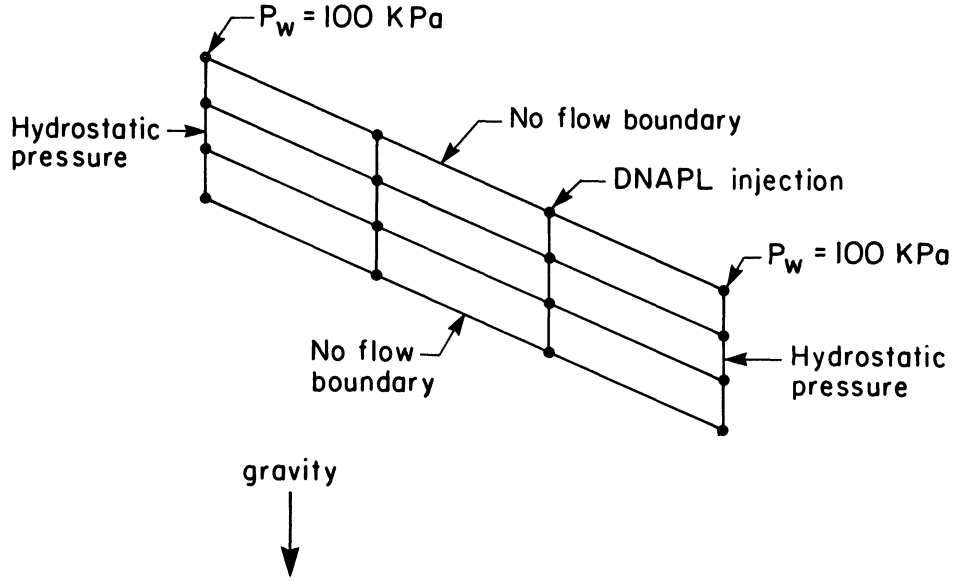


FIG. 4. *Domain and node locations for the coarse grid, nonphysical flow example.*

DNAPL (NAPL with density greater than water) is injected at the node shown in Fig. 4. Physically, we expect that the DNAPL will sink down to the bottom of the region, and then move towards the right-hand boundary.

An initial triangulation of the region shown in Fig. 4 was developed by dividing each quadrilateral into two triangles by joining two diagonally opposite nodes. In order to develop a pathological case, the diagonal was selected so that diagonally opposite nodes generated negative transmissibilities. We will refer to this triangulation

as the negative transmissibility triangulation (28 percent negative transmissibilities). A second triangulation was generated from this initial triangulation using the edge swapping algorithm described in the previous section. In order to get a discretization having all positive transmissibilities, it was necessary to also use the boundary node addition procedure (i.e., some extra nodes were added at various points along the boundary of the domain). We refer to this second triangulation as the positive transmissibility triangulation.

Figure 5(a) shows the steady state DNAPL saturation values at the grid nodes for the negative transmissibility triangulation. Figure 5(b) shows the steady state saturations at the same nodes for the positive transmissibility triangulation.

Figure 5(b) shows the expected result; the DNAPL sinks down and moves to the right, under the action of gravity and the ambient water flow. The maximum DNAPL saturation occurs at the lower boundary. Note that the DNAPL saturation is identically zero for all nodes to the left of the injection node.

However, Fig. 5(a) shows a completely unrealistic result; some DNAPL has "floated" upward and to the left. Consequently, all nodes have an appreciable DNAPL saturation. This nonphysical motion is a result of the negative transmissibility which

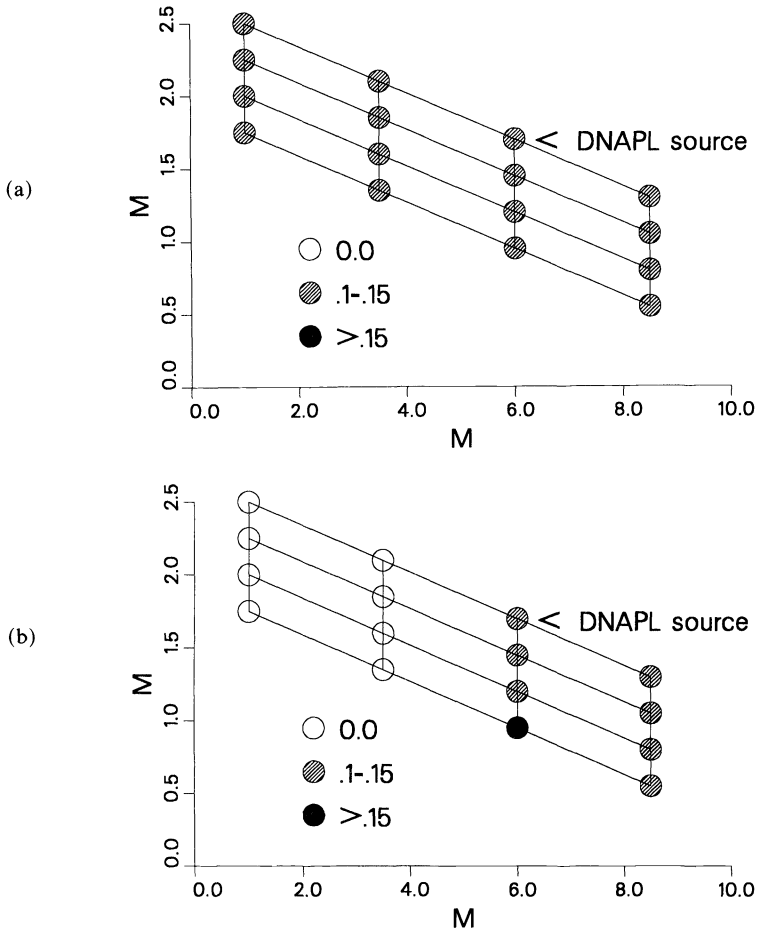


FIG. 5. *Steady state* DNAPL *saturations at the node positions for the example of Fig.* 4. (a) 28 *percent negative transmissibilities.* (b) *All positive transmissibilities.*

connects the upper left node of each quadrilateral to the lower right node. This causes a nonphysical zigzag "floating" motion towards the upper left.

This nonphysical behaviour is solely a result of the geometry of the triangulation, not the coarseness of the grid. This example is clearly a pathological case, but does serve to demonstrate that any triangulation which has negative transmissibilities can exhibit locally nonphysical solutions.

However, if the number of negative transmissibilities is a small fraction of the total number of edges, then we expect that this local nonphysical behaviour will not affect the macroscopic solution too greatly. This will be shown in the next two sections.

Of course, if the fraction of negative transmissibilities is large, and in particular if there is a path of negative transmissibilities from one boundary of the domain to another boundary, as in the above example, then we can expect macroscopically incorrect results.

6. DNAPL contamination in a homogeneous region. The first example consists of a NAPL contaminant which is denser than water (DNAPL specific gravity = 1.2). The homogeneous region for this problem is shown in Fig. 6. In this particular example, we expect that the DNAPL will sink through the unsaturated zone above the water table, through the saturated zone, until encountering the no-flow bottom boundary. The contaminant should then flow along the boundary until leaving the region through the constant pressure edges (the pressure at the side boundaries is set to the hydrostatic

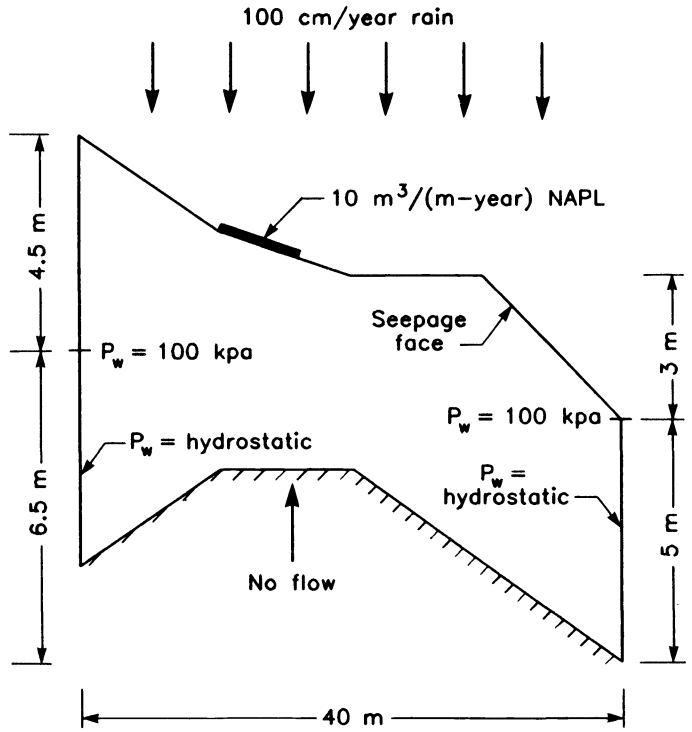


FIG. 6. *Domain for* DNAPL *and* LNAPL *contaminant examples in a homogeneous region.*

pressure of water, as shown in Fig. 6). A detailed description of the data for this example is given in the Appendix.

Two levels of grids will be used in the following: a coarse grid and a fine grid which is constructed from the coarse grid by joining the midpoints of the sides of the coarse grid triangulation (see Fig. 7).

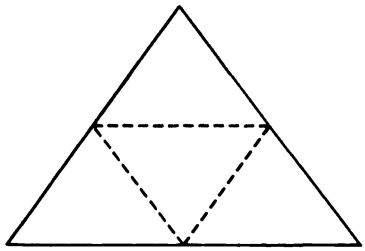


FIG. 7. *Method for constructing a fine triangulation (dotted plus solid lines) from a coarse triangulation (solid line only).*

In order to investigate the effect of a small number of negative transmissibility edges, we will also use two approaches to producing a coarse grid triangulation. The first technique uses local edge swapping and boundary node addition to ensure that all edges have positive transmissibility. The second technique does not add boundary nodes.

Figure 8(a) shows the fine grid without boundary node addition (1089 nodes) and Fig. 8(b) shows the fine grid with boundary node addition (1161 nodes). The grid in Fig. 8(a) has 16 out of 2379 edges with negative transmissibilities. These negative transmissibilities all occur on either the top or bottom boundary, near the location of the extra boundary nodes (near solid colour in Fig. 8(b)). Figure 8(b) has no negative transmissibility edges.

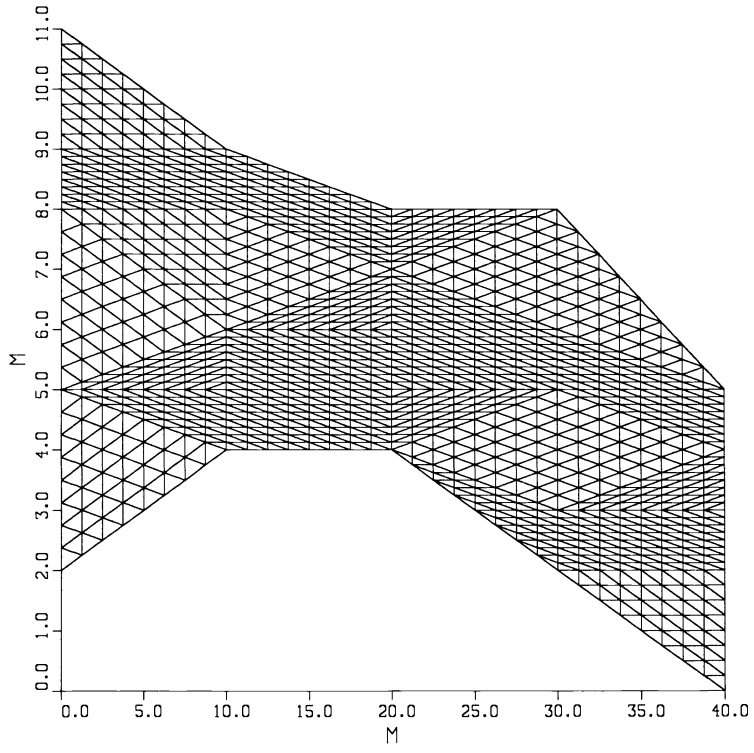
Consequently, any difference between computations on the two grids should be apparent when the DNAPL flows along the bottom edge, between 20 and 30 m from the origin. In particular, we expect to observe the phenomena shown in Fig. 9. If edge AB is a boundary edge, then physically a dense fluid will flow along the path from A to B. However, if AB has a negative transmissibility, then a dense fluid must first flow from A to C, and then from C to B. Of course, for a very fine grid, we expect that this nonphysical flow path will have a vanishingly small effect. However, it remains to be seen if this nonphysical behaviour has an observable effect on coarse grids.

The contamination problem of Fig. 6 was first initialized in an arbitrary state, and then allowed to run for one year with no DNAPL injection. This produced a steady state water saturation profile. After the first year, DNAPL was injected at the site indicated for one year (see Fig. 6).

Figure 10 shows the DNAPL saturation values after one year of injection, for both coarse grids. Figure 10(a) shows results for the grid with negative transmissibilities along the bottom boundary, while Fig. 10(b) shows the results for the coarse grid with all positive transmissibilities. The somewhat ragged appearance of the interface between zones having different S_n levels is magnified by the shading algorithm, which shades the control volumes associated with each node.

A detailed examination of the results reveals that the nonphysical flow path phenomena described in Fig. 9 does occur for the grid with negative transmissibilities.

(a)



(b)

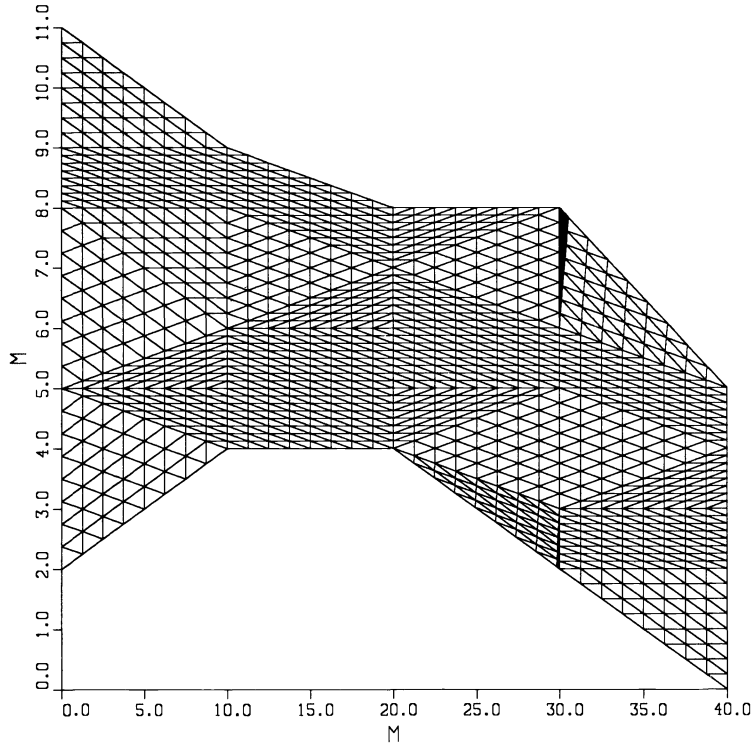


FIG. 8. *Triangulation for domain of Fig.* 4. (a) 1089 *nodes* (*some negative transmissibilities*). (b) 1161 *nodes* (*all positive transmissibilities*). *Solid colour indicates high density triangulation near extra boundary nodes.*

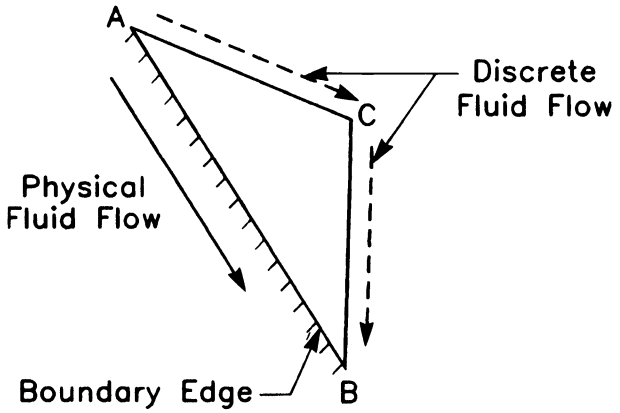


FIG. 9. *Physical flow path (AB) and discrete flow path (AC − CB) when AB has a negative transmissibility.*

This small effect can be seen by comparing Fig. 10(a) to Fig. 10(b) (along the lower boundary from 20 to 30 m). Figure 11 shows the fine grid results for both types of grids. As expected, the interface zones become smoother as the grid is refined, and the graphs (Fig. 11(a) and 11(b)) become quite similar.

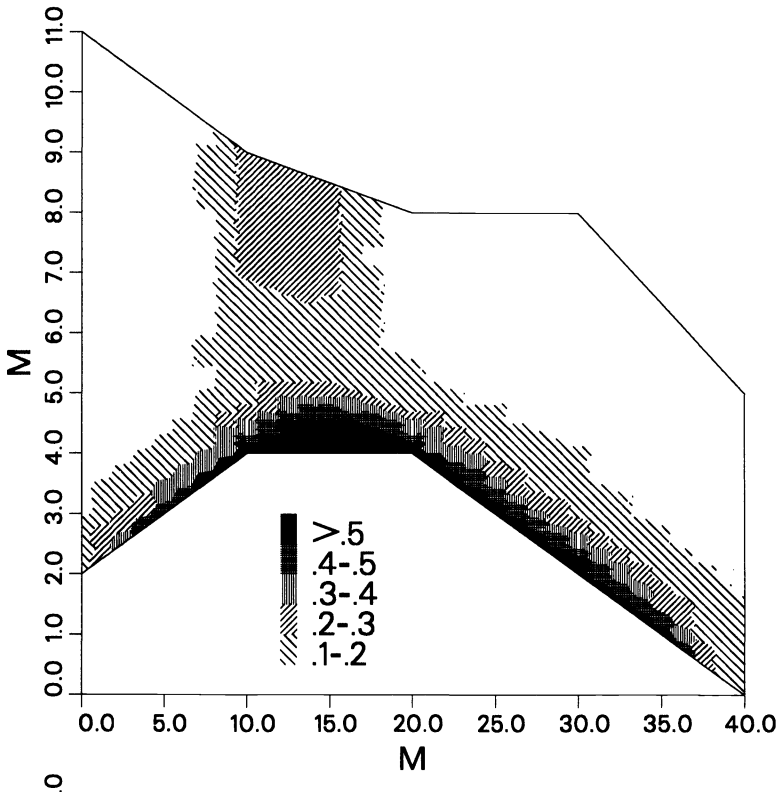
A more quantitative description of the difference between computations for the two grid types (as shown in Fig. 8) is given in Table 1. This table shows the amount of DNAPL remaining in the system after one year. The discretization error for this macroscopic quantity is about 3 percent, which would certainly be adequate accuracy in any practical problem. Note that the differences in DNAPL retention for the grids of Figs. 8(a) and 8(b) are less than the level of discretization error.

Clearly, there is a small observable effect on computational results due to negative transmissibilities. However, if the number of edges having negative transmissibilities is small, then the effect on macroscopic parameters is also small. Either computation using the grid of Fig. 8(a) or Fig. 8(b) is acceptable. However, since the boundary node addition procedure sometimes adds nodes which are quite close to existing nodes, this generally causes more Newton iterations. (Closely spaced nodes will produce nodes with small effective volumes.) Consequently, more computational work is required for a grid where (31) is always true. Of course, this problem of closely spaced nodes can be alleviated by some type of node movement, but, as discussed in the Introduction, we have excluded this possibility from consideration.

7. LNAPL contamination in a homogeneous region. As a second example, the same problem as defined in § 6 was run with a specific gravity of NAPL set to .95. In this light NAPL (LNAPL) scenario, we expect that the LNAPL will tend to float on the water table, and have less tendency to penetrate the aquifer, compared to the DNAPL case.

Figure 12 shows the NAPL saturation values after one year of injection. Figure 12(a) corresponds to the coarse grid version of the grid shown in Fig. 8(a), while Fig. 12(b) corresponds to the coarse grid version of Fig. 8(b). As expected, the highest saturation values "sit" on top of the water table. However, the contaminant does penetrate quite far into the aquifer. This is due to the head of NAPL in the unsaturated zone (the capillary fringe is quite narrow). Figure 13 shows the fine grid results corresponding to Fig. 12. Examination of these graphs shows that there is little difference between Figs. 12(a) and 12(b), and between Figs. 13(a) and 13(b). This indicates, once again, that a small number of edges with negative transmissibilities is quite acceptable.

(a)



(b)

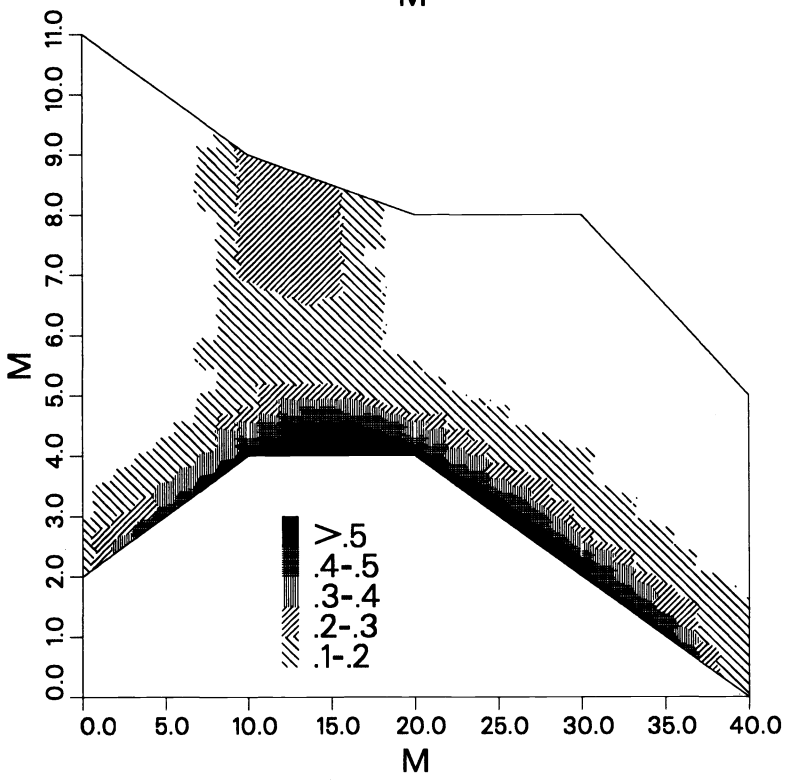


FIG. 10. DNAPL *saturation values after one year (coarse grid).* (a) *Some negative transmissibilities* (289 *nodes*). (b) *All positive transmissibilities* (309 *nodes*).

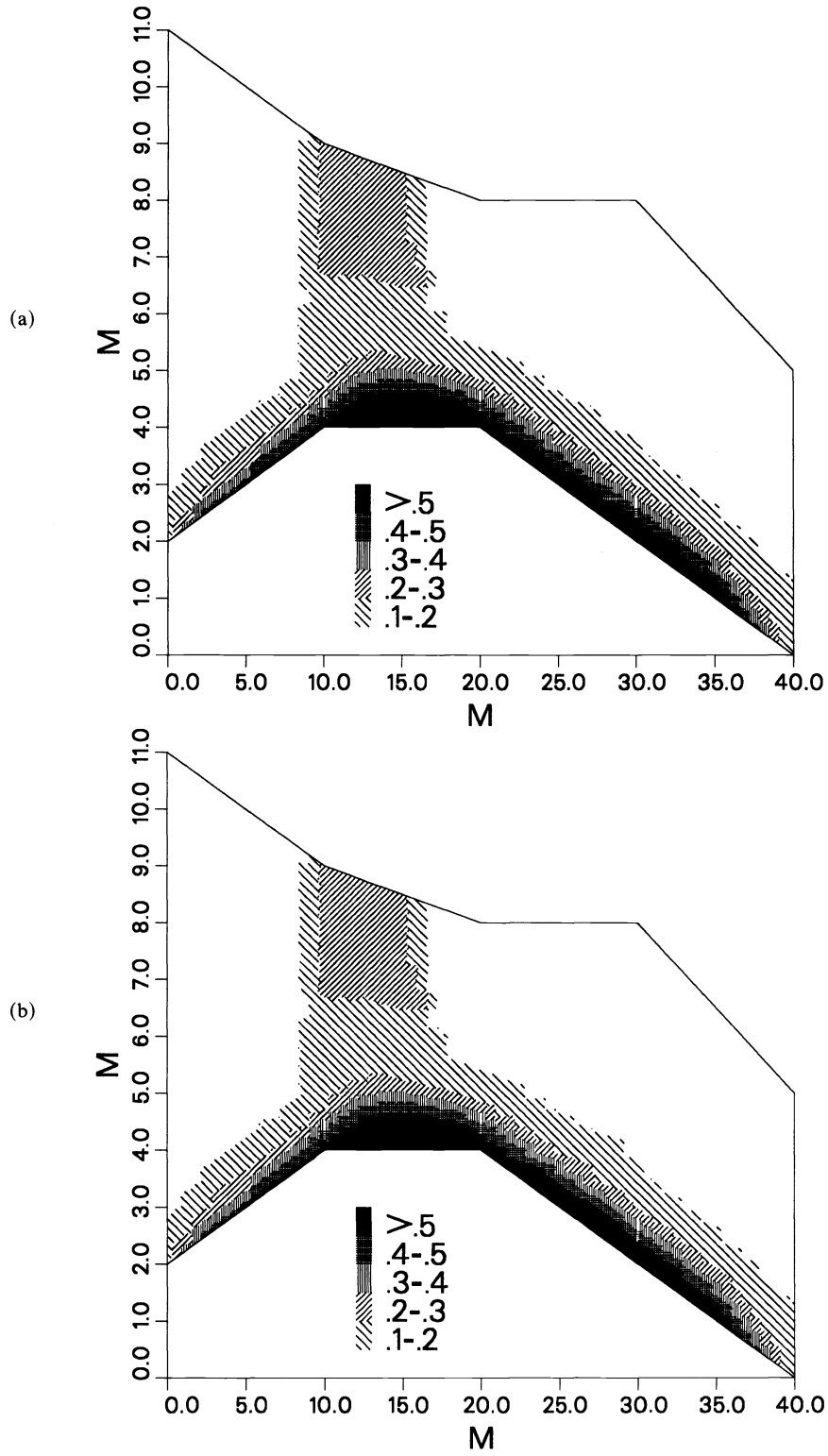


FIG. 11. DNAPL *saturation values after one year (fine grid)*. (a) *Some negative transmissibilities* (1089 *nodes*). (b) *All positive transmissibilities* (1161 *nodes*).

TABLE 1

Amount of DNAPL *remaining in the system after one year* (10 m^3 *injected, homogeneous example*).

	DNAPL volume (m^3)	
	Coarse grid	Fine grid
Some negative transmissibilities (Fig. 8(a))	7.26	7.05
All positive transmissibilities (Fig. 8(b))	7.38	7.13

8. DNAPL contamination in a heterogeneous region. An example of a heterogeneous region is shown in Figs. 14 and 15. The high-P_c region in Fig. 15 has nodes which use a different capillary pressure table than used elsewhere. This high-P_c zone is considered to consist of a porous medium with a large NAPL-water capillary pressure. Regions with a large capillary pressure can be effective at blocking NAPL flow [38].

The absolute permeability is isotropic everywhere, and $K = 1000$ md except in the designated zones shown in Fig. 15, where $K = 0.1$ md. Complete details of the data for this problem are given in the Appendix.

The fine grid discretization for this problem is shown in Fig. 16. The edge swapping procedure produced a final discretization with no negative transmissibilities.

This example was initially allowed to run for one year with no DNAPL injection, in order to obtain a steady state profile. DNAPL was injected for one year, and then DNAPL injection ceased while water injection (rainfall) continued for another two years.

The DNAPL saturation profiles are shown in Fig. 17 for both coarse (400 nodes) and fine (1521 nodes) grids. It is clear that heterogeneous systems result in very complex contaminant motion. Since heterogeneous systems are typical of actual sites [2], [3], [38], DNAPL contaminant will tend to be dispersed over large areas. This will make site remediation and cleanup very difficult, due to the large volumes contaminated with low saturation (and hence almost immobile) DNAPL.

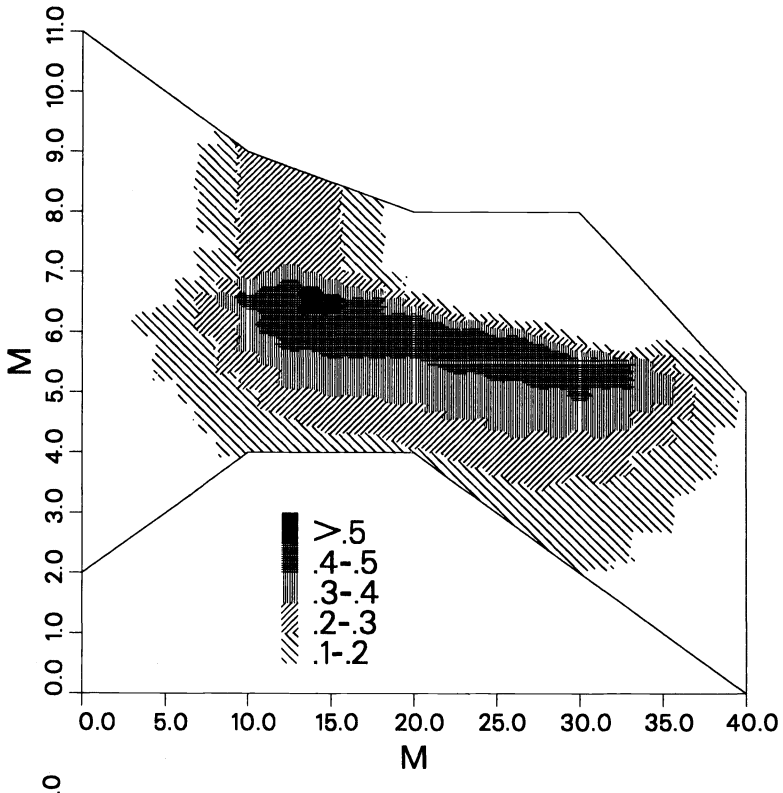
Note that the high S_n profile ($S_n > 0.5$) is quite jagged at the bottom of the high-P_c zone in Fig. 17(b). This is because all the input data are specified on the coarse grid, and interpolated onto the fine grid. In the case of the P_c tables, the default table was used for fine grid nodes unless all coarse grid nodes neighbouring fine grid nodes were in the high P_c region. This had the effect of magnifying the staircase pattern present on the coarse grid.

9. Effect of negative transmissibilities on computational efficiency. In order to test the effects of triangulations having a large fraction of negative transmissibilities, the three previous runs were repeated using an initial triangulation with no edge swapping. The coarse grids were used in all cases. For a given run, the node placement and boundary edge locations were identical. Only the triangle incidences were different.

In Table 2, the computational statistics for these runs (large fraction of negative transmissibilities) are compared to the statistics for the same runs with triangle incidences which have a small fraction of negative transmissibilities. Total inner iterations refers to the total number of iterations required for solution of the Jacobian.

Table 2 shows that for the DNAPL simulations, the total number of Newton iterations required is about 50–100 percent more for triangulations with a large fraction (greater than 20 percent) of negative transmissibilities. Careful examination of the

(a)



(b)

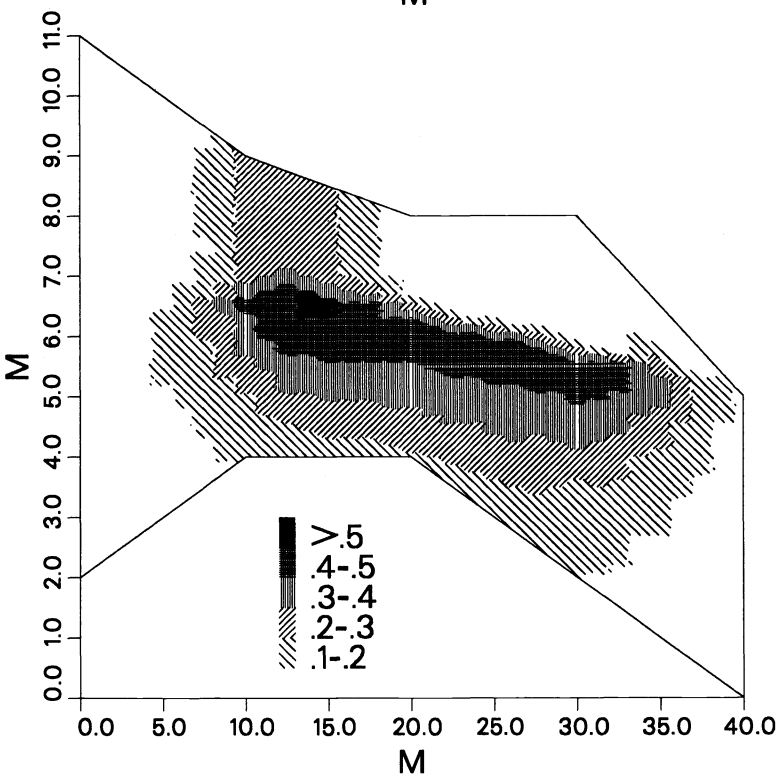


FIG. 12. LNAPL *saturation values after one year* (*coarse grid*). (a) *Some negative transmissibilities* (289 *nodes*). (b) *All positive transmissibilities* (309 *nodes*).

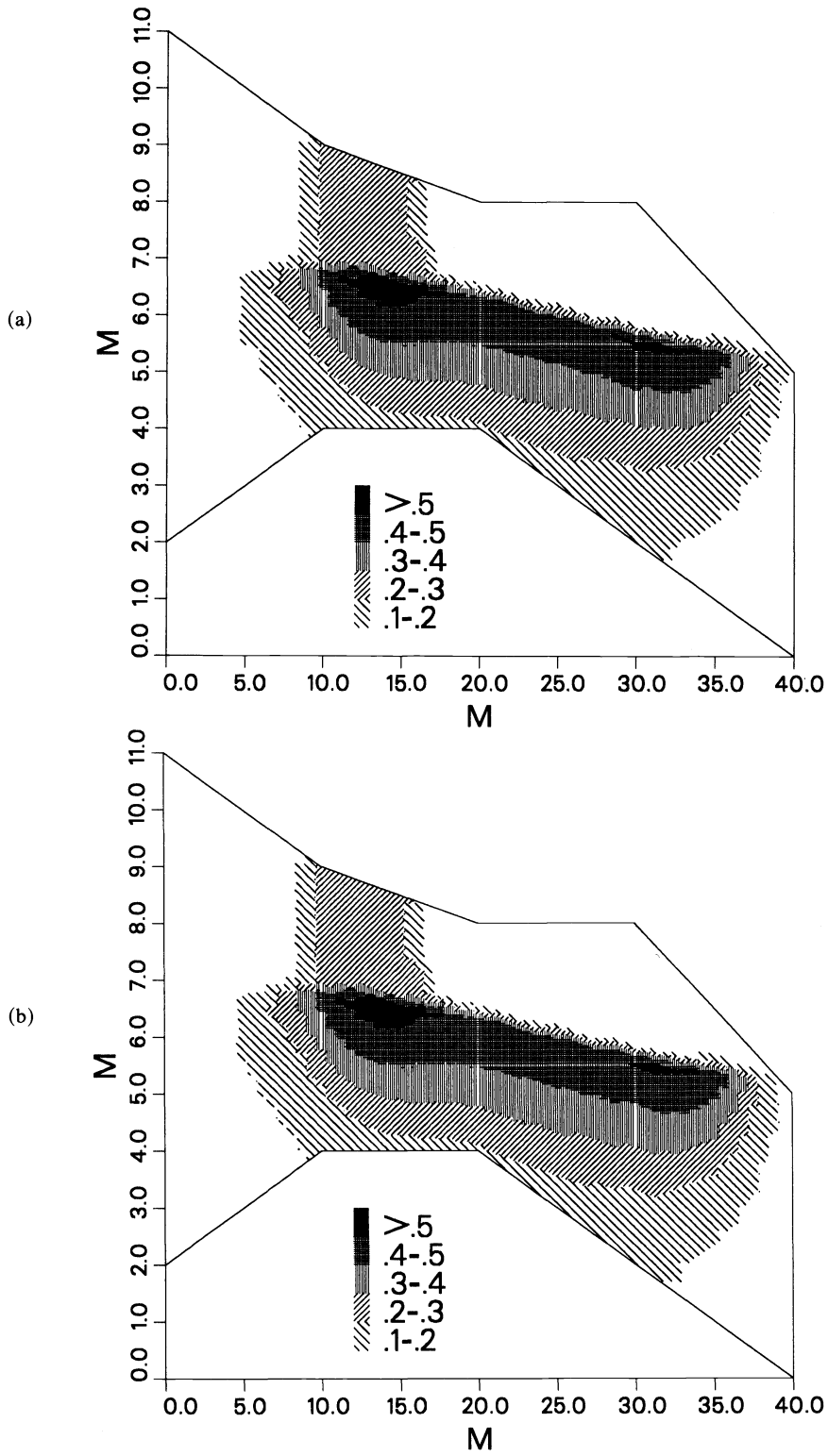


FIG. 13. LNAPL *saturation values after one year (fine grid)*. (a) *Some negative transmissibilities* (1089 *nodes*). (b) *All positive transmissibilities* (1161 *nodes*).

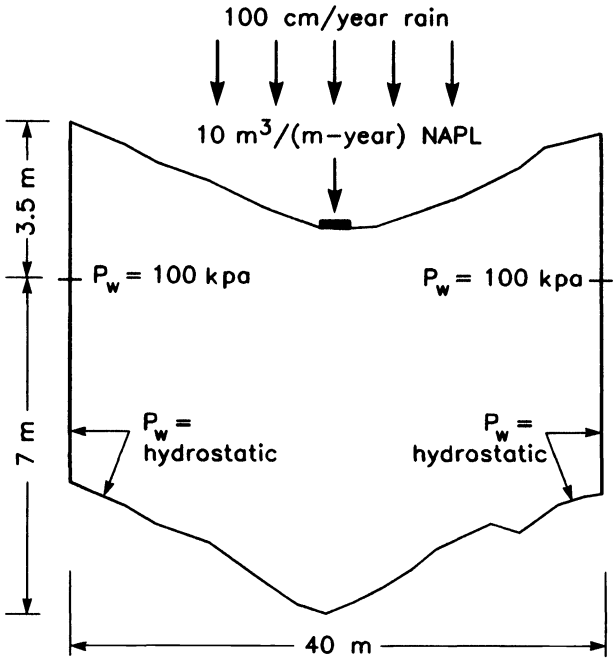


FIG. 14. *Domain of* DNAPL *heterogeneous problem.*

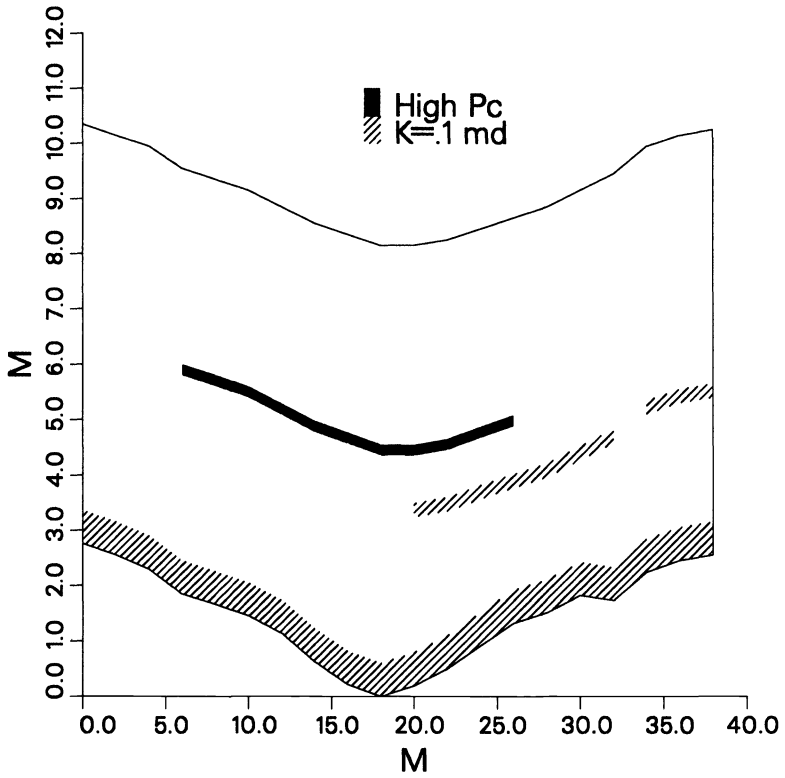


FIG. 15. *Details of heterogeneities for Fig.* 14.

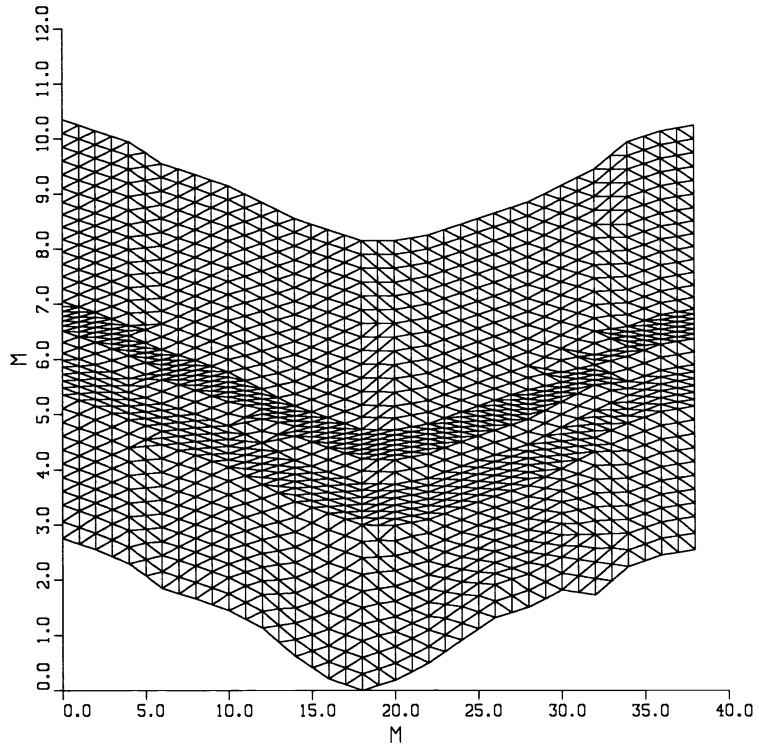


FIG. 16. *Fine grid triangulation for Fig.* 14 (1521 *nodes*).

Newton iterations revealed an interesting physical explanation for this phenomenon. Initially, the DNAPL would flow downward (under the action of gravity) along an edge with positive transmissibility. However, if the receiving node was connected to a node above it by an edge with negative transmissibility, the DNAPL would move back up along this edge (in the negative physical flow direction). The next Newton iteration attempts to correct this by sending more fluid down the positive transmissibility edge, which in turn causes more dense fluid to rise, and so on. This causes the Newton iteration to cycle, and may cause nonconvergence unless the timestep size is reduced. This effect is, of course, absent from nodes having all positive transmissibilities.

For the LNAPL simulation, the LNAPL does not penetrate the water table as much as the DNAPL. Hence the Newton iterations for the LNAPL run with a large fraction of negative transmissibilities are not greatly increased (see Table 2). However, the total number of inner iterations required for the Jacobian solve increased by about 50 percent. The iterative solver used here was a reduced system /ILU type [39]. In single phase regions, the negative transmissibilities will produce non-M matrices for the pressure, which can be expected to cause difficulties for the iterative solver.

The macroscopic saturation profiles for simulations with a large fraction of negative transmissibilities were similar to the saturation profiles for the corresponding simulations with a small fraction of negative transmissibilities.

Experiments were carried out on various grids having heterogeneous properties. In all cases, the edge swapping procedure and boundary node addition produced triangulations with only a small fraction (less than 2 percent) of negative transmissibilities.

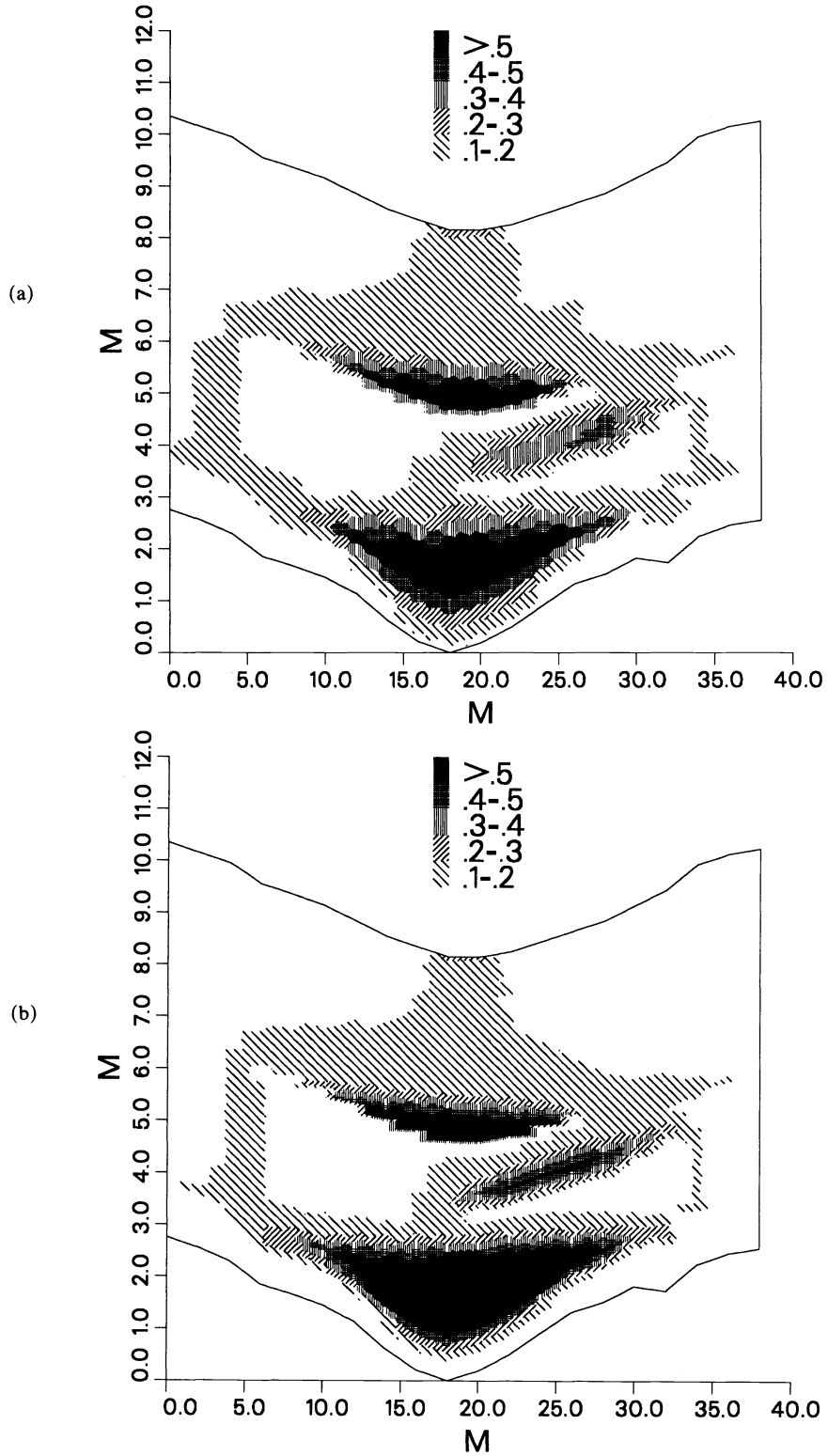


FIG. 17. DNAPL *saturation values, heterogeneous problem, after two years.* (a) *Coarse grid* (400 *nodes*). (b) *Fine grid* (1521 *nodes*).

TABLE 2

Effect of negative transmissibilities on computational efficiency.

Run	Fraction of negative transmissibilities	Total Newton iterations	Total inner iterations	Average degree of implicitness
DNAPL (homogeneous)	0.0	535	2853	0.36
	0.31	1218	5559	0.51
DNAPL (heterogeneous)	0.01	457	3923	0.29
	0.26	658	9026	0.41
LNAPL	0.01	450	4082	0.32
	0.26	465	6149	0.39

10. Conclusions. Previous attempts to use control volume-finite element methods for unstructured grid multiphase flow problems did not ensure that positive transmissibilities resulted from a given node placement. This can have undesirable consequences, since the discrete fluid motion follows a nonphysical flow path. It is also possible that nonphysical local maxima and minima may appear in the pressure field.

In two dimensions, bilinear basis functions defined on quadrilaterals are inappropriate for geometries of interest. However, for convex regions with a constant permeability tensor, use of C^0 linear triangular basis functions, combined with the edge swapping algorithm suggested in this work, guarantees that all internal edges have positive transmissibilities. This procedure does not require any node movement. Possible negative transmissibilities at boundary edges can be removed by adding suitably placed boundary nodes.

In the general case of nonconvex, heterogeneous problems which arise in practice, it is probably not possible to eliminate all negative transmissibilities without node movement. However, various tests indicated that the edge swapping algorithm reduced the occurrence of negative transmissibility edges to a small fraction of the total number of edges. In all cases (positive or negative transmissibilities) the selection of the upstream point ensures that the solution of the implicit discrete equations forces the saturations to lie in the physically relevant range.

It is possible to generate a simple pathological case where negative transmissibilities cause completely unrealistic behaviour (dense fluids "float," and light fluids "sink"). However, in practice, if the number of negative transmissibilities is small, and these transmissibilities are restricted to a finite number of disjoint regions, then it is expected that the macroscopic effect of the local nonphysical fluid motion will be small.

TABLE 3

Physical data for the homogeneous examples. The same data were used for the heterogeneous example, except where indicated in Fig. 15.

Absolute permeability ($K_x = K_y = K_z$)	$10^{-2}\ \mathrm{m}^2$
Porosity, ϕ	0.3
Densities	$\rho_w = 1000\ \mathrm{Kg/m}^3$
	$\rho_n(\mathrm{DNAPL}) = 1200\ \mathrm{Kg/m}^3$
	$\rho_n(\mathrm{LNAPL}) = 950\ \mathrm{Kg/m}^3$
Viscosities	$\mu_w = 1.0$ cp
	$\mu_n = 1.0$ cp

TABLE 4

Relative permeability and capillary pressure tables, used in the homogeneous examples, and everywhere except as indicated in Fig. 15 for the heterogeneous example.

NAPL-water data

S_w	K_{rw}	K_{rn}	P_{cnw}(Kpa)
0.2	0.0	0.68	9.0
0.3	0.04	0.55	5.4
0.4	0.10	0.43	3.9
0.5	0.18	0.31	3.3
0.6	0.30	0.20	3.0
0.7	0.44	0.12	2.7
0.8	0.60	0.05	2.4
0.9	0.80	0.0	1.5
1.0	1.0	0.0	0.0

Liquid-air data

$(1-S_a)$	K_{ra}	K_{rna}	P_{can}(Kpa)	P_{caw}(Kpa)
0.2	0.64	0.0	9.0	6.6
0.32	0.46	0.0	3.0	4.5
0.40	0.36	0.0009	0.24	3.9
0.50	0.25	0.045	2.1	3.6
0.60	0.16	0.116	1.8	3.3
0.70	0.09	0.210	1.5	3.0
0.80	0.04	0.34	1.2	2.0
0.90	0.01	0.49	0.90	1.0
1.0	0.0	0.68	0.0	0.0

$S_n^* = 0.1$ (see eq. (6))

Test computations were carried out on some NAPL problems using a grid having all positive transmissibilities. Although there was a small observable effect on the computational results for nodes having negative transmissibilities, there was little difference in macroscopic parameters. As discussed above, this indicates that a grid with a small number of negative transmissibility edges is probably acceptable for many practical problems. However, even if the number of negative transmissibilities is small, it is possible that poor results can be obtained if the flow behaviour near these edges is important. Consequently, care should be taken to ensure that nodes which correspond to wells, for example, should not have any negative transmissibilities.

Tests with grids containing a large fraction (greater than 20 percent) of negative transmissibilities indicated that for DNAPL problems with rapid downward movement of dense fluids, the negative transmissibilities caused 50–100 percent more Newton iterations (compared to grids with a small fraction of negative transmissibilities). This problem was traced to the fact that the negative transmissibilities caused nonphysical fluid motion. Even for LNAPL problems, an increase of 50 percent in total inner iterations was observed on grids with a large fraction of negative transmissibilities.

Consequently, both in terms of quality of solution, and in terms of computational efficiency, it is desirable to use a discretization which contains a minimal number of negative transmissibilities.

TABLE 5

Relative permeability and capillary pressure data for the high-P_c zone in the heterogeneous problem (Fig. 15).

S_w	K_{rw}	K_{rn}	P_{cnw}(Kpa)
0.2	0.0	0.68	50.0
0.3	0.04	0.55	30.0
0.4	0.10	0.43	20.0
0.5	0.18	0.31	15.0
0.6	0.30	0.20	13.0
0.7	0.44	0.12	12.0
0.8	0.60	0.05	11.0
0.9	0.80	0.0	10.5
1.0	1.0	0.0	10.0

Liquid-air data

$(1-S_a)$	K_{ra}	K_{rna}	P_{can}(Kpa)	P_{cnw}(Kpa)
0.2	0.64	0.0	10.0	10.0
0.32	0.46	0.0	5.0	5.0
0.40	0.36	0.0009	4.5	4.5
0.50	0.25	0.045	4.0	4.0
0.60	0.16	0.116	3.5	3.5
0.70	0.09	0.210	3.0	3.0
0.80	0.04	0.34	2.0	2.0
0.90	0.01	0.49	1.0	1.0
1.0	0.0	0.68	0.0	0.0

$S_n^* = 0.1$ (see eq. (6))

The control volume-finite element technique described in this work can be applied to fully three-dimensional unstructured grids. In order to avoid negative transmissibilities, the obvious choice is C^0 linear tetrahedral elements. However, even for problems with a constant permeability tensor, a three-dimensional Delaunay triangulation [27] does not, in general, produce positive transmissibilities [37]. This indicates that generation of three-dimensional unstructured grids with positive transmissibilities will be considerably more difficult than the two-dimensional case.

Appendix. This Appendix contains the details of the input data for the homogeneous and heterogeneous examples.

Table 3 shows the data for the homogeneous DNAPL example. The same physical data were also used for the LNAPL case. Table 4 gives the relative permeability and capillary pressure data.

The heterogeneous problem used the same physical data as in Table 3. The relative permeability and capillary pressure data were the same as used for the homogeneous problems (Table 4), except for the high-P_c zones as indicated in Fig. 15. For the high-P_c zones, the data given in Table 5 were used.

In all cases, three phase relative permeabilities were computed using Stone's second method [29].

REFERENCES

[1] C. FAUST, *Transport of immiscible fluids within and below the unsaturated zone*, Water Res. Res., 21 (1985), pp. 587-596.

[2] M. OSBOURNE AND J. SYKES, *Numerical modelling of immiscible organic transport in the Hyde Park landfill*, Water Res. Res., 22 (1986), pp. 25-33.
[3] L. M. ABRIOLA AND G. F. PINDER, *A multiphase approach to the modelling of porous media contamination by organic compounds* 1. *Equation development*, Water Res. Res., 21 (1985), pp. 11-18.
[4] ———, *A multiphase approach to the modelling of porous media contamination by organic compounds* 2. *Numerical simulation*, Water Res. Res., 21 (1985), pp. 19-26.
[5] G. F. PINDER AND L. M. ABRIOLA, *On the simulation of nonaqueous phase organic compounds in the subsurface*, Water Res. Res., 22 (1986), pp. 109S-119S.
[6] A. L. BAEHR AND M. Y. A. CORAPCIOGLU, *A compositional multiphase model for groundwater contamination by petroleum products.* 2. *Numerical solution*, Water Res. Res., 23 (1987), pp. 201-213.
[7] D. P. HOCHMUTH AND D. K. SUNADA, *Groundwater model for two phase immiscible flow in coarse material*, Groundwater, 23 (1985), pp. 617-626.
[8] T. KUPPUSAMY, J. SHENG, J. PARKER, AND R. LEHHARD, *Finite element analysis of multiphase immiscible flow through soils*, Water Res. Res., 23 (1987), pp. 625-631.
[9] P. A. FORSYTH, *Simulation of nonaqueous phase groundwater contamination*, Adv. Water Res., 11 (1988), pp. 74-83.
[10] J. KALUARACHCHI AND J. PARKER, *An efficient finite element method for modelling multiphase flow*, Water Res. Res., 25 (1989), pp. 43-54.
[11] P. A. FORSYTH, *Comparison of the single phase and two phase numerical model formulation for saturated-unsaturated groundwater flow*, Comput. Methods Appl. Mech. Engrg., 69 (1988), pp. 243-259.
[12] P. S. HUYAKORN AND G. F. PINDER, *A new finite element technique for the solution of two-phase flow through porous media*, Adv. Water Res., 5 (1978), pp. 285-298.
[13] P. H. SAMMON, *An analysis of upstream differencing*, Soc. Pet. Engrg. J. Res. Engrg., 3 (1988), pp. 1053-1056.
[14] P. A. FORSYTH, *Adaptive implicit criteria for two-phase flow with gravity and capillary pressure*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 227-252.
[15] A. BERMAN AND R. PLEMNONS, *Nonnegative Matrices in the Physical Sciences*, Academic Press, New York, 1979.
[16] C. PRAKESH, *Examination of the upwind formulation in the control volume finite element method for fluid flow and heat transfer*, Numer. Heat Transfer, 11 (1987), pp. 401-416.
[17] G. SCHNEIDER AND M. J. RAW, *A skewed, positive influence coefficient upwinding procedure for control volume based finite element convection diffusion computation*, Numer. Heat Transfer, 9 (1986), pp. 1-26.
[18] C. S. RAFFERTY, M. PINTO, AND R. W. DUTTON, *Iterative methods in semiconductor device simulation*, IEEE Trans. Computer-Aided Design, (1985), pp. 462-471.
[19] V. DALEN, *Simplified finite-element models for reservoir flow problems*, Soc. Pet. Engrg. J., 19 (1979), pp. 333-343.
[20] T. POTEMPA, *A numerical model of two dimensional, two component, single phase miscible displacement in a porous medium*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 582-598.
[21] ———, *Mobility weighting in numerical reservoir simulation*, Soc. Pet. Engrg. J., 25 (1985), pp. 565-572.
[22] B. ROZEN, *A generalized finite volume discretization method for reservoir simulation*, paper SPE 18414, presented at the 10th SPE Symposium on Reservoir Simulation, Houston, TX, 1989.
[23] Z. HEINEMANN AND C. BRAND, *Modeling reservoir geometry with irregular grids*, paper SPE 18412, presented at the 10th SPE Symposium on Reservoir Simulation, Houston, TX, 1989.
[24] R. BANK AND D. ROSE, *Some error estimates for the box method*, SIAM J. Numer. Anal., 24 (1987), pp. 777-787.
[25] R. SIBSON, *Locally equiangular triangulations*, Comput. J., 21 (1978), pp. 243-245.
[26] B. JOE, *Delaunay triangular meshes in convex polygons*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 514-539.
[27] ———, *Delaunay triangulations from local transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 718-741.
[28] E. F. D'AZEVEDO AND R. B. SIMPSON, *On optimal interpolation triangle incidences*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1063-1075.
[29] K. AZIZ AND A. SETTARI, *Petroleum Reservoir Simulation*, Applied Science, London, 1979.
[30] O. AXELSSON AND V. BARKER, *Finite Element Solution of Boundary Value Problems*, Academic Press, Toronto, 1984.
[31] P. S. HUYAKORN, S. D. THOMAS, AND B. M. THOMPSON, *Techniques for making finite elements competitive in modeling flow in variably saturated porous media*, Water Res. Res., 20 (1984), pp. 1099-1115.

[32] P. S. HUYAKORN, E. P. SPRINGER, V. GUVANASEN, AND T. D. WADSWORTH, *A three dimensional finite element model for simulating water flow in variably saturated porous media*, Water Res. Res., 22 (1986), pp. 1790–1808.

[33] P. A. FORSYTH AND P. H. SAMMON, *Practical considerations for adaptive implicit methods in reservoir simulation*, J. Comput. Phys., 62 (1986), pp. 265–281.

[34] R. A. FREEZE AND J. A. CHERRY, *Groundwater*, Prentice-Hall, Englewood Cliffs, NJ, 1979.

[35] P. A. FORSYTH, *A control volume finite element method for local mesh refinement*, paper SPE 18415, presented at the 10th SPE Symposium on Reservoir Simulation, Houston, TX, 1989.

[36] R. B. SIMPSON, *A two dimensional mesh verification algorithm*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 455–473.

[37] F. W. LETNIOWSKI, *Three dimensional Delaunay triangulations for finite element approximations to a second order diffusion operator*, SIAM J. Sci. Statist. Comput., 1992, to appear.

[38] S. FEENSTRA AND J. A. CHERRY, *Subsurface contamination by dense non-aqueous phase liquid* (DNAPL) *chemicals at Tyson's site, Montgomery County, Pennsylvania: Concepts and Implications*, Zenon Environmental Inc., 1986.

[39] G. BEHIE AND P. A. FORSYTH, *Incomplete factorization methods for fully implicit simulation of enhanced oil recovery*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 543–561.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1058–1073, September 1991
© 1991 Society for Industrial and Applied Mathematics
004

STABILITY ANALYSIS AND IMPROVEMENT OF THE BLOCK GRAM-SCHMIDT ALGORITHM*

W. JALBY† AND B. PHILIPPE†

Abstract. The advent of supercomputers with hierarchical memory systems has imposed the use of block algorithms for the linear algebra algorithms. Although block algorithms may result in impressive improvements in performance, their numerical properties are quite different from their scalar counterpart and deserve an in-depth study. In this paper, the numerical stability of block Gram-Schmidt orthogonalization is studied and a variant is proposed which has numerical properties similar to the modified Gram-Schmidt procedure while retaining most of the performance advantages of the block formulation.

Key words. Gram-Schmidt orthogonalization, stability, block algorithms

AMS(MOS) subject classifications. 65F25, 65G05, 68Q10

1. Introduction. The Gram-Schmidt algorithm is one of the most widely used algorithms to orthogonalize a set of vectors. Let $A \in \mathfrak{R}^{n\times m}$ $(n \geqq m)$ be the matrix of which columns (a_i) have to be orthogonalized. The Gram-Schmidt method generates the orthogonal matrix $Q = [q_1; \cdots; q_m]$, which corresponds to the Q factor of the QR decomposition of A, according to the following scheme:

$q_1 := a_1 / \|a_1\|;$

for $k = 1, m-1$

$a'_{k+1} := L_k a_{k+1};$

$q_{k+1} := a'_{k+1} / \|a_{k+1}\|;$

endfor;

where L_k stands for the projection onto the orthogonal complement of the subspace spanned by $(q_1, \cdots, q_k)$. Several implementations of the procedure lie under this common presentation (depending upon the representation of L_k).

If V_k denotes the matrix $[q_1; \cdots; q_k]$, L_k may be expressed as the matrix $I - V_k V_k^t$; this corresponds to the classical version (CGS). The matrix L_k may also be expressed as the composition of the projections onto the orthogonal complements of the one-dimensional subspaces spanned by $q_1, \cdots, q_k$:

$$L_k = (I - q_k q_k^t) \cdots (I - q_1 q_1^t). \tag{1}$$

The corresponding algorithm is the modified version (MGS).

Although (CGS) and (MGS) are equivalent in exact arithmetic, they behave quite differently in finite arithmetic: (CGS) is considered unreliable, as we will see in the next section. The advent of vector and parallel computers using hierarchical memory systems has emphasized the use of matrix multiplication as an efficient primitive which allows efficient data management and has considerably renewed the interest in block algorithms based on matrix multiplications [3], [4]. For example, in the case of the Gram-Schmidt procedure, a natural block version can be obtained by using blocks of

* Received by the editors January 24, 1990; accepted for publication (in revised form) August 13, 1990. This research was supported primarily by National Science Foundation grant CCR-8717942 and AT&T grant AFFL 67 Sameh, when the authors were visiting the Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois 61801.

† IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France.

consecutive columns to orthogonalize the blocks with respect to each other, (MGS) being used to orthogonalize inside a block. In such a case, L_k can be expressed as

$$L_k = (I - q_k q_k^t) \cdots (I - q_l q_l^t)(I - Q_\beta Q_\beta^t) \cdots (I - Q_1 Q_1^t) \tag{2}$$

by assuming that q_{k+1} is a column of the block $Q_{\beta+1}$, whose first column is q_l. The corresponding algorithm is:

$Q_1 := \mathrm{MGS}(A_1)$;
for $\beta = 1, \nu - 1$
  $B_{\beta+1} := A_{\beta+1}$;
  for $\alpha = 1, \beta$
    $S := Q_\alpha^t B_{\beta+1}$;
    $B_{\beta+1} := B_{\beta+1} - Q_\alpha * S$;
  endfor;
  $Q_{\beta+1} := \mathrm{MGS}(B_{\beta+1})$;
endfor;

In the sequel, the block procedure described above will be denoted as (BGS).

The paper addresses the estimation of the numerical reliability of (BGS) which, when quickly examined, exhibits some similarities with (CGS). After a brief description of the loss of orthogonality in the Gram-Schmidt procedure (§ 2), the error introduced by a projection is studied in detail (§ 3). Based on this result, the overall error resulting from (BGS) procedure is analyzed and bounded (§ 4). The estimation of the loss of orthogonality is heavily based on results obtained by Björck, who first characterized the situation for (MGS) [1]. Then, a variant of the (BGS) procedure is introduced and shown to achieve similar numerical quality as (MGS) (§ 5). This variant based on adding extra reorthogonalization steps has also been proposed by Björck [2]. A detailed performance analysis of the variant as well as experimental results are presented in § 6. The major interest of the variant is that numerical properties are greatly enhanced at a moderate price in terms of performance.

2. An illustration of the scene. Björck [1] proved that if $(\bar{Q}, \bar{R})$ are the factors obtained by the (MGS) procedure, there exist constants $(K_i)_{i=1,3}$, only depending upon n and m, such that[1]:

$$\|\bar{Q}\bar{R} - QR\|_F \leqq K_1 \|A\|_F \varepsilon,$$
$$\|\bar{Q}^t\bar{Q} - I\|_2 \leqq K_2 \|A\|_F \|R^{-1}\|_2 \varepsilon,$$
$$\|U\bar{R}\|_F \leqq K_3 \|A\|_F \varepsilon$$

where (Q, R) are the exact factors of A, U is the upper triangular part of the matrix $\bar{Q}^t\bar{Q} - I$, and ε is the precision parameter. These results demonstrate the importance of the condition number of A ($\chi_2(A)$), which satisfies

$$\chi_2(A) \leqq \|A\|_F \|R^{-1}\|_2 \leqq \sqrt{m}\, \chi_2(A).$$

[1] In this paper, bounds are expressed with constants K, which only depend upon n and m, and the block size p. We do not provide precise estimation of these constants because we feel that they are very pessimistic and would only muddle the presentation. They are interesting because of their existence. For the same reason, we systematically neglect the $O(\varepsilon^2)$ terms.

For instance, let us consider the following matrix:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ \sigma & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & \sigma \end{bmatrix}.$$

Its condition number is $\chi(A) = \sqrt{3+\sigma^2}/\sigma$. The loss of orthogonality when applying (CGS) or (MGS) is reported in Table 1 for several values of σ.

TABLE 1

Loss of orthogonality in (MGS) *and* (CGS). (*Experiments performed with* MATLAB *and* $\varepsilon = 2^{-52} \approx 2.10^{-16}$.)

		$\|\bar{Q}^t\bar{Q} - I\|_2$	
σ	$\chi(A)$	MGS	CGS
10^{-4}	$\sqrt{3}\times 10^4$	3×10^{-13}	2×10^{-9}
10^{-5}	$\sqrt{3}\times 10^5$	7×10^{-13}	5×10^{-8}
10^{-6}	$\sqrt{3}\times 10^6$	7×10^{-11}	5×10^{-5}
10^{-7}	$\sqrt{3}\times 10^7$	2×10^{-9}	1×10^{-2}

It should be noted that for a block size of 1 or m, (BGS) corresponds exactly to (MGS). The only case where (BGS) corresponds to (CGS) and not to (MGS) arises when $m = 3$ for a block size of 2. Then, the very special example described above can be viewed as an illustration of the potential bad properties of (BGS).

3. Error in a projection. The elementary step of (BGS) is the projection of a block A_β^α onto the orthogonal complement of $r(Q_\alpha)$ where $\alpha < \beta$ and $r(Q_\alpha)$ is the vector space span by the columns of Q_α. This step corresponds to the computation involved in the innermost loop of (BGS). The sequence A_β^α is then defined by

$$\begin{aligned} &A_\beta^1 = A_\beta \quad \text{and} \\ &A_\beta^{\alpha+1} = P_\alpha A_\beta^\alpha \quad \text{where } P_\alpha = I - Q_\alpha Q_\alpha^t \quad \text{for } 1 < \alpha < \beta \leqq \nu. \end{aligned} \tag{3}$$

Due to rounding error effects, the computed sequences are not $\{A_\beta^\alpha\}$ and $\{Q_\alpha\}$, but $\{\bar{A}_\beta^\alpha\}$ and $\{\bar{Q}_\alpha\}$ and the recurrence formula (3) becomes

$$\begin{aligned} &\bar{A}_\beta^1 = A_\beta \quad \text{and} \\ &\bar{A}_\beta^{\alpha+1} = \bar{A}_\beta^\alpha - \bar{Q}_\alpha \bar{R}_\beta^\alpha + \Delta_\beta^\alpha, \qquad 1 \leqq \alpha < \beta \leqq \nu, \end{aligned} \tag{4}$$

where $\bar{R}_\beta^\alpha = \bar{Q}_\alpha^t \bar{A}_\beta^\alpha$. Let $\bar{P}_\alpha = I - \bar{Q}_\alpha \bar{Q}_\alpha^t$.

Because our purpose is to look at the loss of orthogonality, we do not assume that $\bar{Q}_\alpha^t \bar{Q}_\alpha = I$. Let $D_\alpha = \bar{Q}_\alpha^t \bar{Q}_\alpha - I$. In such a case, $\bar{P}_\alpha$ is, a priori, no longer a projection. Let us decompose any vector u as the sum of its components over $r(\bar{Q}_\alpha)$ and $r(\bar{Q}_\alpha)^\perp$, i.e.,

$$u = \bar{Q}_\alpha v + w \quad \text{with } \bar{Q}_\alpha^t w = 0.$$

Then, we obtain

$$\bar{P}_\alpha u = -\bar{Q}_\alpha D_\alpha v + w,$$

which proves that $\bar{P}_\alpha$ is the identity over $r(\bar{Q}_\alpha)^\perp$, but is not the null operator over $r(\bar{Q}_\alpha)$, as it would be if $\bar{Q}_\alpha$ was orthogonal.

Below are some estimations which will be useful for the following section.

LEMMA 3.1. *Let $\bar{Q}_\alpha$ and D_α be defined as above; then the following inequalities hold:*

- $\|\bar{Q}_\alpha\|_2 = \sqrt{1+\|D_\alpha\|_2} \leqq 1 + \frac{1}{2}\|D_\alpha\|_2$;
- *Under the assumption that $\|\bar{Q}_\alpha D_\alpha\|_2 \leqq 1$:*

$$\|\bar{P}_\alpha\|_2 = 1.$$

Proof. The proof is obvious. □

Let η be the largest number such that $\eta(1+\eta/2) \leqq 1$ ($\eta \approx 0.78$). We will assume that

$$\text{for all } \alpha, \quad \|D_\alpha\|_2 \leqq \eta \tag{5}$$

and therefore $\|\bar{P}_\alpha\|_2 = 1$.

The error Δ_β^α is bounded by the following estimation.

PROPOSITION 3.1. *There exists a sequence of constant numbers K_β^α such that*

$$\|\Delta_\beta^\alpha\|_F \leqq K_\beta^\alpha \|\bar{A}_\beta^\alpha\|_F \varepsilon.$$

Proof. The computation of $\bar{A}_\beta^{\alpha+1}$ consists of three steps. Dropping indices α and β for the sake of clarity, these steps are:

$$W^{(1)} = fl(\bar{Q}^t \bar{A}) = \bar{Q}^t \bar{A} + V^{(1)},$$
$$W^{(2)} = fl(\bar{Q} W^{(1)}) = \bar{Q}\bar{Q}^t \bar{A} + \bar{Q} V^{(1)} + V^{(2)},$$
$$W^{(3)} = fl(\bar{A} - W^{(2)}) = \bar{A} - \bar{Q}\bar{Q}^t \bar{A} - \bar{Q} V^{(1)} - V^{(2)} + V^{(3)}.$$

Using the classical results on the error bounds for matrix multiplications [5], we derive the following inequalities:

$$\|V^{(1)}\|_F \leqq K(n)\|\bar{Q}\|_2 \|\bar{A}\|_F \varepsilon, \quad \text{and then}$$
$$\|\bar{Q} V^{(1)}\|_F \leqq K(n)\|\bar{Q}\|_2^2 \|\bar{A}\|_F \varepsilon, \quad \text{and}$$
$$\|V^{(2)}\|_F \leqq K(p)\|\bar{Q}\|_2^2 \|\bar{A}\|_F \varepsilon + O(\varepsilon^2)$$

where $K(\ )$ is a polynomial of low order and $\bar{Q} \in \mathfrak{R}^{n \times p}$.

The error due to the substraction may be bounded by

$$\begin{aligned} \|V^{(3)}\|_F &\leqq \|\bar{A} - W^{(2)}\|_F \varepsilon \\ &\leqq \|\bar{I} - \bar{Q}\bar{Q}^t\|_2 \|\bar{A}\|_F \varepsilon + O(\varepsilon^2) \\ &\leqq \|\bar{A}\|_F \varepsilon + O(\varepsilon^2) \end{aligned}$$

since we have assumed (5). Putting together all these bounds we obtain

$$\|\Delta\|_F \leqq \|\bar{Q} V^{(1)}\|_F + \|V^{(2)}\|_F + \|V^{(3)}\|_F,$$
$$\|\Delta\|_F \leqq K \|\bar{A}\|_F \varepsilon. \qquad \square$$

LEMMA 3.2. *There exists a constant $\tau = O(\varepsilon)$ such that $\|\bar{A}_\beta^\alpha\|_F \leqq (1+\tau)^{\alpha-1}\|A_\beta\|_F$ for all $\alpha \leqq \beta \leqq \nu$.*

Proof. The proof is obvious from Lemma 3.1 and Proposition 3.1 with

$$\tau = \max_{\alpha \leqq \beta \leqq \nu} (K_\beta^\alpha \varepsilon). \qquad \square$$

Once $\bar{A}_\beta^\beta$ is computed, a Modified Gram-Schmidt process is applied to get $\bar{Q}_\alpha$.

Let R'_β and $\tilde{R}_\beta$ be the exact and the computed R-factor of the QR decomposition of $\bar{A}^\beta_\beta$, respectively. From [1], we have the following estimations:

$$\|\bar{A}^\beta_\beta - \bar{Q}_\beta \tilde{R}_\beta\|_F \leqq K_1 \|\bar{A}^\beta_\beta\|_F \varepsilon,$$

$$\|\bar{Q}^t_\beta \bar{Q}_\beta - I\|_2 \leqq K_2 \|\bar{A}^\beta_\beta\|_F \|R'^{-1}_\beta\|_2 \varepsilon,$$

$$\|(\bar{Q}^t_\beta \bar{Q}_\beta - I)\tilde{R}_\beta\|_F \leqq K_3 \|\bar{A}^\beta_\beta\|_F \varepsilon.$$

Let $C(\beta) = \|\bar{A}^\beta_\beta\|_F \|R'^{-1}_\beta\|_2$. If orthogonalization was exact, we would have $\|A\|_F \|R^{-1}\|_2 \geqq C(\beta)$ and generally, it would be far from equal. We assume that it is still the case with finite arithmetic. The quantity $C = \max_{\beta \leqq \nu} C(\beta)$ is of importance in the sequel.

Let $\bar{R}^\beta_\beta$ be the matrix $\bar{Q}^t_\beta \bar{A}^\beta_\beta$. From the equality

$$\bar{R}^\beta_\beta - \tilde{R}_\beta = (\bar{Q}^t_\beta \bar{Q}_\beta - I)\tilde{R}_\beta + \bar{Q}^t_\beta (\bar{A}^\beta_\beta - \bar{Q}_\beta \tilde{R}_\beta)$$

we obtain

$$\|\bar{R}^\beta_\beta - \tilde{R}_\beta\|_F \leqq (K_3 + \|\bar{Q}_\beta\|_2 K_1)\|\bar{A}^\beta_\beta\|_F \varepsilon,$$

and if

$$\Delta^\beta_\beta = -\bar{A}^\beta_\beta + \bar{Q}_\beta \bar{R}^\beta_\beta \tag{6}$$

we may write

$$\Delta^\beta_\beta = -\bar{A}^\beta_\beta + \bar{Q}_\beta \tilde{R}_\beta + \bar{Q}_\beta(-\tilde{R}_\beta + \bar{R}^\beta_\beta),$$

which implies the following proposition.

PROPOSITION 3.2. *There exists a constant K^β_β such that*

$$\|\Delta^\beta_\beta\|_F \leqq K^\beta_\beta \|\bar{A}^\beta_\beta\|_F \varepsilon.$$

4. Loss of orthogonality in the whole process. By adapting Björck's procedure for blocks, we define the following matrices:

$$U^\alpha_\beta = \begin{cases} \bar{Q}^t_\alpha \bar{Q}_\beta & \text{for } 1 \leqq \alpha < \beta \leqq \nu, \\ 0 & \text{for } 1 \leqq \beta \leqq \alpha \leqq \nu, \end{cases}$$

and

$$D_\alpha = \bar{Q}^t_\alpha \bar{Q}_\alpha - I \quad \text{for } 1 \leqq \alpha \leqq \nu.$$

Let U and D be the upper block triangular and the block diagonal matrices defined by the blocks $\{U^\alpha_\beta\}$ and $\{D_\alpha\}$, respectively.

From the previous section, we may ensure that

$$\|D\|_2 \leqq K_2 C \varepsilon \tag{7}$$

where $C = \max_\alpha C(\alpha)$.

To estimate a bound on $\|U\|_2$, we first estimate a bound of $\|U\bar{R}\|_2$ where $\bar{R}$ is the upper triangular matrix defined by the blocks $\{\bar{R}^\alpha_\beta\}_{1 \leqq \alpha \leqq \beta \leqq \nu}$.

PROPOSITION 4.1. *There exist two constants K_4 and K_5 such that*

$$\|U\bar{R}\|_F \leqq (K_4 C + K_5)\|A\|_F \varepsilon.$$

Proof. By writing (4) for $\alpha = \nu+1, \cdots, \beta-1$, and from the definition of Δ^β_β, we obtain the following equation by accumulating the equalities:

$$\bar{A}^{\nu+1}_\beta = \sum_{\alpha=\nu+1}^{\beta} \bar{Q}_\alpha \bar{R}^\alpha_\beta - \sum_{\alpha=\nu+1}^{\beta} \Delta^\alpha_\beta.$$

The block (μ, β) of $U\tilde{R}$ can then be expressed as

$$[U\bar{R}]_{\mu\beta} = \sum_{\alpha=\mu+1}^{\beta} U_{\mu\alpha}\bar{R}_{\beta}^{\alpha}$$

$$= \bar{Q}_{\mu}^{t}\left(\bar{A}_{\beta}^{\mu+1} + \sum_{\alpha=\mu+1}^{\beta} \Delta_{\beta}^{\alpha}\right).$$

But from (4),

$$\bar{Q}_{\mu}^{t}\bar{A}_{\beta}^{\mu+1} = (\bar{Q}_{\mu}^{t} - (I + \bar{D}_{\mu})\bar{Q}_{\mu}^{t})\bar{A}_{\beta}^{\mu} + \bar{Q}_{\mu}^{t}\Delta_{\beta}^{\mu}$$
$$= -\bar{D}_{\mu}\bar{Q}_{\mu}^{t}\bar{A}_{\beta}^{\mu} + \bar{Q}_{\mu}^{t}\Delta_{\beta}^{\mu},$$

which implies

$$\|\bar{Q}_{\mu}^{t}\bar{A}_{\beta}^{\mu+1}\|_F \leqq \|\bar{D}_{\mu}\|_2\|\bar{Q}_{\mu}\|_2\|\bar{A}_{\beta}^{\mu}\|_F + \|\bar{Q}_{\mu}\|_2\|\Delta_{\beta}^{\mu}\|_F$$

and

$$\|[U\bar{R}]_{\mu\beta}\|_F \leqq \|\bar{D}_{\mu}\|_2\|\bar{Q}_{\mu}\|_2\|\bar{A}_{\beta}^{\mu}\|_F + \|\bar{Q}_{\mu}\|_2 \sum_{\alpha=\mu}^{\beta} \|\Delta_{\beta}^{\alpha}\|_F.$$

From

$$\|U\bar{R}\|_F \leqq \sum_{1\leqq\mu\leqq\beta\leqq\nu} \|[U\bar{R}]_{\mu\beta}\|_F$$

and from Lemma 3.2 and Propositions 3.1 and 3.2, the result of the proposition is obtained. □

Let $E_1 = \bar{A} - QR$ where $\bar{A} = \bar{Q}\bar{R}$ and (Q, R) is the result of the QR factorization of A. Then the following result can be proved.

PROPOSITION 4.2. *There exists a constant K_6 such that*

$$\|E_1\|_F = \|\bar{Q}\bar{R} - QR\|_F \leqq K_6\|A\|_F\varepsilon$$

and the condition $\|E_1\|_F\|R^{-1}\|_2 < \sqrt{2} - 1$ implies that $\bar{A}$ has full rank.

Proof. By summing equations (3) from $\alpha = 1$ to $\alpha = \beta - 1$, we obtain

$$\bar{A}_{\beta}^{\beta} = A_{\beta} - \sum_{\alpha=1}^{\beta-1} \bar{Q}_{\alpha}\bar{R}_{\beta}^{\alpha} + \sum_{\alpha=1}^{\beta-1} \Delta_{\beta}^{\alpha}.$$

Since $\bar{A}_{\beta}^{\beta} = \bar{Q}_{\beta}\bar{R}_{\beta}^{\beta} - \Delta_{\beta}^{\beta}$ it follows that

$$\|E_1\|_F = \sum_{\beta=1}^{\nu} \sum_{\alpha=1}^{\beta} \|\Delta_{\beta}^{\alpha}\|_F.$$

Propositions 3.1 and 3.2 provide bounds for $\{\Delta_{\beta}^{\alpha}\}$:

$$\|E_1\|_F \leqq \left(\sum_{\beta=1}^{\nu} \sum_{\alpha=1}^{\beta} K_{\beta}^{\alpha}\|\bar{A}_{\beta}^{\alpha}\|_F\right)\varepsilon.$$

Let $K_6 = \max_{\alpha\leqq\beta} K_{\beta}^{\alpha}(1+\tau)^{\alpha-1}$, where τ is the $O(\varepsilon)$ quantity that has been introduced in Lemma 3.2. Then

$$\|E_1\|_F \leqq K_6 \sum_{\beta=1}^{\nu} \|\bar{A}_{\beta}\|_F\varepsilon \leqq K_6\|\bar{A}\|_F\varepsilon,$$

which corresponds to the first part of the proposition.

For the second part, Björck's proof can be used:

$$\bar{A}^t\bar{A} = R^t(I+F_1)R \tag{8}$$

where $F_1 = (Q^tE_1R^{-1})^t + Q^tE_1R^{-1} + (E_1R^{-1})^tE_1R^{-1}$.

Since

$$\|F_1\|_2 \leqq 2\|E_1\|_F\|R^{-1}\|_2 + \|E_1\|_F^2\|R^{-1}\|_2^2,$$

condition $\|E_1\|_F\|R^{-1}\|_2 < \sqrt{2}-1$ implies $\|F_1\|_2 < 1$ and therefore the nonsingularity of $(I+F_1)$. □

THEOREM 4.1. *Let* $C = \max_\alpha C(\alpha)$. *For sufficiently small* $(C\varepsilon)$ *and* $(\|A\|_F\|R^{-1}\|_2\varepsilon)$, *there exists a constant* K_7 *such that*

$$\|I - \bar{Q}^t\bar{Q}\|_2 \leqq K_7 C\|A\|_F\|R^{-1}\|_2\varepsilon.$$

Proof. From the definitions of matrices U and D, we have

$$\|I - \bar{Q}^t\bar{Q}\|_2 \leqq 2\|U\|_2 + \|D\|_2.$$

Since $\|D\|_2$ has already been bounded in Theorem 4.1, we only have to consider $\|U\|_2$; because $\|U\|_2 \leqq \|U\bar{R}\|_F\|\bar{R}^{-1}\|_2$, we only need to focus on $\|\bar{R}^{-1}\|_2$. Here again, we adapt Björck's proof by using the following formula:

$$\bar{R}^t\bar{R} = R^t(I+F_2)R \tag{9}$$

where

$$F_2 = F_1 - (R^{-1})^t(U\bar{R})^t(\bar{R}R^{-1}) - (\bar{R}R^{-1})^t(U\bar{R})R^{-1} - (\bar{R}R^{-1})^tD(\bar{R}R^{-1})$$

with F_1 being the matrix introduced at the end of the proof of Proposition 4.2. From (9), it follows that

$$\|\bar{R}R^{-1}\|_2^2 \leqq 1 + \|F_2\|_2.$$

Let $x = \sqrt{1+\|F_2\|_2}$ and $\alpha = \|A\|_F\|R^{-1}\|_2\varepsilon$. From the definition of F_1 and from Proposition 4.2, we obtain

$$\begin{aligned}\|F_1\|_2 &\leqq 2\|E_1\|_F\|R^{-1}\|_2 + \|E_1\|_F^2\|R^{-1}\|_2^2 \\ &\leqq 2K_6a + K_6^2a^2 + O(\varepsilon^2).\end{aligned}$$

Then from the definition of F_2, from (9), and from Proposition 4.2, we may ensure that

$$x^2 \leqq 1 + 2K_6a + K_6^2a^2 + 2a(K_4C + K_5)x + K_2Cx^2\varepsilon.$$

Then x is such that the following relation is true:

$$(1 - K_2C\varepsilon)x^2 - 2a(K_4C + K_5)x - 1 - 2K_6a - K_6^2a^2 \leqq 0.$$

If $(C\varepsilon)$ is small enough to keep $(1-K_2C\varepsilon)$ positive, then the polynomial has two zeros: one is negative and the other is greater than 1. The second root is denoted $1+\rho$; $\rho > 0$ and $\rho = O(\varepsilon)$. In conclusion, we have $\|F_2\|_2 \leqq \rho$.

Let us assume that the quantities $(\|A\|_F\|R_{-1}\|_2\varepsilon)$ and $(C\varepsilon)$ are small enough to ensure that $\rho < 1$. Then, by considering the inverse of both sides of (9), we obtain

$$\|\bar{R}^{-1}\|_2^2 \leqq \|R^{-1}\|_2^2\|(I+F_2)^{-1}\|_2 \leqq (1-\rho)^{-1}\|R^{-1}\|_2^2.$$

This last bound ends the proof of the theorem. □

The worst bound can be obtained when one block has a condition number almost equal to the condition number of the whole system. In that situation, the quantity $(C\|A\|_F\|R^{-1}\|_2)$ is of the order of the square of the condition number of A.

5. (B2GS) A block method as stable as (MGS). Theorem 4.1 shows the importance of the constant C in the error bound. Even when it is small, the error that occurs within the orthogonalization of a block may significantly impact on the precision of the following steps. By reorthonormalizing every block, i.e., by applying twice (MGS) to every block, the constant C would disappear from the estimation, since it would correspond to the QR factorization of the $\{\bar{Q}_\beta\}$ blocks, which have condition numbers close to unity. This remark, leads directly to the algorithm (B2GS), which has exactly the same structure as (BGS) except that the reorthonormalization procedure is applied twice on each block.

To illustrate the stability improvement in (B2GS) compared to (BGS), some numerical results are presented below.

For the first series of experiments, the results obtained by (BGS) and (B2GS) are compared using MATLAB. The matrix under consideration is the Hilbert matrix:

$$A = (a_{ij}) \in \mathfrak{R}^{20\times 10} \quad \text{where } a_{ij} = 1/(i+j-1).$$

The characteristics of the matrix are

$$\|A\|_F = 1.9,$$
$$\|R^{-1}\|_2 = 1.4\times 10^{11},$$
$$\chi(A) = 2.6\times 10^{11}.$$

The value of the residuals, with respect to the block size, are reported in Table 2.

A second series of experiments was performed by implementing the codes corresponding to the three algorithms ((MGS), (BGS), and (B2GS)) in FORTRAN on the CRAY2. The matrix used was a matrix A of size 1024×512 obtained by the multiplication of three matrices such that

$A = (I+\alpha H_1)MH_2$ where

$$M \in \mathfrak{R}^{1024\times 512} \quad \begin{cases} m_{1j} = 1, & j = 1,\cdots,512, \\ m_{ij} = \varepsilon\delta_{i-1,j}, & i = 2,\cdots,1024 \quad \text{and} \quad j = 1,\cdots,512, \end{cases}$$

$H_1 \in \mathfrak{R}^{1024\times 1024}$ h_{1ij} randomly selected in $[-1,+1]$,

$H_2 \in \mathfrak{R}^{512\times 512}$ h_{2ij} randomly selected in $[-1,+1]$

with $\alpha = 10^{-3}$ and $\varepsilon = 10^{-2}$. The condition number of the resulting matrix A was estimated by a LINPACK procedure and its value was $\chi(A) = 0.505E+07$.

Table 3 compares the Frobenius norm of the residuals for various block sizes and for (BGS) and (B2GS). These results show clearly the interest of (B2GS) which gives results with an improvement in the order of magnitude of 3 to 4 compared to the standard (BGS). Furthermore, for all the block sizes, (B2GS) achieves results very close to the standard (MGS) procedure (results of which are given in the first row of Table 3).

TABLE 2

Comparison of loss of orthogonality for (BGS) *and* (B2GS) *with reorthogonalization for each block size on a Hilbert matrix* (MATLAB).

Block size	1	2	3	4	5
(BGS)	5.2×10^{-6}	2.8×10^{-6}	2.3×10^{-5}	1.1×10^{-4}	5.2×10^{-3}
(B2GS)	5.2×10^{-6}	3.0×10^{-6}	4.3×10^{-6}	3.1×10^{-6}	4.0×10^{-6}

TABLE 3

Comparison of residuals for (MGS), (BGS), *and* (B2GS).

(MGS) *residual*: 0.171×10^{-7}.

Block size	(BGS) residuals	(B2GS) residuals
16	0.253×10^{-4}	0.592×10^{-7}
32	0.703×10^{-4}	0.696×10^{-7}
48	0.103×10^{-3}	0.527×10^{-7}
64	0.126×10^{-3}	0.464×10^{-7}
80	0.180×10^{-3}	0.501×10^{-7}
96	0.186×10^{-3}	0.466×10^{-7}
112	0.258×10^{-3}	0.623×10^{-7}
128	0.304×10^{-3}	0.577×10^{-7}
144	0.277×10^{-3}	0.511×10^{-7}
160	0.242×10^{-3}	0.534×10^{-7}
176	0.268×10^{-3}	0.483×10^{-7}
192	0.327×10^{-3}	0.619×10^{-7}
208	0.350×10^{-3}	0.674×10^{-7}

6. Performance analysis of (B2GS). Because one of the major advantages of block algorithms is to provide a good data locality together with a good potential for vectorization and parallelization, our performance analysis will be carried out along two axes: number and characteristics of the arithmetic operations (i.e., number of operations, vectorization, and parallelization properties) and data locality. First, the methodology for the performance analysis will be briefly described, then the two axes mentioned above will be studied. The behavior of (B2GS) will be systematically compared with that of (BGS) and the choice of the block size will be analyzed. Finally, some experimental results on an ALLIANT FX80 and a CRAY-2 will be presented.

6.1. Performance model. Throughout § 6, we will use the same framework used in [3] and [4] for studying block algorithms. Let us recall briefly its major characteristics. The target machine will be assumed to be a shared memory multi-(vector) processor, using a memory system consisting of a cache (of size CS) with fast access and a large memory with slower access. According to the methodology developed in [3] and [4], the total execution time will be split into two components:

(1) T_a: Arithmetic time. The total computation time assuming that the cache has an infinite size and that initially all the data reside in cache.

(2) T_l: Load time. The extra time spent in the loads to be performed from the memory due to the finite cache size (including the initial data loads as well as the ones occurring because all the data cannot fit in the cache).

The first component (T_a) takes into account all the parallelization and vectorization properties of the algorithm studied, while the second one (T_l) attempts to quantify its data locality characteristics. This second measure is extremely difficult to evaluate accurately because it depends upon many intrinsic details of a particular cache organization; as it was proposed in [3] and [4], instead of considering the problem of optimizing T_l we will consider the problem of optimizing N_l, which is the total number of loads from memory. For determining N_l, we will assume that the data transfers between the cache and the memory are under software control (i.e., we can specify which data will reside in cache). Such a measure is much simpler to compute and allows capture

of most of the trends in the performance behavior of the algorithm. Furthermore, we will evaluate the ratio $\mu = N_l/N_a$, where N_a is the total number of floating point operations. This quantity enables us to get a better appreciation of the relative cost of the loads from memory.

Our analysis of (B2GS) is greatly simplified because the differences between (BGS) and (B2GS) are minor. For example, since (B2GS) is using exactly the same computational primitives as (BGS), all the results relative to the parallelization and vectorization properties can be carried out from (BGS) to (B2GS); the only difference is the different weight associated with each primitive.

6.2. Optimization of the arithmetic time. First, let us evaluate the cost in terms of number of floating point operations of the extra computation involved in the reorthonormalization. The regular version of (BGS) has the same complexity as (MGS) or (CGS). For an $n \times m$ system, this number of operations is given by

$$N_{\text{op}}(\text{BGS}) = N_{\text{op}}(\text{MGS}) \simeq 2nm^2.$$

Let us assume that $m = k\omega$ where ω is the block size. Then, an extra reorthonormalization of every block leads to a total of number of operations for (B2GS) given by

$$N_{\text{op}}(\text{B2GS}) \simeq 2nm^2 + 2kn\omega^2 = N_{\text{op}}(\text{BGS}) + 2nm\omega. \tag{10}$$

This clearly shows the interest of keeping ω relatively small. Additionally, it also indicates that the idea of varying block sizes is not worthwhile from the arithmetic point of view. In fact, if we assume nonconstant block sizes (i.e., a sequence $\{\omega_i\}_{i=1,k}$), the extra cost associated with the reorthogonalization step is $\sum_{i=1}^{k} n\omega_i^2$. For a fixed number of blocks, this quantity will be minimal if all the block sizes are constant and equal to m/k. In terms of relative costs,

$$\frac{N_{\text{op}}(\text{B2GS})}{N_{\text{op}}(\text{BGS})} = 1 + \frac{\omega}{m} = 1 + \frac{1}{k}.$$

Besides the extra number of operations, (B2GS) differs from (BGS) by the work repartition between the different primitives. Although all the primitives used in (BGS) (and therefore in (B2GS)) offer a good degree of vectorization and parallelization, there will be a sensible discrepancy in performance between the block factorization ((MGS) primitive on a block) and the matrix multiply primitives, because matrix multiply lends itself to a very efficient use of chaining and registers and has a much simpler synchronization graph than the (MGS) primitive.

More formally, if V_{MGS} (respectively, V_{MAT}) denotes the speed in megaflops of the (MGS) primitive (respectively, the matrix multiply primitive), we end up with

$$T_a(\text{BGS}) = \frac{2mn\omega}{V_{\text{MGS}}} + \frac{2mn(n-\omega)}{V_{\text{MAT}}},$$

$$T_a(\text{B2GS}) = \frac{4mn\omega}{V_{\text{MGS}}} + \frac{2mn(n-\omega)}{V_{\text{MAT}}}.$$

The difference between (BGS) and (B2GS) is therefore

$$\frac{T_a(\text{B2GS}) - T_a(\text{BGS})}{T_a(\text{BGS})} = \frac{V_{\text{MAT}}/V_{\text{MGS}}}{(V_{\text{MAT}}/V_{\text{MGS}}) + k - 1}.$$

If we assume that $V_{\text{MAT}}/V_{\text{MGS}} \leqq 6$, which is reasonable for many practical machines, this gives

$$\frac{T_a(\text{B2GS}) - T_a(\text{BGS})}{T_a(\text{BGS})} \leqq \frac{6}{k+5}.$$

This last estimation insures that as long as the number of blocks (k) is large enough, the discrepancy in performance (due to the extra arithmetic operations) between BGS and B2GS will be low. In practice, for large problems, it is relatively easy to obtain a large number of blocks: for instance, for 512 vectors and a block size of 16, the discrepancy is less than 16 percent.

6.3. Optimization of data locality. Again, because (B2GS) uses the same primitives as (BGS), similar conclusions can be derived: increasing the block size increases data locality. For simplifying the analysis, we can assume without loss of generality that $\omega < CS/n$. This corresponds to the case where the orthogonalization of a block $n \times \omega$ fits in cache. In the case of (B2GS), such a restriction is really not too constraining because the use of block sizes larger than CS/n would generally result in a prohibitive penalty for the number of floating operations.

The good point of (B2GS) is that the two orthogonalizations primitives are applied one after another for each block. This means that no additional loads will be involved due to reorthonormalization:

$$N_l(\text{B2GS}(\omega)) = N_l(\text{BGS}(\omega)).$$

Using the results presented in [4] on (BGS) behavior, we derive that

$$\mu(\text{B2GS}) = \frac{1}{m}\left(1 + k - \frac{2}{k}\right) + \frac{1}{n}\left(1 - \frac{1}{k}\right).$$

As expected, μ is an increasing function with respect to k. This trend is exactly contradictory with the one observed for the optimization of the arithmetic time. A precise determination of the best choice of the number of blocks requires a precise knowledge of the cost of the memory fetch versus a floating point operation in order to find the right trade-off between arithmetic time minimization and memory load minimization. However, it should be noted that a value of $k = 10$, which gives a relatively good arithmetic optimization, would result in a value for μ(B2GS) given by

$$\mu(\text{B2GS}) \simeq \frac{10.8}{m} + \frac{0.9}{n},$$

which gives reasonably good value for μ under the conditions that m and n are large enough. In fact, if these conditions are not met, as we will see in the next section on experimental results, (B2GS) as well as (BGS) performances are sensibly affected.

6.4. Experimental results. In this section, experimental results obtained on a CRAY-2 (1CPU) and an ALLIANT FX80 (8 processors) are presented. These results support the performance analysis results by exhibiting trends as predicted and showing that (B2GS) offers comparable performance with BGS. These machines were chosen because they had both a hierarchical memory system with three levels: vector registers (eight vector registers of 64 elements for the CRAY-2, eight vector registers of 32 elements for the ALLIANT), an intermediate memory level (a private local memory of 16K words for the CRAY-2, a shared cache of 64K words for the ALLIANT), and finally, the main memory.

6.4.1. Various block sizes. In this experiment, (BGS) and (B2GS) were run on a 1024×512 matrix with various block sizes (machine used: CRAY-2, one CPU). The timings obtained are presented in Fig. 1, where the result for the classical (MGS) is plotted as a reference point. As predicted by the performance analysis, we clearly see two conflicting trends: at the beginning, increasing the block size decreases the total execution time (the benefit from minimizing the load is bigger than the extra arithmetic

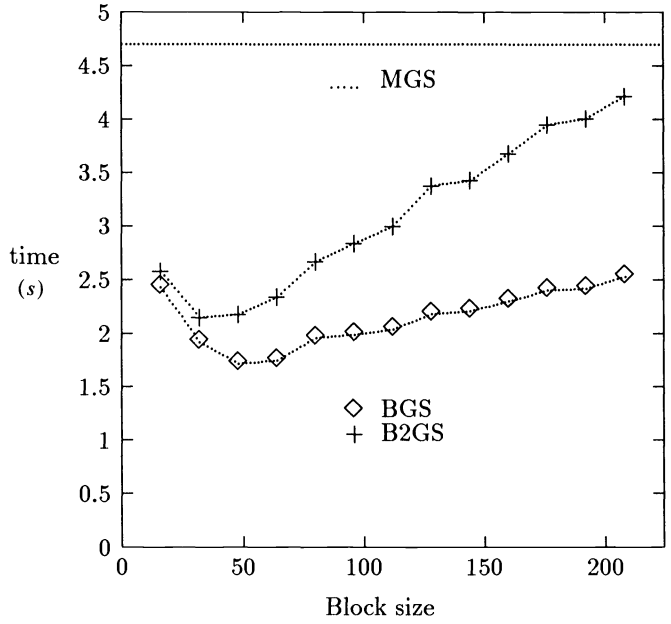


FIG. 1. *Running times for* (MGS), (BGS), *and* (B2GS), *rectangular matrix* (1024×512), (CRAY-2 1CPU).

work), then continuing to increase the block size reverses the situation: the penalty due to the extra operations is overwhelming, resulting in an increase of the execution time.

For (BGS), the minimal execution time of 1.72s is obtained for a block size of 48, while for (B2GS), the minimal execution time of 2.15s corresponds to a block size of 32. The extra cost of (B2GS) is 25 percent. A part of this penalty is due to the fact that our implementation of (B2GS) on the CRAY2 did not keep the block of columns in the local memory between two successive orthonormalizations (requiring a slight modification of the performance analysis), which implies that this implementation (B2GS) does not only increase the number of operations but also the number of loads. Such a characteristic also explains why (B2GS) times are ramping up much faster for large block sizes than (BGS). However, (B2GS) still achieves a speedup of over 2 when compared to the standard (MGS).

6.4.2. Various matrix sizes. In these experiments, (MGS), (BGS), and (B2GS) were run for different matrix shapes and sizes on an ALLIANT FX80. The results are presented in Fig. 2 (square matrix $n \times n$), Fig. 3 (rectangular matrix $1024 \times n$), and Fig. 4 (rectangular matrix $2048 \times n$). In all these figures, the y-axis corresponds to the normalized megaflops, which is obtained by dividing for the three codes the same number of floating point operations (corresponding to (MGS)) by the timing. This allows a fair comparison among the three algorithms: essentially, if the normalized megaflop rate is three times higher for (BGS) than for (MGS), that (BGS) execution time is three times smaller.

The experiments were performed by surrounding the code to be measured by a repetition loop in order to reduce the impact of the clock accuracy. This has an adverse effect for the small matrix sizes, which entirely fit in cache; in such cases, the first iteration will load the matrix in the cache, then all the subsequent operations will be performed with the data entirely resident in cache. This explains the strange shape of the curves for MGS, where the performance first increases then decreases when the

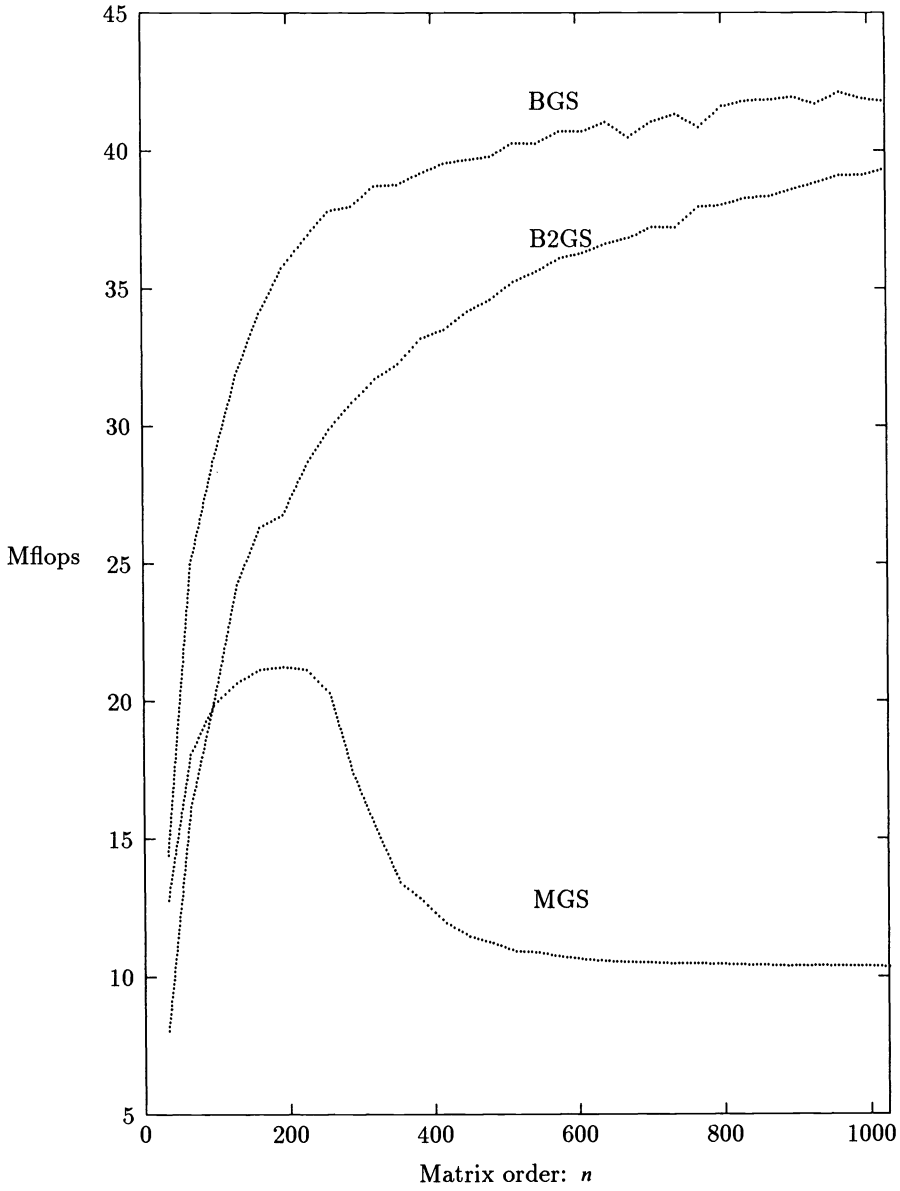


FIG. 2. *Normalized megaflops for* (MGS), (BGS), *and* (B2GS), *square matrix* ($n \times n$) (ALLIANT FX80).

data do not fit any more in cache. It should be noted that (BGS) and (B2GS) are not affected by such a phenomenon because their structure in blocks allows them to keep most of their references to cache.

The block sizes were chosen according to the following rules:

- Square matrix, $n \times n$ (Fig. 2):

BGS: Block size = 16 for $32 \leqq n \leqq 96$

Block size = 32 for $96 < n \leqq 1024$

B2GS: Block size = 16 for $32 \leqq n \leqq 160$

Block size = 32 for $160 < n \leqq 1024$.

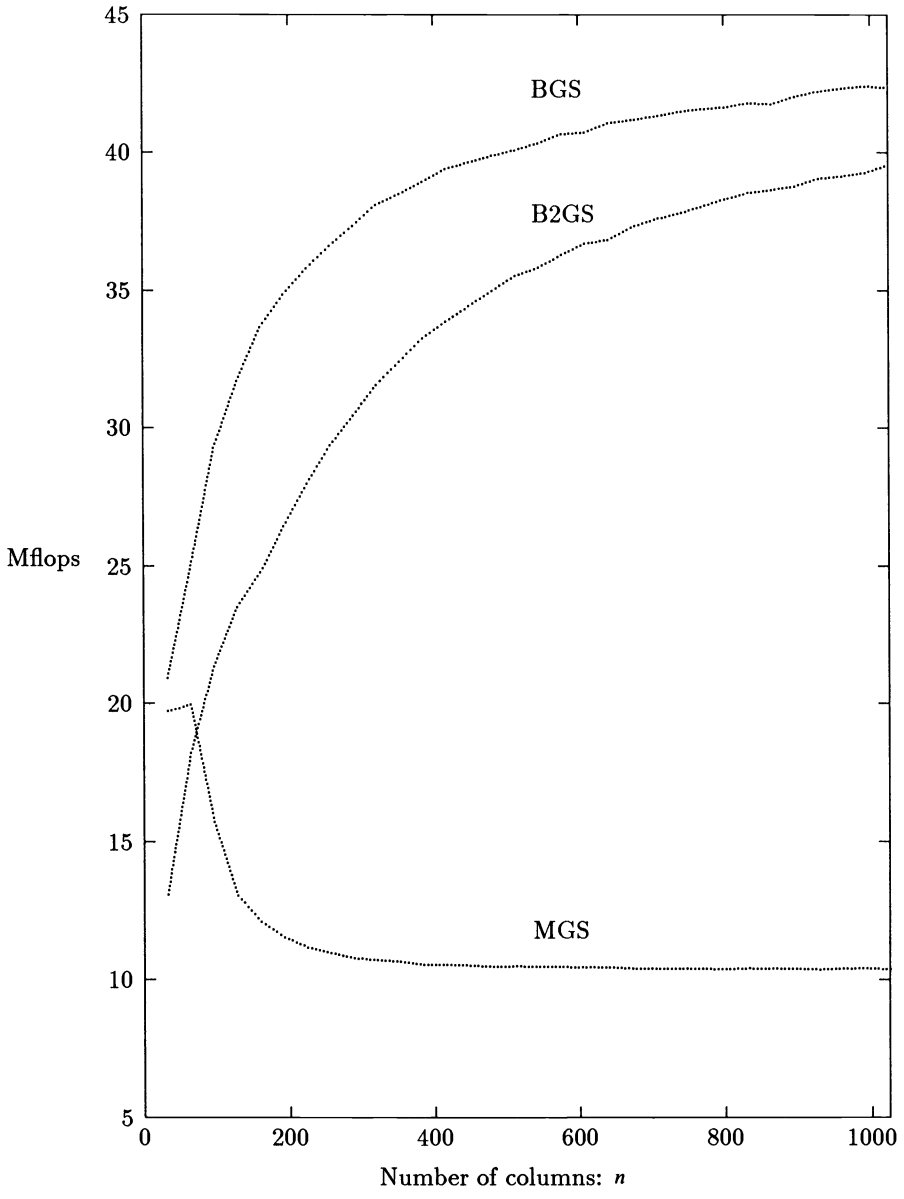


FIG. 3. *Normalized megaflops for* (MGS), (BGS), *and* (B2GS), *rectangular matrix* $(1024 \times n)$ (ALLIANT FX80).

- Rectangular matrix, $1024 \times n$ (Fig. 3):

BGS: Block size $= 16$ for $32 \leqq n \leqq 64$

Block size $= 32$ for $64 < n \leqq 1024$

B2GS: Block size $= 16$ for $32 \leqq n \leqq 128$

Block size $= 32$ for $128 < n \leqq 1024$.

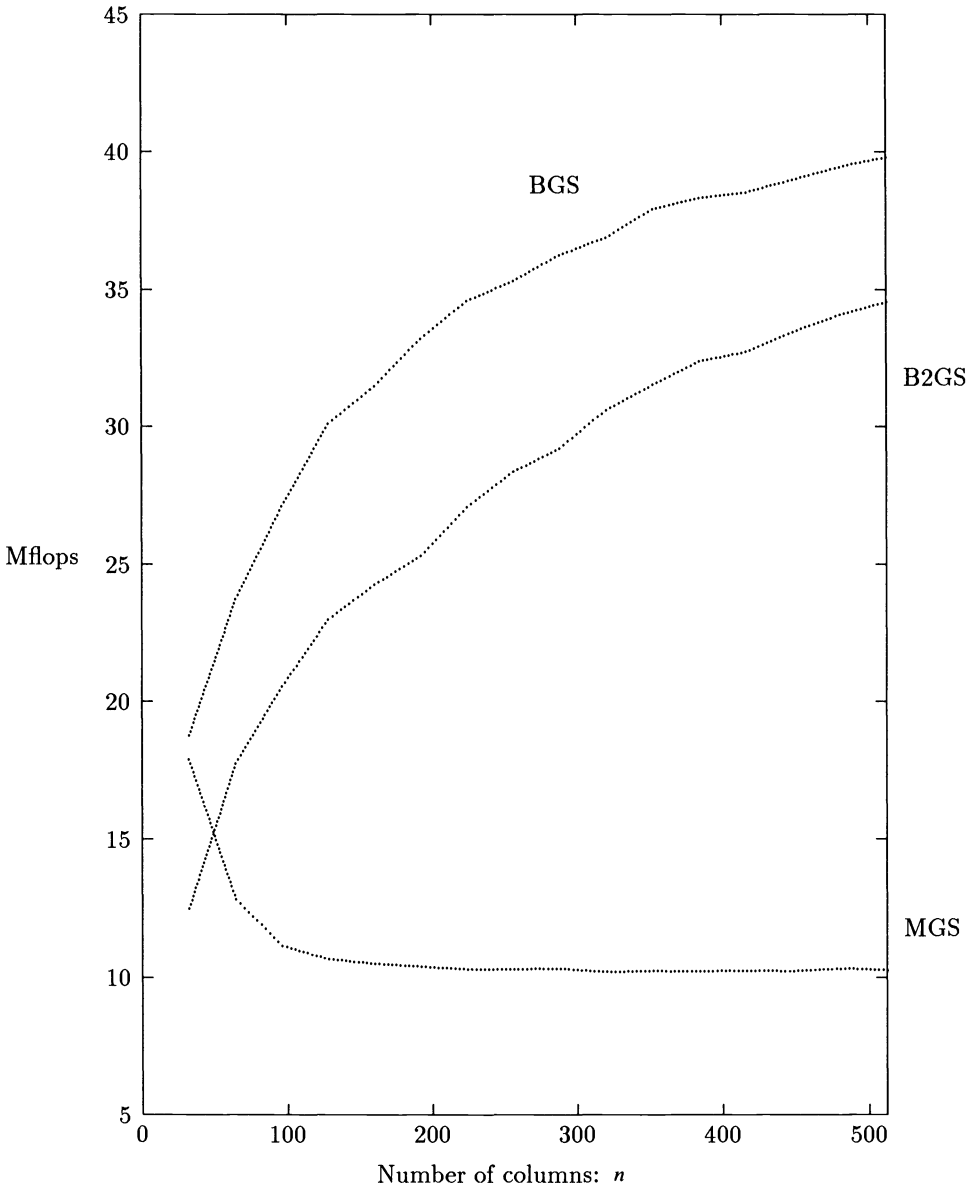


FIG. 4. *Normalized megaflops for* (MGS), (BGS), *and* (B2GS), *rectangular matrix* $(2048 \times n)$ (ALLIANT FX80).

- Rectangular matrix, $2048 \times n$ (Fig. 4):

BGS: Block size $= 16$ for $32 \leqq n \leqq 64$

Block size $= 32$ for $64 < n \leqq 512$

B2GS: Block size $= 16$ for $32 \leqq n \leqq 160$

Block size $= 32$ for $128 < n \leqq 512$.

Such block sizes were chosen according to experimental results.

The main conclusion of the experimental results is that (B2GS) costs between 12 and 25 percent in terms of performance, compared with (BGS); however, the speedups over classical (MGS) are still impressive—over 3 for large matrices. The only negative point is small size problems, where (B2GS) is not competitive, but it should be noted that (BGS) performance is also severly affected in such cases. However, for such small problems, the best choice seems to be classical (MGS), because in such cases the whole data set is close to fit into the cache and therefore does not require specific block algorithms.

7. Conclusion. In this paper it has been proven that by using a reorthogonalization procedure on blocks at an additional cost of low order, the (B2GS) algorithm numerically behaves very closely to the (MGS) algorithm. Another alternative consists of replacing the (MGS) reorthonormalization procedure on each block by another orthogonalization procedure such as the one based on polar decomposition. Since a block $\bar{Q}_\beta$ is nearly orthonormal, its orthogonalization may be performed in the following way [6]:

$$\begin{aligned} \bar{D} &= \bar{Q}_\beta^t \bar{Q}_\beta - I && np^2 \text{ flops (not considering symmetry)}, \\ T &= (I + \bar{D})^{-1/2} && O(p^3) \text{ flops}, \\ \bar{Q}_\beta^c &= \bar{Q}_\beta T && np^2 \text{ flops}. \end{aligned}$$

With such a choice, using reorthonormalization costs twice as much as using (MGS), but only involves matrix multiplication. The use of such primitives (resulting in speedup over 2 compared to Blas1 or Blas2 primitives) may in fact offset the additional price in number of operations, and constitute an interesting alternative.

REFERENCES

[1] A. BJÖRCK, *Solving linear least squares problems by Gram-Schmidt orthogonalization*, BIT, 7 (1967), pp. 1-21.

[2] ——, *Pivoting and Condition Estimation in Algorithms for Linear Least Squares Problems*, 1990 IMSL User Group Conference, Bologna, Italy, March 26-28, 1990.

[3] K. GALLIVAN, W. JALBY, AND U. MEIER, *The use of* BLAS3 *in linear algebra on a parallel processor with a hierarchical memory*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 1079-1084.

[4] K. GALLIVAN, W. JALBY, U. MEIER, AND A. SAMEH, *Impact of hierarchical memory systems on linear algebra algorithm design*, Internat. J. Supercomputer Appl., 2 (1988), pp. 12-48.

[5] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[6] B. PHILIPPE, *An algorithm to improve nearly orthonormal sets of vectors on a vector processor*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 396-403.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1074–1108, September 1991
© 1991 Society for Industrial and Applied Mathematics
005

CELL DISCRETIZATION OF NONSELFADJOINT LINEAR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS*

JOHN GREENSTADT†

Abstract. The cell discretization (CD) algorithm is generalized to handle nonselfadjoint elliptic partial differential equations. A primal-dual functional is used as the basis for the discretization process, combined with the standard moment collocation for interface constraints which was used for selfadjoint equations. The unknown function is expanded, as usual, as a linear combination of basis functions within each subdomain, or cell, while the moment collocation uses weight functions defined over the interfaces. The transformation of discrete variables which simplifies the interface conditions, coupled with the conjugacy condition on the transformation coefficients, carries over to the nonselfadjoint case and decouples the intracell and interface variables. This makes almost all the computations independent cell-by-cell, rendering it very suitable for parallelizing. Two model convection-diffusion problems are solved with diffusivities down to .0001. The results are presented in the form of surface plots, showing the effect of increasing the number of collocations and cell parameters and the use of margins to contain boundary layers. Comparisons are made among the three conjugate direction iterative procedures used to solve the discrete equations. An effort is made to relate the CD procedure to various classical mathematical relationships.

Key words. cell discretization, partial differential equations, finite-element method, convection-diffusion problems

AMS(MOS) subject classifications. 65N30, 65N35

1. Introduction. In [5], the general scheme for the application of the cell discretization (CD) method to a selfadjoint elliptic partial differential equation was described. In this paper, we shall show how the CD method may be extended to apply to a nonselfadjoint partial differential equation (PDE). We shall apply this modified procedure to two convection-diffusion problems and exhibit graphical results showing the boundary layers exhibited by the solution near the boundaries. The calculations are done with a Fortran program, which presently is set up to handle equations with constant coefficients in rectangular geometries. Provision has been made in the program, however, to install facilities for handling variable coefficients and boundary data (based on adaptive numerical quadrature). The present program can also handle a wide class of systems of equations, but this will be described in a future publication.

2. The primal-dual variational formulation for the nonselfadjoint case. The equation we shall study has the form

$$-\nabla\cdot(\vec{\vec{a}}\cdot\nabla\psi)+\vec{b}\cdot\nabla\psi+c\psi=d. \tag{2.1}$$

The arrow indicates that $\vec{b}$ is a vector and the double arrow that $\vec{\vec{a}}$ is a diadic (a second-rank tensor). $\vec{\vec{a}}$, $\vec{b}$, c, and d may be functions of the independent variables $\{x_i\}$, defined in the (open) domain Ω (in R^n) with boundary Γ ($\bar{\Omega}$ will stand for Ω plus its boundary). Without loss of generality, we can assume $\vec{\vec{a}}$ to be symmetric, so that (2.1) is the same as the equation treated in [5], except for the first-order terms involving $\vec{b}$.

* Received by the editors May 29, 1989; accepted for publication (in final revised form) March 12, 1990.
† Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge CB3 9EW, England. Present address, 752 Mariposa Ave., Mountain View, California 94041.

With nonselfadjoint PDEs it is necessary to introduce a *dual* variable ψ_d to form a simple bilinear functional which will yield (2.1) as its Euler equation:

$$\Phi_0 = \int_\Omega \{\nabla\psi_d \cdot \vec{\vec{a}} \cdot \nabla\psi_p + \psi_d \vec{\alpha} \cdot \nabla\psi_p + (\vec{\beta} \cdot \nabla\psi_d)\psi_p + \gamma\psi_d\psi_p + \psi_p\eta_d + \psi_d\eta_p\}\, d\Omega. \tag{2.2}$$

(When an equation or a remark holds for both the primal and dual quantities, we shall simply leave off the distinguishing subscript.) This functional, modelled on those used in the finite element and related literature serves, like them, to circumvent the necessity of calculating second derivatives.

If the standard variational procedure [4] is applied to (2.2) by varying the dual function ψ_d, the resulting Euler equation will be just (2.1), provided $\vec{\alpha}$, $\vec{\beta}$, γ, η_p, and η_d are assigned the right values. We perform this variation in the usual way by augmenting ψ_d by $\varepsilon_d\delta\psi_d$, where $\delta\psi_d(x)$ is an arbitrary function and ε_d is a parameter. The necessary condition for Φ_0 to be stationary is then

$$\left(\frac{d\Phi_0}{d\varepsilon_d}\right)_{\varepsilon_d=0} = 0, \tag{2.3}$$

which, after the standard integrations by parts, etc. results in the equation

$$\int_\Omega \delta\psi_d\{-\nabla\cdot(\vec{\vec{a}}\cdot\nabla\psi_p) + (\vec{\alpha}-\vec{\beta})\cdot\nabla\psi_p + (\gamma - \nabla\cdot\vec{\beta})\psi_p + \eta_p\}\, d\Omega + \int_\Gamma \delta\psi_d(\psi_{pn} + \beta_n\psi_p)\, d\Gamma = 0. \tag{2.4}$$

The combination $\hat{n}\cdot\vec{\vec{a}}\cdot\nabla\psi_p$ is known as the *coderivative* of ψ_p at the boundary Γ; it is the covariant normal derivative and we shall denote it henceforth simply by $\partial\psi_p/\partial n$ or merely by ψ_{pn}.

Since $\vec{\vec{a}}$ is symmetric, the brace contents in (2.4) match the original equation (2.1) if the following relations hold:

$$\vec{\alpha} - \vec{\beta} = \vec{b}, \qquad \gamma - \nabla\cdot\vec{\beta} = c, \qquad \eta_p = -d \tag{2.5}$$

with η_d undetermined.

There still remains some room for different choices of $\vec{\alpha}$ and $\vec{\beta}$. While no uniquely "best" choice is obvious, it is usually the case that $\vec{\alpha} = \vec{b}$ and $\vec{\beta} = 0$ work well.

The *imposed* boundary conditions are just the same as in [5], except that we have two sets. We define the *trace operator* $\mathcal{U}$, which has the effect of mapping any function $\psi(x)$ defined in the interior of Ω into a function defined on the boundary Γ. (For conciseness, we shall indicate a point in the interior of Ω by x and a point on the boundary Γ by s.) We have

$$\mathcal{U}_p[\psi(x)] \equiv P_p(s)\psi_p(s) + Q_p(s)\frac{\partial\psi_p}{\partial n}(s). \tag{2.6}$$

(In this context, $\partial\psi_p/\partial n$ stands for either the covariant normal derivative or the ordinary normal derivative, whichever is chosen for the problem.)

For suitable boundary data $R_p(s)$ on Γ, the boundary condition itself can be written:

$$\mathcal{U}_p[\psi_p] = R_p \tag{2.7}$$

with a corresponding boundary condition for the dual function.

It is *characteristic* of the cell discretization method that the boundary and interface constraints are incorporated a posteriori by means of linear relations based on moment collocation.

For conciseness, we introduce the usual notation for duality pairing: for two functions $A(s)$ and $B(s)$, defined on Γ, the generalized inner product is

$$\langle A, B\rangle \equiv \int_{\Gamma} A(s)B(s)\, d\Gamma. \tag{2.8}$$

With this notation, we may write the Lagrange multiplier form of the constraint as follows:

$$\langle \lambda_{\mathbf{d}}, \mathcal{U}_{\mathbf{p}}[\psi_{\mathbf{p}}] - R_{\mathbf{p}}\rangle = 0. \tag{2.9}$$

For arbitrary $\lambda_{\mathbf{d}}(s)$, (2.9) is equivalent to (2.7). This, and the corresponding dual constraint, may then be incorporated into a combined functional (sometimes called simply the Lagrangian):

$$\Phi \equiv \Phi_0 - \langle \lambda_{\mathbf{d}}, \mathcal{U}_{\mathbf{p}}[\psi_{\mathbf{p}}] - R_{\mathbf{p}}\rangle - \langle \lambda_{\mathbf{p}}, \mathcal{U}_{\mathbf{d}}[\psi_{\mathbf{d}}] - R_{\mathbf{d}}\rangle. \tag{2.10}$$

3. Moment collocation for "weak" boundary conditions. By highlighting the connection between the classical continuous formulation of a problem and its discrete CD version, we obtain a rather perspicuous derivation of boundary moment collocation. First, we note that, if we perform a variation on $\lambda_{\mathbf{d}}(s)$ in (2.10) (in the sense of the calculus of variations), we get the exact ("strong") boundary conditions:

$$\frac{\delta\Phi}{\delta\lambda_{\mathbf{d}}} = \mathcal{U}_{\mathbf{p}}[\psi_{\mathbf{p}}] - R_{\mathbf{p}} = 0. \tag{3.1}$$

Since our aim is to *weaken* the boundary condition, we restrict $\lambda_{\mathbf{d}}$ to be a *finite* linear combination of functions $\{w_{\mathbf{d}\alpha}(s)\}$. These form a Schauder basis of a weight space $\mathcal{W}(\Gamma)$. We have

$$\lambda_{\mathbf{d}}(s) = \sum_{\alpha=1}^{L} \lambda_{\mathbf{d}\alpha} w_{\mathbf{d}\alpha}(s). \tag{3.2}$$

An approximate version of Φ is then

$$\Phi[\psi, \lambda] = \Phi_0 - \left\langle \sum_{\alpha} \lambda_{\mathbf{d}\alpha} w_{\mathbf{d}\alpha},\ \mathcal{U}_{\mathbf{p}}[\psi_{\mathbf{p}}] - R_{\mathbf{p}} \right\rangle - \cdots. \tag{3.3}$$

We then obtain a "weak" boundary condition by differentiating Φ with respect to $\lambda_{\mathbf{d}\alpha}$:

$$\frac{\partial\Phi}{\partial\lambda_{\mathbf{d}\alpha}} = \langle w_{\mathbf{d}\alpha}, \mathcal{U}_{\mathbf{p}}[\psi_{\mathbf{p}}] - R_{\mathbf{p}}\rangle = 0, \tag{3.4}$$

which is the standard form for boundary moment collocation. The functions $\{w_{\mathbf{d}\alpha}\}$ may be regarded as weights because of their role in the moment formula.

One of the main reasons for writing the integral over Γ as an inner product is to emphasize the primal-dual relationship between the weights and the trace function. This relationship is, in turn, connected to the relationship between $\psi_{\mathbf{p}}$ and $\psi_{\mathbf{d}}$, based on their roles in the functional Φ. Corresponding to the inner product $\langle,\rangle$, we introduce one defined with respect to Ω instead of Γ. We let

$$(A, B) \equiv \int_{\Omega} A(x)B(x)\, d\Omega. \tag{3.5}$$

We can give a qualitative description of the relationships of importance to the cell method based on Sobolev spaces (see, e.g., [9]). Clearly, if a $\psi(x) \in H^1$, its first derivatives are L^2, so that they may be considered to be in H^0. Strictly speaking, it is not necessary that the gradients of $\psi_{\mathbf{p}}$ and $\psi_{\mathbf{d}}$ both be in H^0; the only real requirement is that they should be in dual spaces defined over Ω. However, in the limiting case of $\vec{b} \to 0$ (the selfadjoint case), we would certainly want both ψ's to be in the same space H^1.

For a $\psi(x) \in H^1$ defined in the interior of Ω, its boundary values must be considered to be the limiting values of ψ as $x \to \Gamma$. The set of these limiting values, known as a *trace* of ψ, we write as $\psi(s)$, the argument indicating that it is evaluated on Γ. The ensemble of these trace functions forms another Sobolev space $\mathcal{T}_{\mathscr{D}}$ (for Dirichlet), which has been shown to be $H^{1/2}(\Gamma)$. If it is the normal derivative that is evaluated, then the corresponding trace space $\mathcal{T}_{\mathscr{N}}$ (for Neumann) is $H^{-1/2}$.

As was pointed out in [5], point collocation of a trace function on a boundary leads to regularity difficulties. It may be regarded as a special case of moment collocation when the weight functions are

$$w_{\mathbf{d}\alpha}(s) = \delta(s - s_\alpha), \tag{3.6}$$

where $\delta(s)$ is the Dirac delta function. It is known that the δ-function is in $H^{-n/2-\eta}$, where n is the dimension of the boundary and η is any positive number. Thus, for a two-dimensional problem, $n = 1$, so that $\delta(s) \in H^{-1/2-\eta}(\Gamma)$. On the other hand, the dual of $\mathcal{T}_{\mathscr{D}}$ is $H^{1/2}(\Gamma)$, so that there is a mismatch. In fact, for point collocation, $\psi(x)$ should be in $H^{1+\eta}(\Omega)$, so that an "excessive" regularity is required. The mismatch is even worse for three-dimensional problems where $n = 2$, and for Neumann boundary conditions.

This difficulty is easily dealt with by moment collocation. It is only necessary to have $\mathcal{W} = H^{-1/2}(\Gamma)$, i.e., to use basis functions in that space. If the Neumann trace is involved, $\mathcal{T}_{\mathscr{N}} = H^{-1/2}(\Gamma)$, so that $\mathcal{W}$ must then be $H^{1/2}(\Gamma)$. For the more general mixed boundary condition that we have assumed, it is sufficient to choose $\mathcal{W} = H^{1/2}(\Gamma)$. The coefficient $P(s)$ may then be, if required by the problem, sufficiently irregular as to put $P(s)\psi(s)$ into $H^{-1/2}(\Gamma)$.

It is sometimes necessary that $\psi_{\mathbf{p}}$ and $\psi_{\mathbf{d}}$ be in different spaces, in which case their traces are also in different spaces, and it is then necessary to choose $\mathcal{W}_{\mathbf{p}}$ and $\mathcal{W}_{\mathbf{d}}$ to be different.

4. Full discretization. In [5], the unknown function $\psi(x)$ was expanded as a linear combination of basis functions defined over Ω:

$$\psi(x) = \sum_{\mu=1}^{M} \phi_\mu(x)\theta_\mu. \tag{4.1}$$

The only change for the nonselfadjoint case is to have two separate expansions for the primal and dual functions, respectively. Thus,

$$\psi_{\mathbf{p}}(x) = \sum_{\mu=1}^{M_{\mathbf{p}}} \phi_{\mathbf{p}\mu}(x)\theta_{\mathbf{p}\mu}, \qquad \psi_{\mathbf{d}}(x) = \sum_{\nu=1}^{M_{\mathbf{d}}} \delta_{\mathbf{d}\nu}(x)\theta_{\mathbf{p}\mu}. \tag{4.2}$$

It is possible to have M different for the primal and dual expansions, but this leads to difficult complications and does not seem to offer any advantages. We shall therefore assume $M_{\mathbf{p}} = M_{\mathbf{d}} = M$. We shall discuss the primal part explicitly, with the understanding that the dual part is the same mutatis mutandis.

Since $\mathcal{U}_p$ is a linear trace operator, we have

$$\mathcal{U}_p[\psi_p] = \sum_\mu \mathcal{U}_p[\phi_{p\mu}]\theta_{p\mu} \tag{4.3}$$

so that the collocated boundary condition is

$$\langle w_{d\alpha}, \mathcal{U}_p[\psi_p]\rangle = \sum_\mu \langle w_{d\alpha}, \mathcal{U}_p[\phi_{p\mu}]\rangle\theta_{p\mu} = \langle w_{d\alpha}, R_p\rangle. \tag{4.4}$$

In [5], this array of inner products was collected as a matrix U_p as follows:

$$U_p \equiv \{U_{p\alpha\mu}\} = \langle w_{d\alpha}, \mathcal{U}_p[\phi_{p\mu}]\rangle. \tag{4.5}$$

U_p is an $(L\times M)$ matrix and, if we define

$$\begin{aligned} &\theta_p \equiv \{\theta_{p\mu}\} \quad (\text{an } M\text{-vector}),\\ &R_p \equiv \{R_{p\alpha}\} = \{\langle w_{d\alpha}, R_p\rangle\} \quad (\text{an } L\text{-vector}) \end{aligned} \tag{4.6}$$

we can write (4.4) in the short form:

$$U_p\theta_p = R_p. \tag{4.7}$$

Our next step is to substitute the linear expansions into Φ_0. This is facilitated by defining the following matrices:

$$\begin{aligned} S_{\mu\nu} \equiv \int_\Omega \{&\nabla\phi_{d\mu}\cdot\vec{a}\cdot\nabla\phi_{p\nu} + \phi_{d\mu}(\vec{b}+\vec{\beta})\cdot\nabla\phi_{p\nu}\\ &+(\vec{\beta}\cdot\nabla\phi_{d\mu})\phi_{p\nu} + (c+\nabla\cdot\vec{\beta})\phi_{d\mu}\phi_{p\nu}\}\, d\Omega \end{aligned} \tag{4.8}$$

and

$$T_{p\mu} \equiv -\int_\Omega \eta_p\phi_{d\mu}\, d\Omega, \qquad T_{d\mu} \equiv -\int_\Omega \eta_d\phi_{p\mu}\, d\Omega. \tag{4.9}$$

Note that we have substituted for some of the quantities according to (2.5). S is basically a *stiffness-mass matrix* such as appears in the finite-element method and the T's are the primal and dual forcing or source terms. The result of the substitution is the bilinear form

$$\Phi_0 = \sum_{\mu\nu} \theta_{d\mu}S_{\mu\nu}\theta_{p\nu} - \sum_\mu \theta_{d\mu}T_{p\mu} - \sum_\nu \theta_{p\nu}T_{dn}u, \tag{4.10}$$

which can be written in matrix terms:

$$\Phi_0 = \theta_d^T S\theta_p - \theta_d^T T_p - \theta_p^T T_d. \tag{4.11}$$

We can now collect all of the parts that have been reduced to matrix form with the result:

$$\Phi = \theta_d^T S\theta_p - \theta_d^T T_p - \theta_p^T T_d - \lambda_d^T[U_p\theta_p - R_p] - \lambda_p^T[U_d\theta_d - R_d]. \tag{4.12}$$

Differentiating with respect to θ_d, we obtain the principal subset of the discrete equations

$$\frac{\partial\Phi}{\partial\theta_d} = S\theta_p - T_p - U_d^T\lambda_p = 0, \tag{4.13}$$

the other part being just the imposed boundary condition

$$U_p\theta_p = R_p. \tag{4.14}$$

In principle, these equations can be solved directly, it being fairly easy if S is not singular. Unfortunately, that is the most important case, viz., for Laplace's equation, using a typical basis set which contains a constant. Fortunately, there is a much better way of solving the discrete equations, which we shall describe next.

5. The decoupling transformation to new variables. As in [5], we transform the θ's to new variables to effect a remarkable reduction in the complexity of the boundary conditions (we shall treat interface conditions when we discuss the multicell use). The present treatment differs somewhat from that for the selfadjoint case.

If there are M basis functions $\{\phi_\mu\}$ and there are L moments applied at the boundary, we have seen that θ is an M-vector, R is an L-vector, and U is an $(L\times M)$ matrix. We construct an $(M\times L)$ matrix V and an $(M\times J)$ matrix Z (where $J=M-L$) with the following properties:

$$UV=I, \qquad UZ=0, \tag{5.1}$$

where I and 0 are unit and null matrices, respectively, of appropriate orders. Obviously, V's columns span the range space of U and Z's columns span its null space—by definition. It is not hard to compute these matrices efficiently, *assuming U to have full column rank*, a most essential requirement. (Obviously, this requires that $L\leqq M$.) The method used to find V and Z in the author's Fortran program is to do a Gauss-Jordan elimination on the columns of U and to collect the elementary column operations. If column interchanges are arranged so as to reduce U to a unit matrix in the first L columns and zero for the rest, then the resulting matrix has V for its first L columns and Z for its last J columns. This Gauss-Jordan-based method has been found to be quite adequate for the relatively small matrices involved in the method.

Now, we introduce an L-vector σ and a J-vector ρ and let

$$\theta = V\sigma + Z\rho. \tag{5.2}$$

We set up transformations of this kind for both the primal and dual variables

$$\theta_{\mathrm{p}} = V_{\mathrm{p}}\sigma_{\mathrm{p}} + Z_{\mathrm{p}}\rho_{\mathrm{p}}, \qquad \theta_{\mathrm{d}} = V_{\mathrm{d}}\sigma_{\mathrm{d}} + Z_{\mathrm{d}}\rho_{\mathrm{d}} \tag{5.3}$$

and substitute these formulas into the boundary conditions. We obtain for the primal part

$$R_{\mathrm{p}} = U_{\mathrm{p}}\theta_{\mathrm{p}} = U_{\mathrm{p}}(V_{\mathrm{p}}\sigma_{\mathrm{p}} + Z_{\mathrm{p}}\rho_{\mathrm{p}}) = U_{\mathrm{p}}V_{\mathrm{p}}\sigma_{\mathrm{p}} + U_{\mathrm{p}}Z_{\mathrm{p}}\rho_{\mathrm{p}} = \sigma_{\mathrm{p}}, \tag{5.4}$$

which greatly simplifies the boundary condition. A similar result holds for the duals.

We now substitute (5.3) into (4.12). The result is

$$\begin{aligned}\Phi = {} & \sigma_{\mathrm{d}}^T V_{\mathrm{d}}^T S V_{\mathrm{p}}\sigma_{\mathrm{p}} + \rho_{\mathrm{d}}^T Z_{\mathrm{d}}^T S Z_{\mathrm{p}}\rho_{\mathrm{p}} + \sigma_{\mathrm{d}}^T V_{\mathrm{d}}^T S Z_{\mathrm{p}}\rho_{\mathrm{p}} + \rho_{\mathrm{d}}^T Z_{\mathrm{d}}^T S V_{\mathrm{p}}\sigma_{\mathrm{p}} \\ & - \sigma_{\mathrm{d}}^T V_{\mathrm{d}}^T T_{\mathrm{p}} - \sigma_{\mathrm{p}}^T V_{\mathrm{p}}^T T_{\mathrm{p}} - \rho_{\mathrm{p}}^T Z_{\mathrm{d}}^T T_{\mathrm{p}} - \rho_{\mathrm{p}}^T Z_{\mathrm{p}}^T T_{\mathrm{d}} \\ & - \lambda_{\mathrm{d}}^T(\sigma_{\mathrm{p}} - R_{\mathrm{p}}) - \lambda_{\mathrm{p}}^T(\sigma_{\mathrm{d}} - R_{\mathrm{d}}).\end{aligned} \tag{5.5}$$

We can derive the discrete equations in the (σ, ρ) variables by differentiating Φ with respect to each of them. When we differentiate with respect to the dual variables, we get

$$\begin{aligned}\frac{\partial\Phi}{\partial\sigma_{\mathrm{d}}} &= V_{\mathrm{d}}^T S V_{\mathrm{p}}\sigma_{\mathrm{p}} + V_{\mathrm{d}}^T S Z_{\mathrm{p}}\rho_{\mathrm{p}} - V_{\mathrm{d}}^T T_{\mathrm{p}} - \lambda_{\mathrm{p}} = 0, \\ \frac{\partial\Phi}{\partial\rho_{\mathrm{d}}} &= Z_{\mathrm{d}}^T S V_{\mathrm{p}}\sigma_{\mathrm{p}} + Z_{\mathrm{d}}^T S Z_{\mathrm{p}}\rho_{\mathrm{p}} - Z_{\mathrm{d}}^T T_{\mathrm{p}} = 0\end{aligned} \tag{5.6}$$

and there are corresponding dual relations. Note that these equations can be *decoupled* by causing the cross-terms to vanish identically. It is necessary to adjust V_{p} and V_{d} so that

$$V_{\mathrm{d}}^T S Z_{\mathrm{p}} = 0, \qquad Z_{\mathrm{d}}^T S V_{\mathrm{p}} = 0. \tag{5.7}$$

In order to make this adjustment, we need to calculate the matrix A defined by

$$A = Z_{\mathrm{d}}^T S Z_{\mathrm{p}}, \tag{5.8}$$

which we must assume is *nonsingular*, so that it has an inverse A^{-1}. Obviously, its transpose A^T then has an inverse that we shall denote by A^{-T}. Then, if we denote the *old, uncorrected* V by V^0, the formulas

$$V_{\mathrm{p}} = [I - Z_{\mathrm{p}} A^{-1} Z_{\mathrm{d}}^T S] V_{\mathrm{p}}^0, \qquad V_{\mathrm{d}} = [I - Z_{\mathrm{d}} A^{-T} Z_{\mathrm{p}}^T S^T] V_{\mathrm{d}}^0 \tag{5.9}$$

give the required matrices. This conjugacy relation between the primal and dual V's and Z's is extremely important, as we shall see in the multicell case, because it enables us to reduce almost all the work to cell-by-cell calculations, thereby rendering cell discretization very suitable for parallel processing. It is not hard to show that a V_{p} and V_{d} that satisfy (5.1) and (5.7) are unique, provided A is nonsingular. The Z's are unique only to within a nonsingular transformation of their columns. With these V's, (5.6) reduces to

$$V_{\mathrm{d}}^T S V_{\mathrm{p}} \sigma_{\mathrm{p}} - V_{\mathrm{d}}^T T_{\mathrm{p}} = \lambda_{\mathrm{p}}, \qquad Z_{\mathrm{d}}^T S Z_{\mathrm{p}} \rho_{\mathrm{p}} - Z_{\mathrm{d}}^T T_{\mathrm{p}} = 0. \tag{5.10}$$

6. The multicell case. As in [5], we partition Ω into K subdomains, or cells, $\{\Omega_1, \Omega_2, \cdots, \Omega_K\}$. Between any two *contiguous* cells, say, Ω_k and Ω_m, we have an interface Γ_{km}. In cell discretization, *if two cells are not contiguous, they are not neighbors.* Each cell Ω_k, say, will now have a boundary made up of more than one segment; we can label these segments by using the notation $\{m[k]\}$ to represent the set of labels for those cells which are contiguous neighbors of Ω_k, say, N_k in number.

The function $\psi_{\mathrm{p}}(x)$ is now represented by an *ensemble* of representations $\{\psi_{\mathrm{p}k}(x, \theta_{\mathrm{p}k})\}$, one for each cell:

$$\psi_{\mathrm{p}k}(x) = \sum_{\mu=1}^{M_k} \phi_{\mathrm{p}k\mu}(x) \theta_{\mathrm{p}k\mu} \tag{6.1}$$

with a similar formula for the dual. The functional Φ_0 is now given as the sum of integrals over the cells

$$\begin{aligned} \Phi_0 = \sum_k \int_{\Omega_k} \{ & \nabla \psi_{\mathrm{d}k} \cdot \vec{a}_k \cdot \nabla \psi_{\mathrm{p}k} + \psi_{\mathrm{d}k} \vec{\alpha}_k \cdot \nabla \psi_{\mathrm{p}k} + (\vec{\beta}_k \cdot \nabla \psi_{\mathrm{d}k}) \psi_{\mathrm{p}k} \\ & + \gamma_k \psi_{\mathrm{d}k} \psi_{\mathrm{p}k} + \psi_{\mathrm{p}k} \eta_{\mathrm{d}k} + \Psi_{\mathrm{d}k} \eta_{\mathrm{p}k} \} \, d\Omega_k, \end{aligned} \tag{6.2}$$

which, combined with (6.1) becomes

$$\Phi_0 = \sum_k \{ \theta_{\mathrm{d}k}^T S_k \theta_{\mathrm{p}k} - \theta_{\mathrm{d}k}^T T_{\mathrm{p}k} - \theta_{\mathrm{p}k}^T - \theta_{\mathrm{p}k}^T T_{\mathrm{d}k} \} \tag{6.3}$$

with

$$\begin{aligned} (S_k)_{\mu\nu} = \int_{\Omega_k} \{ & \nabla \phi_{\mathrm{d}k\mu} \cdot \vec{a}_k \cdot \nabla \phi_{\mathrm{p}k\nu} + \phi_{\mathrm{d}k\mu} (\vec{b}_k + \vec{\beta}_k) \cdot \nabla \phi_{\mathrm{p}k\nu} \\ & + (\vec{\beta}_k \cdot \nabla \phi_{\mathrm{d}k\mu}) \phi_{\mathrm{p}k\nu} + (c_k + \nabla \cdot \vec{\beta}_k) \phi_{\mathrm{d}k\mu} \phi_{\mathrm{p}k\nu} \} \, d\Omega_k \end{aligned} \tag{6.4}$$

and

$$T_{\mathbf{p}k\mu} = -\int_{\Omega_k} \eta_{\mathbf{p}k}\phi_{\mathbf{d}k}(x)\, d\Omega_k, \qquad T_{\mathbf{d}k\mu} = -\int_{\Omega_k} \eta_{\mathbf{d}k}\phi_{\mathbf{p}k}(x)\, d\Omega_k. \tag{6.5}$$

Besides boundary conditions, we also have *interface* conditions on all those segments (cell faces) that separate contiguous cells. We label the set of coefficient matrices U with the labels of both of the cells, so that the interface conditions on Γ_{km} are

$$\begin{aligned} U_{\mathbf{p}km}\theta_{\mathbf{p}k} - R_{\mathbf{p}km} &= U_{\mathbf{p}mk}\theta_{\mathbf{p}m} - R_{\mathbf{p}mk}, \\ U_{\mathbf{d}km}\theta_{\mathbf{d}k} - R_{\mathbf{d}km} &= U_{\mathbf{d}mk}\theta_{\mathbf{d}m} - R_{\mathbf{d}mk}, \end{aligned} \tag{6.6}$$

where we have adopted the convention that the label "km" refers to quantities evaluated on Γ_{km} in the kth cell and "mk" refers to similar quantities evaluated on Γ_{km}, but from the other side, in Ω_m. (If we denote the region of R^n *exterior* to Ω by Ω_0, the notation suffices to include the boundary conditions.) The corresponding complete functional is

$$\begin{aligned} \Phi = \Phi_0 &- \sum_{km} \lambda_{\mathbf{d}km}\{[U_{\mathbf{p}km}\theta_{\mathbf{p}k} - R_{\mathbf{p}km}] - [U_{\mathbf{p}mk}\theta_{\mathbf{p}m} - R_{\mathbf{p}mk}]\} \\ &- \sum_{mk} \lambda_{\mathbf{p}mk}\{[U_{\mathbf{d}km}\theta_{\mathbf{d}k} - R_{\mathbf{d}km}] - [U_{\mathbf{d}mk}\theta_{\mathbf{d}m} - R_{\mathbf{d}mk}]\}. \end{aligned} \tag{6.7}$$

Note that the quantities within braces are antisymmetric in km.

We shall now apply the transformation to the σ and ρ variables, after finding the appropriate V's and Z's. We note first that for each cell Ω_k, the $U_{km[k]}$ may be collected into a "global" U_k whose rows are the assembled rows of all of the individual coefficient matrices, each associated with a single face of Ω_k. Thus,

$$U_k = \begin{bmatrix} U_{km_1} \\ U_{km_2} \\ \vdots \\ U_{km_{N_k}} \end{bmatrix}, \tag{6.8}$$

where, as noted before, N_k is the number of faces of Ω_k. We have assumed L_{km} moments on Γ_{km}, and M_k coefficients $\{\theta_{k\mu}\}$, so that U_{km} is an $(L_{km} \times M_k)$ matrix. U_k, therefore, is in turn an $(L_k \times M_k)$ matrix, where

$$L_k \equiv \sum_{m[k]} L_{km[k]}. \tag{6.9}$$

We assume, as before, that U_k is nondegenerate, i.e., it has full row rank (requiring that $L_k \leqq M_k$), so that we can construct V_k and Z_k using the procedure given in § 5. The result is an $(M_k \times L_k)$ matrix V_k and an $(M_k \times J_k)$ matrix Z_k, where $J_k = M_k - L_k$. We again have

$$U_k V_k = I, \qquad U_k Z_k = 0 \tag{6.10}$$

with the understanding that the foregoing relations hold for primal and dual quantities. In addition, we adjust the V's for each cell so that

$$V_{\mathbf{d}k}^T S_k Z_{\mathbf{p}k} = 0, \qquad Z_{\mathbf{d}k}^T S_k V_{\mathbf{p}k} = 0. \tag{6.11}$$

We now *repartition* each V_k into a set $\{V_{km[k]}\}$ of submatrices, each with the same

number of columns as the corresponding $U_{km[k]}$ has rows. Thus, matching (6.8), we have

$$V_k = [V_{km_1}, V_{km_2}, \cdots, V_{km_{N_k}}]. \tag{6.12}$$

Z_k remains a matrix associated with Ω_k as a whole, and not with any particular face or faces. These new matrices satisfy the following relationships, based on (6.10) and (6.11):

$$U_{kp}V_{kq} = \delta_{pq}I, \qquad U_{kp}Z_k = 0 \tag{6.13}$$

for both primal and dual, and

$$V^T_{\mathbf{d}km}S_kZ_{\mathbf{p}k} = 0, \qquad Z^T_{\mathbf{d}k}S_kV_{\mathbf{p}km} = 0. \tag{6.14}$$

We now change variables in Ω_k. The transformation corresponding to (5.2) is

$$\theta_k = \sum_{p[k]} V_{kp}(\sigma_{kp} + R_{kp}) + Z_k\rho_k, \tag{6.15}$$

where the summation is, as the notation states, over the labels of the contiguous neighbors of Ω_k. When we apply the discrete operator U_{km} to θ_k, we obtain

$$\begin{aligned} U_{km}\theta_k &= \sum_{p[k]} U_{km}V_{kp}(\sigma_{kp} + R_{kp}) + U_{km}Z_k\rho_k \\ &= \sum_{p[k]} \delta_{mp}(\sigma_{kp} + R_{kp}) + 0\cdot\rho_k \\ &= \sigma_{km} + R_{km} \end{aligned} \tag{6.16}$$

with a matching reduction for $U_{mk}\theta_m$. Equation (6.6) then reduces to

$$(\sigma_{km} + R_{km}) - R_{km} = (\sigma_{mk} + R_{mk}) - R_{mk} \tag{6.17}$$

or

$$\sigma_{km} = \sigma_{mk} \tag{6.18}$$

for primal and dual quantities.

Next, we substitute (6.15) into (6.3) (with a change of index names) to obtain the result:

$$\begin{aligned} \Phi = \sum_{q=1}^{K} [\Phi_{\sigma\sigma,q} + \Phi_{\rho\rho,q} + \Phi_{\sigma\rho,q} + \Phi_{c,q}] \\ - \sum_{\substack{q=1 \\ r=1}}^{K} \lambda_{\mathbf{d}qr}(\sigma_{\mathbf{p}qr} - \sigma_{\mathbf{p}rq}) - \sum_{\substack{q=1 \\ r=1}}^{K} \lambda_{\mathbf{p}qr}(\sigma_{\mathbf{d}qr} - \sigma_{\mathbf{d}rq}) \end{aligned} \tag{6.19}$$

with

$$\begin{aligned} \Phi_{\sigma\sigma,q} = \sum_{\substack{r[q] \\ s[q]}} [\sigma^T_{\mathbf{d}qr}V^T_{\mathbf{d}qr}S_qV_{\mathbf{p}qs}\sigma_{\mathbf{p}qs} + \sigma^T_{\mathbf{d}qr}V^T_{\mathbf{d}qr}S_qV_{\mathbf{p}qs}R_{\mathbf{p}qs} + R^T_{\mathbf{d}qr}V^T_{\mathbf{d}qr}S_qV_{\mathbf{p}qs}\sigma_{\mathbf{p}qs}] \\ - \sum_{r[q]} [\sigma^T_{\mathbf{d}qr}V^T_{\mathbf{d}qr}T_{\mathbf{p}q} + T^T_{\mathbf{d}q}V_{\mathbf{p}qr}\sigma_{\mathbf{p}qr}], \end{aligned} \tag{6.20}$$

$$\begin{aligned} \Phi_{\rho\rho,q} = \rho^T_{\mathbf{d}q}Z^T_{\mathbf{d}q}S_qZ_{\mathbf{p}q}\rho_{\mathbf{p}q} + \sum_{r[q]} [R^T_{\mathbf{d}qr}V^T_{\mathbf{d}qr}S_qZ_{\mathbf{p}q}\rho_{\mathbf{p}q} + \rho^T_{\mathbf{d}q}Z^T_{\mathbf{d}q}S_qV_{\mathbf{p}qr}R_{\mathbf{p}qr}] \\ - \rho^T_{\mathbf{d}q}Z^T_{\mathbf{d}q}T_{\mathbf{p}q} - T^T_{\mathbf{d}q}Z_{\mathbf{p}q}\rho_{\mathbf{p}q}, \end{aligned} \tag{6.21}$$

$$\Phi_{\sigma\rho,q} = \sum_{r[q]} [\sigma^T_{\mathbf{d}qr} V^T_{\mathbf{d}qr} S_q Z_{\mathbf{p}q} \rho_{\mathbf{p}q} + \rho^T_{\mathbf{d}q} Z^T_{\mathbf{d}q} S_q V_{\mathbf{p}qr} \sigma_{\mathbf{p}qr}], \tag{6.22}$$

$$\begin{aligned}\Phi_{c,q} = &\sum_{\substack{r[q]\\ s[q]}} [R^T_{\mathbf{d}qr} V^T_{\mathbf{d}qr} S_q V_{\mathbf{p}qs} R_{\mathbf{p}qs}] \\ &- \sum_{r[q]} [R^T_{\mathbf{d}qr} V^T_{\mathbf{d}qr} T_{\mathbf{p}q} + T^T_{\mathbf{d}q} + T^T_{\mathbf{d}q} V_{\mathbf{p}qr} R_{\mathbf{p}qr}],\end{aligned} \tag{6.23}$$

and the understanding that the double summations are only other values of q and r that label contiguous neighbors.

We can ignore the constant part $\Phi_{c,q}$, because we are going to differentiate Φ with respect to the variables σ and ρ. It is extremely useful to decouple σ and ρ, so that the discrete equation for one will not involve the other. We see that this confers the very attractive feature of being able to do almost everything cell-by-cell, as mentioned already in § 5. The term interfering with this is $\Phi_{\sigma\rho,q}$, which vanishes because of (6.14).

To shorten the equations, we define

$$H_{qrs} \equiv V^T_{\mathbf{d}qr} S_q V_{\mathbf{p}qs}, \qquad A_q \equiv Z^T_{\mathbf{d}q} S_q Z_{\mathbf{p}q}, \tag{6.24}$$

$$\begin{aligned}G_{\mathbf{p}qr} &\equiv V^T_{\mathbf{d}qr} T_{\mathbf{p}q} - \sum_{s[q]} H_{qrs} R_{\mathbf{p}qs}, \\ G_{\mathbf{d}qr} &\equiv V^T_{\mathbf{p}qr} T_{\mathbf{d}q} - \sum_{s[q]} H^T_{qrs} R_{\mathbf{d}qs},\end{aligned} \tag{6.25}$$

$$J_{\mathbf{p}q} \equiv Z^T_{\mathbf{d}q} T_{\mathbf{p}q}, \qquad J_{\mathbf{d}q} \equiv Z^T_{\mathbf{p}q} T_{\mathbf{d}q}. \tag{6.26}$$

Then we have, for Φ,

$$\begin{aligned}\Phi = \sum_{q=1}^{K} \Bigg\{ &\sum_{\substack{r[q]\\ s[q]}} \sigma^T_{\mathbf{d}qr} H_{qrs} \sigma_{\mathbf{p}qs} - \sum_{r[q]} [\sigma^T_{\mathbf{d}qr} G_{\mathbf{p}qr} + G^T_{\mathbf{d}qr} \sigma_{\mathbf{p}qr}] \\ &+ \rho^T_{\mathbf{d}q} A_q \rho_{\mathbf{p}q} - \rho^T_{\mathbf{d}q} J_{\mathbf{p}q} - J^T_{\mathbf{d}q} \rho_{\mathbf{p}q} \Bigg\} \\ &- \sum_{\substack{q=1\\ r=1}}^{K} \lambda_{\mathbf{d}qr} (\sigma_{\mathbf{p}qr} - \sigma_{\mathbf{p}rq}) - \sum_{\substack{q=1\\ r=1}}^{K} \lambda_{\mathbf{p}qr} (\sigma_{\mathbf{d}qr} - \sigma_{\mathbf{d}rq}).\end{aligned} \tag{6.27}$$

To obtain the discrete equations for σ and ρ, we differentiate, first with respect to $\rho_{\mathbf{d}k}$. We obtain

$$A_k \rho_{\mathbf{p}k} = J_{\mathbf{p}k}. \tag{6.28}$$

Obviously, each ρ_k can be found for Ω_k independently of any other cell. This part of the solution can therefore readily be parallelized.

We next differentiate Φ with respect to $\sigma_{\mathbf{d}km}$. The result is

$$\sum_{s[k]} H_{kms} \sigma_{\mathbf{p}ks} - G_{\mathbf{p}km} - (\lambda_{\mathbf{p}km} - \lambda_{\mathbf{p}mk}) = 0. \tag{6.29}$$

The antisymmetry of the interface conditions is reflected in the antisymmetry of the combined λ terms. We make use of this important fact to eliminate the λ's. We interchange k and m in (6.29) and add the result to (6.29). The λ terms cancel and we are left with

$$\sum_{s[k]} H_{kms} \sigma_{\mathbf{p}ks} + \sum_{s[m]} H_{mks} \sigma_{\mathbf{p}ms} - G_{\mathbf{p}km} - G_{\mathbf{p}mk} = 0. \tag{6.30}$$

Equation (6.30) has been solved by various forms of preconditioned conjugate gradient and conjugate direction methods, three of which will be mentioned in § 8. The preconditioning matrix used was simply the block-diagonal matrix of (6.30). This may be extracted from (6.30) by withdrawing the term with $s[k]=m$ from the first sum and the term with $s[m]=k$ from the second. Since $\sigma_{\mathbf{p}mk}=\sigma_{\mathbf{p}km}$, we get

$$\Lambda_{km}\sigma_{\mathbf{p}km}+\sum_{s[k]}{}'\,H_{kms}\sigma_{\mathbf{p}ks}+\sum_{s[m]}{}'\,H_{mks}\sigma_{\mathbf{p}ms}-G_{\mathbf{p}km}-G_{\mathbf{p}mk}=0 \tag{6.31}$$

with

$$\Lambda_{km}\equiv H_{kmm}+H_{mkk} \tag{6.32}$$

and the primes on the summations mean that the km and mk terms are left out. Equations (6.29) and (6.31) are the discrete equations actually solved in the computer program, with the Λ's used as preconditioners.

We emphasize again that all of the integrations over cells and interfaces, all of the computations for V, Z, etc., and the computation of H, A, and ρ can be done independently cell-by-cell, provided only that there is no degeneracy. The only computations that involve more cells than one are those for Λ and for the σ's. Hence all of the first group of calculations can be done in parallel without any reorganization of the data or of the order of computation. For the treatment of σ's, on the other hand, some special measures would have to be taken in the computation of inner products, because the basic iterations in the conjugate direction procedures are Jacobi iterations, which are also independent and hence parallelizable. These algebraic considerations apply whatever the cell structure or the particular representations of the solution, provided only that there is no degeneracy.

7. Variationally induced discrete conservation relations. We shall add here to the connection already made between the discrete problem arrived at by CD and its underlying continuous problem, particularly with respect to the variational approach. Our aim is to clarify the way in which conservation laws in the continuous problem are carried over in the discrete approximation.

In § 2, it was only stated what the induced PDE (the Euler equation) and the induced boundary conditions were, without going into detail as to how these were derived. The method is classical and depends on a more detailed consideration of $\delta\psi_{\mathbf{d}}$ in (2.4). The function $\delta\psi_{\mathbf{d}}$ is first chosen so that it vanishes on the boundary; i.e., its Dirichlet trace vanishes. Furthermore, it is constructed so as to vanish everywhere in Ω save in a small neighborhood of a generic interior point x, where it is positive. With this choice, the surface integral vanishes identically and the volume integral reduces to an integral over the small neighborhood. Since $\delta\psi_{\mathbf{d}}$ is positive and the integral vanishes identically, it is proved in books on the calculus of variations [4] that the multiplier of $\delta\psi_{\mathbf{d}}$, viz., the brace contents in (2.4) must vanish at x.

In the next step, the volume integral is disregarded (because it vanishes for all $\delta\psi_{\mathbf{d}}$) and the behavior of $\delta\psi_{\mathbf{d}}$ on the surface is examined. For all intents and purposes, $\delta\psi_{\mathbf{d}}$ *effectively vanishes* in Ω. It is also constructed to vanish everywhere on Γ save in a small neighborhood of the generic point s. The same argument as before shows that the multiplier of $\delta\psi_{\mathbf{d}}$ in the surface integral of (2.4) must vanish at s.

In [8], it was shown how continuous operators $\mathcal{U}$, $\mathcal{Y}$, $\mathcal{V}$, and $\mathcal{Z}$ corresponding to the matrices U, Y, V, and Z can be constructed in terms of kernels $U(s,x)$, $Y(\xi,x)$, $V(x,s)$, and $Z(x,\xi)$. (We have used ξ to indicate explicitly a point in the interior, i.e., in Ω.) The operator $\mathcal{U}$, which we have already considered, maps a function $\psi(x)$, defined over Ω, into a trace function $\mathrm{Tr}[\psi](s)$, defined over Γ. Therefore, $\mathcal{U}$ maps ψ

into a function that has support only on Γ. Similarly, $\mathscr{Y}$ maps $\psi(x)$ into a function whose support is in Ω only. In kernel form, these operators act as follows:

$$\int_\Omega U(s,x)\psi(x)\,d\Omega=\chi(s), \qquad \int_\Gamma Y(\xi,x)\psi(x)\,d\Gamma=\zeta(\xi). \tag{7.1}$$

$\mathscr{V}$ and $\mathscr{Z}$, on the other hand, are operators that map their domain functions, defined over Γ or Ω, into functions defined over $\bar{\Omega}$. If we now form the combination

$$\psi(x)=\int_\Gamma V(x,s)\sigma(s)\,d\Gamma+\int_\Omega Z(x,\xi)\rho(\xi)\,d\Omega \tag{7.2}$$

we see that ψ is made up of two parts: the first term is annihilated by $\mathscr{Y}$, which means it vanishes in Ω, and the second term is annihilated by $\mathscr{U}$, which means it vanishes on Γ. These properties are summarized by the integral relations:

$$\begin{aligned}
&\int_\Omega U(s,x)V(x,t)\,d\Omega=\delta(s,t),\\
&\int_\Omega U(s,x)Z(x,\xi)\,d\Omega=0,\\
&\int_\Omega Y(\xi,x)V(x,t)\,d\Omega=0,\\
&\int_\Omega Y(\xi,x)Z(x,\eta)\,d\Omega=\delta(\xi,\eta).
\end{aligned} \tag{7.3}$$

Hence, (7.2) is just the decomposition we need to perform conveniently the two variations necessary to derive the Euler equation and the induced boundary condition. We simply vary ρ for the first result and σ for the second.

This construction clarifies the connection with the discrete variables ρ and σ used in § 6. It shows that the differentiations with respect to ρ and σ are discrete versions of the selective variations used for the continuous problem. Therefore, it is fair to identify the derivative of the discrete Φ with respect to the discrete ρ with the intracell Euler equation (contained in (2.4)), and the derivative with respect to the discrete σ with the interface current conditions.

The multicell version of (4.12), the discrete functional, is

$$\begin{aligned}
\Phi=&\sum_{r=1}^{K}[\theta_{\mathbf{d}r}^T S_r\theta_{\mathbf{p}r}-\theta_{\mathbf{d}r}^T T_{\mathbf{p}r}-\theta_{\mathbf{p}r}^T T_{\mathbf{d}r}]\\
&-\sum_{rs}\lambda_{\mathbf{d}rs}^T[U_{\mathbf{p}rs}\theta_{\mathbf{p}r}-R_{\mathbf{p}rs}-U_{\mathbf{p}sr}\theta_{\mathbf{p}s}+R_{\mathbf{p}sr}]\\
&-\sum_{rs}\lambda_{\mathbf{p}rs}^T[U_{\mathbf{d}rs}\theta_{\mathbf{d}r}-R_{\mathbf{d}rs}-U_{\mathbf{d}sr}\theta_{\mathbf{d}s}+R_{\mathbf{d}sr}].
\end{aligned} \tag{7.4}$$

We differentiate first with respect to $\rho_{\mathbf{d}k}$ to obtain the primal "Euler equation" for the kth cell in terms of θ:

$$\begin{aligned}
\frac{\partial\Phi}{\partial\rho_{\mathbf{d}k}}&=\frac{\partial\theta_{\mathbf{d}k}}{\partial\rho_{\mathbf{d}k}}\frac{\partial\Phi}{\partial\theta_{\mathbf{d}k}}\\
&=Z_{\mathbf{d}k}^T\left[S_k\theta_{\mathbf{p}k}-T_{\mathbf{p}k}-\sum_{r[k]}U_{\mathbf{d}kr}^T[\lambda_{\mathbf{p}kr}-\lambda_{\mathbf{p}rk}]\right]\\
&=Z_{\mathbf{d}k}^T[S_k\theta_{\mathbf{p}k}-T_{\mathbf{p}k}]=0
\end{aligned} \tag{7.5}$$

with the help of (6.15) and (6.13).

The discrete interface relations, in terms of the θ's, are derived according to the scheme we have put forward above:

$$\begin{aligned}\frac{\partial\Phi}{\partial\sigma_{\mathbf{d}km}} &= \frac{\partial\theta_{\mathbf{d}k}}{\partial\sigma_{\mathbf{d}km}}\frac{\partial\Phi}{\partial\theta_{\mathbf{d}k}} \\ &= V^T_{\mathbf{d}km}\left[S_k\theta_{\mathbf{p}k} - T_{\mathbf{p}k} - \sum_{r[k]} U^T_{\mathbf{d}kr}[\lambda_{\mathbf{p}kr} - \lambda_{\mathbf{p}sk}]\right] \\ &= V^T_{\mathbf{d}km}[S_k\theta_{\mathbf{p}k} - T_{\mathbf{p}k}] - (\lambda_{\mathbf{p}km} - \lambda_{\mathbf{p}mk}) = 0\end{aligned} \tag{7.6}$$

with the help of (6.15) and (6.13). To eliminate the λ's, we interchange k and m and add, as before. The result is

$$V^T_{\mathbf{d}km}[S_k\theta_{\mathbf{p}k} - T_{\mathbf{p}k}] + V^T_{\mathbf{d}mk}[S_m\theta_{\mathbf{p}m} - T_{\mathbf{p}m}] = 0. \tag{7.7}$$

These are the discrete approximations to the induced interface conditions. Returning now to the continuous case, we can construct from (2.2) and (2.10) the complete multicell functional

$$\begin{aligned}\Phi = \sum_q \int_{\Omega_q} &\{\nabla\psi_{\mathbf{d}q}\cdot\vec{a}_q\cdot\nabla\psi_{\mathbf{p}q} + \psi_{\mathbf{d}q}\vec{\alpha}_q\cdot\nabla\psi_{\mathbf{p}q} + (\vec{\beta}_q\cdot\nabla\psi_{\mathbf{d}q})\psi_{\mathbf{p}q} \\ &+ \gamma_q\psi_{\mathbf{d}q}\psi_{\mathbf{p}q} + \psi_{\mathbf{p}q}\eta_{\mathbf{d}q} + \psi_{\mathbf{d}q}\eta_{\mathbf{p}q}\}\, d\Omega_q \\ -\sum_{qr}\int_{\Gamma_{qr}} &\lambda_{\mathbf{d}qr}(\mathcal{U}_{\mathbf{p}qr}[\psi_{\mathbf{p}qr}] - R_{\mathbf{p}qr} - \mathcal{U}_{\mathbf{p}rq}[\psi_{\mathbf{p}rq}] + R_{\mathbf{p}rq})\, d\Gamma_{qr} \\ -\sum_{qr}\int_{\Gamma_{qr}} &\lambda_{\mathbf{p}qr}(\mathcal{U}_{\mathbf{d}qr}[\psi_{\mathbf{d}qr}] - R_{\mathbf{d}qr} - \mathcal{U}_{\mathbf{d}rq}[\psi_{\mathbf{d}rq}] + R_{\mathbf{d}rq})\, d\Gamma_{qr}.\end{aligned} \tag{7.8}$$

We shall illustrate the analogy between the discrete and continuous forms by first reducing one of the boundary traces. We have

$$\begin{aligned}\mathcal{U}[\psi] &\to \int_{\Omega_x} U(s,x)\psi(x)\, d\Omega_x \\ &= \int_{\Omega_x} U(s,x)\left\{\int_{\Gamma_t} V(x,t)\sigma(t)\, d\Gamma_t + \int_{\Omega_x} Z(x,\xi)\rho(\xi)\, d\Omega_\xi\right\} d\Omega_x \\ &= \int_{\Gamma_t}\int_{\Omega_x} U(s,x)V(x,t)\, d\Omega_x\sigma(t)\, d\Gamma_t + \int_{\Omega_\xi}\int_{\Omega_x} U(s,x)Z(x,\xi)\, d\Omega_x\rho(\xi)\, d\Omega_\xi \\ &= \int_{\Gamma_t}\delta(s,t)\sigma(t)\, d\Gamma_t + \int_{\Omega_\xi} 0\cdot\rho(\xi)\, d\Omega_\xi \\ &= \sigma(s)\end{aligned} \tag{7.9}$$

(assuming, naturally, that all the integrations can be interchanged). Similar relations hold among the other sourcelike operators; moreover, the same kinds of relationships hold among the various operators for different cells and interfaces. Thus, the interface constraint terms in (7.8) reduce to

$$\begin{aligned}\Phi_{\text{constraint}} = -\sum_{qr}\int_{\Gamma_{qr}} &\lambda_{\mathbf{d}qr}(s)[\sigma_{\mathbf{p}qr}(s) - \sigma_{\mathbf{p}rq}(s)]\, d\Gamma_{qr} \\ -\sum_{qr}\int_{\Gamma_{qr}} &\lambda_{\mathbf{p}qr}(s)[\sigma_{\mathbf{d}qr}(s) - \sigma_{\mathbf{d}rq}(s)]\, d\Gamma_{qr}.\end{aligned} \tag{7.10}$$

Now the variation process we wish to apply would replace $\psi_{\mathbf{d}q}$ by its representation (7.2), replace $\rho_{\mathbf{d}q}$ by $\rho_{\mathbf{d}q}+\varepsilon_{\mathbf{d}q}\delta\rho_{\mathbf{d}q}$, differentiate with respect to $\varepsilon_{\mathbf{d}k}$, and set $\varepsilon_{\mathbf{d}k}=0$ in the result. The net effect of this, and the usual integrations by parts, etc., applied to Φ is the equation

$$\begin{aligned}&\int_{\Gamma_k} Z_{\mathbf{d}k}(s,\xi)\left(\frac{\partial\psi_{\mathbf{p}k}}{\partial n}+(\beta_n)_k\psi_{\mathbf{p}k}\right)(s)\,d\Gamma_k(s)\\&\quad+\int_{\Omega_k} Z_{\mathbf{d}k}(x,\xi)\mathscr{E}_{\mathbf{p}k}[\psi_{\mathbf{p}k}](x)\,d\Omega_k(x)=0,\end{aligned}\tag{7.11}$$

where $\mathscr{E}_{\mathbf{p}k}$ is the "Euler operator," defined by

$$\mathscr{E}_{\mathbf{p}k}[\psi_{\mathbf{p}k}]\equiv-\nabla\cdot(\vec{a}_k\cdot\nabla\psi_{\mathbf{p}k})+(\vec{\alpha}_k-\vec{\beta}_k)\cdot\nabla\psi_{\mathbf{p}k}+(\gamma_k-\nabla\cdot\vec{\beta}_k)\psi_{\mathbf{p}k}+\eta_{\mathbf{p}k}.\tag{7.12}$$

The treatment of the variation with respect to the $\sigma_{\mathbf{d}}$'s is entirely analogous, except that the separate faces of the cells cannot be combined. The elimination of the $\lambda_{\mathbf{d}km}(s)$ is done just as in (7.6), (7.7). The final result is

$$\begin{aligned}&\int_{\Gamma_{km}} V_{\mathbf{d}km}(s,t)\left(\frac{\partial\psi_{\mathbf{p}k}}{\partial n}+(\beta_n)_k\psi_{\mathbf{p}k}\right)(s)\,d\Gamma_{km}(s)\\&\quad+\int_{\Gamma_{km}} V_{\mathbf{d}mk}(s,t)\left(\frac{\partial\psi_{\mathbf{p}m}}{\partial n}+(\beta_n)_m\psi_{\mathbf{p}m}\right)(s)\,d\Gamma_{km}(s)\\&\quad+\int_{\Omega_k} V_{\mathbf{d}km}(x,t)\mathscr{E}_{\mathbf{p}k}[\psi_{\mathbf{p}k}](x)\,d\Omega_k(x)\\&\quad+\int_{\Omega_m} V_{\mathbf{d}mk}(x,t)\mathscr{E}_{\mathbf{p}m}[\psi_{\mathbf{p}m}](x)\,d\Omega_m(x)=0.\end{aligned}\tag{7.13}$$

Equations (7.12) and (7.13) are the approximate versions of the Euler equation and the induced interface condition, respectively.

In order better to relate these equations to classical results, we shall make some special assumptions. Let us assume that the trace operator $\mathscr{U}$ is, in fact, the *Dirichlet* trace operator, i.e.,

$$\mathscr{U}_{\mathbf{d}km}[\psi_{\mathbf{p}k}(x)]=\psi_{\mathbf{p}k}(s)\quad(s\text{ on }\Gamma_{km}).\tag{7.14}$$

Based on (7.3), it is not hard to show that the following relations hold:

$$\begin{aligned}V_{\mathbf{d}km}(s,t)&=\int_{\Omega_k} U_{\mathbf{d}km}(s,x)V_{\mathbf{d}km}(x,t)\,d\Omega_k=\delta(s,t),\\Z_{\mathbf{d}k}(s,\xi)&=\int_{\Omega_k} U_{\mathbf{d}km}(s,x)Z_{\mathbf{d}k}(x,\xi)\,d\Omega_k=0.\end{aligned}\tag{7.15}$$

This means, in the first place, that the surface term in (7.11) vanishes identically, leaving the second term to fill the role of Euler equation in the discrete approximation.

Turning to (7.13), the surface integrals collapse to the expressions for the currents, evaluated on Γ_{km}. We have

$$\left[\frac{\partial\psi_{\mathbf{p}k}}{\partial n}+(\beta_n)_k\psi_{\mathbf{p}k}\right]_{\Gamma_{km}}+\left[\frac{\partial\psi_{\mathbf{p}m}}{\partial n}+(\beta_n)_m\psi_{\mathbf{p}m}\right]_{\Gamma_{km}}$$

$$+\int_{\Omega_k} V_{\mathbf{d}km}(x,t)\mathscr{E}_{\mathbf{p}k}[\psi_{\mathbf{p}k}](x)\,d\Omega_k(x) \tag{7.16}$$

$$+\int_{\Omega_m} V_{\mathbf{d}mk}(x,t)\mathscr{E}_{\mathbf{p}m}[\psi_{\mathbf{p}m}](x)\,d\Omega_m(x)=0$$

and if the Euler equations are actually satisfied in each of paired contiguous cells, (7.13) reduces to the classical induced interface condition

$$\left[\frac{\partial\psi_{\mathbf{p}k}}{\partial n}+(\beta_n)_k\psi_{\mathbf{p}}k\right]_{\Gamma_{km}}+\left[\frac{\partial\psi_{\mathbf{p}m}}{\partial n}+(\beta_n)_m\psi_{\mathbf{p}m}\right]_{\Gamma_{km}}, \tag{7.17}$$

which expresses the conservation of current across the interface.

Finally, it is interesting that the representation (7.2) is strongly analogous to the classical Green formula for the solution of a Poisson equation viz.

$$\psi(x)=\int_\Omega G(x,\xi)\rho(\xi)\,d\Omega-\int_\Gamma\frac{\partial G}{\partial n_s}(x,s)\psi(s)\,d\Gamma \tag{7.18}$$

with Dirichlet boundary conditions. (The n_s is used to indicate the normal derivative with respect to the second argument of G.) Clearly, Z corresponds to G and V corresponds to $-\partial G/\partial n$. In fact, $-(\partial G/\partial n)(t,s)=\delta(t,s)$, since $\psi(x)$ must take on its trace value on the boundary. Moreover, G is constructed in such a way that $G(t,\xi)=0$, i.e., G vanishes on the boundary.

However, V and Z differ in an important way from G. The latter satisfies a Poisson equation with point source in Ω. This is sufficient to make ψ satisfy a Poisson equation with an arbitrary source ρ. $Z(x,\xi)$, on the other hand, does not satisfy any differential equation in the conventional sense, but does satisfy a weak condition which is related to one. The continuous version of (5.7) is, following the argument earlier in this section,

$$\begin{aligned}&\int_\Omega\{\nabla V_{\mathbf{d}}\cdot\vec{a}\cdot\nabla Z_{\mathbf{p}}+V_{\mathbf{d}}\vec{\alpha}\cdot\nabla Z_{\mathbf{p}}+(\vec{\beta}\cdot\nabla V_{\mathbf{d}})Z_{\mathbf{p}}+\gamma V_{\mathbf{d}}Z_{\mathbf{p}}\}\,d\Omega=0,\\ &\int_\Omega\{\nabla Z_{\mathbf{d}}\cdot\vec{a}\cdot\nabla V_{\mathbf{p}}+Z_{\mathbf{d}}\vec{\alpha}\cdot\nabla V_{\mathbf{p}}+(\vec{\beta}\cdot\nabla Z_{\mathbf{d}})V_{\mathbf{p}}+\gamma Z_{\mathbf{d}}V_{\mathbf{p}}\}\,d\Omega=0.\end{aligned} \tag{7.19}$$

Now, *if* $V_{\mathbf{d}}$ *and* $Z_{\mathbf{d}}$ *were arbitrary* we could apply the basic variational argument and force $V_{\mathbf{p}}$ and $Z_{\mathbf{p}}$ to satisfy the primal Euler equation (in the sense of each being a fundamental solution with an appropriate singularity), in which case they could play the roles of the Green function and its normal derivative. However, such is not the case—$V_{\mathbf{d}}$ and $Z_{\mathbf{d}}$ are operators satisfying only (7.19), so the burden of satisfying the differential equation is still, so to speak, on ρ and σ. They are *not* arbitrary, but must satisfy (6.29) and (6.31). V and Z are extension operators which only cause ψ to satisfy the interface and boundary conditions automatically regardless of the values of ρ and σ. The conditions (5.7) serve only to decouple these new variables.

8. Solution of the discrete equations. Equations (6.29) are solved in the author's Fortran program by Gauss-Jordan elimination. As remarked several times, each set is solved independently of those in any other cell. The system may vary in order from a few (5 to 6) to around 100. The matrix A may sometimes be singular—for ill-posed problems—but other than that, it has never proved to be ill conditioned.

Equations (6.31), on the other hand, may be a rather large system. For a special class of self-adjoint equations, the author solved a system of 3,800 equations some years ago [6] using the generalized conjugate gradient method of Concus, Golub, and

O'Leary [2], which worked outstandingly well. For nonselfadjoint equations, it proved unreliable (indeed, it was never intended for unsymmetric systems), but occasionally succeeded. However, it was only prudent to look for a more robust method, so three other conjugate direction methods were tried—as described very clearly and concisely in the dissertation of Elman [3]. These are (1) the conjugate gradient method applied to the normal equations—called CGNR—("guaranteed" to work for any nonsingular system), (2) the conjugate direction method applied to the normal equations—called CGNE—(also "guaranteed"), and (3) the Orthomin(1) method (so-called by Elman because it is a variant of Orthomin which only "remembers" one old vector). In the description of the numerical results in the following section, an informal comparison will be made for the various problems tried (actually only two in number, but with different cell structures, numbers of interface moments, etc.). All of the methods include the possibility of preconditioning, i.e., pre- or postmultiplying the coefficient matrix H by some nonsingular matrix Q^{-1}, where Q is an approximation to H in some limited sense. This is supposed to render the altered H into something like the unit matrix, which is very well conditioned, and for which all conjugate direction methods work excellently well.

9. Numerical results. We shall show here the results of applying the CD algorithm described above to two model problems suggested by referees of a previous paper.

The first problem is to solve the equation

$$-a\nabla^2\psi + b\frac{\partial\psi}{\partial x} = 0 \tag{9.1}$$

in the unit square, with boundary conditions

$$\begin{aligned} &\psi(x,0) = 0, \qquad \psi(x,1) = 1, \\ &\psi(0,y) = 1, \qquad \psi_x(1,y) = 0. \end{aligned} \tag{9.2}$$

The layout is shown in Fig. 1.

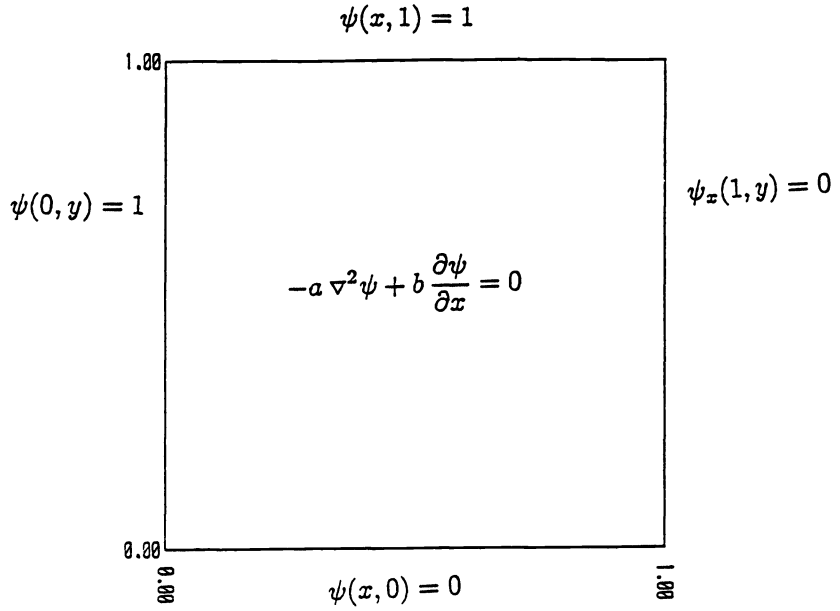


FIG. 1. *Problem* 1. *Definition of problem.*

We can anticipate difficulty from the discontinuity of the boundary data at the origin. To get some idea of the nature and severity of this difficulty, we set $a = 1$ and $b = 1$ and solve this mild problem with four equal cells as shown in Fig. 2. For the first trial we impose only one moment collocation condition per segment. (There are 12 segments, consisting of eight boundary faces and four interfaces.) For each cell, in order to avoid degeneracy (loss of full row rank in U_k), we adjust the number of terms in the intracell expansion, i.e., the degree of the polynomial. (Legendre polynomials were used for basis functions and weights in this study, but ordinary powers and trigonometric basis functions may equivalently be used, as outlined in [5], with only very slight variations in the results.) In this first case, there are six θ's in each cell (quadratic).

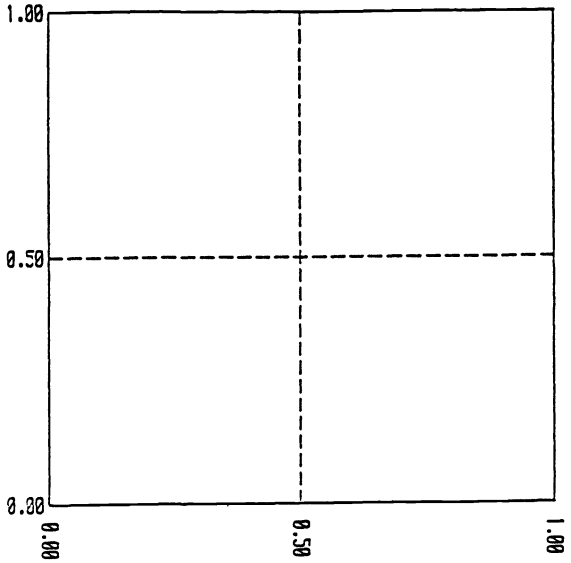


FIG. 2. *Problem* 1. *Initial cell layout.*

Since we solve the σ-system of equations (6.31) for the *interface* variables only, we have a 4×4 system of linear equations. The resulting solution function is plotted as a surface in Fig. 3, where the effect of the moment collocation can be clearly seen. (Unfortunately, there is an unavoidable distortion introduced by the plotting routine, which makes it appear that the mean discrepancy at the interfaces is not zero. In fact, it is.)

For the next trial, we skip to 3 moments per segment. There are 15 θ's in each cell (quartic) and the results are shown in Fig. 4. Next, we skip to 5 moments (Fig. 5), etc. These runs are summarized in Table 1. L is the number of matched moments on each segment, M is the number of terms in the lowest-order complete polynomial of the degree indicated sufficient to remove degeneracy (linear dependence of the rows of U), "Iter." is the number of iterations of the preconditioned CGNR to give a residual of norm less than 10^{-10} of the initial residual, and the solution is shown in the figure indicated.

It is clear that tightening the boundary and interface constraints is futile; the Gibbs disturbance due to the jump in the boundary condition merely propagates and pollutes the solutions in the other cells. What is really required is a representation in the first cell, which can represent the asymptotic solution in the neighborhood of the

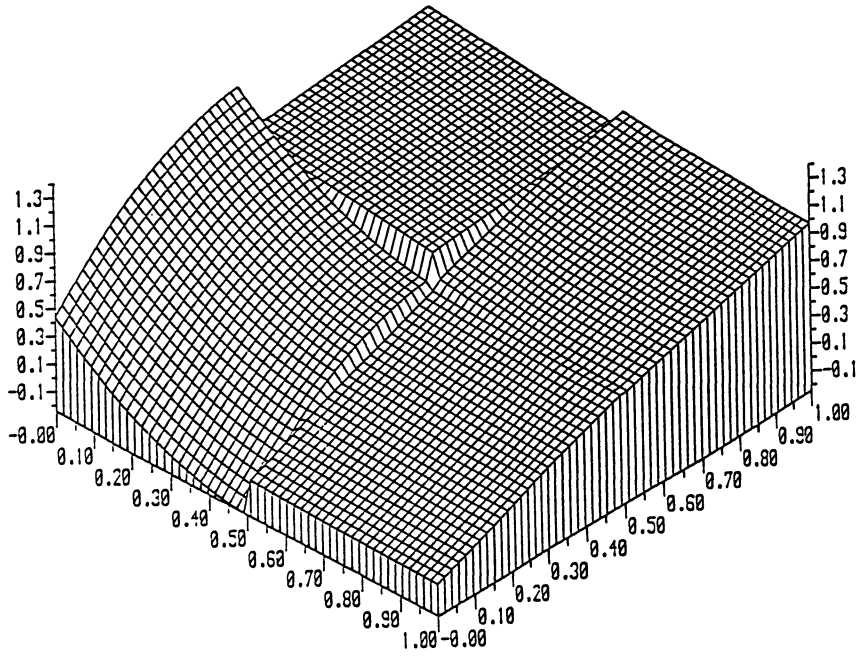


FIG. 3. *Problem* 1. *Solution with* $a = 1.$, $L = 1$, *degree* $= 2$.

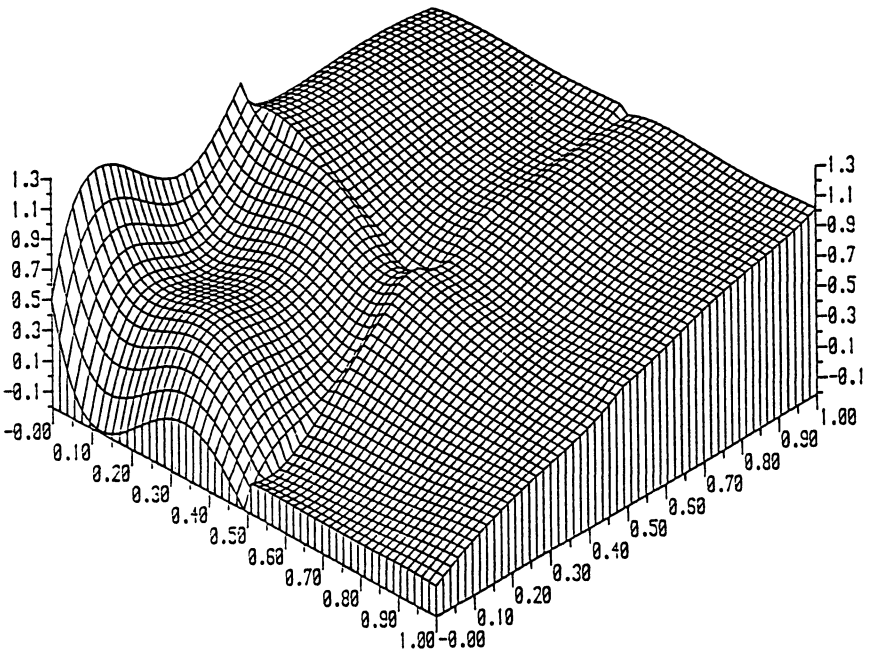


FIG 4. *Problem* 1. *Solution with* $a = 1.$, $L = 3$, *degree* $= 4$.

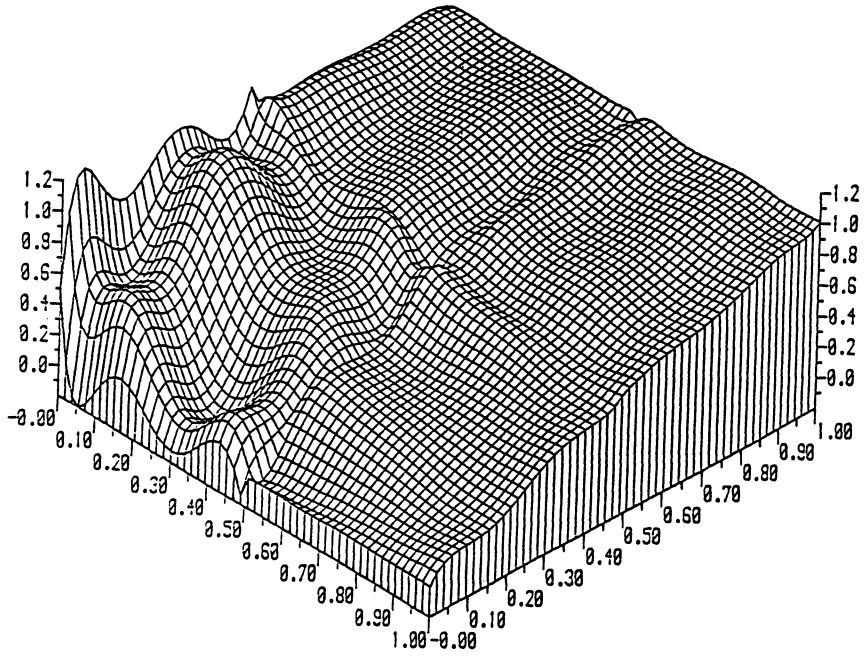


FIG. 5. *Problem* 1. *Solution with* $a = 1.$, $L = 5$, *degree* $= 6$.

corner better than polynomials possibly can. Although there is nothing in the cell method to prevent it from accommodating almost any basis and weight functions (in suitable spaces, of course, for approximating the solution), provision for such unusual representations is not yet available in the author's Fortran program. The appropriate program structure is already installed, but it will still be necessary to code numerical quadrature routines to evaluate the elements of S and U in each cell. This is the next step planned for this program.

In the absence, for the time being, of these capabilities, we must fall back on what amounts to local refinement, although in cell discretization, it amounts to providing the domain with "margins" so as to have a small square around the jump corner, so as to "isolate" the "bad spot" within it. Figure 6 shows the layout. There are 9 cells and 12 interfaces.

The result of a run with the same $L = 1$ is shown in Fig. 7. The number of degrees of freedom M is just sufficient to prevent degeneracy. Figure 8 shows the result with $L = 3$. Although the solution is not violently oscillatory, it is certainly not smooth. Experience has shown that it is often advantageous to have an intracell representation

TABLE 1

Problem 1 ($a = 1$, $b = 1$), 4 *cells. Results for different L values.*

L	M	Degree	Iter.	Figure
1	6	2	4	3
3	15	4	12	4
5	28	6	16	5
7	45	8	20	6

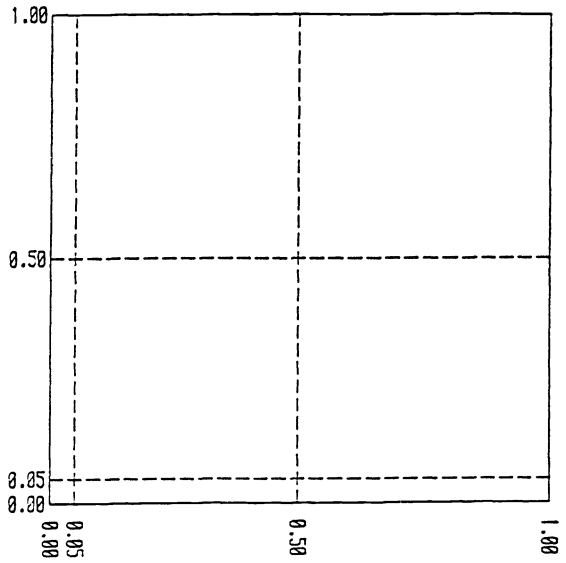


FIG. 6. *Problem* 1. *Cell layout with margins.*

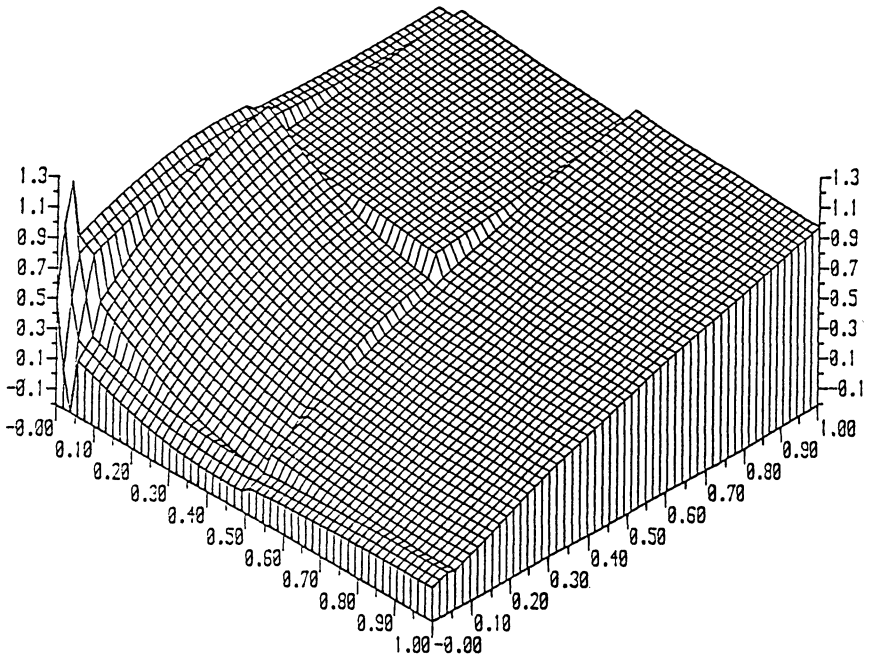


FIG. 7. *Problem* 1. *Solution with* $a = 1.$, $L = 1$, *degree* $= 2$, *margin* $= .05$.

with more degrees of freedom than the minimum necessary to avoid degeneracy. In Fig. 9 is shown the solution with $M = 21$ instead of 15. It is sufficiently improved to merit an additional stepup of M to 28 (sixth degree).

The result is shown in Fig. 10, and is quite good, so much so that L and M are frozen at 3 and 28, respectively, and the value of a is decreased to .1. The result is shown in Fig. 11. Although there is a slight waviness, the solution is still satisfactory.

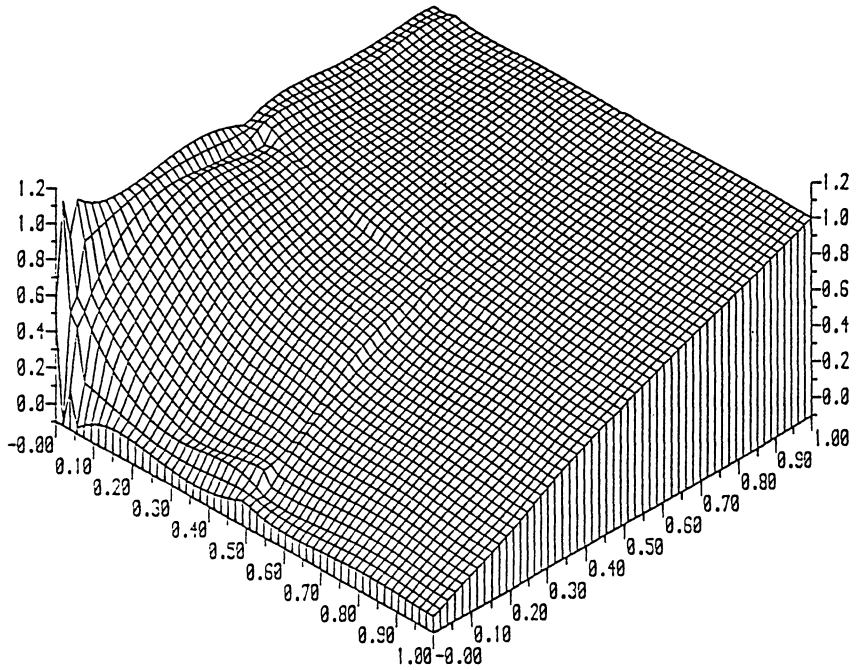


FIG. 8. *Problem* 1. *Solution with* $a = 1.$, $L = 3$, *degree* $= 4$, *margin* $= .05$.

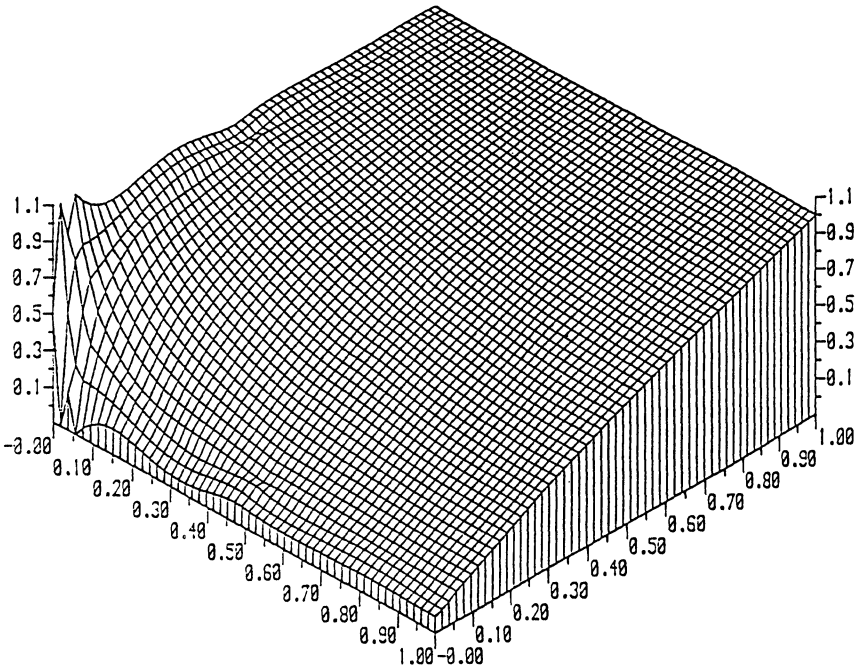


FIG. 9. *Problem* 1. *Solution with* $a = 1.$, $L = 3$, *degree* $= 5$, *margin* $= .05$.

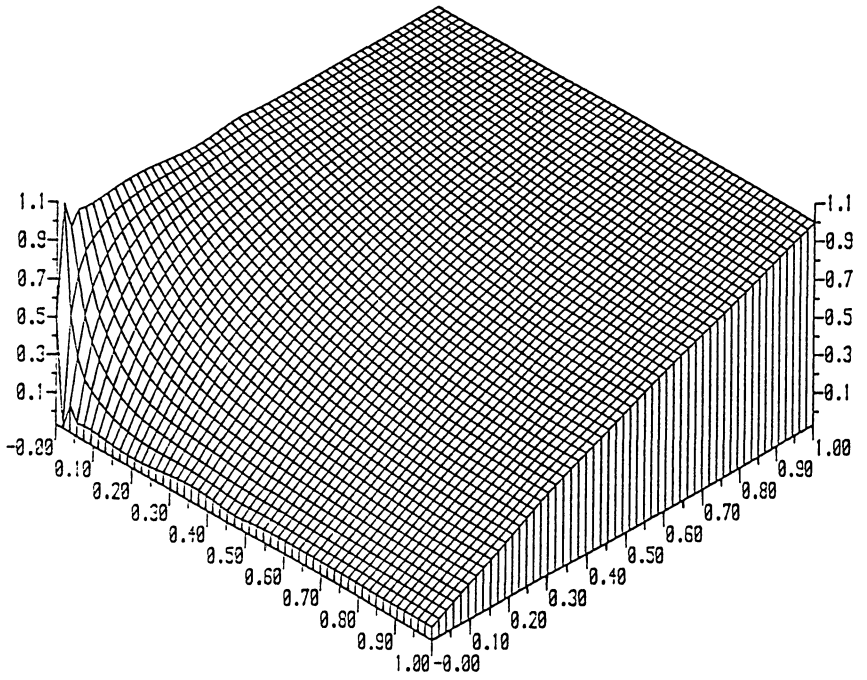


FIG. 10. *Problem* 1. *Solution with* $a = 1.$, $L = 3$, *degree* $= 6$, *margin* $= .05$.

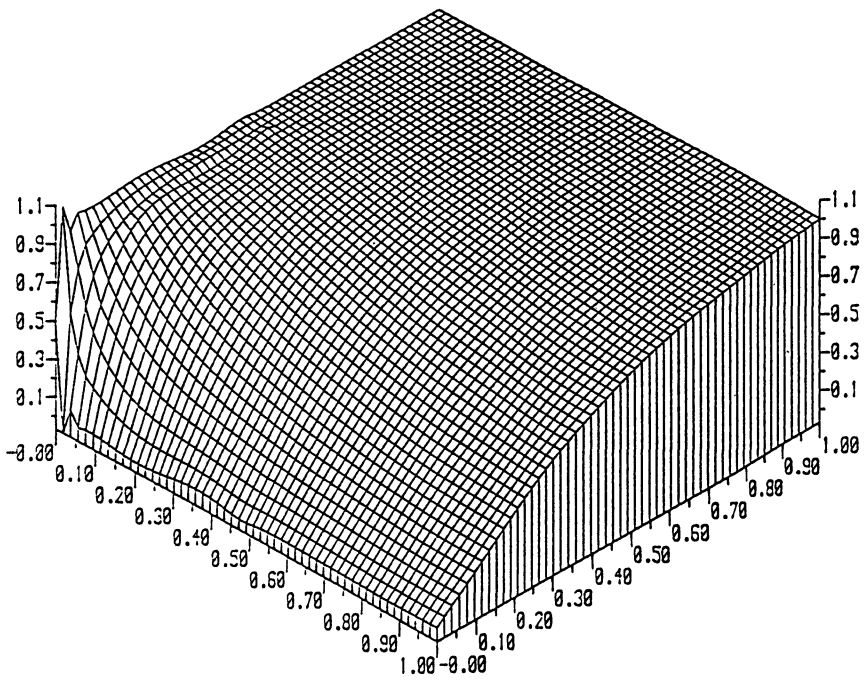


FIG. 11. *Problem* 1. *Solution with* $a = .1$, $L = 3$, *degree* $= 6$, *margin* $= .05$.

The value of a is next reduced to .01 and the result is shown in Fig. 12. Here we can see the beginning of a breakdown in the solution, including ripples and a more pronounced discontinuity along the margin interface. Experimentation shows that by increasing the number of moments collocated *along the vertical interfaces only* (i.e., for constant values of x), the solution can be improved. A further improvement results from widening the margins to .1—the reason for this is that for a problem of this type, the boundary layer near the origin has a thickness of approximately $\sqrt{a}$ [1] (we shall see in Problem 2 that this is not always the case). For sufficiently small a, the solution becomes more sensitive to the margin thickness. Finally, the value of M is increased to 45 (eighth degree). Figure 13 shows the result of all these measures to improve the solution.

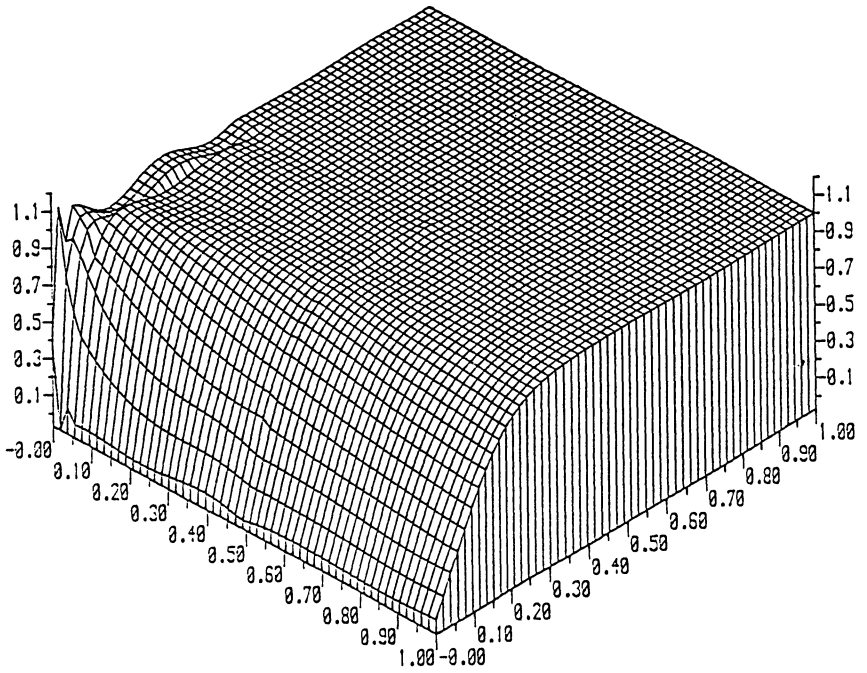


FIG. 12. *Problem* 1. *Solution with* $a = .01$, $L = 3$, *degree* $= 6$, *margin* $= .05$.

A word of explanation: the newness of this method requires a kind of exploration and experimentation to ascertain its powers and limits. These were the first convection-diffusion problems ever attempted by the author with the extended cell method, and he had no idea at the outset how it would cope. In mitigation, note that the cell discretization procedure is applied in its original, unmodified form (with no "adjustments," such as would correspond to upstream differencing, etc.). The only adjustments used here are the margin thicknesses and the discrete values of L and M, the latter being measures of the accuracy of the approximation. It seems evident that adding a proper asymptotic corner solution to the approximation would remove much of the sensitivity.

For the next problem, with $a = .001$, the margins were reset to .03 and the number of "vertical moments" (VL) was set to 7. This produced a satisfactory result, as shown

[1] The author is obliged to Dr. A. Sneyd for pointing this out.

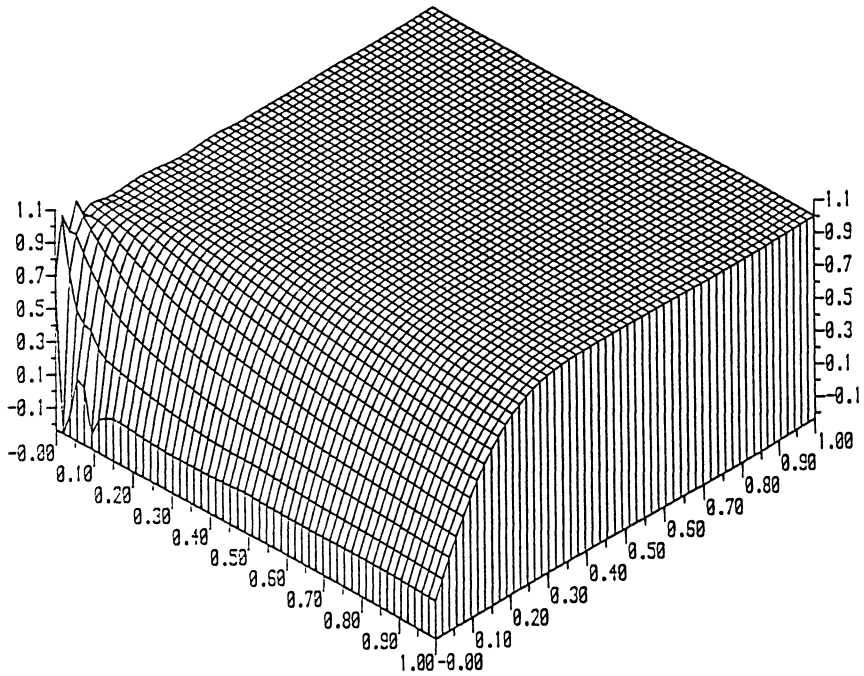


FIG. 13. *Problem* 1. *Solution with* $a = .01$, $L = 3, 5$, *degree* $= 8$, *margin* $= .1$.

in Fig. 14. (There is a slight waviness which could be improved by more moments, etc., but it is really not worth it.)

For $a = .0001$, the margins were reset to .01, VL was set to 11, and M was set to 105 (14th degree); the result is shown in Fig. 15. The boundary layer is not really a sharp edge, as can be shown by plotting only the very narrow strip containing it (Fig. 16). A special view of the bad corner is shown for this case in Fig. 17, covering only 1/10,000 the area of the original plot, to highlight the irregular behavior. Note that the function shown is identical with the original shown in Fig. 16, but is displayed over a different *plotting* grid; the discretization remains exactly the same.

Table 2 summarizes the results of these runs. The entry marked "σ's" shows the total number of variables to be solved for by the CGNR iteration. As can be seen, the linear systems to be solved are quite modest in size, and could readily be solved by direct methods. It is well to bear in mind that the size of the σ-system to be solved depends only on the numbers of interface moments and not on the total number of degrees of freedom (θ's) in the cells. As emphasized earlier, the rest of the variables (ρ's) are solved cell-by-cell.

This is a good place to do a spot comparison of the CGNR, CGNE, and ORTHO algorithms, with and without preconditioning, on some of the problems treated. The preconditioner, as noted in § 8, is the block-diagonal matrix of the Λ's. This either pre- or postmultiplies the rest of the overall H-matrix. The case when $a = .01$, margin $= .1$, $b = 1$, $L = 3, 5$, etc. was solved by various combinations of these procedures, and the numbers of iterations collected in Table 3. (The entry V.S.C. means very slow convergence—practically nonconvergence.)

Although CGNR with postmultiplication has been used for all the runs up to this point, it would appear that CGNE and ORTHO can be faster. However, extensive

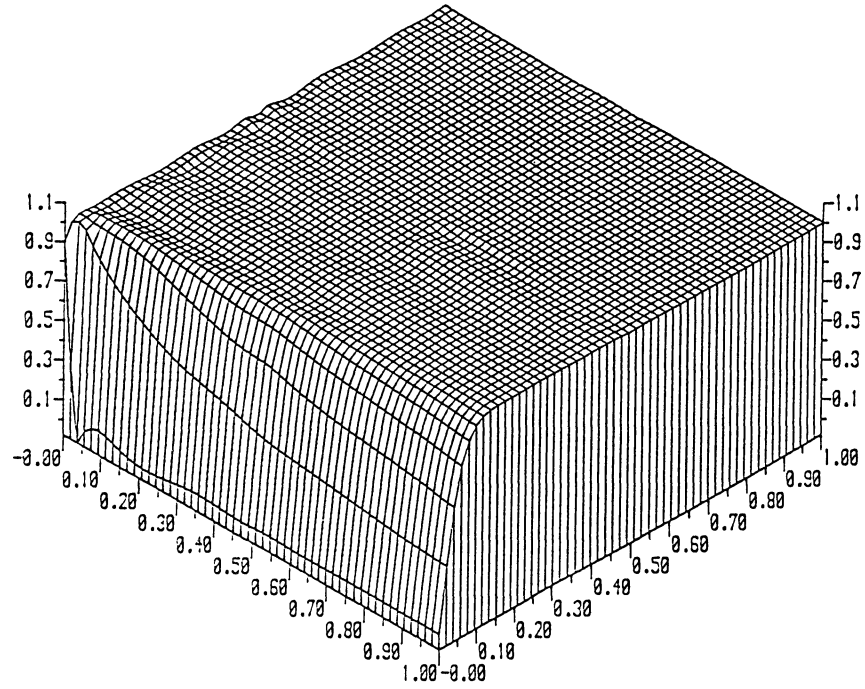


FIG. 14. *Problem* 1. *Solution with* $a = .001$, $L = 3, 7$, *degree* $= 9$, *margin* $= .03$.

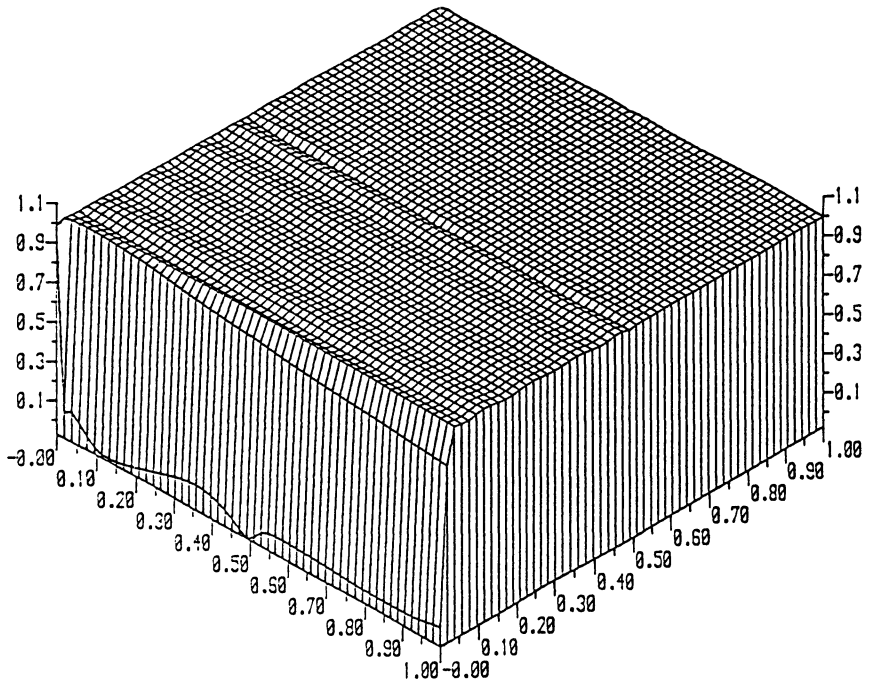


FIG. 15. *Problem* 1. *Solution with* $a = .0001$, $L = 3, 11$, *degree* $= 13$, *margin* $= .01$.

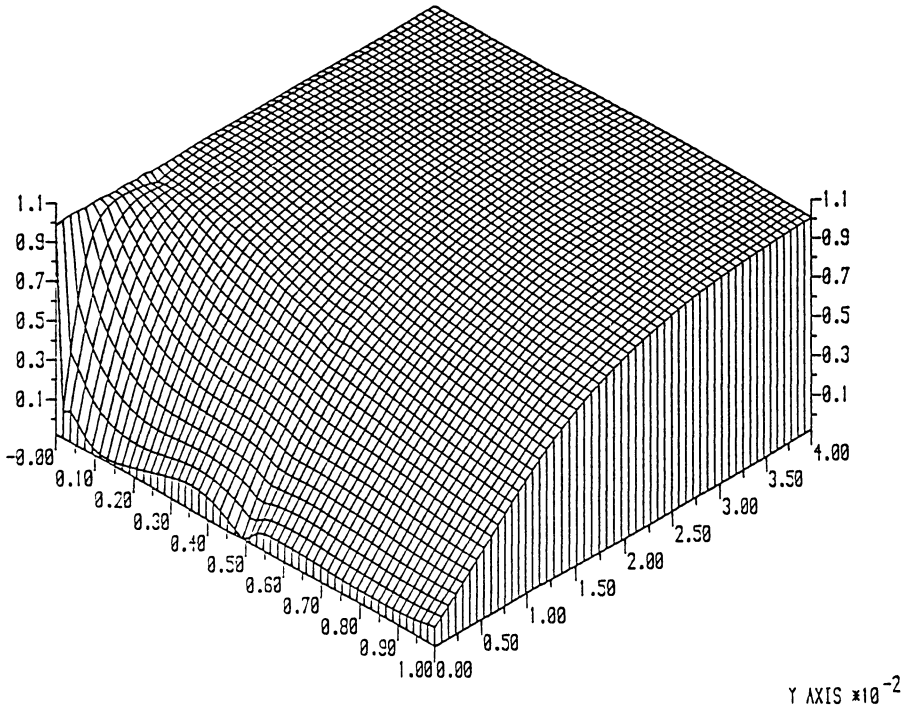


FIG. 16. *Problem* 1. *Enlargement of edge with* $a = .001$, $L = 3, 11$, *degree* $= 13$, *margin* $= .01$.

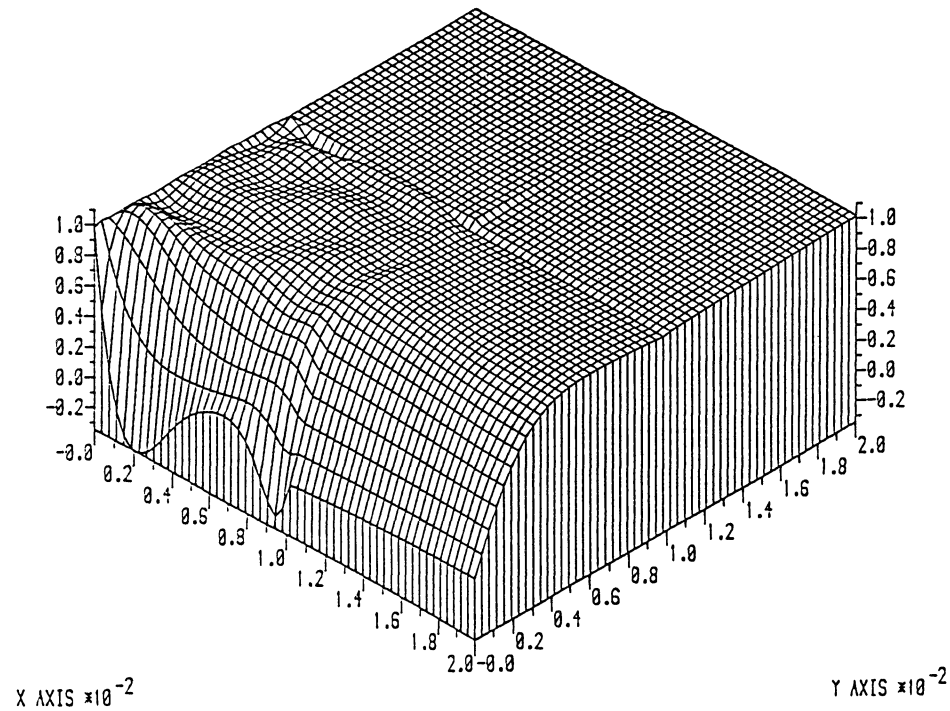


FIG. 17. *Problem* 1. *Closeup of corner with* $a = .001$, $L = 3, 11$, *degree* $= 13$, *margin* $= .01$.

TABLE 2

Problem 1—*margins. Results for various parameter values.*

a	Margin	L	M	Degree	σ's	Iter.	Figure
1.	.05	1	6	2	12	12	7
1.	.05	3	15	4	36	64	8
1.	.05	3	21	5	36	43	9
1.	.05	3	28	6	36	36	10
.1	.05	3	28	6	36	36	11
.01	.05	3	28	6	36	46	12
.01	.1	3, 5	45	8	48	55	13
.001	.03	3, 7	55	9	60	195	14
.0001	.01	3, 11	105	13	84	2100	15, 16, 17

TABLE 3

Problem 1: $a = .01$, *etc. Results for different solution methods.*

	CGNR	CGNE	ORTHO
No multiplication	127	127	277
Premultiplication	51	50	30
Postmultiplication	55	V.S.C.	31

investigation has shown that CGNR with postmultiplication is the most *reliable* of the methods, and this is illustrated by the same study on the case when $a = .001$, $L = 7$, etc., which gives the results in Table 4 (N.C. means no convergence).

Turning now to the second problem, it is summarized in Fig. 18. The lower left section is deleted from the rectangle, and the zero boundary values are imposed as

TABLE 4

Problem 1: $a = .001$, *etc. Results for different solution methods.*

	CGNR	CGNE	ORTHO
No multiplication	655	533	V.S.C.
Premultiplication	N.C.	N.C.	N.C.
Postmultiplication	195	N.C.	N.C.

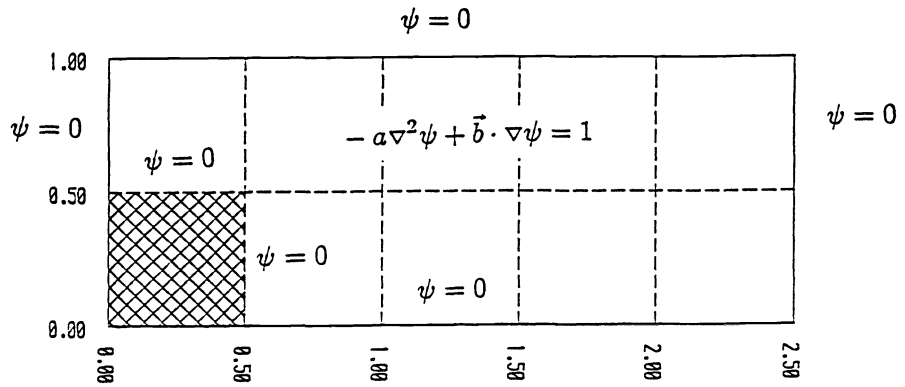


FIG. 18. *Problem* 2. *Definition of problem and initial layout of cells.*

shown. A reasonable-looking cell structure has been selected that makes all the cells square. There are thus 9 cells and 13 interfaces.

$\vec{b}$ is set to the value $(\sqrt{2}/2, \sqrt{2}/2)$, and $a = 1$, initially. The solution is shown in Fig. 19 for $L = 1$. Figure 20 shows the result for $L = 3$. With the degree of each intracell polynomial just high enough to prevent degeneracy, there is a very slight bump in the solution.

The value of a was lowered to .1 and the degree of the intracell polynomials was stepped up by unity. This was done both to speedup convergence of the iterative

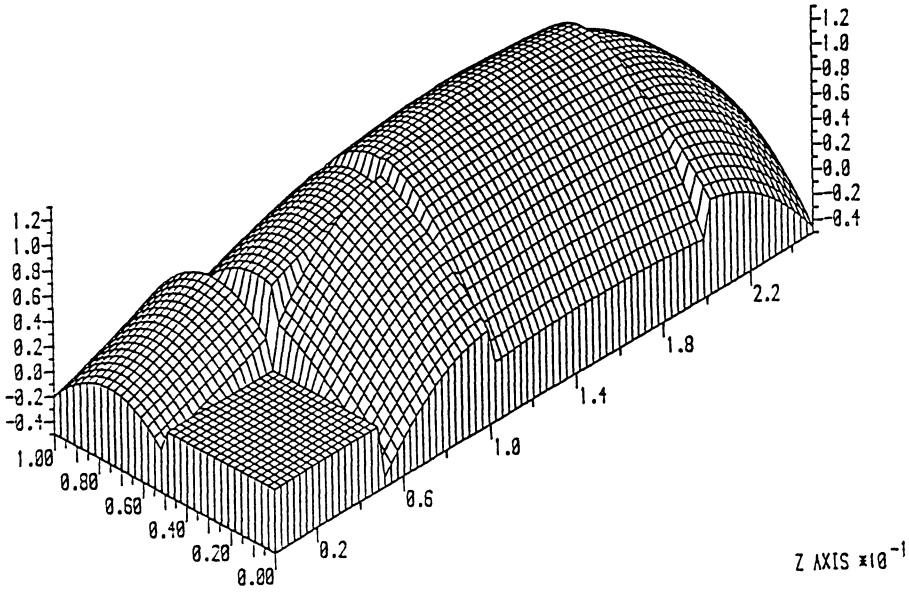


FIG. 19. *Problem 2. Solution with* $a = 1.$, $L = 1$, *degree* $= 2$.

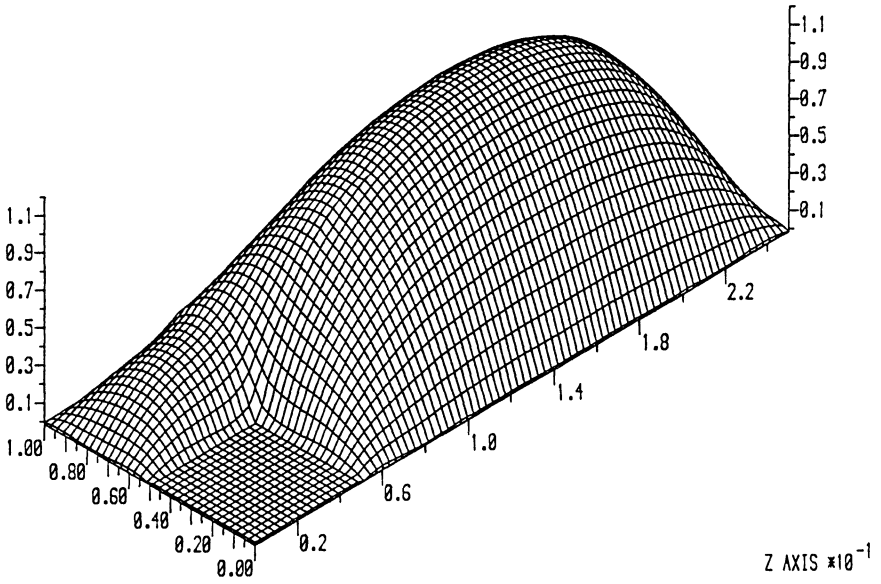


FIG. 20. *Problem 2. Solution with* $a = 1.$, $L = 3$, *degree* $= 4$.

solution of the discrete equations and to improve the solution slightly. The result is shown in Fig. 21, with a "back view" of the solution in Fig. 22.

Lowering the value of a to .01 gave the result shown in Fig. 23, which is quite unsatisfactory. The problem is the boundary layers on the north and east faces. These are known to be exponentials, and polynomials are not very good at approximating exponentials over a wide range of the argument. Therefore, in the absence of the capability of incorporating a genuine exponential into the solution representation, we

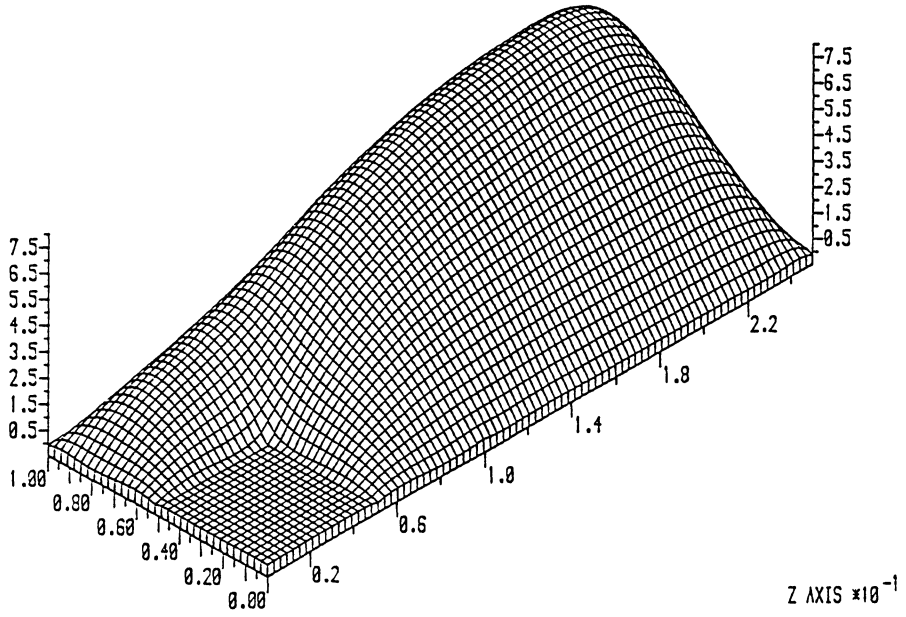


FIG. 21. *Problem 2. Solution with* $a = .1$, $L = 3$, *degree* $= 5$.

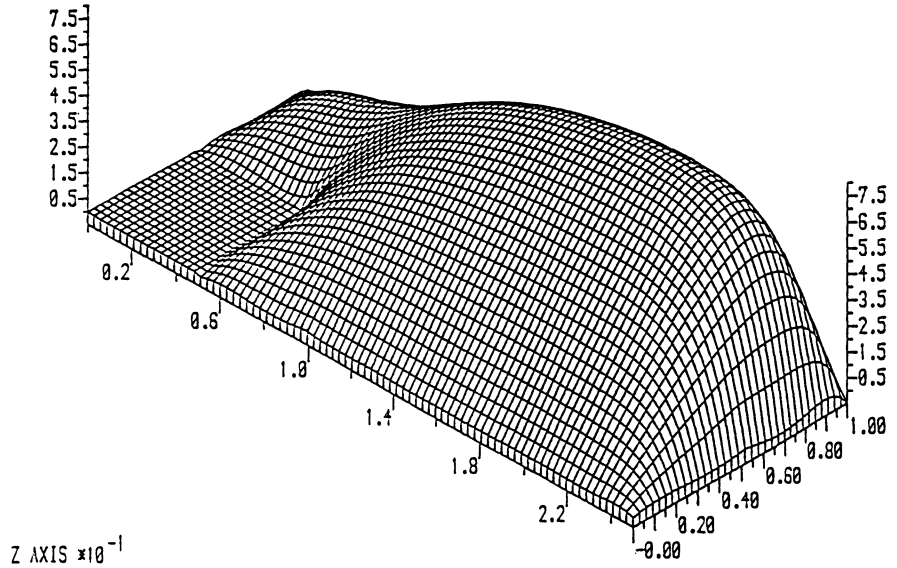


FIG. 22. *Problem 2. Back view of solution with* $a = .1$, $L = 3$, *degree* $= 5$.

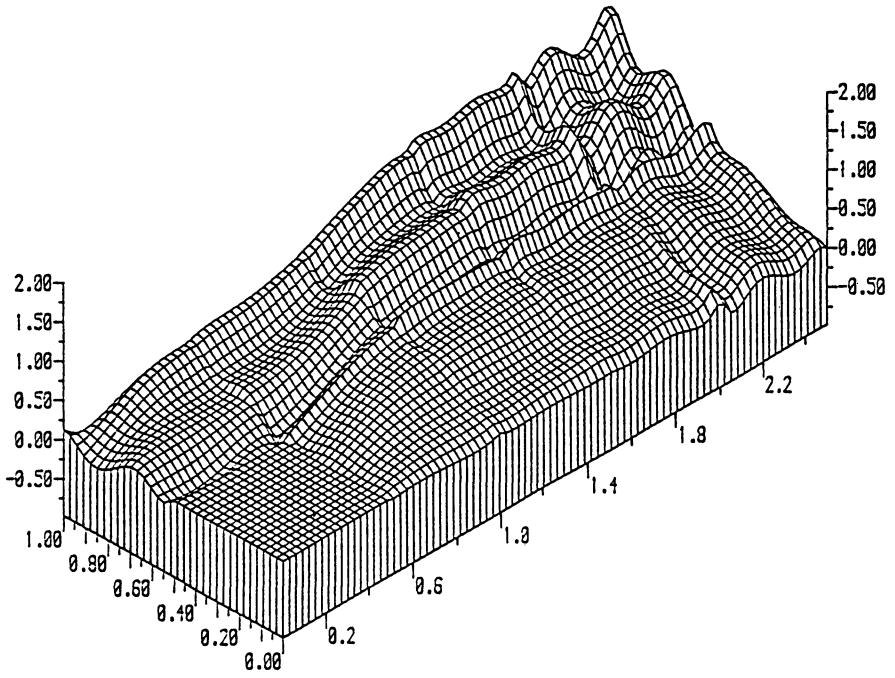


FIG. 23. *Problem 2. Solution with* $a = .01$, $L = 3$, *degree* $= 5$.

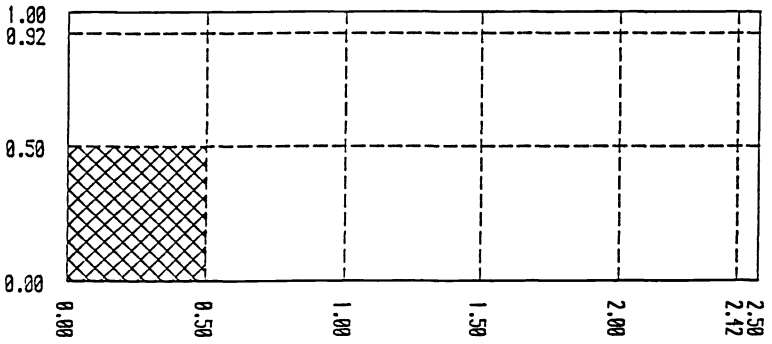


FIG. 24. *Problem 2. Cell layout with margins.*

must again make use of a margin (Fig. 24). To cover the range of exponential values of about $[10^{-3}, 1]$, say, it turns out that we need a margin of about $10 \times a$, although the solution is not very sensitive to the margin thickness from about $6 \times a$ to $20 \times a$.

With this adjustment, the result for $a = .01$ is shown in Fig. 25, with a back view in Fig. 26. The use of the margins has completely eliminated violent ripples and provides a good solution. For $a = .001$, the margin was adjusted appropriately and the degree of the intracell polynomials stepped up by two. This seems to make little difference in the solution, but speeds up the convergence of the conjugate gradient iterations very substantially. The solution for this value of a is shown in Fig. 27. For $a = .0001$, the margin was again adjusted, but the degree was kept fixed. The result is shown in Fig. 28, with the back view shown in Fig. 29. With the plotting grid used in these two figures, the boundary layers look like sharp edges, but by plotting a very small portion

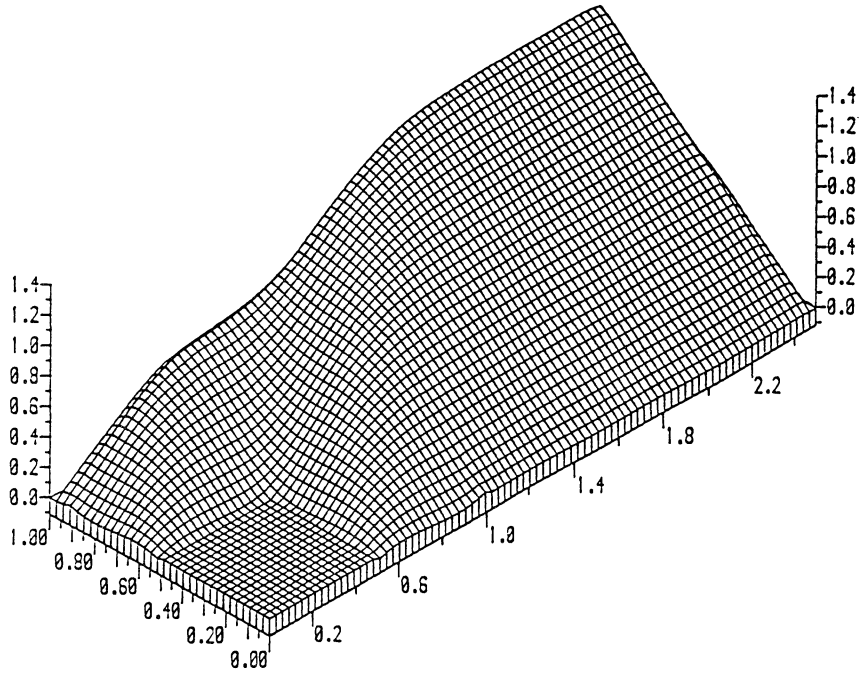


FIG. 25. *Problem* 2. *Solution with* $a = .01$, $L = 3$, *degree* $= 5$, *margin* $= .08$.

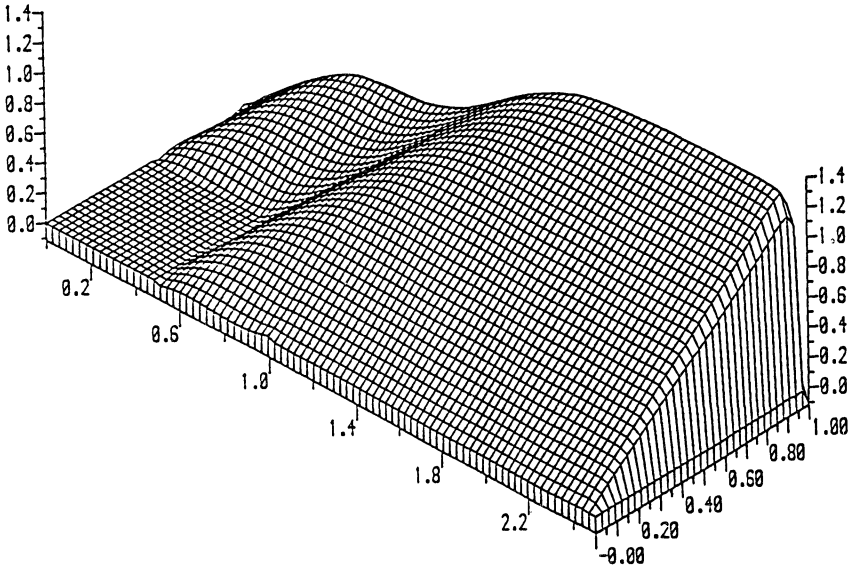


FIG. 26. *Problem* 2. *Back view of solution with* $a = .01$, $L = 3$, *degree* $= 5$, *margin* $= .08$.

of the northeast corner, they can be seen in Fig. 30 to be smooth functions, although of course quite steep.

These results, given the limitation of polynomial representations only, seem to be quite good. In Table 5 are summarized all of the results. The maximum value of the solution *over the plotting mesh* is also given. Since the latter is rather fine, the true

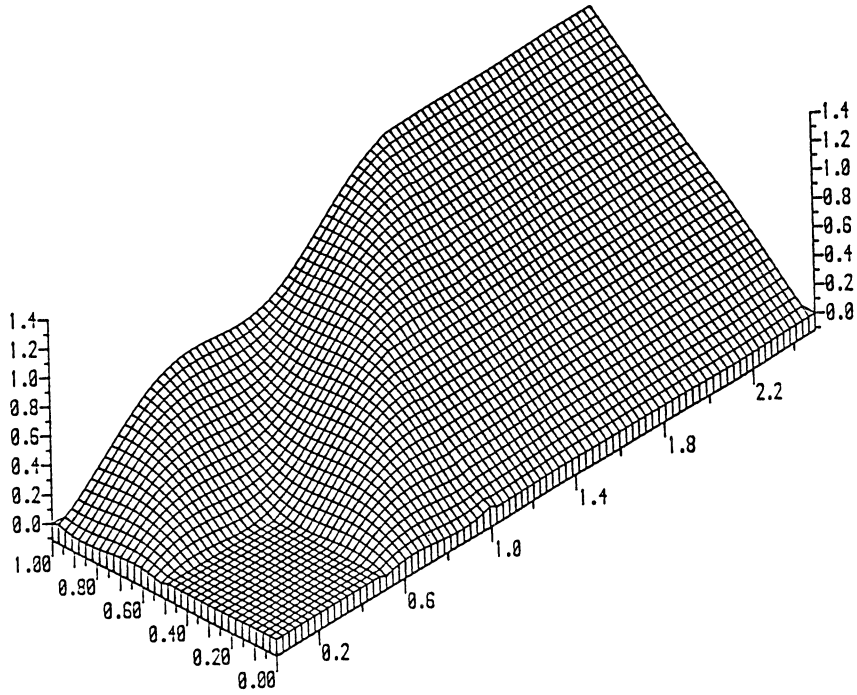


FIG. 27. *Problem* 2. *Solution with* $a = .001$, $L = 3$, *degree* $= 7$, *margin* $= .008$.

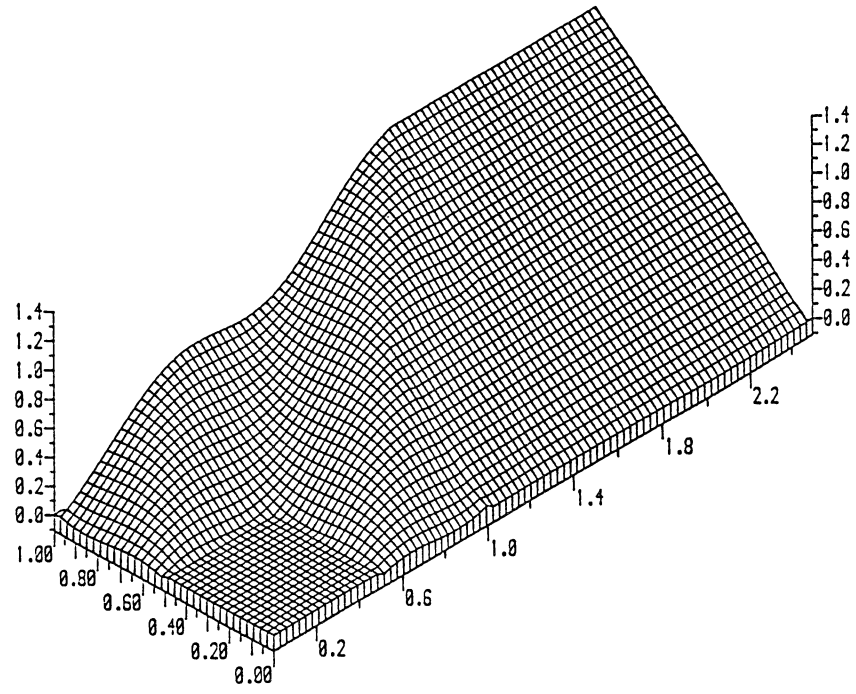


FIG. 28. *Problem* 2. *Solution with* $a = .0001$, $L = 3$, *degree* $= 7$, *margin* $= .0008$.

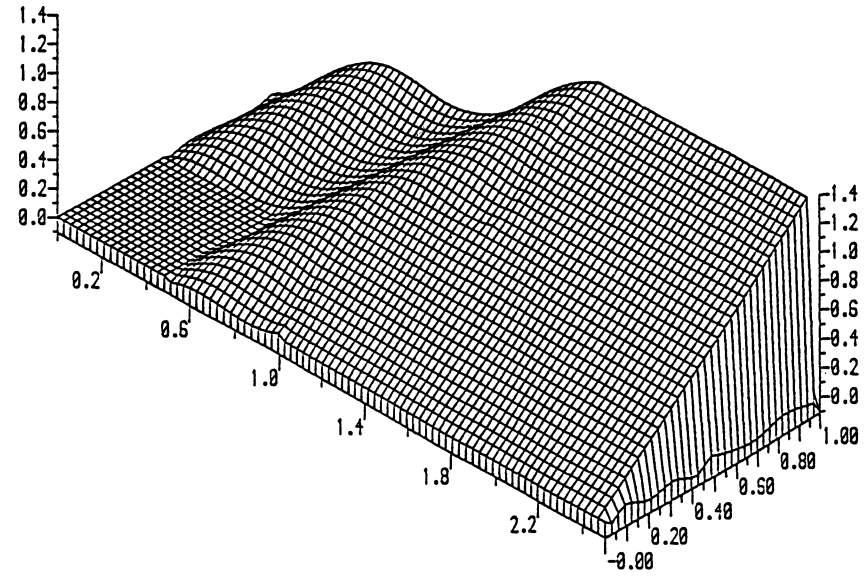


FIG. 29. *Problem 2. Back view of the solution with* $a = .0001$, $L = 3$, *degree* $= 7$, *margin* $= .0008$.

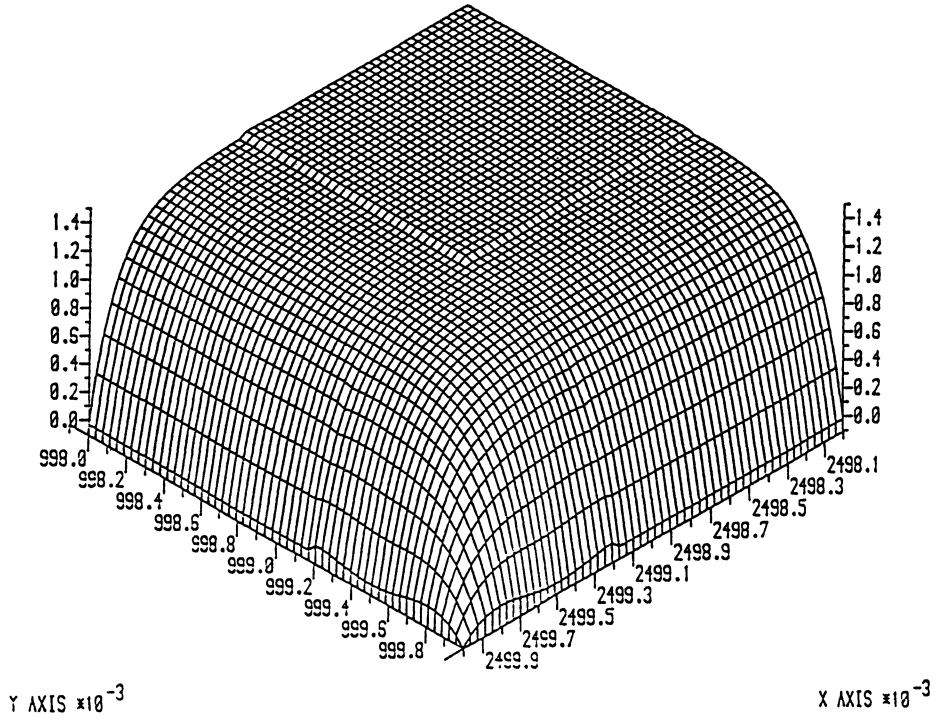


FIG. 30. *Problem 2. Closeup of the northeast corner of the solution with* $a = .0001$, $L = 3$, *degree* $= 7$, *margin* $= .0008$.

TABLE 5

Problem 2—all runs.

a	Margin	L	M	Degree	σ's	Iter.	Max. Value	Figure
1.	None	1	6	2	11	11	.1205	19
1.	None	3	15	4	33	44	.1140	20
.1	None	3	21	5	33	39	.7800	21, 22
.01	None	3	21	5	33	60	1.779	23
.01	.08	3	21	5	33	223	1.315	25, 26
.001	.008	3	36	7	33	319	1.376	27
.0001	.0008	3	36	7	33	1781	1.380	28, 29, 30

TABLE 6

Problem 2: $a = .01$, etc. Results for different solution methods.

	CGNR	CGNE	ORTHO
No multiplication	493	444	1145
Premultiplication	N.C.	N.C.	N.C.
Postmultiplication	223	N.C.	216

maximum cannot be very much larger. This value is given so that those who know the true solution of this problem may gauge the accuracy of the cell solution.

Finally, we present in Table 6 another spot survey of the performance of the several conjugate direction methods with and without preconditioning. Again CGNR with postmultiplication shows up as the most reliable of the methods.

10. Conclusions. The cell discretization algorithm has been applied to a standard class of nonselfadjoint partial differential equations in an obvious, direct way without any important alterations. Although it is not strictly necessary, it is a help in constructing the algorithm, to start with a primal-dual variational formulation. Apart from the interface collocation, the result is only the Petrov-Galerkin algorithm over the cells (these interface constraints not existing in the P-G method).

The range space-null space (V-Z) transformation carries over nicely in the new formulation, as does the conjugacy relation connecting V, Z, and S which serves to decouple the intracell and interface variables. Thus, all the coefficients and source terms in the nonselfadjoint case can still be computed cell-by-cell, thus rendering the computation almost entirely parallelizable in a most natural way. (By suitably relating simple finite-difference schemes—those in widest use—to the CD formalism, it is also possible to construct natural finite-difference domain decompositions using the methods described here.)

In applying the nonselfadjoint procedure to the simple (but not easy) convection-diffusion problems described above, it is gratifying to observe that, with the single exception of the use of margins, the unadulterated method gives very good results, almost entirely free of spurious oscillations and other disturbances. It is almost certain that, with the use of special representations (using nonmeromorphic functions, etc.), even the need for margins will disappear, with even better solution results. (*Note added in proof.* Very recent calculations with an expanded program have borne out this supposition exactly.)

It is evident that the cell method converges quite nicely with increasing numbers of moment collocations. In fact, for the selfadjoint case, the method is known to converge, as has been proved by Swann [10]. There is no reason to suppose that the nonselfadjoint algorithm will present insurmountable new difficulties for convergence proofs, since the procedures are very similar in both the selfadjoint and nonselfadjoint cases. Error estimates are another matter, because of the "Fourier-like" moment collocation relations and the more general basis functions, as compared with polynomial-based representations in the finite-element and finite-difference methods. Nevertheless, Swann has recently derived estimates for the symmetric case.

So far, nothing has been done on time-dependent problems, with the exception of the work of Coffey [1] on parabolic equations, but the CD procedure generalizes quite easily for time-dependent problems when moment collocation on "timelike" interfaces is included. Otherwise, the procedure outlined here would, of course, apply to parabolic equations which are nonselfadjoint (in the time).

In the future, once the computer program is equipped with the capability of handling variable coefficients and sources, as well as of accepting specially programmed nonpolynomial functions, it will be interesting to see how shock capturing and tracking behave, along the lines of the approach taken by the author in [7].

Acknowledgment. I am very deeply indebted to Professor M. J. D. Powell for giving me the opportunity of pursuing this research in his group in the Department of Applied Mathematics and Theoretical Physics at Cambridge University.

REFERENCES

[1] M. W. COFFEY, *Some aspects of treating parabolic differential equations with the cell discretization algorithm*, Report G320-3466, IBM Palo Alto Science Center, Palo Alto, CA, 1984.

[2] P. CONCUS, G. H. GOLUB, AND D. P. O'LEARY, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976.

[3] H. C. ELMAN, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, Research Report #229, Department of Computer Science, Yale University, New Haven, CT, 1982.

[4] M. GELFAND AND S. V. FOMIN, *Calculus of Variations*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

[5] J. GREENSTADT, *The cell discretization algorithm for elliptic partial differential equations*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 261–288.

[6] ———, *The application of cell discretization to nuclear reactor problems*, Nuclear Sci. Engrg., 82 (1982), pp. 78–95.

[7] ———, *The use of nonrational functions to represent steep front solutions to partial differential equations*, J. Comput. Phys., 53 (1984), pp. 299–318.

[8] ———, *On the reconciliation of clashing boundary conditions in cell discretization*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1282–1306.

[9] J. T. ODEN AND J. N. REDDY, *An Introduction to the Mathematical Theory of Finite Elements*, Wiley-Interscience, New York, 1976.

[10] H. SWANN, *On the use of non-conforming methods and Lagrange multipliers to approximate solutions of elliptic equations*, SIAM J. Sci. Statist. Comput., submitted.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1109-1130, September 1991
© 1991 Society for Industrial and Applied Mathematics
006

AN IMPROVED NEWTON ITERATION FOR THE GENERALIZED INVERSE OF A MATRIX, WITH APPLICATIONS*

VICTOR PAN† AND ROBERT SCHREIBER‡

Abstract. Pan and Reif have shown that Newton iteration may be used to compute the inverse of an $n \times n$, well-conditioned matrix in parallel time $O(\log^2 n)$ and that this computation is processor efficient. Since the algorithm essentially amounts to a sequence of matrix-matrix multiplications, it can be implemented with great efficiency on systolic arrays and parallel computers.

Newton's method is expensive in terms of the arithmetic operation count. In this paper the cost of Newton's method is reduced with several new acceleration procedures. A speedup by a factor of two is obtained for arbitrary input matrices; for symmetric positive definite matrices, the factor is four. It is also shown that the accelerated procedure is a form of Tchebychev acceleration, whereas Newton's method uses a Neumann series approximation.

In addition, Newton-like procedures are developed for a number of important related problems. It is also shown how to compute the nearest matrices of lower rank to a given matrix A, the generalized inverses of these nearby matrices, their ranks (as a function of their distances from A), and projections onto subspaces spanned by singular vectors; such computations are important in signal processing applications. Furthermore, it is demonstrated that the numerical instability of Newton's method when applied to a singular matrix is absent from these improved methods. Finally, the use of these tools to devise new polylog time parallel algorithms for the singular value decomposition is explored.

Key words. matrix computation, Moore-Penrose generalized inverse, singular value decomposition, Newton iteration, parallel computing

AMS(MOS) subject classifications. 65F10, 6520, 15A09, 65W05

1. Introduction.

Pan and Reif [11], [12] have shown that Newton iteration may be used to compute the inverse of an $n \times n$, well-conditioned matrix in parallel time proportional to $\log^2 n$ using a number of processors that is within a factor of $\log n$ of the optimum. Newton iteration is simple to describe and to analyze, and is strongly numerically stable for nonsingular input matrices; this is not true of earlier polylog time matrix inversion methods. Moreover, since it is rich in matrix-matrix multiplications, it can be implemented with great efficiency on systolic arrays and parallel computers.[1]

Unfortunately, the computation can be rather expensive. It has been shown that the number of matrix multiplications required can be as high as $4 \log \kappa(A)$, where $\kappa(A) \equiv \|A\|_2 \|A^{-1}\|_2$.

* Received by the editors December 26, 1989; accepted for publication (in revised form) September 5, 1990.

† Computer Science Department, State University of New York, Albany, New York 12222, and Mathematics Department, Lehman College, City University of New York, Bronx, New York 10468. The research of this author was supported by National Science Foundation grant CCR-8805782 and PSC-CUNY award 668541.

‡ Research Institute for Advanced Computer Science, Moffett Field, California 94035. The research of this author was supported by Office of Naval Research contract N00014-86-K-0610, Army Research Office grant DAAL03-86-K-0112 to the Rensselaer Polytechnic Institute, and by the Numerical Aerodynamic Simulation Systems Division via Cooperative Agreement NCC2-387 between the National Aeronautics and Space Administration and the Universities Space Research Association.

[1] (On the Connection Machine [7] matrix products may be computed at 5×10^9 operations per second, but matrix computations done in standard languages rarely can exceed 10^8 operations per second. Furthermore, computer manufacturers are currently being encouraged to provide the fastest matrix product software possible, since other matrix computations (QR and LU decomposition, in particular) may be computed with algorithms that are rich in matrix multiply [4].)

Here we show how to reduce this cost, in some cases quite dramatically. We accelerate and extend the usual Newton iteration in several ways. In particular, we

- Substantially accelerate the convergence of the iteration;
- Ensure its stability even when A is singular;
- Show how to compute the matrix $A(\varepsilon)$ obtained from A by setting to zero any of its singular values that are less than ε. Thus $A(\varepsilon)$ is a closest lower rank approximation to A (see Theorem 2.1 below);
- Compute $A^{+}(\varepsilon)$, the Moore-Penrose generalized inverse (also called the pseudoinverse) of $A(\varepsilon)$;
- Compute the rank of $A(\varepsilon)$;
- Compute the matrix $P(\varepsilon)$ that projects orthogonally onto the range of $A(\varepsilon)$.

Concerning acceleration, we obtain a twofold speedup by scaling the iterates at each step, and we prove that the scaled iterates are defined by Tchebychev polynomials that are, in the usual minimax sense, optimal in a subspace of polynomials in $A^{T}A$. In the case of symmetric positive-definite matrices, we get another speedup by a factor of two through a new means of constructing the initial iterate. We also consider adaptive procedures for which we prove no such a priori lower bound on speedup, but which promise to be useful in practice. Our results have further applications, in particular, to signal processing and to computing the SVD of a matrix.

Concerning efficiency in highly parallel computing environments, we do not claim that the present methods are always advantageous. The alternative, for most computations discussed here, is a parallel computation of the full SVD of A. The computation of the SVD in a highly parallel environment is best done using a square array of $n/2 \times n/2$ processors, implementing a Jacobi-like method due to Kogbetliantz [3]. (This method is not competitive with standard methods in terms of operation count, but the standard methods are not well suited to highly parallel architectures.) Good experimental evidence is available to show that from six to ten sweeps of the Kogbetliantz method are needed. For real A, each sweep requires $8n^3$ multiplications. The sequential operation count of this method is therefore the same as that of 32–52 Newton iterations. Thus, Newton's method is competitive with a Kogbetliantz SVD for these problems on highly parallel computers. It is a clear winner if the condition number of $A(\varepsilon)$ is not too large, or if the adaptive acceleration we employ is especially successful, so that far fewer than 30 Newton steps are needed, or if we can exploit sparsity or other properties of A to reduce its cost. Furthermore, it is often the case on parallel machines that matrix products, the core of the Newton iteration, can be computed especially fast. (On the Connection Machine, microcoded matrix multiply runs an order of magnitude faster than code written in Fortran, C^{*}, or *LISP.)

Of course, other parallel methods for the SVD may arise. And when the number of processors is not large, so that $O(n^2)$ processors are not available, then more modestly parallel, but less costly methods for the SVD are better.

1.1. Contents. We organize the paper as follows. Section 2 is for definitions. In § 3, we recall the customary Newton iteration for matrix inversion. In § 4 we present a Tchebychev acceleration procedure and in § 5 an adaptive acceleration using cubic polynomials rather than the quadratics used in Newton's method. We give a method for finding an improved initial iterate for symmetric positive-definite A in § 6. In §§ 7–9 we give methods for computing the matrices $A(\varepsilon)$ and $A^{+}(\varepsilon)$ (see above), prove the stability of these computations, show how to find subspaces spanned by singular vectors, and show some further applications. Some numerical experiments are presented in § 10.

2. Notation. We denote most matrices by upper case italic letters, diagonal matrices by upper case Greek letters, and the elements of a matrix by the corresponding lower case Greek letter. We also use lower case Greek letters for various scalars, lower case boldface letters for vectors and in particular, columns of matrices. The letters *i, j, k, m, n, r,* and *s* are used for integers. We assume that all the quantities are real; extension to the complex case is straightforward.

The basis for our analysis is the *singular value decomposition* (hereafter referred to as the SVD) of $A \in R^{m\times n}$,

$$A = U\Sigma V^T, \tag{2.1}$$

where $U = [\mathbf{u}_1, \cdots, \mathbf{u}_m]$ and $V = [\mathbf{v}_1, \cdots, \mathbf{v}_n]$ are square orthogonal matrices and $\Sigma = \operatorname{diag}(\sigma_1 \geqq \sigma_2 \geqq \cdots \geqq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0)$. Here, $r = \operatorname{rank}(A)$. The generalized inverse of A is

$$A^+ = V\Sigma^+ U^T$$

where

$$\Sigma^+ \equiv \operatorname{diag}(\sigma_1^{-1}, \sigma_2^{-1}, \cdots, \sigma_r^{-1}, 0, \cdots, 0).$$

If A is symmetric and positive definite, then $m = n$ and $U = V$. In this case, the eigenvalues of A are σ_j and the eigenvectors are $\mathbf{v}_j$, $1 \leqq j \leqq n$.

We shall use the Euclidean vector norm $\|\mathbf{x}\|_2 \equiv (\mathbf{x}^T\mathbf{x})^{1/2}$ and the following matrix norms:

$$\|A\|_2 \equiv \max_{x\neq 0} \|A\mathbf{x}\|_2 / \|\mathbf{x}\|_2,$$

$$\|A\|_1 \equiv \max_j \sum_1 |\alpha_{ij}|,$$

$$\|A\|_\infty \equiv \max_i \sum_j |\alpha_{ij}|,$$

$$\|A\|_F \equiv \left(\sum_{i,j} \alpha_{ij}^2\right)^{1/2}.$$

It is known [6] that if A has the SVD (2.1) then $\|A\|_2 = \sigma_1$, and $\|A\|_F = \|\Sigma\|_F = (\sum_j \sigma_j^2)^{1/2}$. The following theorem (the Eckart-Young theorem) can be found in [6, p. 19].

THEOREM 2.1. *Let A be a matrix of rank r with* SVD (2.1). *Let $s \leqq r$ be an integer, and let*

$$A_s \equiv \sum_{j=1}^{s} \mathbf{u}_j \sigma_j \mathbf{v}_j^T.$$

Then

$$\|A - A_s\|_2 = \min_{\operatorname{rank}(B)=s} \|A - B\|_2 = \sigma_{s+1}.$$

We shall also make use of the spectral condition number $\kappa(A)$ of A, which is defined by

$$\kappa(A) \equiv \|A\|_2 \|A^+\|_2 = \sigma_1/\sigma_r. \tag{2.2}$$

The chief reason for being interested in the generalized inverse is that the solution to the least squares problem

$$\min \|A\mathbf{x} - b\|_2$$

having smallest norm $\|\mathbf{x}\|_2$ is A^+b.

3. Newton iteration. Newton iteration

$$X_{k+1} = X_k(2I - AX_k) = (2I - X_kA)X_k \tag{3.1}$$

was proposed by Schulz in 1933 for computing the inverse of a nonsingular matrix A [16]. Much later, Ben-Israel and Cohen proved that the iteration converges to A^+ provided that

$$X_0 = \alpha_0 A^T, \tag{3.2}$$

with α_0 positive and sufficiently small [1], [2].

All the following discussion employs an SVD-based analysis of the method, first developed by Soderstrom and Stewart [17], who observed that (2.1), (3.1), and (3.2) imply that each iterate has an SVD of the form

$$X_k = V\Xi_k U^T$$

with the matrices U and V of (2.1). The products X_kA therefore satisfy

$$X_kA = VR_kV^T \tag{3.3}$$

where $R_k = \Xi_k\Sigma \equiv \operatorname{diag}(\rho_1^{(k)}, \rho_2^{(k)}, \cdots, \rho_n^{(k)})$; moreover, for all $1 \leqq j \leqq n$ and all $k \geqq 0$,

$$1 - \rho_j^{(k+1)} = (\rho_j^{(k)} - 1)^2. \tag{3.4}$$

Clearly, $\rho_j^{(0)} = \alpha_0\sigma_j^2$. For any $\alpha_0 < 2/\sigma_1^2$, (3.4) implies the quadratic convergence of $\rho_j^{(k)}$ to one for all j, $1 \leqq j \leqq r$, and, therefore, of X_k to A^+. The optimum choice of α_0 in (3.2), which minimizes $\|I - X_0A\|_2$ by making $\rho_1^{(0)} - 1 = 1 - \rho_r^{(0)}$, is

$$\alpha_0 = \frac{2}{\sigma_1^2 + \sigma_r^2}. \tag{3.5}$$

Let $\kappa(A)$ be given by (2.2). Then with the choice (3.5),

$$\rho_r^{(0)} = \frac{2}{1 + \kappa(A)^2}, \tag{3.6}$$

so that $\rho_r^{(0)} \ll 1$ if $\kappa(A)$ is large. In practice, information about σ_r is hard to get. We may instead use a suboptimal but nevertheless safe alternative, such as

$$\alpha_0 = \frac{1}{\|A\|_1\|A\|_\infty}. \tag{3.7}$$

Other choices of α_0 which do not require an estimate of σ_r are available in [1], [2], [11], and [12].

It follows from (3.4) that small singular values $\rho_j^{(k)}$ approximately double at every step. Therefore, it takes about $2\log_2 \kappa(A)$ steps of (3.1) to get to the point where $\rho_r^{(k)} \geqq \frac{1}{2}$ and an additional log log $(1/\varepsilon)$ iterations for convergence of all the ρ_j to within ε of 1. Thus, if the floating-point operations are the measure of cost, the method is more expensive than the conventional alternatives. The reason for our interest in this method is that (3.1) essentially amounts to two matrix multiplications, which can be very efficiently implemented on systolic arrays and on vector and parallel computers [13], [14]. Additional savings are possible: if A is sparse, then X_kA is less costly to compute; in the case of Toeplitz and many other structured matrices, AX_k amounts to a few matrix-vector products (see [8]–[10]). Finally, as only the product of A and some vectors is required, we do not need to form A, which is convenient in some applications.

In operations counts, we use the term "flop" to mean a multiply-add pair. For unstructured $m \times n$ matrices, each iteration step (3.1) essentially amounts to two matrix multiplications, that is to $1.5mn^2$ flops, exploiting the symmetry of AX_k. In § 5 we show how to reduce this to about $mn^2+\frac{1}{2}n^3$ in a practical implementation.

4. Convergence acceleration by scaling. Schreiber [14] presented the scaled iteration

$$X_{k+1} = \alpha_{k+1}(2I - X_kA)X_k, \qquad k = 0, 1, \cdots \tag{4.1}$$

where X_0 is given by (3.2). Here, we employ an acceleration parameter $\alpha_{k+1} \in [1, 2]$ chosen so as to minimize a bound on the maximum distance of any nonzero singular value of $X_{k+1}A$ from 1. Let us assume that we know bounds $\sigma_{\min}$ and $\sigma_{\max}$ on the singular values of A satisfying

$$\sigma_{\min} \leqq \sigma_r^2 \leqq \sigma_1^2 \leqq \sigma_{\max}.$$

In this case, every eigenvalue of A^TA lies in the interval $[\sigma_{\min}, \sigma_{\max}]$. (We assume this interval has nonzero length; otherwise A is a scalar multiple of an orthogonal matrix and X_0 is its inverse.) We then choose X_0 according to (3.2) with

$$\alpha_0 = \frac{2}{\sigma_{\min} + \sigma_{\max}}. \tag{4.2}$$

It follows from (2.1), (3.2), (4.1), and (4.2) that for all j and k,

$$\rho_j^{(0)} = \frac{2\sigma_j^2}{\sigma_{\min} + \sigma_{\max}},$$

and

$$\rho_j^{(k+1)} = \alpha_{k+1}(2 - \rho_j^{(k)})\rho_j^{(k)}.$$

To determine the acceleration parameters α_k, for $k > 0$, we let

$$\underline{\rho}^{(0)} = \alpha_0\sigma_{\min} = \frac{2\sigma_{\min}}{\sigma_{\min} + \sigma_{\max}} \tag{4.3}$$

and

$$\bar{\rho}^{(0)} = \alpha_0\sigma_{\max} = \frac{2\sigma_{\max}}{\sigma_{\min} + \sigma_{\max}}; \tag{4.4}$$

these being lower and upper bounds on the singular values of X_0A. Then take

$$\alpha_{k+1} = \bar{\rho}^{(k+1)} = \frac{2}{1 + (2 - \underline{\rho}^{(k)})\underline{\rho}^{(k)}}, \tag{4.5}$$

which is both an acceleration parameter and an upper bound on $\{\rho_j^{(k+1)}\}$, and

$$\underline{\rho}^{(k+1)} = \alpha_{k+1}(2 - \underline{\rho}^{(k)})\underline{\rho}^{(k)}, \tag{4.6}$$

which is a lower bound on $\{\rho_j^{(k+1)}\}$. The definitions (4.3)–(4.6) imply that for all $1 \leqq j \leqq r$ and $k \geqq 1$,

$$\underline{\rho}^{(k)} \leqq \rho_j^{(k)} \leqq \bar{\rho}^{(k)},$$

and

$$\underline{\rho}^{(k)} = 2 - \bar{\rho}^{(k)}. \tag{4.7}$$

(Note that (4.7) follows immediately from (4.5) and (4.6). The upper bound $\rho_j^{(k)} \leqq \bar{\rho}^{(k)}$ is likewise straightforward. Finally, if

$$0 \leqq \underline{\rho}^{(k)} \leqq \rho_j^{(k)} \leqq (2 - \underline{\rho}^{(k)}),$$

then

$$(2 - \underline{\rho}^{(k)})\underline{\rho}^{(k)} \leqq (2 - \rho_j^{(k)})\rho_j^{(k)} \leqq 1,$$

whence, by (4.6) and the definition of $\rho_j^{(k)}$, the lower bound $\underline{\rho}^{(k)} \leqq \rho_j^{(k)}$ follows.)

Except for the last few iterations before convergence, $\underline{\rho}^{(k)} \ll 1$, which implies that $\alpha_{k+1} \approx 2$. Thus, $\rho_r^{(k+1)} \approx 4\rho_r^{(k)}$. Therefore,

$$-\log_4 \rho_r^{(0)} \approx \log_2 [(\kappa(A)^2 + 1)^{1/2}] = \log_2 \kappa(A) + O(1/\kappa(A)^2)$$

steps suffice to bring all the singular values of $X_k A$ up to $\frac{1}{2}$, which is half as many as for the unaccelerated version.

We shall now derive a theorem concerning the optimality of the acceleration parameters given by (4.5). Let the symmetric residual matrix initially be

$$E \equiv I - \alpha_0 A^T A = I - X_0 A. \tag{4.8}$$

It is straightforward to show that Newton's method, starting with $X_0 = \alpha_0 A^T$, produces iterates X_k satisfying

$$X_k A = I - E^m \tag{4.9}$$

where $m = 2^k$. For a nonsingular matrix A and for the choice (4.2) for α_0, the eigenvalues of E lie in the open interval $(-1, 1)$, and we have that $E^m \to 0$ and, therefore, $X_k A \to I$ as $k \to \infty$.

Furthermore, (4.8) and (4.9) imply that

$$X_k = (I + E + \cdots + E^{m-1})\alpha_0 A^T.$$

Thus, Newton's method is related to the Neumann series expansion

$$(I - E)^{-1} = I + E + E^2 + \cdots.$$

We therefore ask whether the accelerated method (3.2), (4.1)-(4.6) is related to a better polynomial approximation to $(I - E)^{-1}$. In fact, it is exactly equivalent to approximation of this inverse by a Tchebychev polynomial in E, as we now show.

Let

$$T_{2^k}(\zeta) \equiv \cos(2^k \cos^{-1} \zeta)$$

be the Tchebychev polynomial of degree 2^k on $(-1, 1)$. Recall that $T_0(\zeta) = 1$, $T_1(\zeta) = \zeta$, and

$$T_{2^{k+1}}(\zeta) \equiv 2T_{2^k}^2(\zeta) - 1. \tag{4.10}$$

The scaled Tchebychev polynomials on $(\sigma_{\min}, \sigma_{\max})$ are defined by

$$t_k(\zeta) \equiv \frac{T_{2^k}(\gamma\zeta + \delta)}{T_{2^k}(\delta)}$$

where $\gamma = 2/(\sigma_{\max} - \sigma_{\min})$ and $\delta = -(\sigma_{\max} + \sigma_{\min})/(\sigma_{\max} - \sigma_{\min})$. Surely, t_k is a polynomial of degree 2^k, and $t_k(0) = 1$, so that

$$t_k(\zeta) = 1 - \zeta \bar{t}_k(\zeta)$$

for some polynomial $\bar{t}_k$ of degree 2^k-1. Furthermore, it is a classical result that among all such polynomials, t_k minimizes the norm $\sup_{\sigma_{\min}\leq\zeta\leq\sigma_{\max}}|t_k(\zeta)|\equiv\|t_k\|_{\infty,(\sigma_{\min},\sigma_{\max})}$. They also satisfy a recurrence like (4.10). Let $\beta_k\equiv T_{2^k}(\delta)$. Then by (4.10),

$$\beta_k t_k(\zeta)=2(\beta_{k-1}t_{k-1}(\zeta))^2-1. \tag{4.11}$$

THEOREM 4.1. *Let the sequence of matrices X_k, $k=0,1,\cdots$ be generated by* (3.2), (4.1)-(4.6). *Then*

$$X_k=\bar{p}_k(A^TA)A^T \tag{4.12}$$

where $\bar{p}_k(\zeta)$ is a polynomial of degree 2^k-1. The polynomial $1-\zeta\bar{p}_k(\zeta)$ is the scaled Tchebychev polynomial $t_k(\zeta)$ of degree 2^k on $(\sigma_{\min},\sigma_{\max})$.

Remark. Before proving the theorem, we point out that (4.12) implies that

$$X_kA=\bar{p}_k(A^TA)A^TA,$$

and therefore that

$$\|I-X_kA\|_2=\max_{1\leq j\leq n}|1-\sigma_j^2\bar{p}_k(\sigma_j^2)|,$$

which shows the relevance of the theorem's conclusion.

An analogue of this result also holds for X_0 any matrix of the form $r(A^TA)A^T$, with r a polynomial, such that the 2-norm of $I-X_0A$ is less than 1.

Proof. The claim (4.12) is clearly true when $k=0$, with $\bar{p}_0(\zeta)\equiv\alpha_0$. With the choice (4.2), $1-\zeta\bar{p}_0(\zeta)=1-2\zeta/(\sigma_{\max}+\sigma_{\min})=t_0(\zeta)$ is the appropriate scaled Tchebychev polynomial.

Now use induction on k. A straightforward calculation using (4.5) and (4.7) shows that, for all $k\geq 1$, $1<\alpha_k<2$ and $\beta_k=1/(\alpha_k-1)$, or equivalently that $\alpha_k=(1+\beta_k)/\beta_k$. It also follows from (4.1) and the inductive hypothesis in the form (4.12), after multiplying by A, that (4.12) holds for the polynomial

$$\bar{p}_k(\zeta)=\alpha_k(2-\zeta\bar{p}_{k-1}(\zeta))\bar{p}_{k-1}(\zeta).$$

From this and the relations between α_k and β_k we derive the recurrence

$$\beta_k(1-\zeta\bar{p}_k)=2\beta_{k-1}^2(1-\zeta\bar{p}_{k-1})^2-1.$$

Thus, $\beta_k(1-\zeta\bar{p}_k)$ satisfies the Tchebychev recurrence (4.11), which proves the theorem. □

5. Convergence acceleration with cubic polynomials. In § 3 we saw that with the unaccelerated Newton method, the convergence of $\rho_j^{(k)}$ to one is slow for all j such that $\rho_j^{(0)}$ lies near zero or two. For many input matrices, and for moderately large k, the set $\{\rho_j^{(k)}\}_{j=0}^n$ produced by Newton's method without acceleration consists of two clusters, one lying near zero and the other near one. In this section, we present an alternative to the acceleration method of § 4 that, in the case of such a large gap in the spectrum of X_kA, results in much faster convergence.

Let X be a fixed matrix satisfying $XA=VRV^T$, $R=\operatorname{diag}(\rho_1,\cdots,\rho_n)$ where $0\leq\rho_j\leq 2$ for all j. We seek an improved approximation X_1 to A^+ of the form

$$X_1=(\gamma_3(XA)^2+\gamma_2XA+\gamma_1I)X.$$

We choose $\{\gamma_1,\gamma_2,\gamma_3\}$ so that the cubic polynomial $c(\rho)=\gamma_1\rho+\gamma_2\rho^2+\gamma_3\rho^3$ satisfies

(i) $c(1)=1$,
(ii) $c'(1)=0$,
(iii) $c(0)=0$,
(iv) $c'(0)\gg 2$.

The idea here is that small singular values are amplified by the factor $c'(0)$ while those near 1 continue to converge. It is quite evident, however, that $c(\rho)$ will take large values for some $\rho \in (0, 1)$, so we must exercise caution.

We begin by finding $\underline{\rho} > 0$ such that we are certain that there are no eigenvalues ρ_j in $(\underline{\rho}, 1-\underline{\rho})$ or in $(1+\underline{\rho}, 2]$. Let $T = XA$ and compute T^2 and $\delta \equiv \|T - T^2\|_F$. Now, it follows from (3.3) that

$$\delta^2 = \sum_{j=1}^{n} [\rho_j(1-\rho_j)]^2; \tag{5.1}$$

whence, for all $1 \leqq j \leqq n$,

$$\rho_j|1-\rho_j| \leqq \delta.$$

If $\delta \geqq \frac{1}{4}$, this provides us with no useful information. If, on the contrary, $0 < \delta < \frac{1}{4}$, then we conclude that all the eigenvalues ρ_j lie in the two closed intervals: $[0, \underline{\rho}]$ and $[1-\underline{\rho}, 1+\bar{\rho}]$, where

$$0 \leqq \underline{\rho} \equiv \tfrac{1}{2} - \sqrt{\tfrac{1}{4} - \delta} < \tfrac{1}{2}, \qquad \tfrac{1}{2} < 1 - \underline{\rho} = \tfrac{1}{2} + \sqrt{\tfrac{1}{4} - \delta},$$
$$1 \leqq 1 + \bar{\rho} = \tfrac{1}{2} + \sqrt{\tfrac{1}{4} + \delta} < 1 + \underline{\rho}.$$

(See Fig. 1.) Thus, for $j = 1, \cdots, n$,

$$\rho_j \in [0, \underline{\rho}] \cup [1-\underline{\rho}, 1+\bar{\rho}]. \tag{5.2}$$

In other words ρ_j is in neither $(\underline{\rho}, 1-\underline{\rho})$ nor $(1+\bar{\rho}, 2]$. Now we show how to choose $c(\rho)$ so as to satisfy the criteria (i)-(iv), as well as the equally important criteria

(v) $c : [0, \underline{\rho}] \to [0, 1]$,

(vi) $c : [1-\underline{\rho}, 1+\bar{\rho}) \to [1, 1+\underline{\rho})$.

To determine c, we enforce (i)-(iii) together with

(vii) $c(\underline{\rho}) = 1$.

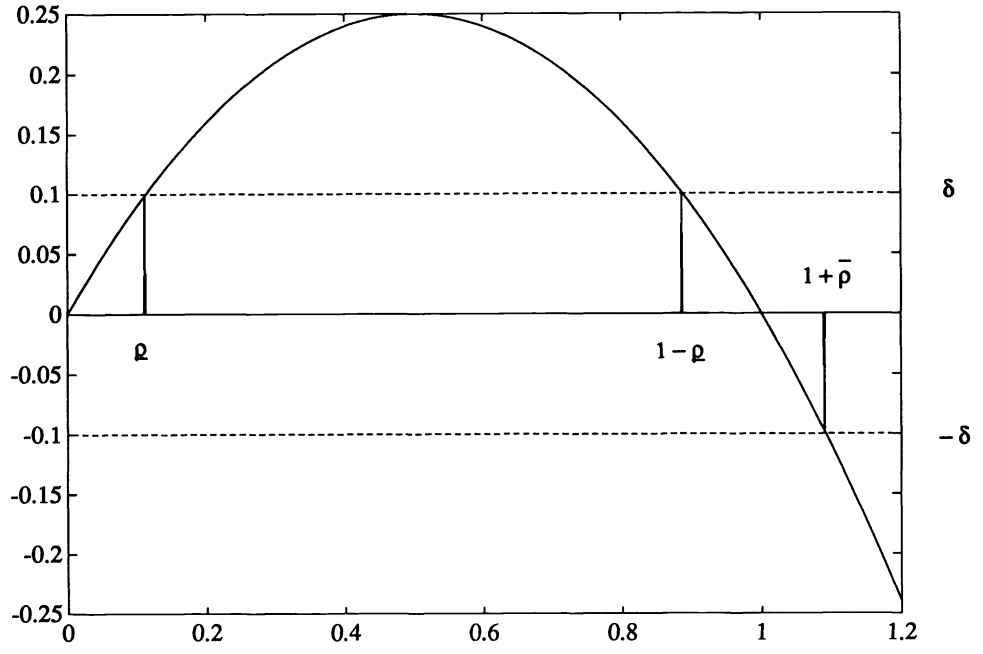


FIG. 1. *The intervals* $[0, \underline{\rho}]$ *and* $[1-\underline{\rho}, 1+\bar{\rho}]$ *contain all the* ρ_j *when* $\delta = \|XA - (XA)^2\|_F < \frac{1}{4}$.

The unique solution is

$$c(\rho)=\frac{1}{\underline{\rho}}(\rho^2-(2+\underline{\rho})\rho+(1+2\underline{\rho}))\rho,$$

which may be rewritten as

$$c(\rho)=\frac{1}{\underline{\rho}}((\rho-1)^3+(1-\underline{\rho})(\rho-1)^2)+1.$$

THEOREM 5.1. *Let* $0\leqq\underline{\rho}\leqq\frac{1}{2}$. *If* $c(\rho)$ *is the unique cubic satisfying* (i)-(iii) *and* (vii) *above, then* (v) *and* (vi) *hold.*

Proof. To show that (v) holds, we recall that a nonvanishing cubic polynomial $c(\rho)$ has at most two critical points (where $c'(\rho)\equiv 0$). One of these is $\rho=1$ (condition (ii)). By (i), (vii), and Rolle's theorem, the other one is in $(\underline{\rho}, 1)$, so $c(\rho)$ is monotone on $[0, \underline{\rho}]$ and must therefore map $[0, \underline{\rho}]$ onto $[0, 1]$.

To establish (vi), note that

$$c(\rho)=1+\frac{(\rho-1)^2}{\underline{\rho}}(\rho-\underline{\rho}).$$

Now let $\rho=1-\varepsilon$ with $|\varepsilon|\leqq\underline{\rho}<\frac{1}{2}$. Then $c(\rho)-1=\varepsilon^2(1-\varepsilon-\underline{\rho})/\underline{\rho}$, and since the second factor is bounded by 0 and 1, the right-hand side is bounded by 0 and $\underline{\rho}$, establishing (vi). □

Thus, if $\delta\geqq\frac{1}{4}$ we cannot accelerate. In this case, we let $X_1=(2I-T)X$ and proceed to the next iteration (i.e., X_1 is the result of a Newton step (3.1)). On the other hand, if $\delta<\frac{1}{4}$ we compute $\underline{\rho}:=\frac{1}{2}-\sqrt{\frac{1}{4}-\delta}$ and let $X_1=(1/\underline{\rho})(T^2-(2+\underline{\rho})T+(1+2\underline{\rho})I)X$ (i.e., X_1 is the result of a cubic step).

Here is a practical algorithm incorporating this idea for computing A^+.

ALGORITHM CUINV.
INPUT: A
OUTPUT: A^+
STEP 0 [INITIALIZE]:
    Choose $X=\alpha_0 A^T$, with α_0 given by (4.2) or (3.7);
    $T:=XA$;
    T2VALID := false;
STEP 1 [NEWTON STEP]:
    if X is sufficiently close to A^+ **then return**(X);
    $X:=(2I-T)X$;
    if T2VALID **then**
        $T:=2T-T_2$
    else
        $T:=XA$;
    endif
    T2VALID := false;
    if trace(T) $\geqq n-\frac{1}{2}$ **then**
        goto STEP 1;
STEP 2 [TEST FOR SMALL CHANGE]:
    $T_2:=T^2$;
    $\delta:=\|T-T_2\|_F$;
    if $\delta\geqq\frac{1}{4}$ **then**
        T2VALID := true;

goto STEP 1;
else
goto STEP 3;
STEP 3 [USE ACCELERATION]:
$\underline{\rho} := \frac{1}{2} - \sqrt{\frac{1}{4} - \delta}$;
$X := (1/\underline{\rho})(T_2 - (2+\underline{\rho})T + (1+2\underline{\rho})I)X$;
$T := XA$;
goto STEP 1;

COMMENTS on ALGORITHM CUINV.

(1) Stopping criteria are discussed by Soderstrom and Stewart [17].

(2) First, we have made use of the fact that after a Newton step (3.1),

$$\begin{aligned} T_{k+1} &\equiv X_{k+1}A \\ &= (2I - X_kA)X_kA \\ &= (2I - T_k)T_k. \end{aligned}$$

Thus, if we decide to reject the use of a cubic acceleration step, the computation of T^2 in STEP 2 is not wasted, because it saves us at least that much work in the following Newton step (STEP 1).

(3) We do not use cubic steps exclusively. We do need at least one Newton step for each cubic step because without Newton steps, we cannot find intervals $[0, \underline{\rho}]$ and $[1-\underline{\rho}, 1+\underline{\rho}]$ known to contain all the eigenvalues. Furthermore, the Newton steps finish the job of forcing eigenvalues near 1 to actually converge.

(4) We use only Newton steps when all the singular values of XA are close to 1. This is signaled by the convergence of trace (XA) to within one half of its limiting value of n.

If A is an $m \times n$ matrix, then we assume that the cost of computing the product XA is $\frac{1}{2}mn^2$ flops (we exploit the symmetry of XA), the cost of computing the product T^2 is $\frac{1}{2}n^3$ flops, and the cost of computing X in STEP 1 or STEP 3 is mn^2 flops. Then, if we skip STEP 3, an iteration of CUINV costs

$$n^2m + \tfrac{1}{2}n^3 \text{ flops},$$

assuming that T2VALID was true. The cost of an iteration in which we take STEPS 1-3 is

$$3mn^2 + \tfrac{1}{2}n^3,$$

assuming that T2VALID was false. These assumptions are warranted since it is the iterations that take STEP 3 that cause T2VALID to be false. Thus, for $n \approx m$, we pay a premium of only about 16 percent compared with two Newton steps.

What is the effect of a full iteration assuming STEP 3 is taken? Formally, the eigenvalues of XA are mapped as follows:

$$\rho \to c((2-\rho)\rho) = 1 + \frac{(1-\underline{\rho})}{\underline{\rho}}(1-\rho)^4 - \frac{1}{\underline{\rho}}(1-\rho)^6 \equiv C(\rho; \underline{\rho}). \tag{5.3}$$

For small ρ, $C(\rho; \underline{\rho}) \approx (4+2/\underline{\rho})\rho$. And because it will often be the case that $\underline{\rho} \ll 1$, the combined iteration greatly amplifies the small eigenvalues. Those near 1 continue to

converge superlinearly; equation (5.3) shows that

$$|1-C(\rho;\underline{\rho})| = \frac{1-\rho}{\underline{\rho}}(1-\rho)^4 + O((1-\rho)^6)$$
$$\leqq (1-\underline{\rho})(1-\rho)^3 + O((1-\rho)^6) = O((1-\rho)^3),$$

since any eigenvalue ρ of XA exceeding $\underline{\rho}$ must be in $[1-\underline{\rho}, 1+\underline{\rho}]$, so $\underline{\rho} \geqq 1-\rho$.

See § 10 for some experimental evidence of the efficiency and reliability of ALGORITHM CUINV.

6. An improved initial iterate in the symmetric positive-definite case. Let us consider the case of a symmetric positive-definite matrix A, which is important in many applications [6]. Given a matrix A with the SVD

$$A = V\Sigma V^T, \qquad \Sigma = \operatorname{diag}(\sigma_1 \geqq \cdots \geqq \sigma_n > 0) \tag{6.1}$$

(which is its eigendecomposition, too) we shall choose

$$X_0 = \beta I + \alpha_0 A \tag{6.2}$$

so as to optimally place the singular values of X_0A. From (6.1) and (6.2)

$$X_0A = V(\beta\Sigma + \alpha_0\Sigma^2)V^T.$$

We choose (β, α_0) so that, with $p(\sigma) \equiv \beta\sigma + \alpha_0\sigma^2$, the largest possible initial error

$$\|I - X_0A\|_2 \equiv \max_{\sigma_n \leqq \sigma \leqq \sigma_1} |p(\sigma) - 1|$$

is minimized. By standard arguments (see [5, Chap. 9]), $1-p$ is a scaled Tchebychev polynomial on (σ_n, σ_1):

$$1 - p(\sigma) = T_2(\sigma) = \frac{2(\sigma - \sigma_{\text{mid}})^2 - \delta^2}{2\sigma_{\text{mid}}^2 - \delta^2}$$

where $\sigma_{\text{mid}} = \frac{1}{2}(\sigma_n + \sigma_1)$ and $\delta = \sigma_1 - \sigma_{\text{mid}}$. Moreover,

$$\|I - X_0A\|_2 = \max_{\sigma_n \leqq \sigma \leqq \sigma_1} |T_2(\sigma)| = T_2(\sigma_1) = \frac{\delta^2}{2\sigma_{\text{mid}}^2 - \delta^2}.$$

We are most concerned about $\rho_n^{(0)}$ the smallest singular value of X_0A, which is

$$p(\sigma_n) = \frac{2(\sigma_{\text{mid}}^2 - \delta^2)}{2\sigma_{\text{mid}}^2 - \delta^2}.$$

Let $\omega \equiv \delta/\sigma_{\text{mid}} < 1$. The condition number κ is given by $\kappa = \sigma_1/\sigma_n$. Thus

$$\kappa = \frac{\sigma_{\text{mid}} + \delta}{\sigma_{\text{mid}} - \delta} = \frac{1+\omega}{1-\omega};$$

whence

$$\omega = \frac{\kappa - 1}{\kappa + 1}.$$

Thus, the smallest initial singular value of X_0A is

$$\begin{aligned} p(\sigma_n) &= \frac{2(1-\omega^2)}{2-\omega^2} \\ &= \frac{2((\kappa+1)^2 - (\kappa-1)^2)}{2(\kappa+1)^2 - (\kappa-1)^2} \\ &= \frac{8\kappa}{\kappa^2 + 6\kappa + 1}. \end{aligned} \tag{6.3}$$

This asymptotically is $8\kappa^{-1}+O(\kappa^{-2})$, which is far larger than the $2\kappa^{-2}$ provided by the "optimal" choice $X_0=\alpha_0 A$ (see (3.6))! To find β, α_0, and hence X_0, we use the equations

$$\begin{aligned}\beta\sigma+\alpha_0\sigma^2=p(\sigma)&=1-T_2(\sigma)\\&=\frac{2\sigma_{\text{mid}}^2-\delta^2-2(\sigma-\sigma_{\text{mid}})^2+\delta^2}{2\sigma_{\text{mid}}^2-\delta^2}\\&=\frac{-2\sigma^2+4\sigma\sigma_{\text{mid}}}{2\sigma_{\text{mid}}^2-\delta^2}.\end{aligned}$$

Hence

$$\beta\equiv\frac{4\sigma_{\text{mid}}}{2\sigma_{\text{mid}}^2-\delta^2},\qquad \alpha_0\equiv-\frac{2}{2\sigma_{\text{mid}}^2-\delta^2}. \tag{6.4}$$

Remark. Even without computing estimates or bounds for the extreme singular values we may improve the initial approximation when A is symmetric positive definite by choosing $X_0=(1/\|A\|_F)I$, for in this case $\|I-X_0A\|_2\leqq 1-1/(n^{1/2}\kappa)$.

7. Suppressing the smaller singular values. In this section, given a matrix A and a positive scalar ε, we show how to compute $A(\varepsilon)$ and $A^+(\varepsilon)$, where $A(\varepsilon)$ is obtained from A by suppressing (that is, setting to zero) all the singular values of A that do not exceed ε. (As noted in § 2 above, $A(\varepsilon)$ is a closest approximation to A by matrices whose rank is that of $A(\varepsilon)$. In practice, solving least squares problems often requires the use of this form of regularization; for when A is very badly conditioned, its generalized inverse is largely unknowable due to perturbations in A, but the reduced-rank generalized inverses are much better conditioned. This idea is discussed more fully by Golub and Van Loan [6, § 5.5].)

In order to compute these rank-reduced generalized inverses, we require a polynomial $c(\rho)$ that gives fast convergence to 0 near $\rho=0$ and to 1 near $\rho=1$. Consider first the iteration

$$X_{k+1/2}=(2I-X_kA)X_k,\qquad X_{k+1}=X_{k+1/2}AX_{k+1/2} \tag{7.1}$$

that was proposed by Schreiber [14]. The eigenvalues of X_kA satisfy the equations $\rho_j^{(k+1)}=((2-\rho_j^{(k)})\rho_j^{(k)})^2$ for all j and k. Figure 2 illustrates the effect of this mapping. The fixed points are $\hat\rho_0=0$, $\hat\rho_1=(3-\sqrt5)/2=.3819\cdots$, $\hat\rho_2=1$, and $\hat\rho_3=(3+\sqrt5)/2=2.618\cdots$ which are the roots of the quartic $(2-\hat\rho)^2\hat\rho^2=\hat\rho$. Consider especially the intervals $\{\rho: 0<\rho<\hat\rho_1\}$ and $\{\rho:\hat\rho_1<\rho<2-\hat\rho_1\}$ (note that $2-\hat\rho_1=(1+\sqrt5)/2=1.618\cdots$). Evidently, the eigenvalues from the former interval are sent toward zero and from the latter interval toward one. The convergence is ultimately quadratic but is slow near $\hat\rho_1$ and $2-\hat\rho_1$. To compute $A^+(\varepsilon)$, it suffices to apply the iteration (4.1)–(4.6) with appropriate $\sigma_{\min}$ and $\sigma_{\max}$ until the following relations hold:

$$0\leqq\rho_j^{(k)}<\hat\rho_1 \qquad \text{if and only if } \sigma_j<\varepsilon, \tag{7.2}$$

$$\hat\rho_1<\rho_j^{(k)}<2-\hat\rho_1 \qquad \text{otherwise.} \tag{7.3}$$

We can satisfy (7.2) for $k=0$ by setting $\alpha_0=\hat\rho_1/\varepsilon^2$, but then (7.3) may not hold. Therefore, we proceed as follows:

(0) Compute an upper bound $\sigma_{\max}$ on σ_1^2.

(1) Set $\alpha_0=\min\{2/(\sigma_{\max}+\varepsilon^2),\hat\rho_1/\varepsilon^2\}$. This ensures that the eigenvalues $\rho_j^{(0)}$ (of X_0A) corresponding to the singular values $\sigma_j\geqq\varepsilon$ (of A) are all closer to 1 than the smaller eigenvalues. Set $\bar\rho^{(0)}=\alpha_0\sigma_{\max}$, $\underline\rho^{(0)}=\alpha_0\varepsilon^2$.

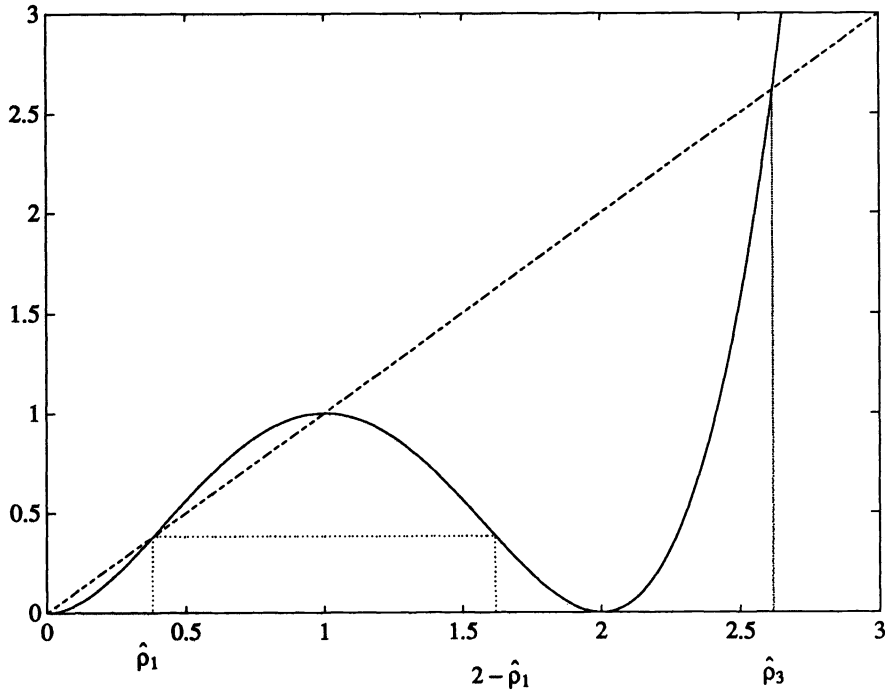


FIG. 2. *The mapping* $(2-\rho)^2\rho^2$.

(2) Apply the iteration (4.1) with parameters given by (4.5) and (4.6) until $\underline{\rho}^{(k)} \geqq \hat{\rho}_1$.
(3) Set $X_k := (\hat{\rho}_1/\underline{\rho}^{(k)})X_k$.
(4) Apply the iteration (7.1) until the matrix $A^+(\varepsilon)$ has been computed with the desired accuracy.

The scaling at Step (3) is done to ensure that all the small singular values (less than ε) are in fact suppressed.

The iteration (7.1) associated with the quartic polynomial $(\rho(2-\rho))^2$ is not the most efficient way of computing $A^+(\varepsilon)$. The same objective can be achieved by using the iteration

$$X^{(k+1)} = (-2X^{(k)}A + 3I)X^{(k)}AX^{(k)} \tag{7.4}$$

associated with the cubic polynomial $\tilde{c}(\rho) = -2\rho^3 + 3\rho^2$ (see Fig. 3). Note that $\tilde{c}(1) = 1$, $\tilde{c}(0) = \tilde{c}'(0) = \tilde{c}'(1) = 0$, $\tilde{c}(\frac{1}{2}) = \frac{1}{2}$, so that the mapping $\tilde{c}(\rho)$ has three nonnegative fixed points $0, \frac{1}{2}$, and 1. Let $\hat{\rho}_4 = (1+\sqrt{3})/2 = 1.366\cdots$ be the unique solution to $-2\hat{\rho}_4^3 + 3\hat{\rho}_4^2 = \frac{1}{2}$ greater than 1. The eigenvalues of $X^{(k)}A$ in the interval $\{\rho: 0 \leqq \rho < \frac{1}{2}\}$ are sent toward zero, and the eigenvalues in the interval $\{\rho: \frac{1}{2} < \rho \leqq \hat{\rho}_4\}$ are sent toward 1; the convergence to 0 and 1 is ultimately quadratic but is slow near $\frac{1}{2}$ and $\hat{\rho}_4$.

The iteration (7.4) is simpler than (7.1): it requires three matrix multiplications, one less than is needed in (7.1).

Remark 7.1. We may also compute $A(\varepsilon)$ itself since

$$A(\varepsilon) = AA^+(\varepsilon)A. \tag{7.5}$$

Remark 7.2. We may use either of the methods (7.1) or (7.4) to "split" a matrix A into two better-conditioned matrices $A(\varepsilon)$ and $\bar{A}(\varepsilon)$ such that

$$A = A(\varepsilon) + \bar{A}(\varepsilon).$$

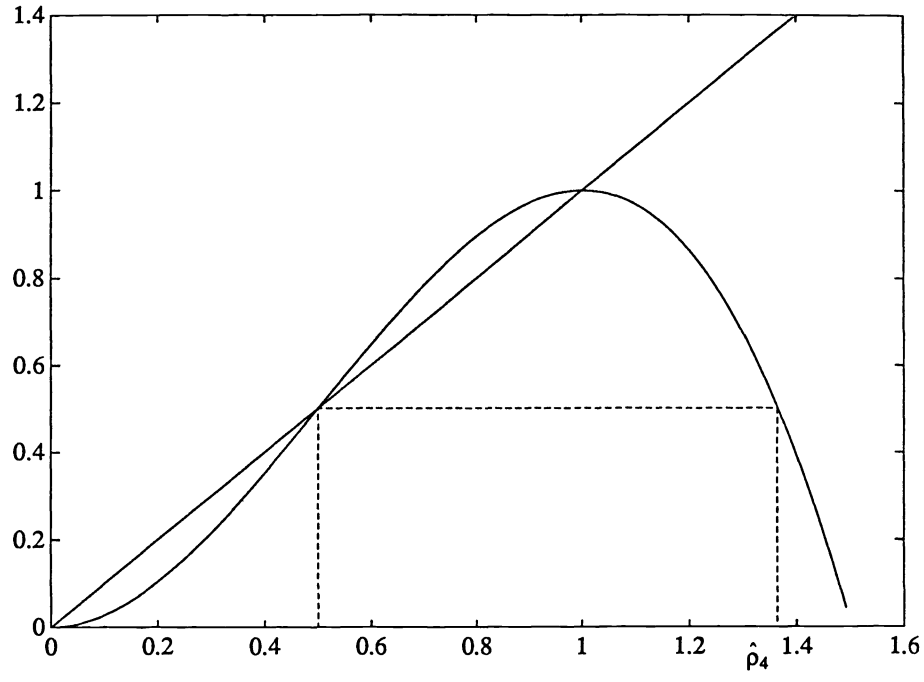


FIG. 3. *The mapping* $-2\rho^3+3\rho^2$.

Moreover, if ε is placed where there is a large gap in the singular values of A, then faster convergence is possible. For A^+ may be computed by the formula

$$A^+ = A^+(\varepsilon)+\bar{A}^+(\varepsilon).$$

The matrices $A(\varepsilon)$ and $\bar{A}(\varepsilon)$ may be much better conditioned than A.

8. Stability of the basic and modified iterations. It is well known that Newton iteration (3.1) is numerically stable and even self-correcting if the input matrix A is nonsingular. If A is singular, however, then it is very mildly unstable.

Let A and A^+ have the following SVDs:

$$A = U\begin{bmatrix}\Sigma & 0\\ 0 & 0\end{bmatrix}V^T, \qquad A^+ = v\begin{bmatrix}\Sigma^+ & 0\\ 0 & 0\end{bmatrix}U^T.$$

In order to analyze the propagation of errors by (3.1), we assume that

$$X_k = A^+ + \hat{E} = V\begin{bmatrix}\Sigma^+ + E_{11} & E_{12}\\ E_{21} & E_{22}\end{bmatrix}U^T, \tag{8.1}$$

where $\hat{E} = VEU^T$ is the current error in X_k. We shall consider whether or not the Newton iteration amplifies these errors. Throughout this section we shall drop all terms of second order in E.

Using (8.1) it is simple to compute that

$$X_{k+1} = X_k + (I - X_kA)X_k = \begin{bmatrix}\Sigma^+ & E_{12}\\ E_{21} & 2E_{22}\end{bmatrix}.$$

Due to the block $2E_{22}$, the iteration (3.1) is mildly unstable if A is singular (in which case the (2, 2) block above is not empty). After $2\log_2 \kappa(A)$ iterations, rounding errors

of order $\kappa^2(A)$ can accumulate. In Fig. 4, this phenomenon is illustrated. Here, A was 6×6 with condition number 30 and rank 4. The method converges in 17 iterations. All logarithms are base 10 and the Frobenius norm is used. Note that the norm of the off-diagonal part of VXV^T grows exponentially. It reaches a value about four orders of magnitude above machine precision (which is roughly 10^{-10}); a loss of four digits of precision results.

On the other hand, the iteration (7.1) is stable even for singular A. Indeed, by (7.1) and (8.1)

$$X^{(k+1/2)} = V\begin{bmatrix} \Sigma^+ & E_{12} \\ E_{21} & 2E_{22} \end{bmatrix} U^T,$$

hence

$$X^{(k+1/2)}A = V\begin{bmatrix} I & E_{12} \\ E_{21}\Sigma & 2E_{22} \end{bmatrix} V^T,$$

and therefore,

$$X^{(k+1)} = X^{(k+1/2)}AX^{(k+1/2)} = V\begin{bmatrix} \Sigma^+ & E_{12} \\ E_{21} & 0 \end{bmatrix} U^T.$$

Thus we deduce that the iteration (7.1) is stable for any matrix A. Similarly, we deduce that the iteration (7.4) is stable for any matrix A. Thus, the methods of § 7 have the dual advantage of stability and a well-conditioned solution, in contrast to the use of Newton iteration (or its accelerated form) on A. (If we wish to compute A^+, then the instability of Newton's method can be partly removed by using a few iterations (7.1)

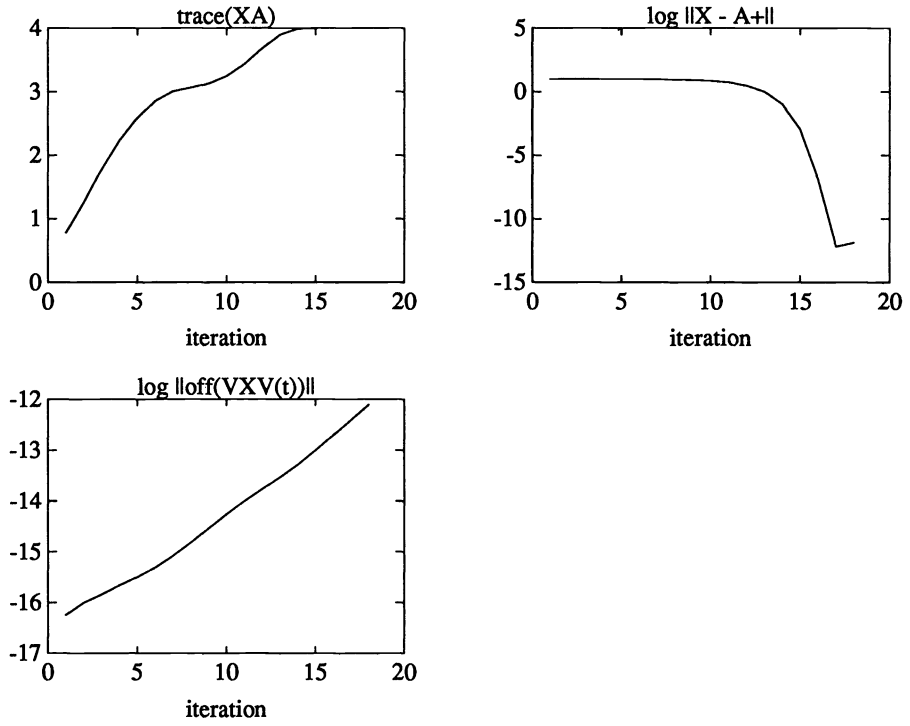


FIG. 4. *Mild instability of Newton iterations in the rank-deficient case.*

or (7.4) after all the significant singular values have converged. Some practical details of this technique are discussed in § 10.)

9. Computing the projection onto a subspace spanned by singular vectors. In this section we discuss a modification of the Newton iterations discussed above that allows us to compute the orthogonal projection matrices onto subspaces spanned by the singular vectors corresponding to either the dominant or the smallest singular values at less expense than computing the generalized inverse of A. Important applications to spectral estimation and direction finding with antenna arrays were developed by Schmidt [15].

Let us define the following matrices:

$$P(\varepsilon) = AA^{+}(\varepsilon) = U\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} U^{T}, \tag{9.1}$$

$$P^{*}(\varepsilon) = A^{+}(\varepsilon)A = V\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} V^{T}, \tag{9.2}$$

where the matrices U and V are from (2.1), I is the $r(\varepsilon)\times r(\varepsilon)$ identity block, and $r(\varepsilon)$ is the number of the singular values of A that are not less than ε. Then $P(\varepsilon)$ and $P^{*}(\varepsilon)$ are the orthogonal projections onto the subspaces spanned by the first $r(\varepsilon)$ columns of the matrices U and V, respectively. Our previous results already give us some iterative algorithms for computing $A(\varepsilon) = (A^{+}(\varepsilon))^{+} = AA(\varepsilon)^{+}A$, as well as $P(\varepsilon)$ and $P^{*}(\varepsilon)$, but there are simpler and more efficient algorithms that we shall give shortly. In addition to the signal processing application mentioned above, we may use this technique as an alternative to the methods of § 7 for computing $A(\varepsilon)$, since

$$A(\varepsilon) = P(\varepsilon)A = AP^{*}(\varepsilon). \tag{9.3}$$

The following iteration extends (7.4) and converges to $P(\varepsilon)$, unless ε is a singular value of A. Let

$$P_0 = \alpha_0 AA^{T} + \beta I, \tag{9.4}$$

where we choose $\alpha_0 > 0$, $\beta \geqq 0$, to satisfy

$$\alpha_0\varepsilon^{2} + \beta = \tfrac{1}{2}, \qquad \alpha_0\sigma_1^{2} + \beta < \hat{\rho}_4 = 1.37\cdots, \tag{9.5}$$

then iterate as follows:

$$P_{k+1} = (-2P_k + 3I)P_k^{2} = (I - 2(P_k - I))P_k^{2}, \qquad k = 0, 1, \cdots. \tag{9.6}$$

The convergence of P_k to $P(\varepsilon)$ immediately follows from the considerations of § 7.

The iteration (9.4)-(9.6) converges to $P^{*}(\varepsilon)$ if we replace AA^{T} by $A^{T}A$ in (9.4). Furthermore, we may compute $A(\varepsilon)$ by using (9.3), which is superior to the solution given in § 7 because we now need to compute a single generalized inverse (rather than two). Also, each iteration step (9.6) only involves two matrix multiplications. And finally, if A is rectangular then one of these iterations is less expensive than (7.4) (for example) because it involves smaller symmetric matrices.

Remark 9.1. The stability analysis of § 8 can be immediately extended to the iteration (9.6).

Remark 9.2. We may accelerate the cubic iteration for $P(\varepsilon)$ as follows. At the early stages of the iteration, it is more important to move singular values away from 0.5. After a step $\tilde{P} = 3P^{2} - 2P^{3}$, we have that the spectrum of $\tilde{P}$ lies in the closed interval $[0, 1]$. We then replace $\tilde{P}$ by $\alpha\tilde{P} + \beta I$ where $\alpha\rho + \beta$ is a line that maps $[0, 1]$ into $(-(\hat{\rho}_4 - 1), \hat{\rho}_4)$. To get the best possible speedup by this method, we choose such a line and also require that $\alpha(\frac{1}{2}) + \beta = \frac{1}{2}$. See Fig. 5.

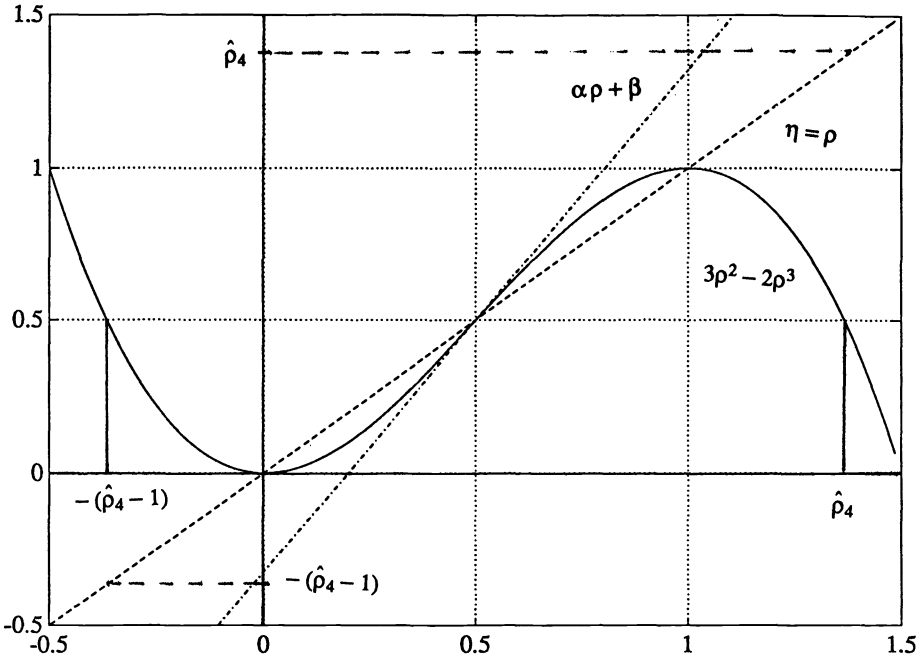


FIG. 5. *Acceleration by scaling and shifting,* $P := \alpha P + \beta I$.

Remark 9.3. Our ability to compute the projectors $P(\varepsilon)$ allows us, with a little additional computation, to do the following:

(1) The projector $P(\varepsilon)$ defines the rank of the matrices $A(\varepsilon)$ and $A^+(\varepsilon)$, for

$$\text{rank } A(\varepsilon) = \text{rank } A^+(\varepsilon) = \|P(\varepsilon)\|_F = \text{trace } (P(\varepsilon)).$$

This observation may be used as the basis of a bisection strategy for computing the singular values of A in polylog time. Indeed, the singular values of A are those of $A(\varepsilon)$ together with those of $A - A(\varepsilon)$. We may in this way reduce the problem to that of computing the positive singular value of a matrix with only one positive singular value. This we discuss in point (4). It is straightforward to develop a similar polylog algorithm for the eigenvalues of any symmetric matrix.

(2) We may compute projectors $P(\varepsilon_1, \varepsilon_2)$ onto subspaces spanned by singular vectors belonging to all the singular values in $[\varepsilon_1, \varepsilon_2)$, since $P(\varepsilon_1, \varepsilon_2) = P(\varepsilon_1) - P(\varepsilon_2)$.

(3) We may determine easily, for a given vector $\mathbf{x}$, whether or not $\mathbf{x} \in S(\varepsilon)$ where $S(\varepsilon)$ is the span of the singular vectors corresponding to singular values greater than or equal to ε, by checking whether or not $\mathbf{x} = P(\varepsilon)\mathbf{x}$.

(4) We may rapidly compute any singular value, regardless of multiplicity, as soon as we have found an interval $[\varepsilon_1, \varepsilon_2]$ that contains this singular value, σ, and no other. For σ is the only singular value of $A(\varepsilon_1, \varepsilon_2) = (P(\varepsilon_1) - P(\varepsilon_2))A$. Its multiplicity k is given by trace $(P(\varepsilon_1) - P(\varepsilon_2))$. And $\sigma^2 = \text{trace } (A^T(\varepsilon_1, \varepsilon_2)A(\varepsilon_1, \varepsilon_2))/k$.

10. Experimental results. We generated a random 64×64 matrix, then changed its singular values so as to create an ill-conditioned matrix A whose singular values lie in two clusters. There are 32 singular values in the interval $[1, 7.6]$ and 32 others in the interval $[10^{-7}, 10^{-6}]$. We used algorithm CUINV to compute A^{-1}. The initial iterate was that of (3.2) and (3.7). The results are shown in Fig. 6; the data are in

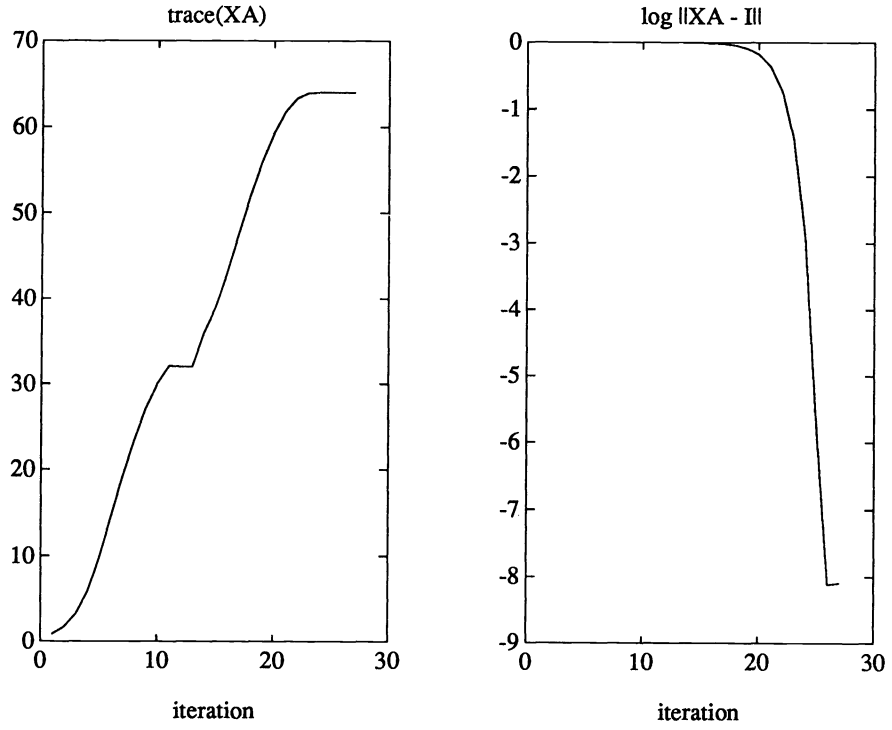


FIG. 6. *Convergence history for first test problem.*

Table 1, which gives trace (X_kA), $\|X_kA-I\|_2$, and the computed bound $\underline{\rho}$ for $k=0, 1, \cdots, 12$. Where $\underline{\rho}=0$ the algorithm skipped the cubic acceleration step. Early on this is because δ is too large, as the first 32 large singular values converge. By way of comparison, Newton's method takes 60 iterations to obtain the solution produced by CUINV in 25.

Next, we repeated this experiment with a second random matrix of order 64, having all its singular values in [0.066, 1]. The method chose unaccelerated Newton steps (19 are required) every time. Cubic steps were never used; this was due to the absence of any gap in spectrum of X_kA.

We therefore modified the algorithm so that when cubic acceleration is ruled out it uses a step of adaptive Tchebychev acceleration as discussed in § 4. For the practical application of this idea, we require a means of finding a bound $\rho_*>0$ such that $\rho_r^{(k)}<\rho_*$. We then use as the acceleration parameter

$$\alpha_k=\frac{2}{1+(2-\rho_*)\rho_*}.$$

This ensures that the acceleration process does not cause a large singular value to be mapped to the left of the smallest, thereby making the problem more difficult. Assume that the test $\delta<\frac{1}{4}$ at STEP 2 of CUINV fails. Let

$$\underline{\delta}=\delta/\sqrt{n}.$$

Then, if $\underline{\delta}<\frac{1}{4}$, we take

$$\rho_*=\tfrac{1}{2}-\sqrt{\tfrac{1}{4}-\underline{\delta}}.$$

TABLE 1

Results for first test matrix.

Iteration	Trace (XA)	$\|XA-I\|$	ρ
0	8.6801E−01	1.0000E+00	0
1	1.6882E+00	1.0000E+00	0
2	3.1987E+00	1.0000E+00	0
3	5.7784E+00	1.0000E+00	0
4	9.6436E+00	1.0000E+00	0
5	1.4398E+01	1.0000E+00	0
6	1.9113E+01	1.0000E+00	0
7	2.3340E+01	1.0000E+00	0
8	2.7108E+01	1.0000E+00	0
9	3.0015E+01	1.0000E+00	0
10	3.2111E+01	1.0000E+00	2.0481E−01
11	3.2003E+01	1.0000E+00	3.0331E−03
12	3.2014E+01	9.9998E−01	8.4572E−06
13	3.5992E+01	9.9357E−01	7.0355E−03
14	3.8974E+01	9.8718E−01	0
15	4.3075E+01	9.7452E−01	0
16	4.7663E+01	9.4970E−01	0
17	5.2046E+01	9.0192E−01	0
18	5.5954E+01	8.1346E−01	0
19	5.9225E+01	6.6172E−01	0
20	6.1749E+01	4.3787E−01	0
21	6.3309E+01	1.9173E−01	0
22	6.3906E+01	3.6761E−02	0
23	6.3997E+01	1.3514E−03	0
24	6.4000E+01	1.8267E−06	0
25	6.4000E+01	7.6424E−09	0

Now, since (5.1) holds,

$$\min_{1\leqq j\leqq n}(\rho_j-\rho_j^2)\leqq \underline{\delta}.$$

Now it must be the case that ρ_r (the smallest positive singular value of X_kA) is less than ρ_* or else all of the singular values are in $(1-\rho_*, 1]$. To rule out the latter possibility, we check whether

$$\text{trace}\,(X_kA)\geqq n(1-\rho_*)$$

and, if not, we accelerate.

With this change, the number of iterations required for convergence dropped to 13 for this problem; the operation count went from 20.5e6 to 14.6e6.

Evidently, even for well-conditioned matrices, modified CUINV can be far more efficient than Newton's method. For moderately ill-conditioned matrices, the differences are pronounced.

Next, we generated a random 64×64 matrix, then changed its singular values so as to create an ill-conditioned matrix A with 54 singular values in $[10^{-16}, 10^{-11}]$ and the remaining 10 in $[0.01, 1]$. We computed the generalized inverse of A(1.e-10), the matrix of rank 10 obtained by suppressing the small singular values of A. We first used algorithm CUINV with A, choosing $\alpha_0=1$. In addition we monitored the growth of $\bar{\varepsilon}$ which was initially set equal to $\varepsilon^2=10^{-20}$ and which is updated according to the

recursions:

$$\bar{\varepsilon} := (2-\bar{\varepsilon})\bar{\varepsilon}$$

in Step 1 and

$$\bar{\varepsilon} := \frac{1}{\underline{\rho}}(\bar{\varepsilon}^2-(2+\underline{\rho})\bar{\varepsilon}+(1+2\underline{\rho}))\bar{\varepsilon}$$

in Step 3. Thus, $\bar{\varepsilon}$ separates the singular values of XA that we wish to suppress from those that we wish to map to 1. At the first iteration in which the parameter $\underline{\rho}$, computed in Step 3, is less than $\bar{\varepsilon}$, we stop and switch to the iteration (7.4) for, since (5.2) holds, the unwanted singular values of XA must now be smaller than $\frac{1}{2}$. In this example, the switch occurred after the seventeenth iteration. Table 2 gives the results; see also Fig. 7.

TABLE 2
Computing $A(\varepsilon)^+$.

Iteration	Trace (XA)	$\|XA(\varepsilon)-A(\varepsilon)^+A\|$	ε	$\underline{\rho}$
1	3.9667E−01	9.9997E−01	1.0000E−20	0
2	7.2843E−01	9.9995E−01	2.0000E−20	0
3	1.2410E+00	9.9989E−01	4.0000E−20	0
4	1.8730E+00	9.9978E−01	8.0000E−20	0
5	2.4361E+00	9.9957E−01	1.6000E−19	0
6	2.8749E+00	9.9913E−01	3.2000E−19	0
7	4.5728E+00	9.9271E−01	2.6999E−18	4.5074E−01
8	5.3217E+00	9.8548E−01	5.3998E−18	0
9	6.0588E+00	9.7117E−01	1.0800E−17	0
10	6.7139E+00	9.4318E−01	2.1599E−17	0
11	7.3764E+00	8.8958E−01	4.3198E−17	0
12	8.1084E+00	7.9136E−01	8.6396E−17	0
13	8.7958E+00	6.2625E−01	1.7279E−16	0
14	9.3559E+00	3.9219E−01	3.4558E−16	0
15	1.0110E+01	9.2682E−02	5.3991E−15	1.7207E−01
16	1.0008E+01	8.4375E−03	1.2779E−12	8.5950E−03
17	1.0000E+01	7.1182E−05	3.5906E−08	7.1192E−05
18	1.0000E+01	3.0452E−11	—	—
19	1.0000E+01	3.0452E−11	—	—

The computed generalized inverse is accurate to about 11 digits. This is somewhat fewer than we could wish, given that the condition number of this problem was 100. The loss of accuracy is due to the fact that the accelerated Newton process (CUINV) went "too far" (raising $\bar{\varepsilon}$ by 12 orders of magnitude) before detecting the possibility of switching to the stable procedure (7.4).

11. Discussion. It has been our purpose here to clarify and illustrate the potential for the use of variants of Newton's method to solve problems of practical interest on highly parallel computers. We have shown how to accelerate the method substantially. We have shown how to modify it to successfully cope with ill-conditioned matrices. We have developed practical implementations. We conclude that Newton's method can be of value for some interesting computations, especially in parallel and other computing environments in which matrix products are especially easy to work with.

Acknowledgments. We would like to thank the referees for many valuable suggestions, especially for helping to simplify and clarify the proofs; Roland Freund and

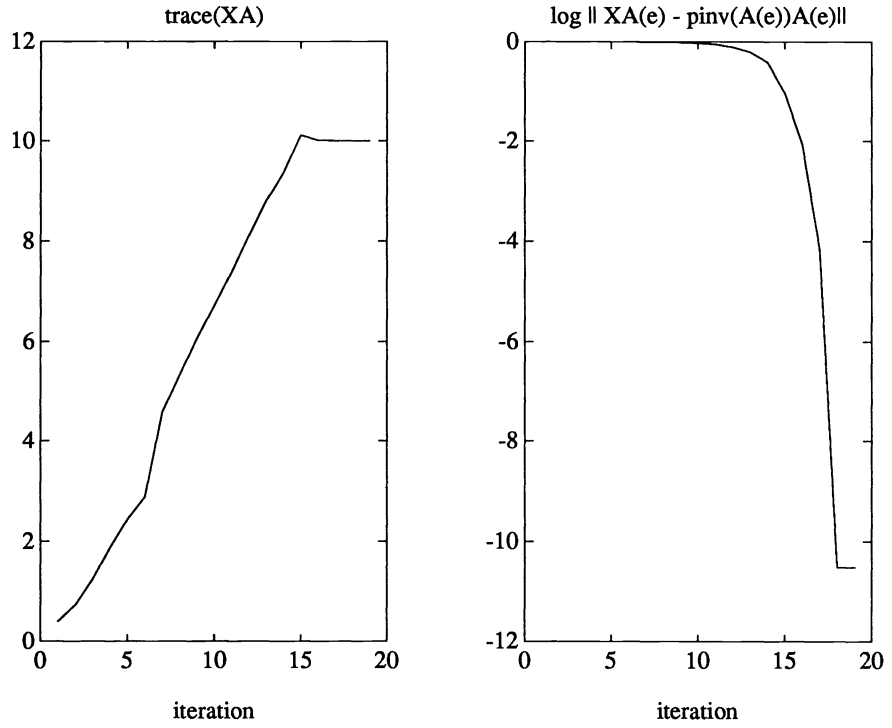


FIG. 7. *Convergence history for second test problem.*

Youcef Saad for suggesting the Tchebychev analysis; and Sally Goodall for expertly typing the manuscript.

REFERENCES

[1] A. BEN-ISRAEL, *A note on iterative method for generalized inversion of matrices*, Math. Comp., 20 (1966), pp. 439-440.

[2] A. BEN-ISRAEL AND D. COHEN, *On iterative computation of generalized inverses and associated projections*, SIAM J. Numer. Anal., 3 (1966), pp. 410-419.

[3] R. P. BRENT, F. T. LUK, AND C. VAN LOAN, *Computation of the singular value decomposition using mesh-connected processors*, J. VLSI Comput. Systems, 1 (1985), pp. 242-270.

[4] J. DONGARRA, J. DUCROZ, I. DUFF, AND S. HAMMARLING, *A set of level 3 basic linear algebra subprograms*, ANL-MCS-P88-1, Argonne National Laboratory, Argonne, IL, 1988.

[5] D. K. FADDEEV AND V. N. FADDEEVA, *Computational Methods of Linear Algebra*, W. H. Freeman, San Francisco, 1963.

[6] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, 2nd ed., Johns Hopkins University Press, Baltimore, MD, 1989.

[7] W. D. HILLIS, *The Connection Machine*, MIT Press, Cambridge, MA, 1985.

[8] V. PAN, *Fast and efficient parallel inversion of Toeplitz and block Toeplitz matrices*, Tech. Report 88-8, Computer Science Department, State University of New York at Albany, Albany, NY, 1988.

[9] ———, *New effective methods for computations with structured matrices*, Tech. Report 88-28, Computer Science Department, State University of New York at Albany, Albany, New York, 1988.

[10] ———, *A new acceleration of the Hilbert-Vandermonde matrix computations by their reduction to Hankel-Toeplitz computations*, Tech. Report 88-34, Computer Science Department, State University of New York at Albany, Albany, NY, 1988.

[11] V. PAN AND J. REIF, *Efficient parallel solution of linear systems*, in Proc. 17th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1985, pp. 143-152.

[12] V. PAN AND J. REIF, *Fast and efficient parallel solution of dense linear systems*, Computers & Math (with Applications), 17 (1989), pp. 1481–1491.
[13] M. J. QUINN, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, New York, 1987.
[14] R. SCHREIBER, *Computing generalized inverses and eigenvalues of symmetric matrices using systolic arrays*, in Computing Methods in Applied Science and Engineering, R. Glowinski and J.-L. Lions, eds., North-Holland, Amsterdam, 1984.
[15] R. O. SCHMIDT, *Multiple emitter location and signal parameter estimation*, IEEE Trans. Antennas and Propagation, 34 (1986), pp. 276–280.
[16] G. SCHULZ, *Iterative Berechnung der Reziproken Matrix*, Z. Angew. Math. Mech., 13 (1933), pp. 57–59.
[17] T. SODERSTROM AND G. W. STEWART, *On the numerical properties of an iterative method for computing the Moore–Penrose generalized inverse*, SIAM J. Numer. Anal., 11 (1974), pp. 61–74.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1131-1139, September 1991
© 1991 Society for Industrial and Applied Mathematics
007

THE FAST GAUSS TRANSFORM WITH VARIABLE SCALES*

JOHN STRAIN†

Abstract. This paper presents a fast algorithm that evaluates a d-dimensional Gaussian convolution sum with scales that vary from point to point. This algorithm evaluates the sum of N Gaussians at M arbitrarily distributed points in $C(N+M)$ work, where C depends only on the precision required and the essential minimum of the scales. It achieves a speedup of almost 2,000 with $N=M=100{,}000$, $d=2$, and scales bounded below by 1/100.

Key words. Gauss transform, fast algorithms, Hermite polynomials

AMS(MOS) subject classifications. 65R10, 33A65, 35K05, 62G20, 65P05

1. Introduction. Many numerical calculations require the evaluation of d-dimensional Gaussian convolution sums having one of the following two forms:

$$Sf(t_i)=\sum_{j=1}^{N} f_j\, e^{-|t_i-s_j|^2/\delta_j} \tag{1}$$

or

$$Tf(t_i)=\sum_{j=1}^{N} f_j\, e^{-|t_i-s_j|^2/\delta_i} \tag{2}$$

at a collection of M points $t=t_i\in\mathbf{R}^d$. Here s_j are given points in $\mathbf{R}^d$, $|x|^2=x_1^2+\cdots+x_d^2$, and δ_i are given positive real numbers with a lower bound δ_m. The difference between (1) and (2) is that in (1) the "scale" δ depends on the source index j, while in (2) δ depends on the target index i.

Since this calculation requires the application of an $M\times N$ matrix to an N-vector, it would seem to require $O(MN)$ computational work, making it CPU-intensive when M and N are large. Since we begin with $O(N)$ data and get $O(M)$ output values, it is natural to seek an algorithm that requires only $O(N+M)$ work. Fortunately, the matrix has enough structure to make this possible.

This paper presents an algorithm for evaluating (1) and (2) in $O(M+N)$ time, with the constant factor in $O(N+M)$ depending only on the precision desired and the lower bound δ_m of the scales δ_i or δ_j. Our algorithm is an extension of the scheme presented in [5], which treated the special case when $\delta_j=\delta$ are all the same. It combines Taylor or Hermite expansion with scaling and direct evaluation, after organizing either sources or targets into a convenient data structure. The algorithms for source-dependent (1) and target-dependent (2) scales are similar in organization.

Section 2 of the paper reviews the necessary facts about Hermite expansions, § 3 presents the source-dependent scale algorithm, and § 4 treats the target-dependent case. Section 5 presents numerical examples that show our algorithm is much faster than direct evaluation for large-scale problems, achieving a speedup of 2,400 when $N=M=100{,}000$ and the scales are not too small. Section 6 presents our conclusions.

* Received by the editors April 23, 1990; accepted for publication November 27, 1990. This research was supported by Defense Advanced Research Projects Agency/Air Force Office of Scientific Research contract F-49620-87-C-0065 and a National Science Foundation Mathematical Sciences Postdoctoral Research Fellowship.

† Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, New York 10012. Present address, Department of Mathematics, Princeton University, Princeton, New Jersey 08540.

2. Hermite expansions. This section describes the properties of Hermite expansions we will need. Good references for this material are [1], [2], [4] and, particularly, Hille's classical paper [6].

The Hermite polynomials $H_n(t)$ are defined by the Rodrigues formula

$$H_n(t) = (-1)^n e^{t^2} D^n e^{-t^2}, \qquad t \in \mathbf{R},$$

where $D = d/dt$. This implies the generating function

$$e^{2ts-s^2} = \sum_{n=0}^{\infty} \frac{s^n}{n!} H_n(t).$$

Multiplying through by e^{-t^2} gives

$$e^{-(t-s)^2} = \sum_{n=0}^{\infty} \frac{s^n}{n!} h_n(t),$$

where the Hermite functions $h_n(t)$ are defined by

$$h_n(t) = e^{-t^2} H_n(t). \tag{3}$$

(Note that these are not the usual orthonormal Hermite functions; these Hermite functions form a biorthogonal set with Hermite polynomials, rather than themselves forming an orthogonal set.) In practice, we will use a shifted and scaled version of this formula: for $s_0 \in \mathbf{R}$ and $\delta > 0$, we have

$$\begin{aligned} e^{-(t-s)^2/\delta} &= e^{-(t-s_0-(s-s_0))^2/\delta} \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} \left(\frac{s-s_0}{\sqrt{\delta}}\right)^n h_n\left(\frac{t-s_0}{\sqrt{\delta}}\right) \\ &= e^{-(t-s_0)^2/\delta} \sum_{n=0}^{\infty} \frac{1}{n!} \left(\frac{s-s_0}{\sqrt{\delta}}\right)^n H_n\left(\frac{t-s_0}{\sqrt{\delta}}\right). \end{aligned}$$

This formula tells us how to evaluate the Gaussian field $e^{-(t-s)^2/\delta}$ at the *target* t due to the *source* at s, as a Hermite expansion centered at s_0. Thus we can shift a Gaussian centered at s to a sum of Hermite polynomials times a Gaussian, all centered at s_0.

We can also interchange t and s to write

$$e^{-(t-s)^2/\delta} = \sum_{n=0}^{\infty} \frac{1}{n!} h_n\left(\frac{s-t_0}{\sqrt{\delta}}\right)\left(\frac{t-t_0}{\sqrt{\delta}}\right)^n. \tag{4}$$

Looked at this way, the expansion expresses a Gaussian with target t as a Taylor series about a nearby target t_0; the coefficients of the Taylor series are the Hermite functions evaluated at t_0. Thus the same expansion serves as both the near-field (Taylor) and a far-field (Hermite) expansion.

The final one-dimensional results we will need are the recurrence relation

$$h_{n+1}(t) = 2th_n(t) - 2nh_{n-1}(t), \qquad t \in \mathbf{R},$$

for Hermite functions and Cramer's inequality for Hermite polynomials:

$$|H_n(t)| \leqq K 2^{n/2} \sqrt{n!}\, e^{t^2/2},$$

where K is a numerical constant less than 1.09 in value.

In summarizing the extensions of these facts to $d>1$ dimensions, we will find it convenient to adopt multi-index notation. A d-dimensional multi-index $\alpha=(\alpha_1,\alpha_2,\cdots,\alpha_d)$ is a d-tuple of nonnegative integers, playing the role of a multi-dimensional index. For any multi-index α and any $t\in\mathbf{R}^d$, we define

$$|\alpha|=\alpha_1+\alpha_2+\cdots+\alpha_d,$$
$$\alpha!=\alpha_1!\,\alpha_2!\cdots\alpha_d!,$$
$$t^\alpha=t_1^{\alpha_1}t_2^{\alpha_2}\cdots t_d^{\alpha_d},$$
$$D^\alpha=\partial_1^{\alpha_1}\partial_2^{\alpha_2}\cdots\partial_d^{\alpha_d},$$

where ∂_i is differentiation with respect to the ith coordinate in $\mathbf{R}^d$. If p is an integer, we say $\alpha\geqq p$ if $\alpha_i\geqq p$ for $1\leqq i\leqq d$.

The multidimensional Hermite polynomials and Hermite functions are then defined by

$$\begin{aligned}H_\alpha(t)&=H_{\alpha_1}(t_1)\cdots H_{\alpha_d}(t_d),\\ h_\alpha(t)&=e^{-|t|^2}H_\alpha(t)=h_{\alpha_1}(t_1)\cdots h_{\alpha_d}(t_d),\end{aligned}\tag{5}$$

where $|t|^2=t_1^2+\cdots+t_d^2$.

The Hermite expansion of a Gaussian in $\mathbf{R}^d$ is then simply

$$e^{-|t-s|^2}=\sum_{\alpha\geqq 0}\frac{(t-s_0)^\alpha}{\alpha!}h_\alpha(s-s_0).\tag{6}$$

Cramer's inequality implies

$$\frac{1}{\alpha!}|h_\alpha(t)|\leqq K\,e^{-|t|^2/2}2^{|\alpha|/2}\frac{1}{\sqrt{\alpha!}},\tag{7}$$

where K is less than $(1.09)^d$.

We can now present the two key expansions on which our algorithms rely.

First, scale the Hermite expansion (6) by δ_j to get

$$\begin{aligned}e^{-|t-s_j|^2/\delta_j}&=\sum_{\beta\geqq 0}\frac{1}{\beta!}h_\beta\left(\frac{s_j-t_C}{\sqrt{\delta_j}}\right)\left(\frac{t-t_C}{\sqrt{\delta_j}}\right)^\beta\\ &=\sum_{\beta\geqq 0}\frac{1}{\beta!}\left(\frac{t-t_C}{\sqrt{\delta}}\right)^\beta\left(\frac{\delta}{\delta_j}\right)^{|\beta|/2}h_\beta\left(\frac{s_j-t_C}{\sqrt{\delta_j}}\right).\end{aligned}\tag{8}$$

This expresses a Gaussian centered at s_j as a Taylor series about an arbitrary target t_C, with a scaling factor δ independent of j. By Cramer's inequality, the error in truncating the Taylor series (8) after the first p^d terms is bounded by

$$|E_p|\leqq K\sum_{\text{some }\beta_i\geqq p}\frac{1}{\sqrt{\beta!}}2^{|\beta|/2}\left(\frac{|t-t_C|_\infty}{\sqrt{\delta_j}}\right)^{|\beta|}.$$

Here $|t|_\infty=\max_{1\leqq i\leqq d}|t_i|$. Suppose t lies in a box C with center t_C and sides of length $r\sqrt{2\delta}\leqq r\sqrt{2\delta_j}$ parallel to the axes. Then Stirling's formula implies that

$$|E_p|\leqq\frac{r_p^p}{1-r_p}\tag{9}$$

whenever $r_p^2=er^2\delta/\delta_j(p+1)<1$. This error bound decreases very rapidly as p increases. If $r=1$ and $\delta_j=\delta$, then $|E_p|\leqq 10^{-4}$ when $p=12$ and $|E_p|\leqq 10^{-6}$ when $p=17$. Even if

$r=2$, accuracy 10^{-4} is attained with $p=24$ and accuracy 10^{-6} with $p=31$. In practice, this error bound overestimates the error by two or more orders of magnitude.

This error bound is more useful than the (erroneous) bound used in [5], because that bound required $r<1$. The current bound allows us to use $r\geqq 1$ if we use more terms in the Hermite series. This is practically useful because we can cluster the points into larger groups and use longer vectors, which makes the code run much faster on vector processors.

Our second formula is the dual of the first, obtained by scaling with a target-dependent scale δ_i rather than δ_j and shifting to a source center s_B rather than a target center t_C. We find

$$e^{-|t_i-s_j|^2/\delta_i}=\sum_{\beta\geqq 0}\frac{1}{\beta!}\left(\frac{\delta}{\delta_i}\right)^{|\beta|/2}h_\beta\left(\frac{t_i-s_B}{\sqrt{\delta_i}}\right)\left(\frac{s_j-s_B}{\sqrt{\delta}}\right)^\beta. \tag{10}$$

This expresses a Gaussian centered at s_j as a scaled Hermite series centered at a nearby source box center s_B, with coefficients independent of i. If the source s_j lies inside a box of side length $r\sqrt{2\delta_i}$ centered at s_B, then the error in truncating the Hermite series after the first p^d terms is bounded by (9).

These two formulas provide the basis for the two fast algorithms developed in this paper.

3. Source-dependent scales. In this section, we present our algorithm for evaluating

$$Sf(t_i)=\sum_{j=1}^{N} f_j\, e^{-|t_i-s_j|^2/\delta_j} \tag{11}$$

at M targets t_i. We assume (by scaling the δ_j's if necessary) that all the targets lie in the unit box $B_0=[0,1]^d$, for convenience of exposition. The sources s_j may lie anywhere in $\mathbf{R}^d$.

Our algorithm requires $O(M+N)$ work; the constant factor in $O(M+N)$ depends only on the precision ε and the minimum δ_m of the scales δ_j.

Choose cutoffs a and b and split the scales into three groups: $0\leqq\delta\leqq a$, $a<\delta<b$, and $b\leqq\delta$. We treat each group by an appropriate method, and choose a and b to minimize the work required.

First consider $0\leqq\delta\leqq a$. Gaussians with small scales decay rapidly away from the source, so each of these sources can affect only targets that are quite nearby. Hence we treat these sources by direct evaluation, with a data structure constructed as in [11].

To do this, let R be the range of the longest-range Gaussian in this group; thus $e^{-R^2/a}\leqq\varepsilon$ and $R=\sqrt{-a\log\varepsilon}$. Divide $[0,1]^d$ into n^d boxes of side length greater than or equal to R, so $n=\lfloor 1/R\rfloor$. Sort the targets t_i into boxes by location. Now run through the sources s_j. Put s_j into a box, B, say, by location, and let $k_j=\lfloor\sqrt{-\delta_j/a}\rfloor$. Then s_j affects boxes with indices within k_j of B, to accuracy ε. Run through these boxes, incrementing $Sf(t_i)$ at each target t_i in each box. This concludes the treatment of sources with $0\leqq\delta_j\leqq a$.

Now consider the large scales, with $b\leqq\delta_j<\infty$. These Gaussians have very long ranges, so direct evaluation is very expensive. For the same reason, however, they are very smooth. If b is chosen large enough, all these sources can be incorporated into a single p^d-term truncated Taylor expansion about the box center $t_0=(\frac{1}{2},\cdots,\frac{1}{2})$ of $[0,1]^d$ with error less than $F\varepsilon$ where $F=\sum|f_j|$. Indeed, (8) implies that

$$\sum_{\delta_j\geqq b} f_j\, e^{-|t-s_j|^2/\delta_j}=\sum_{\beta\leqq p} C_\beta\left(\frac{t-t_0}{\sqrt{\delta}}\right)^\beta+E_p,$$

where

$$C_\beta = \frac{1}{\beta!} \sum_{\delta_j \geqq b} \left(\frac{\delta}{\delta_j}\right)^{|\beta|/2} h_\beta\left(\frac{s_j - t_0}{\sqrt{\delta_j}}\right)$$

and

$$E_p \leqq F \frac{r_p^p}{1 - r_p}, \qquad r_p = \sqrt{\frac{e}{p+1}}\, r < 1.$$

Here $r = 1/\sqrt{2\delta} \leqq 1/\sqrt{2b}$ and $F = \sum |f_j|$.

Thus the large variances are very easy to deal with; we simply form p^d coefficients C_β at a cost of $O(p^d N)$ work and evaluate the Taylor series at M targets, with a total cost of $O(p^d(M+N))$. The coefficient p^d depends only on the desired precision ε.

Finally, we must deal with the intermediate sources with $a < \delta_j < b$. We treat these by a combination of the previous two methods. The targets are sorted into boxes and each source contributes to a Taylor expansion about the center of each box within range. When each Taylor expansion is complete, it is evaluated at each target in the box.

We begin by sorting the targets into boxes C, say, with centers t_C and sides of lengths $r\sqrt{2\delta}$ parallel to the axes. Here $\delta = \min_{a \leqq \delta_j \leqq b} \delta_j \geqq a$. Now each source is converted into a Taylor expansion (8) about the center t_C of each target box C within range $R_j = \sqrt{-\delta_j \log \varepsilon}$ of s_j. Thus we have, for $t \in C$,

$$\sum_{|s_j - t_C| \leqq R_j} f_j e^{-|t-s_j|^2/\delta_j} = \sum_{\beta \leqq p} C_\beta \left(\frac{t - t_C}{\sqrt{\delta}}\right)^\beta + E_p,$$

where

$$C_\beta = \frac{1}{\beta!} \sum_{\delta_j \geqq b} \left(\frac{\delta}{\delta_j}\right)^{|\beta|/2} h_\beta\left(\frac{s_j - t_C}{\sqrt{\delta_j}}\right)$$

and E_p satisfies the error bound

$$E_p \leqq F \frac{r_p^p}{1 - r_p}$$

with $r_p = (e/(p+1))^{1/2} r < 1$. We choose r and p to make $|E_p| \leqq F\varepsilon$. After each source has contributed to all Taylor expansions in range, each Taylor expansion is evaluated at the targets t_i in its box.

Two slight refinements of this procedure reduce the overhead considerably in practice. First, it is inefficient to form a Taylor series and evaluate it for a box containing only a few targets. Thus a breakpoint M_T is introduced and boxes with more than M_T targets receive Taylor expansions, while direct evaluation is more efficient for boxes with fewer than M_T targets. Typically, $M_T = 8$ or 10 is a reasonable value when single precision accuracy is desired. The right value of M_T can be determined by the program when it is run, in a reasonably straightforward way.

The second refinement is introduced in order to avoid a needlessly accurate computation. If a scaling factor $\delta = \min \delta_j$ is used to determine the box size, then many sources will have δ_j much larger than δ. For these sources, p^d terms is far too many to provide the desired accuracy, and p should be reduced. Hence we can compute how many terms are necessary for each source independently, and increment only those coefficients. Thus we compute q^d terms where q is chosen to satisfy the error estimate

$$\frac{r_{qj}^q}{1 - r_{qj}} \leqq \varepsilon$$

with $r_{qj} = r\sqrt{e\delta/\delta_j(q+1)}$. The small overhead required to compute q for each source is compensated for by the reduction in effort required to evaluate Hermite coefficients. In practice, this refinement often produces a factor of two speedup in the calculation.

A third refinement is possible; we can subdivide the center interval (a, b) further, in order to avoid treating longer-range smoother Gaussians on wastefully small scales. A geometric subdivision as in [12] is possible, or some other problem-dependent subdivision may be more appropriate.

The algorithm is completed by the choice of the cutoffs a and b. If the scales δ_j are distributed in a fairly uniform way over the interval $[\delta_m, \delta_M]$, then we can take $a = \delta_m + O(1/N)$ and b fixed, say, $b = 1/10$. The choice of a is made because we can afford to have only a fixed number of sources below a. The choice of $b = 1/10$ is about the smallest for which we can make the Hermite series accurate with a practical number of terms, like twenty or thirty. The choice of a is more important when there are a few outlying scales δ_j, with all the rest lying considerably higher. Then we should choose a to be the essential minimum of the δ_j's, in other words the smallest lower bound for all but a fixed number of the scales as $N \to \infty$. This can substantially improve the performance of the medium scales, because we can take larger boxes, and the cost of evaluating a few sources directly is then worth it.

4. Target-dependent scales. In this section, we describe how to evaluate

$$Tf(t_i) = \sum_{j=1}^{N} f_j e^{-|t_i - s_j|^2/\delta_i}$$

at M points $t_i \in \mathbf{R}^d$, under the assumption (for convenience) that all the sources s_j lie in $[0, 1]^d$. The algorithm is structurally dual to the algorithm of § 3, so we concentrate on the differences.

We divide the δ_i's into three sets in the same way as before, and treat each set of *targets* as a group.

For targets with $0 \leqq \delta_i \leqq a$, we sort the sources into boxes rather than the targets, and apply direct evaluation.

For the middle interval $a < \delta_i < b$, we use the Hermite expansion (10) with $\delta = \min_{a \leqq \delta_i \leqq b} \delta_i \geqq a$. We impose a box structure on the sources and collect all the sources in each box B (with center s_B and sides of length $r\sqrt{2\delta}$ parallel to the axes) into a scaled Hermite expansion about the box center, valid everywhere in $[0, 1]^d$:

$$T_B f(t_i) = \sum_{s_j \in B} f_j e^{-|t_i - s_j|^2/\delta_i} = \sum_{\beta \leqq p} B_\beta \left(\frac{\delta}{\delta_i}\right)^{|\beta|/2} h_\beta\left(\frac{t_i - s_B}{\sqrt{\delta_i}}\right)^\beta + E_p,$$

where

$$B_\beta = \frac{1}{\beta!} \sum_{s_j \in B} f_j \left(\frac{s_j - s_B}{\sqrt{\delta}}\right)^\beta$$

and we make $E_p \leqq F\varepsilon$ by the choice of r and p. Then we evaluate a fixed number of Hermite expansions at each target t_i;

$$Tf(t_i) = \sum_{B \text{ in range of } t_i} T_B f(t_i).$$

Some overhead can be avoided if we form Hermite series only for boxes with more than, say, N_S sources; boxes with fewer than N_S sources are handled by direction evaluation. Also, targets with δ_i much larger than δ need only evaluate a smaller Hermite series, since the error decays faster for them.

Finally, the sources with $\delta_i \geqq b$ are handled by forming all of them into a single Hermite expansion about the center of $[0, 1]^d$ and evaluating it at each target.

The parameters a, b, and N_S are chosen in a way very like the source-dependent algorithm.

5. Numerical results. In this section, we present some numerical results showing the speedups achievable by our algorithms. We tested the source-dependent algorithm on sets of $N = M$ sources and targets; the target-dependent algorithm will exhibit highly similar statistics. We used strengths randomly generated and distributed uniformly on $[0, 1]$, with source and target points randomly generated and uniformly distributed either on $[0, 1]^d$ or on a circle. The scales were randomly generated on $[\delta_m, \delta_M]$ for various choices of δ_m and δ_M.

Tables 1–4 present results for the source-dependent scale fast Gauss transform for four examples. The time taken by the fast algorithm is recorded in the column headed T_f, the time required to evaluate the Gaussian sum directly under T_d, and the error produced by the fast algorithm—relative to $F = \sum |f_j|$—under the heading E_p/F. All computations were carried out in double precision arithmetic with the error tolerance ε set to 10^{-4}. (This was enough to achieve an actual error well below 10^{-6}

TABLE 1

Times and errors for $0.1 \leqq \delta_j \leqq 1.0$, *with targets and sources distributed uniformly in the unit box.*

Case	$N = M$	T_f	T_d	E_p/F
1	100	0.05	0.10	0.222e−06
2	200	0.09	0.40	0.163e−06
3	400	0.18	1.60	0.551e−07
4	800	0.36	6.80	0.110e−07
5	1,600	0.76	25.60	0.449e−07
6	3,200	1.45	99.20	0.128e−07
7	6,400	2.88	406.40	0.186e−07
8	12,800	5.75	1,638.40	0.957e−08
9	25,600	11.45	6,553.60	0.544e−08
10	51,200	22.96	26,060.80	0.114e−07
11	102,400	45.94	104,499.20	0.836e−08

TABLE 2

Times and errors for $0.1 \leqq \delta_j \leqq 1.0$, *with targets and sources distributed uniformly on a circle.*

Case	$N = M$	T_f	T_d	E_p/F
1	100	0.060	0.100	0.115e−06
2	200	0.100	0.400	0.553e−06
3	400	0.160	1.600	0.439e−06
4	800	0.370	6.400	0.403e−06
5	1,600	0.710	24.000	0.352e−06
6	3,200	1.430	102.400	0.335e−06
7	6,400	2.830	409.600	0.181e−06
8	12,800	5.690	1,632.000	0.255e−06
9	25,600	11.380	6,528.000	0.226e−06
10	51,200	22.620	26, 214.400	0.155e−06
11	102,400	45.430	104,192.000	0.254e−06

TABLE 3

Times and errors for $0.01 \leqq \delta_j \leqq 1.0$, *with targets and sources distributed uniformly on a circle.*

Case	$N = M$	T_f	T_d	E_p/F
1	100	0.060	0.100	0.145e−06
2	200	0.120	0.400	0.564e−06
3	400	0.240	1.600	0.420e−06
4	800	0.450	6.400	0.252e−06
5	1,600	0.900	25.600	0.311e−06
6	3,200	1.810	102.400	0.275e−06
7	6,400	3.600	400.000	0.170e−06
8	12,800	7.280	1,657.600	0.238e−06
9	25,600	14.540	6,502.400	0.201e−06
10	51,200	29.030	26,291.200	0.144e−06
11	102,400	58.860	104,499.200	0.236e−06

TABLE 4

Times and errors for $0.002 \leqq \delta_j \leqq 0.2$, *with targets and sources distributed uniformly on a circle.*

Case	$N = M$	T_f	T_d	E_p/F
1	100	0.140000	0.100	0.112e−07
2	200	0.360000	0.300	0.180e−06
3	400	0.730000	1.400	0.123e−06
4	800	1.550000	6.000	0.110e−06
5	1,600	3.180000	25.600	0.623e−07
6	3,200	6.590000	102.400	0.683e−07
7	6,400	13.220000	409.600	0.460e−07
8	12,800	26.880000	1,644.800	0.936e−07
9	25,600	54.150000	6,720.000	0.634e−07
10	51,200	107.850000	26,470.400	0.266e−07
11	102,400	216.640000	108,083.200	0.720e−07

in every case.) This required $p = 10$ with $r = 1$ for sources with $a \leqq \delta_j \leqq b$ and $p = 24$ for $= \delta_j > b$. In Table 1, we used source and target points uniformly distributed on $[0, 1]^d$ and scales δ_j uniformly distributed between 0.1 and 1.0. We took $a = 0$, $b = 0.125$, and $M_T = 8$. The results show a beautifully linear growth of computation time with N for the fast algorithm, against a quadratic time for direct evaluation. A speedup of 2,400 is obtained with $N = 102{,}400$, indicating a projected breakeven point of about 45. The error estimate is seen to be quite sharp, due to the second refinement mentioned in § 3.

In Table 2, we used points randomly chosen on a circle with radius 0.29 and center (0.6, 0.7), with the same parameters as in Table 1. The nonuniformity of the sources and targets did not slow down the algorithm; in fact, it seems to speed it up slightly. This is because fewer boxes are nonempty, so fewer Taylor expansions need to be constructed.

Table 3 presents times and errors for the same calculation as in Table 2, but with the minimum scale reduced to 0.01. This slows down the algorithm only slightly; the speedup goes down to 1,900 with 102,400 points.

Table 4 is similar to Table 3, but with δ_j's lying between 0.002 and 0.2. Even with these fairly small scales, we still achieve a respectable speedup of 500 with $N = 102{,}400$.

6. Conclusions. We have presented fast algorithms that evaluate d-dimensional Gaussian convolution sums (1) and (2) with scales dependent either on the source or the target. Such sums are frequently evaluated in statistical computations [3], [10] and also occur in numerical solution of the heat equation [9], [8] and other applications [7]. Our algorithms evaluate these sums in $O(M+N)$ work, with a constant factor depending on the precision required and the minimum scale that occurs.

Numerical results indicate that our method can speed up large calculations by several orders of magnitude; in one example, the computation time was reduced to a minute instead of the 1.2 *days* required by direct evaluation. Thus this algorithm can be extremely useful in reducing the time required to carry out large-scale computations. We are currently applying a variant of this algorithm to problems in crystal growth, where it is used to evaluate free-space heat potentials.

REFERENCES

[1] H. DYM AND H. P. MCKEAN, *Fourier Series and Integrals*, Academic Press, San Diego, CA, 1972.

[2] A. ERDELYI, W. MAGNUS, F. OBERHETTINGER, AND F. G. TRICOMI, *Higher Transcendental Functions*, Vol. II, McGraw-Hill, New York, 1953.

[3] S. GEMAN AND C. HWANG, *Nonparametric maximum likelihood estimation by the method of sieves*, Ann. Statist., 10 (1982), pp. 401–414.

[4] I. S. GRADSHTEYN AND I. M. RYZHIK, *Tables of Integrals, Series, and Products*, Academic Press, New York, 1980.

[5] L. GREENGARD AND J. STRAIN, *The fast Gauss transform*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 79–94.

[6] E. HILLE, *A class of reciprocal functions*, Ann. Math., 27 (1926), pp. 427–464.

[7] N. LERNER, *Wick-Wigner Functions and Tomographic Methods*, preprint, 1989.

[8] A. MCINTYRE, *Boundary integral solutions of the heat equation*, Math. Comp., 46 (1986), pp. 71–79.

[9] P. J. NOON, *The single layer heat potential and Galerkin boundary element methods for the heat equation*, Ph.D. thesis, University of Maryland, College Park, MD, 1988.

[10] B. W. SILVERMAN, *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, London, 1986.

[11] J. STRAIN, *Fast Potential Theory* II: *Layer Potentials and Discrete Sums*, Report PAM-567, Center for Pure and Applied Mathematics, University of California, Berkeley, CA, April 1990; J. Comput. Phys., submitted.

[12] ———, *A fast Laplace transform based on Laguerre functions*, Math. Comp., to appear.

SIAM J. SCI. STAT. COMPUT.
Vol 12, No. 5, pp. 1140-1161, September 1991
© 1991 Society for Industrial and Applied Mathematics
008

COMPARING AVERAGED-OUT UTILITIES OF PROBABILITY TREES HAVING RANDOM PARAMETERS*

R. D. EISENHUT†, R. S. DITTUS‡, S. D. ROBERTS§, AND J. R. WILSON¶

Abstract. This paper develops a procedure to approximate the marginal distributions of each of the following performance measures defined on a forest of probability trees with random parameters: (a) the averaged-out utility of each individual tree in the forest, and (b) the difference between the averaged-out utilities of each pair of trees in the forest. The parameters of a probability tree are its branching probabilities and node utilities, and they may be specified as mutually independent random variables with given distributions; moreover, any of these parameters may be duplicated at several places in the forest to represent parameter dependencies within and between trees. The approximation procedure incorporates an efficient, numerically robust algorithm for computing up to the first four moments of the selected performance measure. The desired distribution is approximated by an Edgeworth expansion based on the computed moments. With respect to accuracy and computational complexity, the approximation procedure is compared to estimation techniques based on simulation and normal distribution theory. To illustrate how the analysis of probability trees can directly incorporate uncertainty or random variation in the values of tree parameters, the approximation procedure is applied to the comparison of alternative medical protocols for managing patients following myocardial infarction (heart attack). In addition to its applications in medical decision making, the procedure developed in this paper should also have applications in risk analysis, reliability, project planning, marketing, and other fields in which probability trees are used to model various processes of interest.

Key words. decision analysis, distribution function estimation, Monte Carlo, probability trees, sensitivity analysis

AMS(MOS) subject classifications. 65D15, 65U05, 90B50

1. Introduction.

Probability trees can model a wide variety of processes or operations. Nodes in a probability tree represent milestones in the associated process, and the branches emanating from each node are assigned branching probabilities to represent the stochastic behavior of the process. As the process evolves, a utility is accumulated at each node that is reached. The term *utility* is used in this paper to refer to a single performance measure defined on the process—for example, this measure could be a cost, a profit, a time period, a quality-adjusted year of life, a mortality rate, or some dimensionless unit. Typical applications of probability trees include risk analysis, project planning, reliability, marketing, and decision analysis. In this paper we feature applications to medical decision making.

Averaging out a probability tree is the traditional method for analyzing the associated process (Raiffa [17]). The averaged-out or expected utility of a probability tree is computed by an iterative procedure that moves from the leaves of the tree to the root and successively replaces each node and its descendants with the expected utility accumulated for that node and its descendants. If several processes are to be compared, then the best alternative is considered to be the process whose corresponding probability tree has the most favorable (usually the largest) averaged-out utility.

* Received by the editors February 6, 1989; accepted for publication (in revised form) September 7, 1990. This work was partially supported by a grant from the American College of Cardiology and by National Science Foundation grant DMS-8717799.

† Pritsker Corporation, P.O. Box 2413, West Lafayette, Indiana 47906.

‡ Indiana University School of Medicine, Regenstrief Institute for Health Care, 1001 West Tenth Street, Indianapolis, Indiana 46202.

§ Department of Industrial Engineering, North Carolina State University, P.O. Box 7906, Raleigh, North Carolina 27695.

¶ School of Industrial Engineering, Purdue University, West Lafayette, Indiana 47907.

The branching probabilities and node utilities constitute the parameters of a probability tree, and there are two reasons these parameters might not be treated as constants. First, the values of some parameters of the associated process might actually vary according to a random mechanism. Second, the modeler's knowledge of some parameters might be uncertain because authoritative sources differ in the specification of those parameters.

Sensitivity analysis is a conventional approach for dealing with questionable parameters in a probability tree by attempting to characterize the way in which the tree's averaged-out utility depends on those parameters. Each questionable parameter must be varied individually over its relevant range of values. If several parameters are in question, then they must also be varied simultaneously and systematically over their corresponding ranges. In practice, whenever more than two parameters are varied simultaneously, the sensitivity analysis becomes unwieldy and difficult to interpret. Moreover, sensitivity analysis inadequately accounts for the following significant factors: (a) the likelihoods (relative frequencies) associated with different values of parameters that are subject to random variation; and (b) the degree of uncertainty associated with parameters that are subjectively specified by knowledgeable individuals.

A more natural and more powerful approach for handling questionable parameters in a probability tree is to postulate probability distributions for each of these parameters (Doubilet et al. [10], Critchfield and Willard [6], Willard and Critchfield [22]). The result is that the averaged-out utility of the tree becomes a random variable. However, there is another potential complication that must be considered when distributions are specified for the parameters of a probability tree.

If any random parameter is incorporated into a forest of probability trees at more than one location—that is, if the parameter is *duplicated* at several places in the forest—then the same value of that parameter must be used at each of the designated locations in the forest, as opposed to taking independent replications of that random variable at each of those locations. More generally, we can regard duplication as one method for representing stochastic dependencies among the parameters of a forest of probability trees (see Doubilet et al. [10]). Now if all of the random tree parameters are mutually independent, then the usual iterative procedure described in the second paragraph of this section can be adapted to compute the variance and higher moments of the averaged-out utility as well as its expected value. However, if some of the random tree parameters are either duplicated or otherwise interdependent, then this iterative procedure can only be used to calculate the expected value of the averaged-out utility. In many applications we have observed that a set of random parameters associated with a particular subtree must be duplicated at several places in the overall tree; and in the pairwise comparison of trees, we have often found that some set of random parameters must appear in both trees. For example, in § 4 we present an actual medical decision process that has 398 nodes. Although this tree contains 104 nodes with nonzero costs, there are only five distinct random costs that can occur in the associated medical protocol. Such extensive parameter duplication appears to be common in medical processes and possibly in other application areas as well.

In the analysis of probability trees with random parameters, the fundamental problem is an accurate estimation of the cumulative distribution function for (a) the averaged-out utility of a single tree, and (b) the difference between the averaged-out utilities of a pair of trees. One approach to approximating each distribution involves generating repeated realizations of the relevant trees by simulation. Generating a realization of a single tree entails sampling each random parameter so that the same value of each duplicated parameter is taken throughout the tree; then the tree's averaged-out utility is computed (Pass and Goldstein [16]; Doubilet et al. [10];

Critchfield and Willard [6]; Critchfield, Willard, and Connelly [7]). When two trees are compared, a realization of each tree must be generated in such a way that the same value of each duplicated parameter is taken throughout both trees; then the difference between the averaged-out utilities of the two trees is computed. Although in each case independent replications of the simulation will yield an unbiased estimator of the target distribution function, this approach can require an excessive number of replications (with corresponding data-storage requirements) to achieve acceptable accuracy. A second approach employs normal distribution theory. Willard and Critchfield [22] adapted a general-purpose algebraic manipulation algorithm to compute the mean and variance of the relevant averaged-out utility so that they could use a normal approximation for the desired distribution. Unfortunately, the normal fit may be inadequate, particularly in the distribution tails; and even for trees of moderate size, a general algebraic manipulation procedure can require substantially greater computer resources (namely, execution time and memory) to obtain higher moments of the averaged-out utility than a procedure that has been specifically designed to exploit the structure of probability trees. As an alternative method for implementing the normal approximation, Katz and Hui [14] derived an easily computed estimator for the variance of the averaged-out utility by applying the delta method to a truncated Taylor series expansion for the averaged-out utility. No analytical or empirical evidence has been given to quantify the accuracy of this variance estimator or the accuracy of the corresponding normal approximation to the target distribution.

The objective of this paper is to present a practical computational procedure for accurately approximating the distribution of the averaged-out utility (or the difference between averaged-out utilities) for a probability tree (or pair of trees) in which some of the branching probabilities and the node utilities may be independent random variables with given probability distributions; moreover, some of these random parameters may be duplicated at several locations in the tree(s). The procedure incorporates an efficient, numerically robust algorithm for computing up to the first four moments of the selected performance measure. An Edgeworth expansion based on the computed moments is used to approximate the target distribution function. We compare the accuracy and speed of this procedure to that of estimation techniques based on simulation and normal distribution theory. To illustrate the usefulness of the procedure, we apply it to the analysis of medical protocols for treatment of post-myocardial infarction (heart attack) patients.

This paper is organized as follows. In § 2 we formulate the distribution estimation problem for a single tree and for the comparison of two trees, and we discuss the assumptions underlying our solution to these problems. We present the details of our approximation procedure in § 3, and in this section we also evaluate the asymptotic time complexity of the approximation procedure. In § 4 we describe an application of this methodology to a problem in medical decision making. Finally, in § 5 we summarize the conclusions of this work and we make recommendations for further research.

2. Problem formulation, assumptions, and notation.

2.1. Formulation of the distribution estimation problem for a single tree. Figure 1 depicts a probability tree that has nine nodes. For presentation, the nodes are numbered in a "breadth-first" fashion. Nodes 5, 6, 7, 8, and 9 are the *leaves* while node 1 is the *root* of the tree. The *height* of this tree is three since the longest path from the root to a leaf consists of three branches.

In general, a probability tree having the node index-set N is completely described by its *branching probabilities* $\{B_i: i \in N\}$ and *node utilities* $\{U_i: i \in N\}$. For the tree

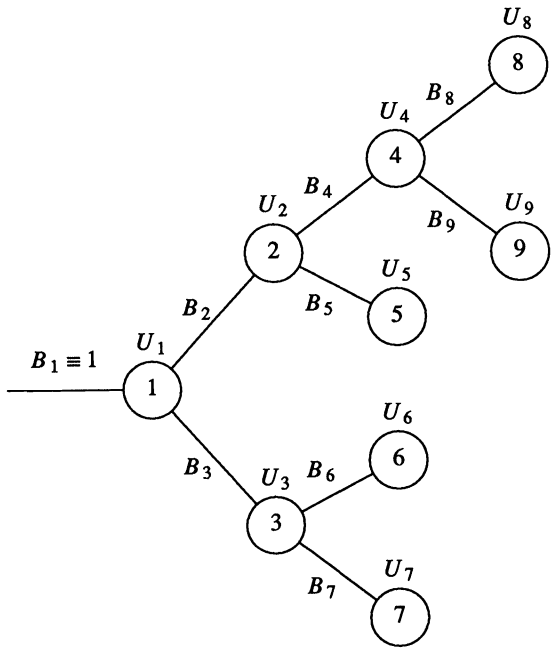


FIG. 1. *Probability tree* 1.

displayed in Fig. 1, we have $N = \{1, 2, \cdots, 9\}$. We assume that the root is always reached and that subsequent nodes are reached according to the following conditional probabilities:

$$B_i \equiv \begin{cases} 1, & \text{if node } i \text{ is the root,} \\ \Pr\{\text{node } i \text{ reached} \mid \text{parent of node } i \text{ reached}\} & \text{otherwise.} \end{cases}$$

The process represented by the tree accumulates the additional utility U_i if node i is reached. For each node $i \in N$, we define the following index set:

$$Q(i) \equiv \{j\text{: node } j \text{ is on the path from the root to node } i\}.$$

Thus the *partial path probability* to node i is the product of the branching probabilities for each node on the path from the root to node i:

$$P_i \equiv \prod_{j \in Q(i)} B_j \quad \text{for every } i \in N. \tag{1}$$

Since P_i is the probability that the utility U_i is accumulated, the averaged-out utility of the tree is

$$A \equiv \sum_{i \in N} P_i U_i. \tag{2}$$

If the tree has random parameters, then some of the $\{B_i, U_i : i \in N\}$ are described by probability distributions; and in this case A is also a random variable. We seek to estimate the cumulative distribution function (CDF) of A:

$$F_A(a) \equiv \Pr\{A \leqq a\}, \qquad -\infty < a < \infty.$$

We restrict attention to probability trees that satisfy the assumptions discussed in § 2.3. The procedure to estimate F_A for a single tree is detailed in § 3.

2.2. Formulation of the distribution estimation problem for comparing two trees. Figure 2 depicts a second probability tree whose averaged-out utility is to be compared to that of the tree shown in Fig. 1. The scheme for labeling the branches and nodes of the second tree is a continuation of the scheme used in the first tree, except that the original node utilities in the second tree are denoted by an asterisk to clarify subsequent operations on these quantities. Thus equations (1) and (2) also apply to the second tree when we take $N = \{10, 11, \cdots, 14\}$ and we replace U_i with U_i^* for all $i \in N$.

Conceptually we can combine trees 1 and 2 to estimate the distribution of the difference between their averaged-out utilities as follows: (a) we define new utilities for the second tree by reversing the sign of each of the original utilities in this tree; (b) we connect the roots of both trees to a new "superroot" (say, node 0) having utility zero to form a combined tree; and (c) we estimate the distribution of the averaged-out utility of the combined tree using the Edgeworth approximation developed for the analysis of a single tree as described in § 3. Figure 3 depicts the result of combining trees 1 and 2. Strictly speaking, we note that a supertree constructed in this way is not a proper probability tree since the weights on the two branches emanating from the superroot do not sum to one. We could avoid this anomaly by assigning the weight 0.5 to each of the two branches emanating from the superroot and by doubling the utilities assigned to each node in the supertree; however, this modification would incur substantial computational overhead without altering the final results in any way. Although it is technically a misnomer to refer to a construct like the one in Fig. 3 as a probability tree, no confusion can result from this usage. Thus equations (1) and (2) apply to the supertree in Fig. 3 when we take $N = \{0, 1, \cdots, 14\}$; and we see that, in general, the procedure for comparing two probability trees is identical to the procedure for analyzing a single probability tree except that the random variable A for a combined tree represents the difference between the averaged-out utilities of the compared trees. It should be understood that in all subsequent references to a probability tree, the tree may be a model of a single process or it may be a model of the difference between two processes.

2.3. Assumptions underlying the analysis. The following properties are assumed to hold throughout the rest of this paper:

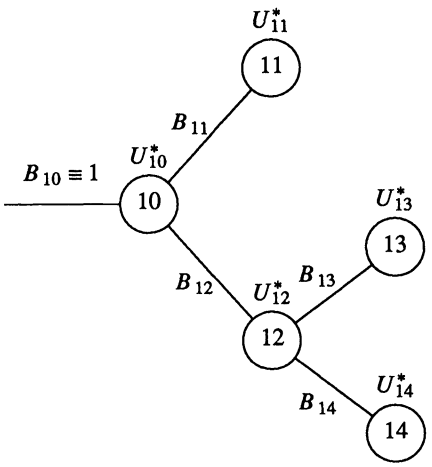


FIG. 2. *Probability tree* 2.

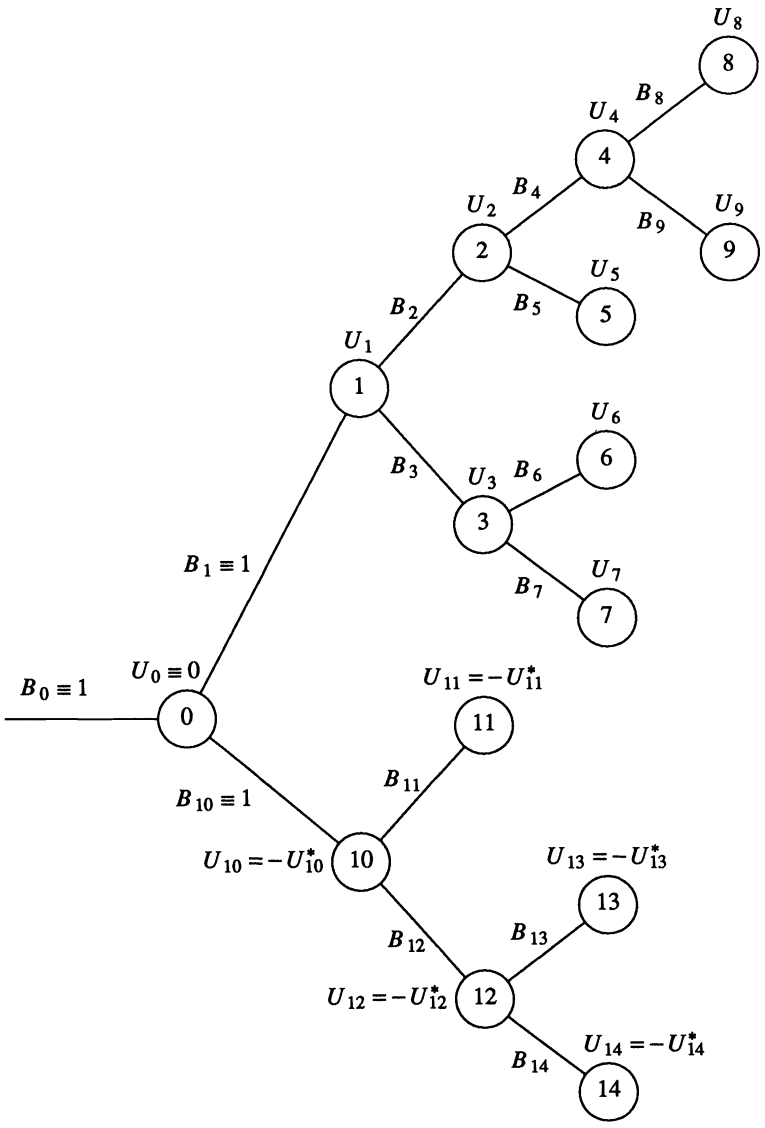


FIG. 3. *Supertree representing the difference between tree* 1 *and tree* 2.

(1) *The target probability tree is binary.* Each node in the tree is assumed to have at most two children. If B_i is the conditional probability of reaching one child given that the parent has been reached, then $1 - B_i$ is the conditional probability of reaching the other child. Now any nonbinary probability tree can be converted into an equivalent binary probability tree by replacing each p-way node with an equivalent sequence of $p - 1$ two-way nodes (see Doubilet et al. [10]). For example, nodes 2, 6, and 7 in Fig. 3 originally may have been on a three-way branch from node 1 and then converted into a sequence of binary nodes by adding node 3. This conversion process can be cumbersome if there are many nonbinary nodes or if some nonbinary nodes have many branches. Conversion is further complicated because specification of the branching probability B_i for each node i in the equivalent binary tree must be conditioned on the failure to reach the sibling of that node in the original nonbinary tree. In the

example mentioned above, the probabilities of reaching nodes 6 and 7 must be conditioned on the failure to reach node 2.

(2) *Distinct (nonduplicated) random tree parameters are mutually independent.* Any subset of the random tree parameters $\{B_i, U_i: i \in N\}$ that does not contain duplicates must consist of mutually independent random variates. In particular, this means that the branching probabilities $\{B_i\}$ must be independent of the node utilities $\{U_i\}$. However, the possibility of duplicate random parameters within the tree allows for dependencies within the sets $\{B_i\}$ and $\{U_i\}$. Moreover, when two trees are being compared, random parameters may be duplicated across the two trees as well as within them.

(3) *All random tree parameters are bounded, and at least one is continuous.* The branching probabilities $\{B_i\}$ are clearly bounded. The node utilities $\{U_i\}$ are bounded when they are conventional utilities or when they represent physical outcomes like cost. Since equations (1) and (2) show that A is a continuous function of the tree parameters, we see that A is also bounded. In this situation A has finite moments $\mathrm{E}(A^w)$, $w = 1, 2, \cdots$; and the target distribution function F_A is uniquely determined by these moments (see Stuart and Ord [20, § 4.22(c)]). By also requiring that at least one of the parameters of the tree is a continuous random variable, we ensure that F_A has an absolutely continuous component. As elaborated in § 3.1 below, F_A has a convergent Edgeworth expansion in the moments of A when F_A has bounded support and an absolutely continuous component.

(4) *A truncated Edgeworth expansion can approximate F_A adequately.* An Edgeworth expansion that is used to approximate F_A and that has been truncated after y terms is uniquely determined by the moments $\mathrm{E}(A^w)$, $w = 1, 2, \cdots, y+1$ (see § 3.1). Specifically, we assume F_A can be adequately approximated by an Edgeworth expansion that has been truncated after two or three terms, hence the approximation can be based on the mean, variance, skewness, and excess kurtosis of A. This is less restrictive than the normality assumption, which presumes that the skewness and excess kurtosis of A must be zero. In § 4 we present empirical evidence to justify the approximation of F_A with a truncated Edgeworth expansion.

(5) *Each compared process handles the same population of transactions (customers).* Given a realization of each random parameter in a probability tree, we can regard the averaged-out utility of the tree as the mean "reward" accumulated by the relevant population of transactions (customers, clients, patients) as they advance through the process or operation represented by the tree. If two or more probability trees are to be compared, then in principle the same population of transactions must be acted upon by each of the alternative processes represented by a tree. This assumption forms the basis for the procedure of § 2.2 in which the comparison of two trees is reduced to the analysis of a combined tree incorporating both of the original trees as subtrees with appropriately signed node utilities.

2.4. Notation. In this section we summarize the statistical notation that is used throughout the rest of the paper. We let $\phi(z)$ and $\Phi(z)$, respectively, denote the standard normal density and distribution function

$$\phi(z) \equiv (2\pi)^{-1/2} \exp\left(-\frac{1}{2} z^2\right) \quad \text{and} \quad \Phi(z) \equiv \int_{-\infty}^{z} \phi(\zeta)\, d\zeta \quad \text{for } -\infty < z < \infty;$$

moreover, we let $\phi^{(w)}(z)$ and $\Phi^{(w)}(z)$ denote the corresponding wth derivatives for $w = 0, 1, 2, \cdots$. The wth Hermite polynomial $H_w(z)$ is defined by the relation

$$\phi^{(w)}(z) \equiv (-1)^w H_w(z) \phi(z), \qquad w = 0, 1, 2, \cdots, \tag{3}$$

so that we have

$$H_0(z)=1,\quad H_1(z)=z,\quad H_2(z)=z^2-1,\quad H_3(z)=z^3-3z,$$
$$H_4(z)=z^4-6z^2+3 \quad \text{and} \quad H_5(z)=z^5-10z^3+15z. \tag{4}$$

In terms of the mean and variance of A:

$$\mu_A \equiv \mathrm{E}(A) \quad \text{and} \quad \sigma_A^2 \equiv \mathrm{E}[(A-\mu_A)^2],$$

the normal approximation to F_A is given by

$$\hat{F}_{A1}(a)=\Phi[(a-\mu_A)/\sigma_A], \qquad -\infty<a<\infty. \tag{5}$$

The wth central and noncentral moments of A are, respectively, defined by

$$\mu_{Aw} \equiv \mathrm{E}[(A-\mu_A)^w] \quad \text{and} \quad \mu'_{Aw} \equiv \mathrm{E}(A^w), \quad w=2,3,\cdots.$$

Thus the skewness and excess kurtosis of A are, respectively, given by

$$\gamma_{A1} \equiv \mathrm{E}[(A-\mu_A)^3/\sigma_A^3] \quad \text{and} \quad \gamma_{A2} \equiv \mathrm{E}[(A-\mu_A)^4/\sigma_A^4]-3.$$

3. The approximation procedure.

3.1. The Edgeworth expansion for F_A. Since, in general, the averaged-out utility A is a weighted sum of random utilities $\{U_i\}$ with stochastically interdependent weights $\{P_i\}$, the normal approximation to the distribution function F_A must ultimately be based on a version of the central limit theorem for dependent variates. (For example, see Chung [3, Thm. 7.3.1].) At best this approach will yield an estimation error $\hat{F}_{A1}(a)-F_A(a)$ that approaches zero as ν, the number of nodes in the tree, tends to infinity. Of course, in an actual application, ν is fixed so that the normal approximation has a set of fixed estimation errors for different values of a; and in practice we have found that these errors can be unacceptably large—particularly in the tails of F_A. Since the normal approximation uses only the first two moments of A, it is natural to seek an improved approximation to F_A that is based on the higher moments of A. We chose to use a series expansion because this appears to be the simplest extension of the normal approximation that makes direct use of the higher moments of A in a closed-form approximation to F_A. Compared to other series expansions and to standard distribution families, the Edgeworth expansion has been found to yield good fits to the empirical CDF for a wide variety of data sets (Cramér [4], Elderton and Johnson [13]). These considerations motivated the approximation method that we used in this work.

Since the random variable A is bounded and has an absolutely continuous component (see assumption (3) of § 2.3), it follows that the distribution function F_A has a convergent Edgeworth expansion of the form

$$F_A(a)=\Phi[(a-\mu_A)/\sigma_A]+\sum_{w=1}^{\infty} G_w(\Phi) \quad \text{for } -\infty<a<\infty, \tag{6}$$

where $G_w(\Phi)$ is a polynomial of degree $3w$ in Φ such that: (a) the coefficients of $G_w(\Phi)$ are determined by the "standardized" central moments μ_{Ay}/σ_A^y, $y=3,\cdots,w+2$; and (b) each factor of the form Φ^y is interpreted as the yth derivative $\Phi^{(y)}[(a-\mu_A)/\sigma_A]$. (Actually, the expansion (6) is convergent under more general conditions than are assumed in this paper; see Wallace [21, pp. 638–639] or Cramér

[5, pp. 223-224] for an elaboration of these conditions.) In particular, we have

$$G_1(\Phi) = -\frac{\gamma_{A1}}{6}\Phi^{(3)}[(a-\mu_A)/\sigma_A], \tag{7}$$

$$G_2(\Phi) = \frac{\gamma_{A2}}{24}\Phi^{(4)}[(a-\mu_A)/\sigma_A] + \frac{\gamma_{A1}^2}{72}\Phi^{(6)}[(a-\mu_A)/\sigma_A]. \tag{8}$$

Combining equations (3), (4), (7), and (8) and letting $z_a \equiv (a-\mu_A)/\sigma_A$, we obtain the two- and three-term Edgeworth approximations for F_A:

$$\hat{F}_{A2}(a) = \Phi(z_a) - \frac{\gamma_{A1}}{6}H_2(z_a)\phi(z_a), \tag{9}$$

$$\hat{F}_{A3}(a) = \Phi(z_a) - \phi(z_a)\left[\frac{\gamma_{A1}}{6}H_2(z_a) + \frac{\gamma_{A2}}{24}H_3(z_a) + \frac{\gamma_{A1}^2}{72}H_5(z_a)\right]. \tag{10}$$

Thus the practical implementation of the Edgeworth approximations (9) and (10) for F_A hinges on the use of an efficient, numerically robust algorithm for computing the skewness and excess kurtosis of A.

3.2. The moment computation algorithm. The evaluation of the skewness and excess kurtosis of A proceeds by computing the first four noncentral moments $\{\mu'_{Aw}: w = 1, \cdots, 4\}$ so that we can use the following standard formulas to obtain the corresponding central moments $\{\mu_{Aw}: w = 1, \cdots, 4\}$ as well as γ_{A1} and γ_{A2}:

$$\begin{aligned} &\mu_{A2} = \sigma_A^2 = \mu'_{A2} - \mu_A^2, \qquad \mu_{A3} = \mu'_{A3} - 3\mu'_{A2}\mu_A + 2\mu_A^3, \\ &\mu_{A4} = \mu'_{A4} - 4\mu'_{A3}\mu_A + 6\mu'_{A2}\mu_A^2 - 3\mu_A^4, \\ &\gamma_{A1} = \mu_{A3}/\mu_{A2}^{3/2} \quad \text{and} \quad \gamma_{A2} = (\mu_{A4}/\mu_{A2}^2) - 3. \end{aligned} \tag{11}$$

Throughout the rest of § 3, we assume the nodes in the target probability tree are labeled sequentially starting at 1 so that the node index-set for the tree is $N = \{1, 2, \cdots, \nu\}$. (For a combined tree, the superroot may simply be ignored in the following discussion.) If we let $X_i \equiv P_i U_i$ for each $i \in N$, then the key to the efficient computation of μ'_{Aw} is the observation that $A = X_1 + \cdots + X_\nu$ is a symmetric function of the $\{X_i\}$. Thus for each desired power w ($w = 1, 2, 3, 4$), the power-sum A^w can be expressed in terms of the *augmented symmetric functions* of the $\{X_i\}$ (Stuart and Ord [20]). Now in typical statistical applications of these functions, the $\{X_i\}$ are independent and identically distributed random variables. However, in the analysis of probability trees, the $\{X_i\}$ are neither independent nor identically distributed; and this substantially complicates the computation of the moments of A.

The noncentral moments of A are evaluated as follows. By assumption (2) of § 2.3, the branching probabilities $\{B_i\}$ are independent of the node utilities $\{U_i\}$. In view of equation (1), we see that the partial path probabilities $\{P_i\}$ are also independent of the $\{U_i\}$. Eliminating all redundant calculations from the formula for each augmented symmetric function that occurs in the power-sum A^w and recognizing the possibility of duplicate variates within the set $\{B_i\}$ and within the set $\{U_i\}$, we obtain the following master computational formulas for μ'_{Aw}, $w = 1, 2, 3, 4$:

$$\mu_A = \sum_{i=1}^{\nu} \mathrm{E}(P_i)\mathrm{E}(U_i), \tag{12}$$

$$\mu'_{A2} = \sum_{i=1}^{\nu} \mathrm{E}(P_i^2)\mathrm{E}(U_i^2) + 2\sum_{i=1}^{\nu-1}\sum_{j=i+1}^{\nu} \mathrm{E}(P_iP_j)\mathrm{E}(U_iU_j), \tag{13}$$

$$\mu'_{A3} = \sum_{i=1}^{\nu} \mathrm{E}(P_i^3)\mathrm{E}(U_i^3) + 3 \sum_{i=1}^{\nu} \sum_{\substack{j=1 \\ j\neq i}}^{\nu} \mathrm{E}(P_i^2 P_j)\mathrm{E}(U_i^2 U_j)$$

$$+2 \sum_{i=1}^{\nu-1} \sum_{j=i+1}^{\nu} \sum_{\substack{k=1 \\ k\neq i,j}}^{\nu} \mathrm{E}(P_i P_j P_k)\mathrm{E}(U_i U_j U_k), \tag{14}$$

$$\mu'_{A4} = \sum_{i=1}^{\nu} \mathrm{E}(P_i^4)\mathrm{E}(U_i^4) + 4 \sum_{i=1}^{\nu} \sum_{\substack{j=1 \\ j\neq i}}^{\nu} \mathrm{E}(P_i^3 P_j)\mathrm{E}(U_i^3 U_j)$$

$$+6 \sum_{i=1}^{\nu-1} \sum_{j=i+1}^{\nu} \mathrm{E}(P_i^2 P_j^2)\mathrm{E}(U_i^2 U_j^2)$$

$$+12 \sum_{i=1}^{\nu-1} \sum_{j=i+1}^{\nu} \sum_{\substack{k=1 \\ k\neq i,j}}^{\nu} \mathrm{E}(P_i P_j P_k^2)\mathrm{E}(U_i U_j U_k^2) \tag{15}$$

$$+4 \sum_{i=1}^{\nu-1} \sum_{j=i+1}^{\nu} \sum_{\substack{k=1 \\ k\neq i,j}}^{\nu-1} \sum_{\substack{l=k+1 \\ l\neq i,j}}^{\nu} \mathrm{E}(P_i P_j P_k P_l)\mathrm{E}(U_i U_j U_k U_l).$$

See Eisenhut [11] for a detailed discussion and justification of (12)-(15).

To complete the computation of the wth noncentral moment μ'_{Aw}, we must evaluate product moments of the form $\mathrm{E}(P_i^r P_j^x P_k^y P_l^z)$ and $\mathrm{E}(U_i^r U_j^x U_k^y U_l^z)$ where r, x, y, and z are nonnegative integers such that $r+x+y+z=w$. The probability trees depicted in Figs. 1 and 2 illustrate the method for evaluating such terms. Suppose that in these trees, the following random parameters are duplicates: (a) U_2, U_8, and U_{14}^*; and (b) B_2, B_6, B_8, and B_{12}. It follows immediately that in the combined tree of Fig. 3,

$$B_2 = B_6 = B_8 = B_{12}, \qquad B_3 = B_7 = B_9 = B_{11} = 1 - B_2, \tag{16}$$

$$U_2 = U_8 = U_{14}^* \quad \text{and} \quad U_{14} = -U_{14}^* = -U_8 = -U_2. \tag{17}$$

In evaluating the summands of the fourth term on the right-hand side of equation (15), we must compute, for example, the terms $\mathrm{E}(P_6^2 P_8 P_{14})$ and $\mathrm{E}(U_6^2 U_8 U_{14})$. The evaluation procedure expresses each relevant partial path probability P_i as a product of its component branching probabilities $\{B_j : j \in Q(i)\}$ by traversing the tree from node i to the root (or superroot for a combined tree), accounting for duplicates and accumulating the exponent of each distinct factor B_j in the product P_i. In the example, display (16) implies that

$$P_6 = B_3 B_6 = (1 - B_2)B_2, \qquad P_8 = B_2 B_4 B_8 = B_2^2 B_4,$$

and

$$P_{14} = B_{12} B_{14} = B_2 B_{14};$$

thus the desired overall product of partial path probabilities has the form

$$P_6^2 P_8 P_{14} = B_2 B_3^2 B_4 B_6^2 B_8 B_{12} B_{14} = B_2^5 (1 - B_2)^2 B_4 B_{14}. \tag{18}$$

The evaluation algorithm accumulates terms like $\mathrm{E}(P_6^2 P_8 P_{14})$ as a running product of expectations of stochastically independent factors involving distinct branching probabilities. For each pair of sibling-nodes whose complementary branching probabilities both appear with nonzero exponents in the overall product of partial path probabilities, a *forking operation* is performed to compute the expected value of the corresponding factor. In the example, nodes 2 and 3 are siblings for which the factor

$B_2^5 B_3^2 = B_2^5(1-B_2)^2$ appears in the overall product (18); and the forking operation yields the intermediate result

$$\xi_1 = \mathrm{E}[B_2^5(1-B_2)^2] = \mathrm{E}(B_2^5) - 2\mathrm{E}(B_2^6) + \mathrm{E}(B_2^7)$$

by the usual binomial expansion. Clearly, this forking operation would be extremely complicated for a nonbinary node, since it would involve a multinomial expansion and would have to be computed from the joint distribution of the branching probabilities for the full set of associated sibling nodes. For a set of more than two sibling nodes, it is not even clear how to specify the joint distribution of the corresponding branching probabilities (Doubilet et al. [10]). These considerations motivated assumption (1) that the probability tree is binary. In the example, equation (18) shows that no other forking operations are required, and the other intermediate results for stochastically independent factors are

$$\xi_2 = \mathrm{E}(B_4), \qquad \xi_3 = \mathrm{E}(B_{14}).$$

The final result is the product of all intermediate results:

$$\mathrm{E}(P_6^2 P_8 P_{14}) = \xi_1 \xi_2 \xi_3.$$

The computation of terms of the form $\mathrm{E}(U_i^r U_j^x U_k^y U_l^z)$ is simpler. Accounting for duplicates and accumulating the exponent of each distinct factor in the product $U_i^r U_j^x U_k^y U_l^z$, the evaluation algorithm computes $\mathrm{E}(U_i^r U_j^x U_k^y U_l^z)$ as a running product of expectations of stochastically independent factors involving distinct utilities. A minor complication arises if two compared trees have been joined to form a combined tree and some of the node utilities U_i, U_j, U_k, U_l are derived from the original node utilities $\{U_h^*\}$ of the second tree. Since $\mathrm{E}(U_i^r U_j^x U_k^y U_l^z)$ must be computed from the noncentral moments of the original node utilities in both compared trees, the product-moment computation for utilities must include accumulation of the exponent of the sign-reversal factor (-1). In the example, display (17) shows that

$$U_6^2 U_8 U_{14} = -U_6^2 U_8 U_{14}^* = -U_6^2 U_2^2 \Rightarrow \mathrm{E}(U_6^2 U_8 U_{14}) = -\mathrm{E}(U_2^2)\mathrm{E}(U_6^2).$$

3.3. Asymptotic time complexity of the approximation procedure. We have implemented our procedure for analyzing probability trees in TreeModeler, a portable public-domain computer program which is available from the authors upon request (Dittus et al. [8]). TreeModeler consists of two major modules: (i) an interactive tree-building module that is equipped with flexible, accessible commands for constructing probability trees as well as extensive error checking and help capabilities (Stouppe [19]); and (ii) an analysis module that incorporates the moment computation algorithm detailed in § 3.2 and the Edgeworth approximations formulated in § 3.1 (Eisenhut [11]). TreeModeler will also optionally perform a simulation of the user-specified trees based on the SLN simulation language (Roberts and Klein [18]). The operation of TreeModeler, as it relates to the Edgeworth approximations for F_A, consists of the following key steps:

(a) The characteristics of the tree are set up either interactively or by reading an input file prepared by the user; and for each node $i \in N$, the required moments $\mathrm{E}(B_i^w)$ and $\mathrm{E}(U_i^w)$ are computed once and stored for use in step (b) below.

(b) The relevant noncentral moments of A are computed according to (12)–(15) based on the approximation (5), (9), or (10) requested by the user.

(c) The relevant central moments of A (possibly together with the skewness γ_{A1} or excess kurtosis γ_{A2}) are computed according to (11) based on the approximation (5), (9), or (10) requested by the user.

(d) The requested approximation $\hat{F}_{A1}(a)$, $\hat{F}_{A2}(a)$, or $\hat{F}_{A3}(a)$ is evaluated over a user-specified interval of values for the cutoff point a.

The computational complexity of TreeModeler is determined by the moment computation algorithm—specifically, step (b) above. Let ν_0 denote the number of nodes with nonzero utilities in the target probability tree. Under certain assumptions about the target tree, we demonstrated that the asymptotic time complexity of the algorithm for computing the first w moments of A is $O[\nu_0^w \log(\nu_0)]$ (see Eisenhut et al. [12]). We also performed an extensive experimental verification of this result on a wide diversity of probability trees. In this investigation we systematically varied the following tree characteristics: (i) the percentage of nonzero utilities, (ii) the ratio of the height of the tree to the number of nodes, (iii) the percentage of duplicated branching probabilities, (iv) the percentage of duplicated utilities, and (v) the computer used. The experimental results showed that in the computation of the wth noncentral moment of A, the dependence of the execution time τ_w on the number of nonzero utilities ν_0 is fairly accurately described by the relation

$$\tau_w = C_w \nu_0^w \log(\nu_0). \tag{19}$$

The constant of proportionality C_w in equation (19) depends on the factors (i)-(v) mentioned above. See Eisenhut [11] for further analysis of the computational complexity of the moment computation algorithm within TreeModeler.

If simulation is used to estimate F_A, then the resulting empirical distribution function $\hat{F}_{A0}$ requires computation time $O(r\nu_0)$, where r is the number of independent replications of the target tree (Eisenhut [11]). As in equation (20) below, r is usually set by the user to yield a prespecified upper bound on the estimation error $|\hat{F}_{A0}(a) - F_A(a)|$ for all values of a; and this replication count does not depend on ν_0. Thus the ratio of computational costs for the w-term Edgeworth approximation $\hat{F}_{Aw}$ relative to the direct-simulation estimator $\hat{F}_{A0}$ is $O(r^{-1}\nu_0^{w-1})$, which favors simulation as the size of the tree grows. However, the w-term Edgeworth approximation has lower computational costs for small to moderate-size trees, especially when we take $w = 2$; moreover, the Edgeworth approximation has the added benefit of providing exact moments of A. In the next section we make a direct comparison of the simulation-based estimation technique versus the Edgeworth approximation in the context of a large-scale application in the field of medical decision making.

4. Application to medical decision processes.

4.1. Patient management following uncomplicated myocardial infarction. Survivors of an uncomplicated myocardial infarction (heart attack) generally suffer from atherosclerosis of the coronary arteries (blockage of the arteries supplying blood to the heart) and are at risk for successive cardiovascular problems, particularly during the six months following infarction. Depending on the location and extent of coronary disease, coronary artery bypass grafting (CABG) surgery can improve the prognosis, especially among patients with obstruction of the left main coronary artery and among patients with obstruction in all three of the major coronary arteries. However, bypass surgery is expensive, and surgical complications can result in additional morbidity and mortality. Moreover, the only definitive diagnostic test to identify a patient's anatomic disease is angiography, an expensive test which carries some risk. Other available diagnostic tests are less accurate than angiography but carry less risk and are less

expensive. Thus, one problem facing clinicians in the management of patients following uncomplicated myocardial infarction is the identification of patients who should be referred for bypass surgery.

Dittus, Roberts, and Adolph [9] studied seven alternative management strategies for this problem and examined the corresponding expected costs, mortality rates, and complications within the six-month period following the initial myocardial infarction. The strategies consisted of selected combinations of one of three testing procedures together with one of two criteria for surgery. The testing procedures were exercise electrocardiography (treadmill), exercise thallium scintigraphy (thallium scan), and coronary cineangiography (angiography). The criteria for bypass surgery were: (i) operate on patients with left main disease and on patients with a strongly positive treadmill or thallium scan; and (ii) operate on patients with a strongly positive treadmill or thallium scan and on patients with left main obstruction, three vessel obstruction, or one or two vessel obstruction with a significant amount of myocardium in jeopardy.

The results of the study indicated that the strategy beginning with treadmill exercise testing (TME) was the most cost-effective. Under this strategy as depicted in Fig. 4, patients with a strongly positive treadmill result are referred for angiography (see Fig. 5) and bypass surgery—in this case, angiography is still necessary to define the anatomy for the surgeons and to determine if the type of obstruction can be bypassed. Patients with an otherwise positive test result are referred for angiography, and bypass surgery is recommended under the second surgical criterion. Patients with a normal treadmill are not tested further and do not undergo bypass surgery. Patients with an indeterminate test result are referred for thallium testing (THAL), a more sensitive and more expensive test than the treadmill test. Like the treadmill test, the thallium test can have one of four results: strongly positive, otherwise positive, indeterminate, or normal. Patients with strongly positive tests are referred for angiography and surgery. Those with otherwise positive or indeterminate tests are referred for angiography, and the decision to perform surgery is also based on the second surgical criterion. Patients with normal tests are managed medically, including the use of a beta-adrenergic blocking agent. Although the treadmill strategy was deemed the most cost-effective, a slightly lower mortality rate was achieved at a somewhat higher cost using the strategy shown in Fig. 5 that was based solely on angiography and the second surgical criterion. As a follow-up to the original analysis of Dittus, Roberts, and Adolph [9], we performed a more extensive comparison of the treadmill exercise strategy and the angiography strategy.

Data for the analysis were provided by members of the Cardiovascular Norms Committee of the American College of Cardiology, who derived estimates from published literature and expert judgment. Because these input estimates were not known with absolute certainty and were also known to vary among patients, a complex multiway sensitivity analysis was indicated. However, the large number of questionable parameters precluded such an approach; and a more realistic representation of the uncertainty was obtained by modeling these parameters as random variables. Sensitivity analysis can demonstrate how one or two input parameters can change the expected value; however, a sensitivity analysis cannot provide an answer to certain important questions. For example, what is the likelihood that in the face of the stated uncertainty, the averaged-out cost difference between the two strategies will exceed $5,000, or any other fixed cutoff value (especially zero)? For cost-effective medical care planning, we need the cumulative probabilities associated with various thresholds for the averaged-out cost difference between alternative medical treatments. We have addressed this problem to illustrate the methodology presented in this paper.

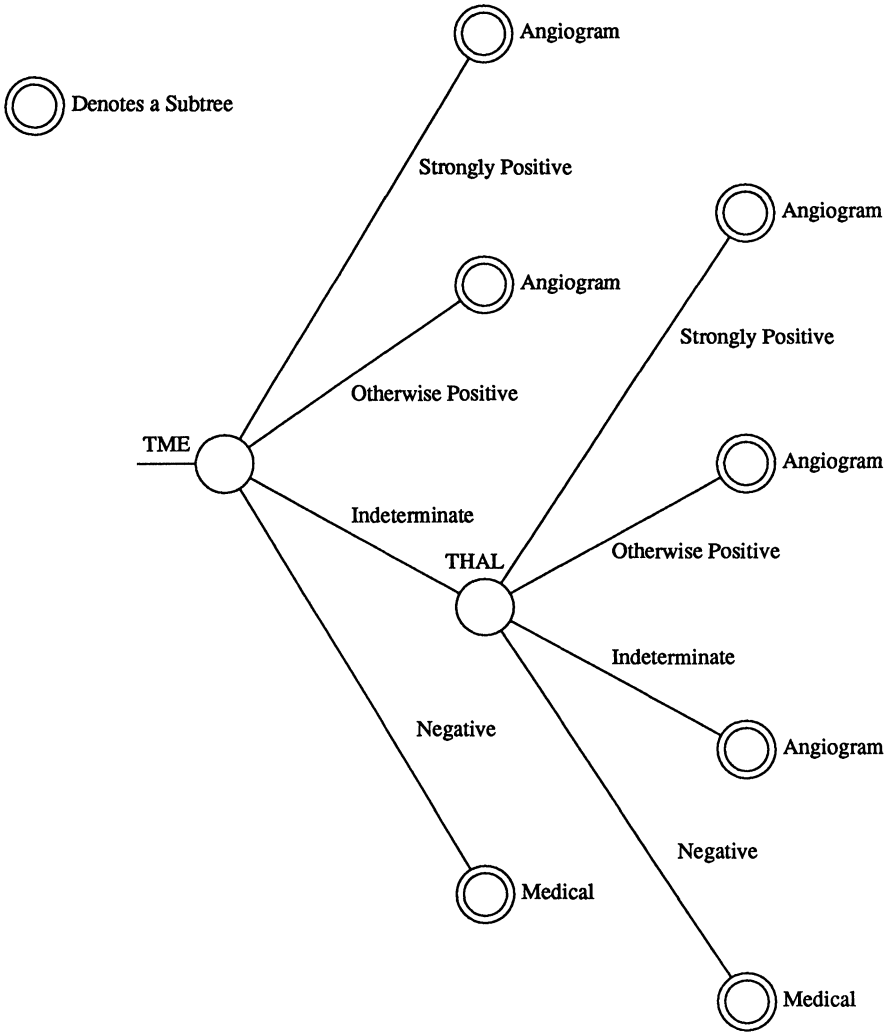


FIG. 4. *Simplified probability tree for the treadmill exercise strategy.*

4.2. Methods used in the evaluation. We restricted our attention in this evaluation to a single performance measure—the expected six-month cost of each management strategy. There was uncertainty in both the branching probabilities and the costs at the nodes. The costs were known to be random, while many of the branching probabilities were uncertain due to the lack of expert agreement on appropriate values. The clinical investigators who supplied the data in the original study were asked to represent uncertainty in the parameters of the probability tree for each strategy by specifying the minimum, mode, and maximum of triangular distributions postulated for these variables. The analysis included the following five random costs with the associated minimum, mode, and maximum values, respectively, given in parentheses: treadmill exercise test ($100, $150, $300); thallium scan ($500, $750, $1,000); angiography ($2,000, $2,500, $3,000); bypass surgery ($12,000, $15,000, $25,000); and hospitalization for nonfatal infarction ($1,000, $1,500, $3,000). The resulting probability tree with random parameters was then converted to an equivalent binary tree (recall assumption (1) of § 2.3).

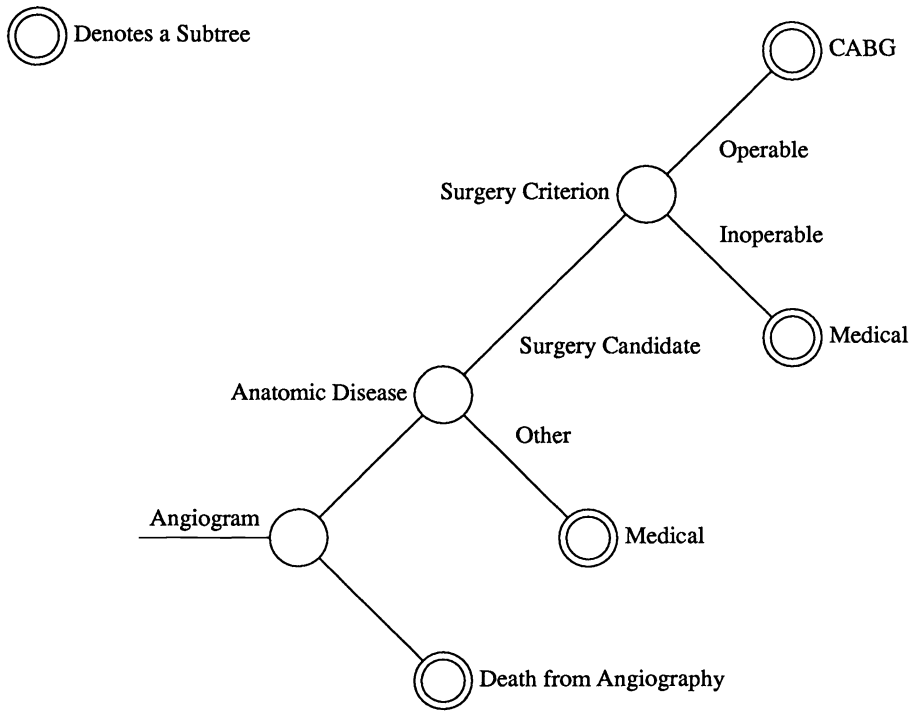


FIG. 5. *Simplified probability tree for the angiography strategy.*

The combined probability tree representing the averaged-out cost difference between the two competing strategies (treadmill testing and angiography) was very large, originally composed of 119 nodes, most of which were nonbinary. When it was converted to an equivalent binary representation, the tree expanded to 398 binary nodes. Since there were only five distinct nonzero costs in this application and since duplicate costs occurred in 27 nodes of the original tree and in 99 nodes of the derived binary tree, altogether there were 32 nonzero cost nodes in the original tree and 104 nonzero cost nodes in the derived binary tree. In addition to the duplication of costs, 27 of the original branching probabilities and 165 of the derived branching probabilities were duplicates.

To complement our analytical comparison of the two treatment strategies using TreeModeler, we also developed a simulation model of these strategies using the SLN simulation language (Roberts and Klein [18]). The simulation used a portable prime modulus random number generator (Marse and Roberts [15]) together with the inverse CDF method (Bratley, Fox, and Schrage [2]) to sample independently from each triangular distribution describing a parameter of the corresponding probability tree. Each replication of the simulation yielded a random sample from the distribution F_A of the averaged-out cost difference between the strategies. The simulation was run for 125,000 independent replications. This large number of replications was chosen to ensure that with probability 0.99, the simulation-generated empirical distribution function $\hat{F}_{A0}$ satisfies the following accuracy specification:

$$|\hat{F}_{A0}(a) - F_A(a)| \leqq 0.005 \quad \text{for } -\infty < a < \infty. \tag{20}$$

Since A is a continuous random variable in this application, the limiting distribution

of the Kolmogorov statistic provides the basis for property (20) (Bickel and Doksum [1]).

4.3. Comparison of the analytical and simulation-based approximations to F_A. Because of the size of the combined tree and the requirement to compute the fourth moment of A as a demonstration of the complete approximation procedure, TreeModeler was run on Purdue University's CYBER 205 supercomputer. In this application the overall execution time for TreeModeler was 88 minutes; however the computation of the first three moments of A required only 87 seconds so that the computation of the fourth moment of A accounted for virtually all of the execution time. (To put these execution times in perspective, we note that the first three moments of A can be calculated in 10.3 minutes on a 25-MHz 80386-based microcomputer with an 80387 numeric coprocessor.) The moments of A were found to be: $\mu_A = \$3{,}071$, $\sigma_A = \$506$, $\gamma_{A1} = 0.507$, and $\gamma_{A2} = 0.118$. The large number of random parameters in the combined tree resulted in a central-limit effect which ensured that the skewness and excess kurtosis of A were not markedly nonnormal. In comparison to the approximation procedure based on the normal distribution, the Edgeworth approximation appears to be most useful when the target distribution displays moderate departures from normality (Stuart and Ord [20]).

To complete the comparison of the angiography and treadmill exercise strategies, we performed 125,000 independent replications of the simulation for the corresponding combined tree, and we computed some standard goodness-of-fit measures for all of the approximation methods. Figure 6 shows plots over the interval $\$0 \leqq a \leqq \$6{,}000$ of the simulation-generated CDF $\hat{F}_{A0}(a)$ and the normal approximation $\hat{F}_{A1}(a)$. Figure 7 shows similar plots of the simulation-generated CDF and the two-term Edgeworth approximation $\hat{F}_{A2}(a)$. Table 1 displays average and maximum values for the absolute deviations $|\hat{F}_{A0}(a) - \hat{F}_{Aw}(a)|$ over the interval $-\infty < a < \infty$ for $w = 1, 2, 3$. Both the graphical and tabular results reveal that the two-term Edgeworth approximation to F_A is almost indistinguishable from the simulation-based approximation, while the normal approximation yields a clearly inferior fit throughout the interval of cutoff values. As shown in Fig. 8, the two-term Edgeworth approximation $\hat{f}_{A2}$ for the probability density function of A provides clear evidence of the nonnormality of A.

Our experience in computing the moments of A for a variety of probability trees is that as the number of nodes increases, the excess kurtosis γ_{A2} tends to zero faster than the skewness γ_{A1} approaches zero; and this effect is even more pronounced when the difference between two trees is being examined. With respect to maximum absolute deviation of the fitted CDF from the simulation-generated empirical CDF, in this application there was a 29 percent reduction in error for the three-term Edgeworth approximation versus the two-term Edgeworth approximation. However, the third term in the Edgeworth expansion for F_A required a 5,900 percent increase in computation time. For a probability tree having 25 to 100 nonzero utility nodes, we recommend using the two-term Edgeworth approximation, which is competitive with the simulation-based approximation both in accuracy and in computational requirements.

4.4. Discussion. The Edgeworth approximations to F_A have proven to be accurate and computationally feasible. Compared to the normal approximation, the two-term Edgeworth approximation provides greater accuracy with little additional computing cost. In contrast to other techniques for approximating the distribution of A, the procedure developed in this paper yields the exact moments of A (at least to the limits of machine accuracy). Simulation-based estimators of the higher moments of A are subject to progressively greater sampling fluctuations (that is, they have progressively

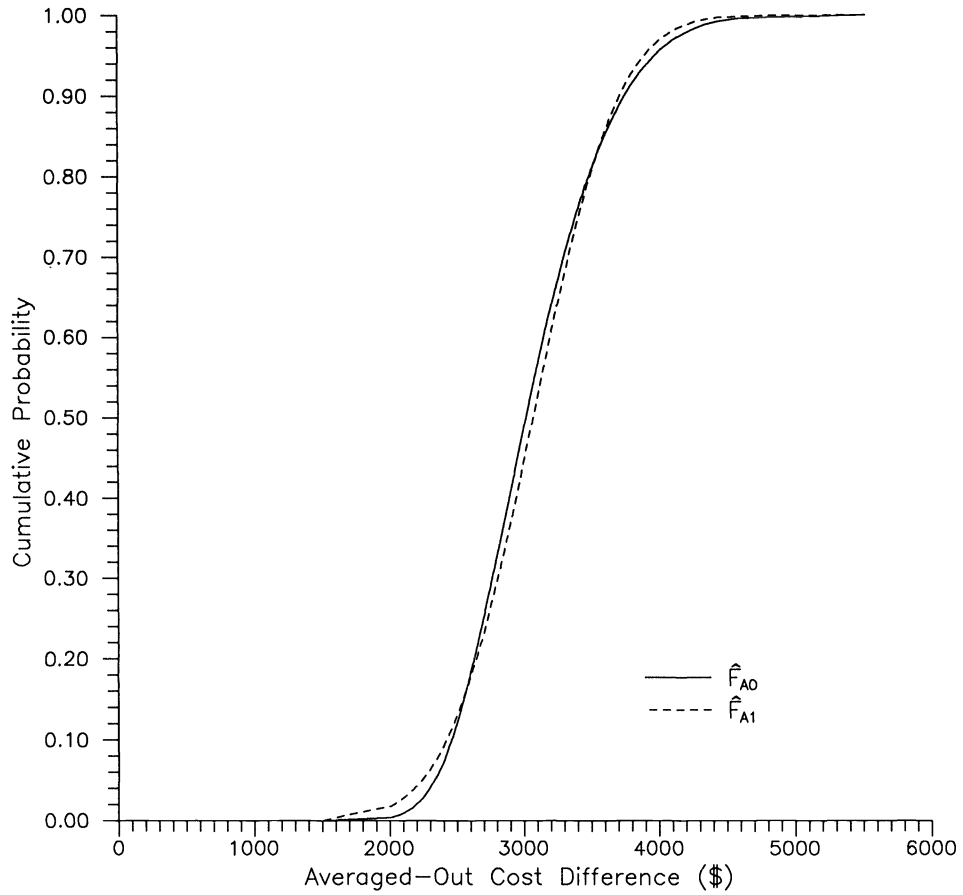


FIG. 6. *$\hat{F}_{A0}$ versus $\hat{F}_{A1}$ for the averaged-out cost difference between the angiography and treadmill exercise strategies.*

larger variances); and reducing this variability to acceptable levels can require prohibitively large sample sizes.

Examination of Fig. 8, the two-term Edgeworth approximation to the probability density function of the averaged-out cost difference between the angiography and treadmill exercise strategies, reveals that this distribution is skewed to the right. There is clearly no chance that angiography will ever be less expensive than the treadmill exercise test, verifying the absolute dominance of treadmill exercise as the less costly strategy. By virtue of the uncertainty inherent in the input data, the actual averaged-out cost difference between the two strategies could be as low as $1,700 or as high as $5,000. Figure 7, the corresponding two-term Edgeworth approximation to the cumulative distribution function, shows there is almost no chance that the cost difference will exceed $5,000. In fact, there is at most a five percent chance that the difference will exceed $4,000.

A probability interpretation could also be of interest in the comparison of strategies whose averaged-out utilities have nearly the same expected value. In such cases, data uncertainty could imply that each of a pair of compared strategies has roughly an equal chance of incurring the lesser averaged-out cost. In such cases, the choice of a treatment strategy would probably be based on other considerations. In the management

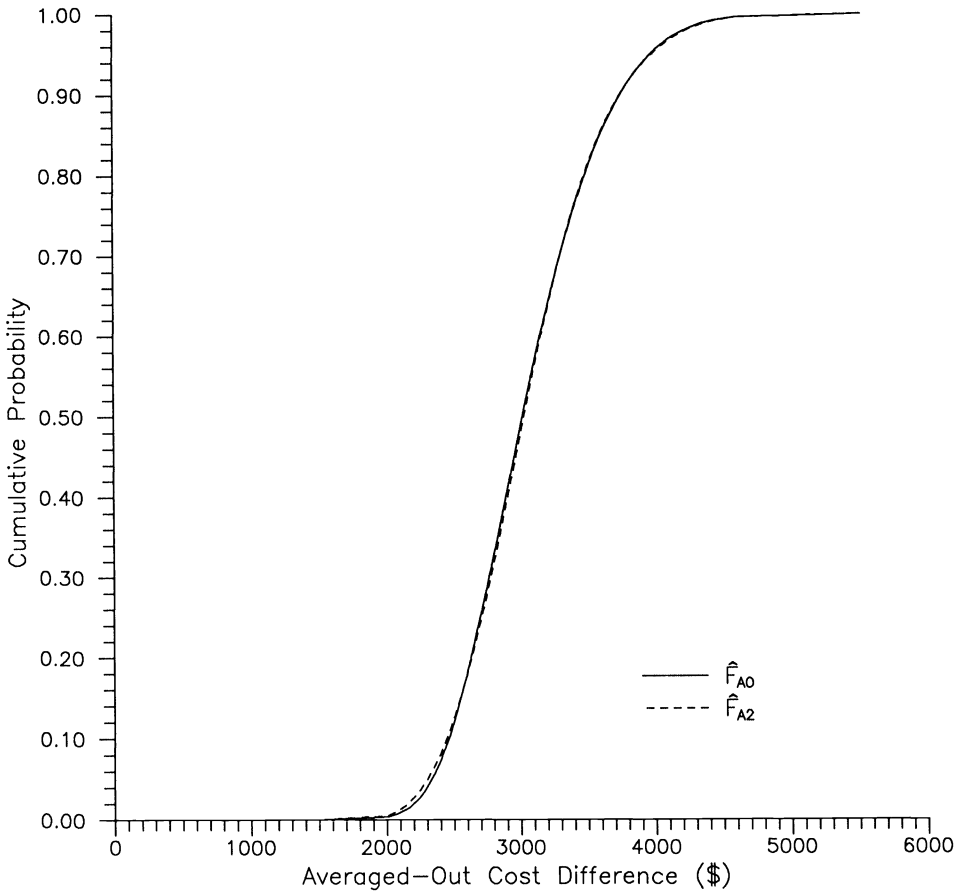


FIG. 7. *$\hat{F}_{A0}$ versus $\hat{F}_{A2}$ for the averaged-out cost difference between the angiography and treadmill exercise strategies.*

TABLE 1

Accuracy of the approximations to F_A for the averaged-out cost difference between the angiography and treadmill exercise strategies.

Deviation from empirical CDF $\hat{F}_{A0}$	Normal approx. $\hat{F}_{A1}$	2-term Edgeworth approx. $\hat{F}_{A2}$	3-term Edgeworth approx. $\hat{F}_{A3}$
Average absolute dev.	0.022	0.005	0.0044
Maximum absolute dev.	0.042	0.011	0.0079

of patients following myocardial infarction, the expected value of the averaged-out cost (but not the mortality) associated with the use of the thallium scan (without treadmill testing) was close to that of the angiography strategy. Figure 9 displays approximations to the distribution function of the averaged-out cost difference (angiography cost minus thallium scan cost) based on the two-term Edgeworth expansion as well as the simulation-generated empirical CDF. This figure reveals that angiography is always more expensive, even when uncertainty in the parameters is taken into account; and there is only a six percent likelihood that the difference in

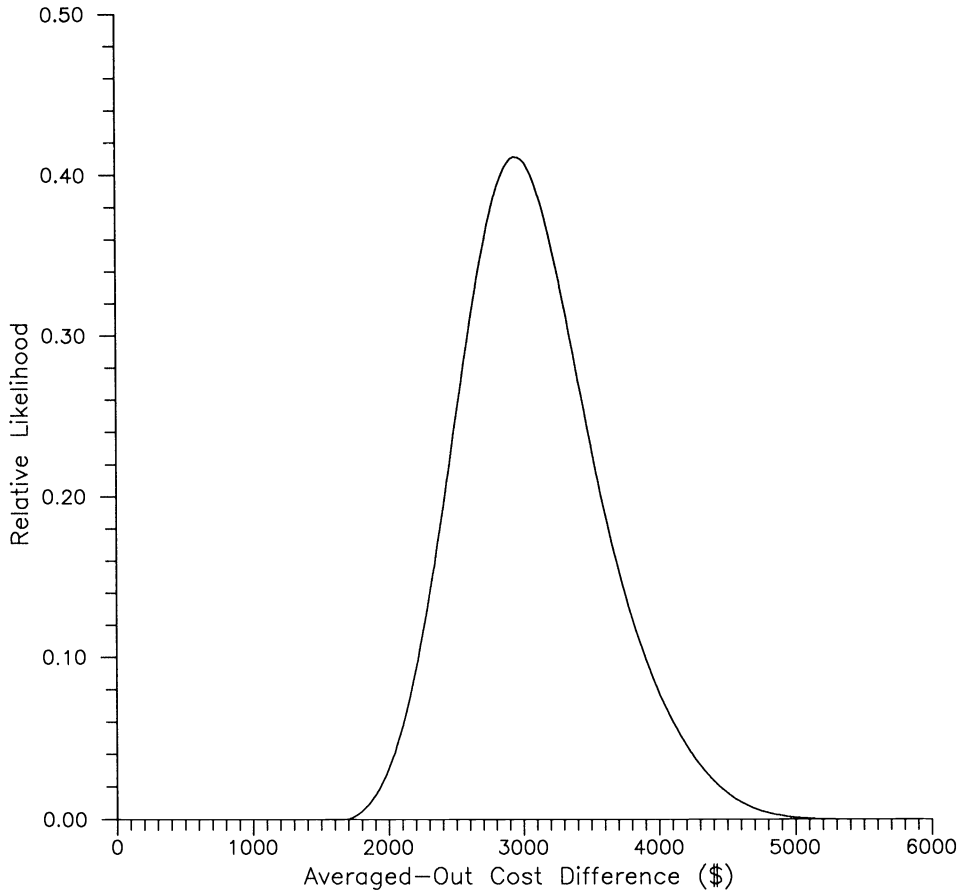


FIG. 8. $\hat{f}_{A2}$ *for the averaged-out cost difference between the angiography and treadmill exercise strategies.*

cost would be less than $500 and a two percent likelihood that the difference in cost would exceed $1,500.

The main advantage of this approach for evaluating the effect of questionable parameters on the conclusions of any study is that the uncertainty in all input variables is simultaneously taken into account. Traditional one-, two-, and three-way sensitivity analyses cannot provide such information. In the interpretation of the clinical problem of post-infarction management, finding upper and lower bounds on the averaged-out cost difference between alternative treatment strategies enables medical care planners to understand more completely the potential financial risk associated with using one management strategy as compared to its alternatives. For example, in the analysis depicted by Figs. 7 and 8, the finding that $5,000 is a "hard" upper bound on the averaged-out cost difference between angiography and treadmill exercise testing might be sufficient information to justify the use of the angiography strategy, with the expected benefit of reducing mortality. In the absence of this information, administrators might discourage routine use of the angiography strategy for fear of substantial financial loss.

5. Conclusions and recommendations for further research. Characterizing the parameters of a probability tree as (possibly duplicated) random variables is a more meaningful representation of the inherent uncertainty in a decision process than is possible with a conventional sensitivity analysis because the former approach incorpo-

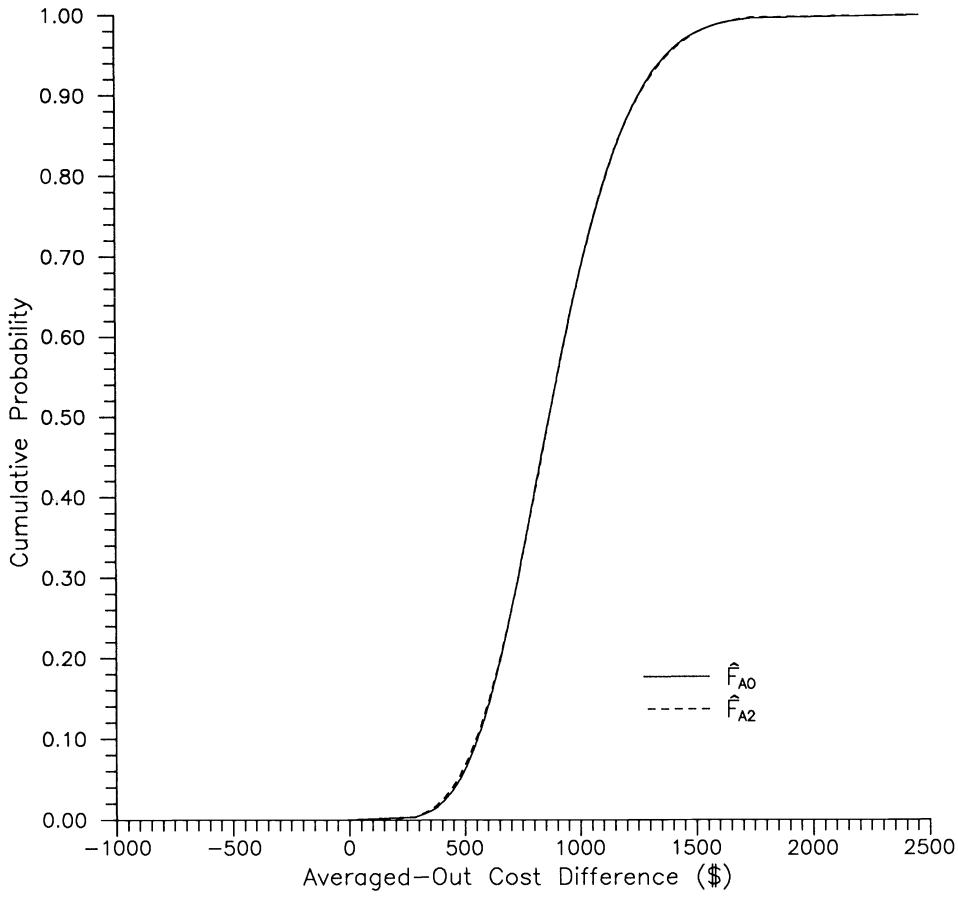


FIG. 9. $\hat{F}_{A0}$ *versus* $\hat{F}_{A2}$ *for the averaged-out cost difference between the angiography and thallium scan strategies.*

rates the likelihoods associated with different values of the parameters in question. Thus the averaged-out utility of a probability tree with random parameters must itself be treated as a random variable whose cumulative distribution function is to be estimated. For instance, the decision analyst can state that there is a five percent probability that the averaged-out utility of a tree is greater than $1,000, or that there is a 95 percent probability that the averaged-out utility will be between $925 and $1,025.

Because the analysis of a single tree can be easily extended to the comparison of two trees, the distributional interpretation applies to the difference between the averaged-out utilities of the two trees. The decision analyst need not rely only on the expected value of this difference. Employing the distribution of the difference in averaged-out utilities, the analyst may state, for instance, that there is an 80 percent probability that tree 1 has a larger averaged-out utility than tree 2. Such information should enhance the comparison of trees by directly incorporating any uncertainty in their parameters. Furthermore, since the two trees are examined under identical conditions, the comparison clearly reveals systematic differences between the underlying processes.

Our experience with the Edgeworth approximation shows that it provides a numerically robust means to estimate the distribution of averaged-out utility for a

probability tree with random parameters. In all of our numerical studies, the two-term Edgeworth approximation substantially outperformed the normal approximation with respect to accuracy while incurring little additional computing cost. The TreeModeler program has been run on a wide variety of machines, from microcomputers to supercomputers, without numerical problems. Although the execution time for the moment computation algorithm can be prohibitive for large trees, the program is practical for small to medium trees. For trees with fewer than 25 nonzero utility nodes, the three-term Edgeworth approximation is a practical alternative to simulation. For trees with 25 to 100 nonzero utility nodes, the two-term Edgeworth approximation is practical—even on microcomputers. Although simulation can never provide exact moments, it becomes the method of choice for approximating F_A in trees with more than 100 nonzero utility nodes because of its lower computational requirements.

Perhaps the most serious limitation of our approximation procedure is the requirement that the random tree parameters must be either mutually independent or perfectly correlated (that is, duplicated). We need a general means of specifying a dependency structure among the random parameters of a probability tree. Moreover, we need an effective method for eliciting information about this dependency structure from knowledgeable individuals. These developments are the subject of future research.

In our limited experience, nonnormality in the distribution of the averaged-out utility is sufficiently pronounced to warrant the improved accuracy afforded by the Edgeworth adjustment. However, the practical consequence of assuming normality when the averaged-out utility is not normal would have its greatest impact when comparisons involve "close calls" and where the probability statements about cost differences are determining factors in the decision. In general, the extra computational cost of the two-term Edgeworth approximation is moderate compared to the computational cost of the normal approximation; and the corresponding improvement in accuracy is easily obtained with a portable software package like TreeModeler.

Acknowledgments. The authors wish to acknowledge the helpful observations of Dr. Suzanne Knoebel of the Krannert Institute of Cardiology, Indiana University School of Medicine, and those of Professor Bruce Schmeiser, Purdue University School of Industrial Engineering. The authors also thank Mr. Robert Klein of the Regenstrief Institute for his insightful critique of the manuscript.

REFERENCES

[1] P. J. BICKEL AND K. A. DOKSUM, *Mathematical Statistics: Basic Ideas and Selected Topics*, Holden-Day, San Francisco, 1977.

[2] P. BRATLEY, B. L. FOX, AND L. E. SCHRAGE, *A Guide to Simulation*, Second Edition, Springer-Verlag, New York, 1987.

[3] K. L. CHUNG, *A Course in Probability Theory*, Second Edition, Academic Press, New York, 1974.

[4] H. CRAMÉR, *On the composition of elementary errors, II: Statistical applications*, Scand. Actuar. J., 11 (1928), pp. 141-180.

[5] ———, *Mathematical Methods of Statistics*, Princeton University Press, Princeton, NJ, 1946.

[6] G. C. CRITCHFIELD AND K. E. WILLARD, *Probabilistic analysis of decision trees using Monte Carlo simulation*, Med. Decis. Making, 6 (1986), pp. 85-92.

[7] G. C. CRITCHFIELD, K. E. WILLARD, AND D. P. CONNELLY, *Probabilistic sensitivity analysis for general decision models*, Comput. & Biomedical Res., 19 (1986), pp. 254-265.

[8] R. S. DITTUS, R. D. EISENHUT, R. W. KLEIN, S. D. ROBERTS, M. S. STOUPPE, AND J. R. WILSON, *TreeModeler: Construction and analysis of probability trees with random parameters*, Research Memorandum No. 90-1, School of Industrial Engineering, Purdue University, West Lafayette, IN, 1990.

[9] R. S. DITTUS, S. D. ROBERTS, AND R. J. ADOLPH, *Cost-effective analysis of patient management alternatives following uncomplicated myocardial infarction: A model*, J. Amer. Coll. Cardiol., 10 (1987), pp. 869-878.

[10] P. DOUBILET, C. B. BEGG, M. C. WEINSTEIN, P. BRAUN, AND B. J. MCNEIL, *Probabilistic sensitivity analysis using Monte Carlo simulation: A practical approach*, Med. Decis. Making, 5 (1985), pp. 157-177.

[11] R. D. EISENHUT, *Probability trees with random variable parameters*, M.S. thesis, School of Industrial Engineering, Purdue University, West Lafayette, IN, 1986.

[12] R. D. EISENHUT, R. S. DITTUS, S. D. ROBERTS, AND J. R. WILSON, *Approximating the distribution of averaged-out utility in probability trees with random parameters*, Research Memorandum No. 87-13, School of Industrial Engineering, Purdue University, West Lafayette, IN, 1987.

[13] W. P. ELDERTON AND N. L. JOHNSON, *Systems of Frequency Curves*, Cambridge University Press, London, 1969.

[14] B. P. KATZ AND S. L. HUI, *Variance estimation for medical decision analysis*, Statist. Med., 8 (1989), pp. 229-241.

[15] K. J. MARSE AND S. D. ROBERTS, *Implementing a portable* FORTRAN *uniform* (0, 1) *generator*, Simulation, 41 (1983), pp. 135-139.

[16] T. M. PASS AND L. P. GOLDSTEIN, *A computerized aid for medical cost-effectiveness analysis*, Med. Decis. Making, 1 (1981), p. 465.

[17] H. RAIFFA, *Decision Analysis: Introductory Lectures on Choices under Uncertainty*, Addison-Wesley, Reading, MA, 1968.

[18] S. D. ROBERTS AND R. W. KLEIN, *The simulation of logical networks* (SLN), Simulation, 43 (1984), pp. 224-232.

[19] M. S. STOUPPE, TREEMAKER: *A tool for constructing probability trees with random parameters*, M.S. thesis, School of Industrial Engineering, Purdue University, West Lafayette, IN, 1988.

[20] A. STUART AND J. K. ORD, *Kendall's Advanced Theory of Statistics, Volume* 1: *Distribution Theory*, Fifth Edition, Oxford University Press, New York, 1987.

[21] D. L. WALLACE, *Asymptotic approximations to distributions*, Ann. Math. Statist., 29 (1958), pp. 635-654.

[22] K. E. WILLARD AND G. C. CRITCHFIELD, *Probabilistic analysis of decision trees using symbolic algebra*, Med. Decis. Making, 6 (1986), pp. 93-100.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1162-1172, September 1991
© 1991 Society for Industrial and Applied Mathematics
009

A STABILIZED TREATMENT OF THE BIHARMONIC OPERATOR WITH SPECTRAL METHODS*

WILHELM HEINRICHS†

Abstract. For the discretization of the biharmonic operator with spectral methods a high condition number like $O(N^8)$, where N is the maximal degree of polynomials, occurs. Two different techniques are proposed for reducing this condition number to $O(N^4)$. The best technique seems to be a splitting of the equation with the biharmonic operator into a system of two equations with the Laplace operator. An effective way of preconditioning is introduced. An effective iterative solver is presented.

Key words. biharmonic operator, spectral methods, stabilized, preconditioning, iterative solver

AMS(MOS) subject classification. 65N35

1. Introduction. For the discretization of the biharmonic operator with spectral methods (Galerkin, tau, or collocation) [5], [8] we obtain a spectral condition number growing as $O(N^8)$, where N is the maximal degree of polynomials. Although in the context of spectral methods the number of degrees of freedom can be taken to be relatively small (compared to finite difference or finite element methods) this high condition number will lead to instabilities caused by rounding errors. For direct solution methods (e.g., the Gauss method) a strong propagation of rounding errors can be expected. For iterative solution techniques (e.g., Richardson iteration or spectral multigrid methods [10], [11], [18], [19]) a very slow convergence rate, which behaves like $1-O(N^{-8})$ for the Richardson iteration, can be expected. For spectral methods it is therefore recommended that preconditioning be used. For second-order problems several techniques of preconditioning are already extensively investigated in the spectral literature (see [5], [8]). For fourth-order problems we investigated preconditioning in [7]. But this technique does not stabilize the evaluation of the biharmonic operator.

In this paper we propose two different techniques for reducing the condition number. The first method (§ 2) was already introduced in [12] for the Laplace operator. The main idea is that we employ polynomials fulfilling the Dirichlet boundary conditions. The biharmonic operator applied to a truncated series of these trial functions is then developed in a series of Chebyshev polynomials. The coefficients of this series are then taken to be equal to the coefficients of the right-hand side expansion. The resulting system that corresponds to the spectral tau method has the improved condition number. We further propose an efficient elimination procedure for the corresponding spectral systems. Subsequently, the system has a block structure and a block-Gauss elimination procedure can be efficiently used for solving. We present numerical results which substantiate the usefulness of our treatment.

The second approach (§ 3) consists of a splitting of the original problem into a system of two equations with the Laplace operator. This is a quite natural idea and was already used in the context of multigrid methods for finite difference schemes (see Brandt [3], [4] and Linden [17]). From the numerical results it becomes obvious that the splitting technique results in a satisfying reduction of the condition number. In comparison to our first treatment we obtain a further improved condition number. In this framework we show that the split system can be efficiently preconditioned with

* Received by the editors June 13, 1989; accepted for publication (in revised form) August 17, 1990.
† Mathematisches Institut der Heinrich-Heine-Universität Düsseldorf, Universitätsstrasse 1, D-4000 Düsseldorf 1, Federal Republic of Germany.

finite differences. Therefore an iterative solver based on this splitting and finite difference preconditioning can be efficiently used for solving our problem. Based on these ideas we intend to construct a spectral multigrid solver for equations with the biharmonic operator.

For further information about spectral methods for the biharmonic problem we refer to Bernardi and Maday [1], [2]; Canuto et al. [5]; and Karageorghis and Phillips [14], [15]. In [2] it is proved that the standard Gauss-Lobatto nodes are not the best choice for the collocation points. Then, they propose a new set of nodes related to some generalized Gauss-type quadrature formulas. Here we employ the Gauss-Lobatto nodes that were also used in [14] and [15]. Numerical examples indicate the high accuracy of the spectral method.

2. A stabilized treatment of the biharmonic operator.

2.1. Spectral tau discretization. We consider the model problem

$$Lu = \Delta^2 u = f \quad \text{in } \Omega = (-1, 1)^2 \tag{2.1}$$

with Dirichlet boundary conditions

$$u = \frac{\partial}{\partial n} u = 0 \quad \text{on } \partial\Omega, \tag{2.2}$$

where $\partial/\partial n$ denotes the outer normal derivative.

For discretization of problem (2.1), (2.2) we use a spectral tau method (see Lanczos [16] and Haidvogel and Zang [9]). In particular, for our problem we employ an approximation of the form

$$u_N(x, y) = (1-x^2)^2(1-y^2)^2 \sum_{p,q=0}^{N} a_{p,q} T_p(x) T_q(y),$$

where $T_p(x) = \cos(p \arccos x)$ $(p = 0, \cdots, N)$ denotes the pth Chebyshev polynomial.

Obviously, u_N fulfills the boundary conditions (2.2). The coefficients are to be determined by equation (2.1). For this purpose we consider

$$\begin{aligned}\Delta^2(1-x^2)^2(1-y^2)^2 T_k(x) T_l(y) \\ = ((1-x^2)^2 T_k(x))^{\mathrm{IV}}(1-y^2)^2 T_l(y) + 2((1-x^2)^2 T_k(x))''((1-y^2)^2 T_l(y))'' \\ + ((1-y^2)^2 T_l(y))^{\mathrm{IV}}(1-x^2)^2 T_k(x).\end{aligned}$$

Hence the expressions $((1-x^2)^2 T_k(x))^{\mathrm{IV}}$, $((1-x^2)^2 T_k(x))''$, and $(1-x^2)^2 T_k(x)$ must be evaluated in a stable way.

For the further investigation we introduce some fundamental results about the representation with Chebyshev series (see also Gottlieb and Orszag [8]).

LEMMA 2.1. *Let* $v_N(x) = \sum_{n=0}^{N} a_n T_n(x)$; *then we have*

$$x v_N(x) = \sum_{n=0}^{N+1} e_n T_n(x),$$

$$(1-x^2) v_N(x) = \sum_{n=0}^{N+2} f_n T_n(x),$$

and

$$v_N'(x) = \sum_{n=0}^{N-1} d_n T_n(x),$$

where

$$e_n = \tfrac{1}{2}(\alpha_{n-1}a_{n-1} + a_{n+1}),$$
$$f_n = -\tfrac{1}{4}\alpha_{n-2}a_{n-2} + (1 - \tfrac{1}{4}(\alpha_n + \alpha_{n-1}))a_n - \tfrac{1}{4}a_{n+2}$$

and

$$d_n = \frac{2}{\alpha_n} \sum_{\substack{p=n+1 \\ p+n \text{ odd}}}^{N} pa_p.$$

Hereby we denote $\alpha_0 = 2$, $\alpha_n = 0$ $(n < 0)$, *and* $\alpha_n = 1$ $(n > 0)$.

In the following we express the relation between $e = (e_n)$, $f = (f_n)$, $d = (d_n)$, and $a = (a_n)$ by matrices $E, F, D \in \mathbb{R}^{N+1,N+1}$, which are constructed such that $e = Ea$, $f = Fa$, and $d = Da$. By using these fundamental matrices, we now construct the spectral tau discretization of (2.1), (2.2). In the following lemma we first give the spectral operators for the one-dimensional case.

LEMMA 2.2. *Let* v_N *be defined as in Lemma* 2.1. *Then it follows that*

$$((1-x^2)v_N(x))^{\mathrm{IV}} = \sum_{n=0}^{N} g_n T_n(x),$$
$$((1-x^2)^2 v_N(x))'' = \sum_{n=0}^{N+2} h_n T_n(x),$$

and

$$(1-x^2)^2 v_N(x) = \sum_{n=0}^{N+4} k_n T_n(x),$$

where $g = (g_n)$, $h = (h_n)$, $k = (k_n)$ *are determined by the relations*

$$g = Ga, \quad h = Ha, \quad k = Ka$$

with

$$G = F^2D^4 - 16EFD^3 + (72E^2 - 24I)D^2 + 96ED + 24I,$$
$$H = F^2D^2 - 8EFD + 12E^2 - 4I,$$

and

$$K = F^2.$$

In the right-hand side series we have neglected the Chebyshev polynomials of degree greater than N.

Proof. The representation of G, H, and K easily follows from

$$((1-x^2)^2 T_k(x))^{\mathrm{IV}} = (1-x^2)^2 T_k^{\mathrm{IV}}(x) - 16x(1-x^2)T_k'''(x) + (72x^2 - 24)T_k''(x) + 96xT_k'(x) + 24T_k(x)$$

and

$$((1-x^2)T_k(x))'' = (1-x^2)^2 T_k''(x) - 8x(1-x^2)T_k'(x) + (12x^2 - 4)T_k(x)$$

by using the standard formulas of Lemma 2.1. □

By using the matrices G, H, and K the spectral tau discretization matrix M for (2.1), (2.2) can be written as

$$M = G \otimes K + 2 \cdot H \otimes H + K \otimes G \quad (\otimes \text{ denotes the tensor product for matrices}).$$

By developing the right-hand side f in a Chebyshev series we obtain

$$f = \sum_{k,l=0}^{N} \hat{f}_{k,l} T_k T_l.$$

With $\hat{f} = (\hat{f}_{k,l})_{k,l=0,\cdots,N}$, the spectral tau problem can now be written as

$$Ma = \hat{f}. \tag{2.3}$$

2.2. Elimination procedure. By using special properties of the spectral matrices we derive a suitable elimination procedure for problem (2.3). First, we observe that the matrices FD^2 and F^2D^3 are upper triangular and the components are columnwise equal (apart from the diagonal components and the components of the first row). Hence we recommend the multiplication of equation (2.3) from the left side by $E1 \otimes E1$, where $E1 = (e1_{k,l})_{k,l=0,\cdots,N}$ with

$$e1_{k,l} = \begin{cases} 2, & l = k = 1, \\ 1, & l = k > 1, \\ -1, & l = k+2, \\ 0 & \text{else.} \end{cases}$$

Hence $E1$ is a sparse matrix that is chosen such that a multiplication by $E1$ causes an elimination in FD^2 and F^2D^3 (since the components are columnwise equal). In order to eliminate the matrix D (from the right side) we recommend the multiplication by $E2 \otimes E2$, where $E2 = (e2_{k,l})_{k,l=0,\cdots,N}$ with

$$e2_{k,l} = \begin{cases} 1, & l = k, \\ -l/(l-2), & l = k+2, \\ 0 & \text{else.} \end{cases}$$

Here $E2$ is a sparse matrix chosen such that the multiplication from the right side causes an elimination in D. This is due to the fact that for $D = (d_{k,l})$,

$$d_{k,l-2} e2_{l-2,l} + d_{k,l} e2_{l,l} = 2(l-2) e2_{l-2,l} + 2l = 0.$$

This elimination procedure yields an elimination matrix

$$\tilde{M} = (E1 \otimes E1) M (E2 \otimes E2)$$

with two lower and four upper nonvanishing diagonals, which are separated by one zero diagonal. Using $\tilde{M}$, problem (2.3) can equivalently be written as

$$\tilde{M}\tilde{a} = \tilde{f}, \tag{2.4}$$

where $\tilde{f} = (E1 \otimes E1) f$ and $\tilde{a} = (E2^{-1} \otimes E2^{-1}) a$.

From (2.4) we observe the even/odd uncoupling of the coefficients $\tilde{a}_{m,n}$. Due to this fact, system (2.4) can be split into four separate systems: n and m even, n odd and m even, n even and m odd, n and m odd. Each of these systems can be solved independently. Due to the band structure of the arising systems, efficient direct band solvers can be used. Their amount of work grows as $O(N^4)$.

TABLE 1

Maximal eigenvalues of M.

N	λ_{max}	λ_{max}/N^4
8	1.22×10^4	2.98
16	1.54×10^5	2.35
32	2.66×10^6	2.54

TABLE 2

Maximal eigenvalues of $\tilde{M}$.

N	λ_{max}	λ_{max}/N^4
8	7.76×10^4	16.49
16	8.89×10^5	13.56
32	1.49×10^7	14.22

TABLE 3

ERR *for example* (2) *in* § 2.3.

k	2	4	8
ERR	3.14×10^{-16}	6.35×10^{-12}	5.32×10^{-4}

In Tables 1 and 2 the condition numbers of M and $\tilde{M}$ are presented. It becomes obvious that these numbers only grow as $O(N^4)$ where the condition number of $\tilde{M}$ is slightly worse than that of M.

2.3. Numerical results. For two examples with exact solution

(1) $u(x, y) = (1-x^2)^2(1-y^2)^2\, e^{x+y}$,

(2) $u(x, y) = (1-x^2)^2(1-y^2)^2 \sin(k\pi x) \sin(k\pi y)$, $k = 2, 4, 8$,

we have numerically calculated the discretization error ERR, defined as

$$\text{ERR} = \frac{1}{N}\left(\sum_{i,j=0}^{N} e_{i,j}^2\right)^{1/2} \quad \text{with error component } e_{i,j},\ i, j = 0, \cdots, N.$$

For example (1) we observe that $\text{ERR} < 10^{-15}$, which corresponds with the double accuracy of our Siemens 7.570-P computer. The errors ERR for example (2) are given in Table 3. The presented results once more show the very high accuracy of our spectral method.

2.4. Variable coefficient problems. For variable coefficient problems we already found in [12] that it is useful to solve the modified singular problem

$$L_s u = g \quad \text{on } \Omega,$$

where

$$L_s = (1-x^2)^2(1-y^2)^2 L, \qquad g = (1-x^2)^2(1-y^2)^2 f.$$

Now it is possible to apply a collocation method. The arising linear systems also have a reduced condition number only growing as $O(N^4)$. For an iterative solution of these spectral systems the preconditioned Richardson iteration is recommended.

For a more detailed description of this method in connection with our problem we once again refer to [12].

2.5. Inhomogeneous boundary conditions. Inhomogeneous boundary conditions are treated in a way similar to that in [12]. We consider the biharmonic problem (2.1) with Dirichlet boundary conditions

$$u = g, \quad \frac{\partial}{\partial n} u = h \quad \text{on } \partial\Omega, \tag{2.5}$$

where g and h denote given functions defined on $\partial\Omega$. For a spectral approximation of problem (2.1), (2.5) the boundary conditions (2.5) are collocated. First, we determine a smooth function u_N^1 that satisfies

$$u_N^1(x_i, y_j) = g(x_i, y_j), \quad \frac{\partial}{\partial n} u_N^1(x_i, y_j) = h(x_i, y_j) \quad \text{for } (x_i, y_j) \in \partial\Omega.$$

Usually, u_N^1 is a suitable polynomial of degree $\leqq N$ in x, y. Furthermore, let u_N^0 denote the spectral tau approximation for the solution u^0 of the problem

$$\Delta^2 u^0 = f - \Delta^2 u_N^1 \quad \text{in } \Omega, \qquad u^0 = \frac{\partial}{\partial n} u^0 = 0 \quad \text{on } \partial\Omega.$$

Hence u_N^0 can be determined by using the stabilizing technique of §§ 2.1 and 2.2. Then $u_N = u_N^1 + u_N^0$ yields a spectral approximation for the solution of problem (2.1), (2.5). The approximation error of this method can be estimated by the interpolation error of u_N^1 on the boundary $\partial\Omega$ and the approximation error of u_N^0.

3. Splitting of the biharmonic operator. For a stabilized treatment of the biharmonic operator with spectral methods we propose splitting it into a system of two equations with the Laplace operator. For this purpose we introduce a function v defined on $\bar{\Omega}$ and now the splitting reads as follows:

$$\Delta u - v = 0 \quad \text{in } \bar{\Omega}, \tag{3.1}$$

$$\Delta v = f \quad \text{in } \Omega. \tag{3.2}$$

For the collocation method we further define nodes:

$$(x_i, y_j) = \left(\cos\frac{i\pi}{N}, \cos\frac{j\pi}{N}\right) \qquad (i, j = 0, \cdots, N).$$

The functions u and v are approximated by polynomials u_N and v_N of the form

$$u_N(x, y) = (1-x^2)^2(1-y^2)^2 \sum_{n,m=0}^{N-4} a_{n,m} T_n(x) T_m(y)$$

and

$$v_N(x, y) = \sum_{n,m=0}^{N} b_{n,m} T_n(x) T_m(y).$$

Hence u_N automatically fulfills the homogeneous boundary conditions. The discrete collocation problem for (3.1), (3.2) now reads as follows.

Find u_N and v_N such that

$$\Delta u_N(x_i, y_j) - v_N(x_i, y_j) = 0 \qquad (i, j = 0, \cdots, N), \tag{3.3}$$

$$\Delta v_N(x_i, y_j) = f(x_i, y_j) \qquad (i, j = 2, \cdots, N-2). \tag{3.4}$$

In (3.3), (3.4) we have $M=(N+1)^2+(N-3)^2$ conditions of collocation for the M unknowns $(a_{n,m})_{n,m=0,\cdots,N-4}$, $(b_{n,m})_{n,m=0,\cdots,N}$. By transforms based on interpolation in the points (x_i, y_j), $i,j=2,\cdots,N-2$, respectively, (x_i, y_j), $i,j=0,\cdots,N$ it is possible to work with a spectral operator applied to $\hat{u}=(u_{N;i,j})_{i,j=2,\cdots,N-2}$ instead of a and $\hat{v}=(v_{N;i,j})_{i,j=0,\cdots,N}$ instead of b.

The spectral problem can now be written as

$$L_{\mathrm{sp}}\begin{pmatrix}\bar{u}\\ \bar{v}\end{pmatrix}=\begin{pmatrix}\bar{0}\\ \bar{f}\end{pmatrix},$$

where

$$\bar{f}=(f_k)_{k=1,\cdots,(N-3)^2}\in\mathbb{R}^{(N-3)^2}$$

with

$$f_k=f(x_i,y_j)\quad\text{for } k=(j-2)(N-3)+i-1,\quad i,j=2,\cdots,N-2,$$

$$\bar{0}\in\mathbb{R}^{(N+1)^2}\quad\text{zero vector},$$

$$L_{\mathrm{sp}}\in\mathbb{R}^{M,M}\quad\text{spectral operator},$$

$$\bar{u}=(u_k)_{k=1,\cdots,(N-3)^2}\in\mathbb{R}^{(N-3)^2}$$

with

$$u_k=(u_N)_{i,j}\quad\text{for } k=(j-2)(N-3)+i-1,\quad i,j=2,\cdots,N-2$$

and

$$\bar{v}=(v_k)_{k=1,\cdots,(N+1)^2}\in\mathbb{R}^{(N+1)^2}$$

with

$$v_k=(v_N)_{i,j}\quad\text{for } k=j(N+1)+i+1,\quad i,j=0,\cdots,N.$$

In a similar way it is possible to derive the finite difference problem

$$L_{\mathrm{FD}}\begin{pmatrix}\bar{u}\\ \bar{v}\end{pmatrix}=\begin{pmatrix}\bar{0}\\ \bar{f}\end{pmatrix}.$$

Here L_{FD} is constructed without using outer points since an introduction of outer points (as done, e.g., in Linden [17]) yields too many unknowns. In order to fulfill the boundary conditions we set

$$u_{i,j}=0\quad\text{if } i\in[0,1,N-1,N]\quad\text{or}\quad j\in\{0,1,N-1,N\},$$

i.e., the values in boundary points and the points next to the boundary are set to zero.

The latter setting realizes only a first-order approximation of the Neumann boundary conditions but since the points are dense near the boundary (with steplength of $O(N^{-2})$) we still obtain a second-order approximation in N. By this setting we reduce the number of unknowns to M. The construction of the finite difference operator can be taken by tensor products from the one-dimensional finite difference approximation for $Lu=u''$. In the gridpoints of Ω we choose the standard second-order finite difference approximation for Lu. It can be represented by means of a three-point star as follows:

$$Lu(x_i)\triangleq-\frac{2}{x_{i+1}-x_{i-1}}\left[\frac{1}{x_i-x_{i-1}},\frac{1}{x_i-x_{i-1}}-\frac{1}{x_{i+1}-x_i},\frac{1}{x_{i+1}-x_i}\right]u$$

for $i=1,\cdots,N-1$.

In order to fulfill (3.1) we must also introduce a finite difference approximation for Lu in the boundary points. We consider $x_0 = 1$. By introducing an outer point $x_{-1} = 2 - \cos(\pi/N)$ and then eliminating this point by means of the Neumann boundary condition we obtain in $x = x_0$

$$u''(x_0) \hat{=} q \cdot u(x_0) - q \cdot u(x_1),$$

where

$$q = -\frac{2}{x_1 - x_{-1}} \cdot \left(\frac{1}{x_0 - x_{-1}} + \frac{1}{x_1 - x_0} \right).$$

Since we have set $u(x_0) = u(x_1) = 0$ we obtain $u''(x_0) = 0$ for the finite difference approximation u.

In Table 4 we present the maximal eigenvalues of L_{sp}. These eigenvalues are calculated by means of the vector iteration. In comparison to the treatment of § 2 we observe much smaller maximal eigenvalues. Hence the technique of splitting the biharmonic operator seems to be the desirable way to reduce the condition number.

In Table 5 we give the minimal and maximal eigenvalues λ_{min} and λ_{max} of L_{sp} for $N = 4, 8, 12$. We calculated the eigenvalues by using standard EISPACK routines. It can be seen that some eigenvalues in the middle of the spectrum are complex. Furthermore, it becomes obvious that the quotient $\lambda_{max}/\lambda_{min}$ now behaves like N^4 multiplied by a small constant. In Table 6 we present the minimal and maximal eigenvalues of the preconditioned operator $L_{FD}^{-1} L_{sp}$. Some eigenvalues in the middle of the spectrum of eigenvalues (around 1) are complex with small imaginary part. The

TABLE 4

Maximal eigenvalues of L_{sp}.

N	λ_{max}	λ_{max}/N^4
8	1.38×10^2	0.034
16	1.56×10^3	0.024
32	2.37×10^4	0.023

TABLE 5

Minimal and maximal eigenvalues of L_{sp}.

N	λ_{min}	λ_{max}	$\lambda_{max}/\lambda_{min} N^4$
4	0.802	1.16×10^1	0.056
8	0.768	1.38×10^2	0.044
12	0.731	5.40×10^2	0.036

TABLE 6

Minimal and maximal eigenvalues of $L_{FD}^{-1} L_{sp}$.

N	λ_{min}	λ_{max}	$\lambda_{max}/\lambda_{min}$
4	0.462	2.00	4.33
8	0.346	2.84	8.21
12	0.321	3.25	10.12

ratio $\lambda_{\max}/\lambda_{\min}$ is slightly increasing in N but the preconditioning seems to be very effective.

We remark that a further improved preconditioning can be achieved by using bilinear finite elements (see [6], [13]).

4. Iterative solver. Since the exact solution of the finite difference problems is too expensive we recommend an iterative procedure (see [10], [11], [18], [19]). There exist two different strategies for the iterative approximation: incomplete LU (ILU)-decomposition and Gauss-Seidel-line relaxation. In [10] we compare both techniques and the line relaxation method turned out to be the most effective strategy. The ILU method requires twice as much work for the initial decomposition.

Now we consider the finite difference problem:

$$L_{\mathrm{FD}}\begin{pmatrix}\vec{u}\\ \vec{v}\end{pmatrix}=\begin{pmatrix}r^1\\ r^2\end{pmatrix},$$

where $r^1\in\mathbb{R}^{(N+1)^2}$, $r^2\in\mathbb{R}^{(N-1)^2}$ denote the spectral residuals. Furthermore, we introduce index sets of boundary points:

$$I_b^1=\{(i,j)\colon i\in\{0,N\},\,0\leqq j\leqq N \text{ or } j\in\{0,N\},\,0\leqq i\leqq N$$
$$\text{or } (i,j)\in\{(1,1),(1,N-1),(N-1,1),(N-1,N-1)\}\}$$

(boundary points)

and

$$I_b^2=\{(i,j)\colon i\in\{1,N-1\},\,2\leqq j\leqq N-2 \text{ or } j\in\{1,N-1\},\,2\leqq i\leqq N-2\}$$

(points next to the boundary)

and

$$I_b=I_b^1\cup I_b^2.$$

The five-point finite difference star for the Laplace operator Δ is written by

$$(\Delta_{\mathrm{FD}}u_N)_{i,j}=\begin{bmatrix}0 & f_{0,1}^{i,j} & 0\\ f_{-1,0}^{i,j} & f_{0,0}^{i,j} & f_{1,0}^{i,j}\\ 0 & f_{0,-1}^{i,j} & 0\end{bmatrix}u_N \quad \text{for } 1\leqq i,j\leqq N-1.$$

Let us denote by u_N^0, v_N^0 the grid function before the relaxation sweep and by u_N^n, v_N^n the grid function after the relaxation sweep. By using $\Delta_{\mathrm{FD}}u_N-v_N=r^1$ in I_b we set, before relaxation,

$$(v_N^n)_{i,j}=-r_{i,j}^1 \quad \text{if } (i,j)\in I_b^1$$

and for $(i,j)\in I_b^2$ we set, for instance,

$$(v_N^n)_{i,j}=f_{1,0}^{1,j}u_{2,j}^0-r_{1,j}^1 \quad \text{for } 2\leqq j\leqq N-2,$$
$$(v_N^n)_{N-1,j}=f_{-1,0}^{N-1,j}u_{N-2,j}^0-r_{N-1,j}^1 \quad \text{for } 2\leqq j\leqq N-2.$$

The same settings are made for $j\in\{1,N-1\}$, $2\leqq i\leqq N-2$. Now we must use a relaxation technique for only the inner points for $2\leqq i,j\leqq N-2$. We must approximate the discrete system:

$$(\Delta_{\mathrm{FD}}u_N-v_N)_{i,j}=r_{i,j}^1,$$
$$(\Delta_{\mathrm{FD}}v_N)_{i,j}=r_{i,j}^2 \quad \text{for } 2\leqq i,j\leqq N-2.$$

For this purpose we choose a line relaxation technique. We first relax along the lines

of the constant x by solving for each $j \in \{2, 4, \cdots, N-2\}$:

$$f^{i,j}_{-1,0}(u^n_N)_{i-1,j} + f^{i,j}_{0,0}(u^n_N)_{i,j} + f^{i,j}_{1,0}(u^n_N)_{i+1,j} - (v^n_N)_{i,j}$$
$$= r^1_{i,j} - f^{i,j}_{0,-1}(u^0_N)_{i,j-1} - f^{i,j}_{0,1}(u^0_N)_{i,j+1},$$
$$f^{i,j}_{-1,0}(v^n_N)_{i-1,j} + f^{i,j}_{0,0}(v^n_N)_{i,j} + f^{i,j}_{1,0}(v^n_N)_{i+1,j}$$
$$= r^2_{i,j} - f^{i,j}_{0,-1}(v^0_N)_{i,j-1} - f^{i,j}_{0,1}(v^0_N)_{i,j+1} \qquad (i = 2, 3, \cdots, N-2)$$

for the new grid functions u^n, v^n. Then the same sweep follows for the odd $j \in \{3, 5, \cdots, N-3\}$. Afterwards an analogous sweep is performed along the lines of the constant y. Hence we obtain a $2(N-3) \times 2(N-3)$-system of equations. We can split the corresponding matrix into four submatrices (see Fig. 1). The matrices on the diagonal are tridiagonal. The upper matrix is diagonal and the lower matrix is identical to the zero matrix. This system can efficiently be solved by some appropriate Gaussian elimination procedure.

Instead of setting the values of v in I_b it is possible to eliminate these values by means of the corresponding values of u. This changes the structure of the system above and now the lower zero matrix is replaced by a matrix with entries in the diagonal. However, this change did not significantly improve the preconditioning.

In Table 7 we present the minimal and maximal eigenvalues of the preconditioned system. We compare NIT = 1 and NIT = 10 steps of preconditioning. It becomes obvious that it is not efficient to employ more than one iteration. However, the preconditioning by means of the line relaxation method yields satisfying results.

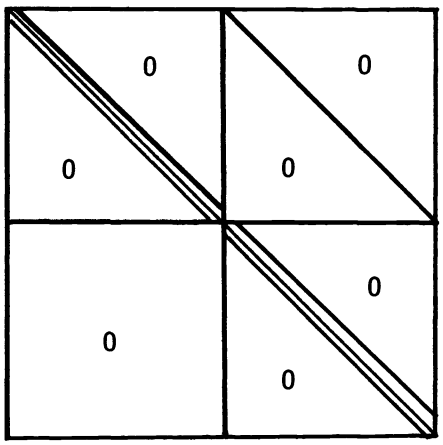


FIG. 1. *Structure of the matrices.*

TABLE 7

Minimal and maximal eigenvalues of the iteratively preconditioned bilaplace operator.

	$\lambda_{\min}$		$\lambda_{\max}$	
N	NIT = 1	NIT = 10	NIT = 1	NIT = 10
4	0.4040	0.4617	2.000	2.000
8	0.3919	0.3461	3.094	2.869
12	0.3255	0.3213	3.319	3.246
16	0.2485	0.3077	3.374	3.398

Unfortunately, we were not successful in finding an effective multigrid method. If we use the relaxation above in a standard multigrid context (see [10], [18]) we obtain very bad convergence factors of about 0.8–0.9. At the moment it is not clear why multigrid methods do not work. We observed that the coarse grid corrections are not good enough. In a future work we will try to clarify these observations.

5. Conclusion. In this paper we present two different techniques to reduce the condition number of the spectral operator for the biharmonic equation. The desirable way results from a splitting of the fourth-order problem into a system of two second-order problems. The resulting spectral system has a bad condition number. We propose a suitable way of finite difference preconditioning. By means of preconditioning, the convergence speed of an iterative method can be dramatically improved. The corresponding iterative method can be employed in a multigrid context.

This work yields a first step in finding an effective solver for the Navier-Stokes equations in streamfunction formulation (see [5, 11.3.4]).

REFERENCES

[1] C. BERNARDI AND Y. MADAY, *Spectral methods for the approximation of fourth-order problems: Applications to the Stokes and Navier-Stokes equations*, Comput. & Structures, 30 (1988), pp. 205–216.

[2] ———, *Some spectral approximations of one-dimensional fourth-order problems*, ICASE Report No. 89-36, ICASE, NASA-Langley Research Center, Hampton, VA, 1989.

[3] A. BRANDT, *Multi-level adaptive solutions to boundary value problems*, Math. Comp., 31 (1977), pp. 333–390.

[4] ———, *Multigrid techniques*: 1984 *guide with applications to fluid dynamics*, GMD-Studies No. 85, St. Augustin, Gesellschaft für Mathematik und Datenverarbeitung, Germany, 1984.

[5] C. CANUTO, M. Y. HUSSAINI, A. QUARTERONI, AND T. A. ZANG, *Spectral Methods in Fluid Dynamics*, Series of Computational Physics, Springer-Verlag, Heidelberg, Berlin, New York, 1989.

[6] C. CANUTO AND P. PIETRA, *Boundary and interface conditions within a finite element preconditioned for spectral methods*, Report Istituto di Analisi Numerica del C.N.R. di Pavia, No. 555, Pavia, Italy; J. Comput. Phys., to appear.

[7] D. FUNARO AND W. HEINRICHS, *Some results about the pseudospectral approximation of one-dimensional fourth-order problems*, Numer. Math., 58 (1990), pp. 399–418.

[8] D. GOTTLIEB AND S. A. ORSZAG, *Numerical Analysis of Spectral Methods: Theory and Applications*, CBMS-NSF Regional Conference Series in Applied Mathematics 36, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1977.

[9] D. B. HAIDVOGEL AND T. A. ZANG, *An accurate solution of Poisson's equation by expansion in Chebyshev polynomials*, J. Comput. Phys., 30 (1979), pp. 167–180.

[10] W. HEINRICHS, *Line relaxation for spectral multigrid methods*, J. Comput. Phys., 77 (1988), pp. 166–182.

[11] ———, *Multigrid methods for combined finite difference and Fourier problems*, J. Comput. Phys., 78 (1988), pp. 424–436.

[12] ———, *Improved condition number for spectral methods*, Math. Comp., 53 (1989), pp. 103–119.

[13] ———, *Finite element preconditioning for spectral multigrid methods*, Appl. Math. Comp., to appear.

[14] A. KARAGEORGHIS AND T. N. PHILLIPS, *Efficient direct methods for solving the spectral collocation equations for Stokes flow in rectangularly decomposable domains*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 89–103.

[15] ———, *Spectral collocation methods for Stokes flow in contraction geometries and unbounded domains*, J. Comput. Phys., 80 (1989), pp. 314–330.

[16] C. LANCZOS, *Applied Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1956.

[17] J. LINDEN, *Mehrgitterverfahren für das erste Randwertproblem der biharmonischen Gleichung und Anwendung auf ein inkompressibles Strömungsproblem*, GMD-Bericht No. 164, R. Oldenbourg Verlag, Munich, Vienna, 1986.

[18] T. A. ZANG, Y. S. WONG, AND M. Y. HUSSAINI, *Spectral multigrid methods for elliptic equations* I, J. Comput. Phys., 48 (1982), pp. 485–501.

[19] ———, *Spectral multigrid methods for elliptic equations* II, J. Comput. Phys., 54 (1984), pp. 489–507.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1173–1183, September 1991

A SCHEME FOR HANDLING RANK-DEFICIENCY IN THE SOLUTION OF SPARSE LINEAR LEAST SQUARES PROBLEMS*

ESMOND NG†

Abstract. Several schemes for computing sparse orthogonal factorizations using static data structures have been proposed recently. One novel feature of some of these schemes is that the data structures are large enough to store both the orthogonal transformations and upper triangular factor explicitly. However, in order to make use of the static storage schemes, the orthogonal factorization has to be computed without column interchanges, which is sufficient when the observation matrix has full rank. When the least squares matrix is rank-deficient, computing the minimum-norm solution to the least squares problem requires the knowledge of the rank of the matrix, which may be difficult to get if the factorization is computed without column pivoting. In this paper an algorithm is developed that makes use of the resulting factorization to solve rank-deficient least squares problems. The new algorithm has three major features. First, it uses the original triangular factor. Second, this factor is not altered. Third, the rank decision is based on the singular value decomposition and not on the diagonal elements of the triangular factor. The techniques used are similar to those employed by Björck.

Key words. sparse least squares problems, sparse orthogonal factorization, updating algorithms, rank-deficiency

AMS(MOS) subject classification. 65F

1. Introduction.

In this paper we consider the direct solution of a linear least squares problem

$$\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2,$$

where $\boldsymbol{A}$ is a sparse $m \times n$ matrix, with $m \geq n$, and $\boldsymbol{b}$ is an m-vector. The solution scheme is based on an orthogonal decomposition of $\boldsymbol{A}$:

$$\boldsymbol{A} = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{pmatrix},$$

where $\boldsymbol{Q}$ is an $m \times m$ orthogonal matrix and $\boldsymbol{R}$ is an $n \times n$ upper triangular matrix. When $\boldsymbol{A}$ has full rank, $\boldsymbol{R}$ is nonsingular and the solution to the least squares problem is obtained by solving $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{c}$, where $\boldsymbol{c}$ contains the first n components of $\boldsymbol{Q}^T\boldsymbol{b}$.

There are several methods for computing the orthogonal decomposition of $\boldsymbol{A}$ when it is sparse and has full rank [11]. In [2], George and Heath described a novel way of computing the upper triangular factor $\boldsymbol{R}$, which is based on the observation that $\boldsymbol{R}$ is mathematically the Cholesky factor of $\boldsymbol{A}^T\boldsymbol{A}$ (apart from possible sign differences in some rows). If $\boldsymbol{A}^T\boldsymbol{A}$ is sparse, techniques developed for solving sparse symmetric positive-definite systems can be employed to preserve and exploit the sparsity of $\boldsymbol{R}$. More precisely, a symbolic Cholesky factorization algorithm can be applied to $\boldsymbol{A}^T\boldsymbol{A}$ to obtain the structure of $\boldsymbol{R}$. This allows a storage scheme to be set up for storing the nonzeros of $\boldsymbol{R}$ [3]. Using the fixed (or static) storage scheme, the upper triangular factor $\boldsymbol{R}$ is then obtained by annihilating the rows of $\boldsymbol{A}$ using Givens rotations. Since

* Received by the editors February 13, 1989; accepted for publication (in revised form) September 24, 1990. This research was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy contract DE-AC-05-84OR21400 with Martin Marietta Energy Systems Inc.

† Mathematical Sciences Section, Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, Tennessee 37831-6367.

the symbolic factorization algorithm does not predict the positions of the rotations, a static data structure cannot be set up to store the rotations. (It was shown in [7] that the symbolic factorization algorithm on $\boldsymbol{A}^T\boldsymbol{A}$ can be modified to predict the positions of the rotations, although the resulting data structures may be too generous.) In [2], the rotations are applied to the vector $\boldsymbol{b}$ during the annihilation process and are not saved. It is well known that the sparsity of $\boldsymbol{R}$ depends not only on the structure of $\boldsymbol{A}^T\boldsymbol{A}$, but also on the ordering of the columns and rows of $\boldsymbol{A}^T\boldsymbol{A}$. Thus, a fill-reducing symmetric ordering for $\boldsymbol{A}^T\boldsymbol{A}$ should be chosen to preserve the sparsity of $\boldsymbol{R}$. See [3] for details. Note that reordering the columns and rows of $\boldsymbol{A}^T\boldsymbol{A}$ symmetrically is equivalent to reordering the columns of $\boldsymbol{A}$.

In [8], George and Ng described an efficient symbolic factorization algorithm that computes the structure of $\boldsymbol{R}$ directly from that of $\boldsymbol{A}$. If $\boldsymbol{Q}$ is a sequence of Householder transformations, the symbolic factorization algorithm also produces the structure of the transformations. Thus, not only can a static data structure be determined for storing the nonzeros of $\boldsymbol{R}$, but a fixed storage scheme can also be set up for storing the nonzeros of the Householder transformations. In [4], George, Liu, and Ng proposed an efficient way of storing and computing $\boldsymbol{R}$ and the Householder transformations. It should be noted that the sparsity of $\boldsymbol{R}$ and the Householder transformations depend on both the structure of $\boldsymbol{A}$ and the initial column ordering.

To summarize, there are efficient methods for reducing a sparse matrix to upper triangular form using orthogonal transformations. However, the column ordering is often chosen beforehand to preserve sparsity of the upper triangular factor $\boldsymbol{R}$. This implies that if a static data structure is used, the orthogonal decomposition must be computed without any column interchanges during the numerical computation phase, since column pivoting will change the sparsity of $\boldsymbol{R}$. Hence, the approaches in [2], [4], and [8] work well when the matrix $\boldsymbol{A}$ has full rank and is well conditioned.

We now consider the case in which $\boldsymbol{A}$ is rank-deficient; that is, we assume that $\boldsymbol{A}$ has well-defined numerical rank (in the sense that there is a well-determined gap between large and small singular values). Suppose $\boldsymbol{R}$ is obtained from $\boldsymbol{A}$ without column pivoting. In exact arithmetic, some of the diagonal elements of $\boldsymbol{R}$ will be zero. However, for those rows whose diagonal elements are zero, the off-diagonal elements need not be zero. Hence, the rank-deficiency of $\boldsymbol{A}$ and $\boldsymbol{R}$ cannot be determined simply by counting the number of zeros on the diagonal in $\boldsymbol{R}$. The situation becomes fuzzier in the presence of round-off errors. Instead of having exact zeros on the diagonal, it is likely that some of the diagonal elements of $\boldsymbol{R}$ will be close to zero or small in magnitude compared to the remaining diagonal elements.

In [10], Heath proposed a technique for handling rank-deficiency in the basic method described in [2]. Let δ be a user-specified tolerance, the choice of which will depend on the magnitude of the diagonal elements of $\boldsymbol{R}$ and the machine precision. The basic idea is to use additional Givens rotations to annihilate the offdiagonal nonzeros in any rows of $\boldsymbol{R}$ whose diagonal elements are less than δ in magnitude. When the annihilation process is completed, any rows in the resulting upper triangular matrix will be regarded as null if their diagonal elements are less than δ in magnitude. Using the resulting upper triangular matrix, Heath developed an algorithm for computing the minimum-norm solution to the least squares problem. In [1], Björck extended the ideas in [10] and developed an algorithm for solving a general sparse constrained linear least squares problem in which the least squares and constraint matrices can have arbitrary ranks.

For the approaches described in [4] and [8], Heath's approach can be employed

to compute the minimum-norm solution as well, so long as only one right-hand side vector $\boldsymbol{b}$ has to be solved, or when numerous right-hand sides are available at the same time. If several right-hand sides are to be solved and if each has to be processed one after another, then the Givens rotations needed to process the upper triangular factor $\boldsymbol{R}$ must be saved. However, the static data structures described in [4] and [8] have space to store only the nonzeros of $\boldsymbol{Q}$ (represented as a sequence of Householder transformations); they do not have space to store the extra transformations needed to postprocess $\boldsymbol{R}$. Thus, we need an algorithm for solving a rank-deficient problem that uses the *original* upper triangular factor obtained in the initial orthogonal factorization without changing this factor. The objective of this note is to describe such an algorithm, in which rows of $\boldsymbol{R}$ with diagonal elements less than δ in magnitude are treated as updates to the problem. Rank-deficiency is determined using the singular value decomposition during the updating process and not based on the magnitude of the diagonal elements of $\boldsymbol{R}$. The techniques are similar to those employed in [1]. An outline of the paper is as follows. In §2, the updating algorithm is derived. Some implementation issues are presented in §3. Finally, some concluding remarks are given in §4.

2. The updating algorithm. Consider the linear least squares problem

$$\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2, \tag{1}$$

where $\boldsymbol{A}$ is large and sparse. Recall that the upper triangular factor $\boldsymbol{R}$ is mathematically the Cholesky factor of $\boldsymbol{A}^T\boldsymbol{A}$. If some of the rows of $\boldsymbol{A}$ are relatively dense, then $\boldsymbol{A}^T\boldsymbol{A}$, and consequently $\boldsymbol{R}$, may be dense. In order to preserve sparsity, it may be desirable to withhold any dense rows from the initial sparse orthogonal factorization. Instead of solving (1), the sparse orthogonal decomposition is used to solve a smaller least squares problem. Then any dense rows that were withheld can be used to update the solution.

The technique of handling dense rows is useful in another context. It is quite often the case that after a least squares problem has been solved, more equations become available. For large and sparse problems, if the number of new equations is small, it is desirable to be able to update the solution to the first least squares problem using the new equations to obtain the new solution.

With the discussion above, we assume that $\boldsymbol{A}$ and $\boldsymbol{b}$ are partitioned, respectively, into $\begin{pmatrix} \boldsymbol{C} \\ \boldsymbol{D} \end{pmatrix}$ and $\begin{pmatrix} \boldsymbol{e} \\ \boldsymbol{f} \end{pmatrix}$. The matrix $\boldsymbol{C}$ is assumed to be large and sparse, but $\boldsymbol{D}$ is small and may be dense. Let $\boldsymbol{C}$ be $m \times n$ and $\boldsymbol{D}$ be $p \times n$. We assume that $n \leq m$ and $p << n$. These assumptions on m, n, and p are not essential for the correctness of the algorithm we will develop, but they are vital for its efficiency. We do not make any assumptions on the ranks of $\boldsymbol{C}$ and $\boldsymbol{A}$. However, for efficiency, it is assumed that the rank of $\boldsymbol{C}$ is close to n. If $\boldsymbol{A}$ is rank-deficient, (1) does not have a unique solution, and in that case, the minimum-norm solution is computed.

We assume that a sparse orthogonal factorization of $\boldsymbol{C}$ has been computed using, for example, the approach and the static storage scheme described in [4] or [8]:

$$\boldsymbol{C} = \boldsymbol{Q}\begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{pmatrix}. \tag{2}$$

Here $\boldsymbol{Q}$ is $m \times m$ and orthogonal, and $\boldsymbol{R}$ is $n \times n$ and upper triangular. As we pointed out in the previous section, an a priori column ordering of $\boldsymbol{C}$ is chosen to preserve sparsity in $\boldsymbol{Q}$ and $\boldsymbol{R}$. Thus, the orthogonal decomposition of $\boldsymbol{C}$ is computed

with a fixed column ordering. It is well known that, in floating-point arithmetic, some of the diagonal elements in the upper triangular factor $\boldsymbol{R}$ are likely to be close to zero or small in magnitude compared to the remaining diagonal elements if $\boldsymbol{C}$ is rank-deficient. Furthermore, since the orthogonal factorization is computed without column interchanges, rows with small diagonal elements (in terms of magnitude) may occur anywhere in $\boldsymbol{R}$. Without loss of generality, we assume that $\boldsymbol{R}$ and $\boldsymbol{D}$ are partitioned into

$$\begin{array}{rl} & \begin{array}{ccc} n_1 & n_2 & n_3 \end{array} \\ \boldsymbol{R} = & \begin{pmatrix} \boldsymbol{R}_1 & \boldsymbol{S}_{12} & \boldsymbol{S}_{13} \\ \boldsymbol{O} & \boldsymbol{R}_2 & \boldsymbol{S}_{23} \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{R}_3 \end{pmatrix} \begin{array}{l} n_1 \\ n_2, \\ n_3 \end{array} \\ & \begin{array}{ccc} n_1 & n_2 & n_3 \end{array} \\ \boldsymbol{D} = & \begin{pmatrix} \boldsymbol{D}_1 & \boldsymbol{D}_2 & \boldsymbol{D}_3 \end{pmatrix} \quad p. \end{array} \tag{3}$$

Here $\boldsymbol{R}_1$, $\boldsymbol{R}_2$, and $\boldsymbol{R}_3$ are upper triangular, and their orders are, respectively, n_1, n_2, and n_3. Thus, $n_1 + n_2 + n_3 = n$. We assume that the diagonal elements of $\boldsymbol{R}_2$ are less than δ in magnitude, but those of $\boldsymbol{R}_1$ and $\boldsymbol{R}_3$ are larger than δ in magnitude, where $\delta \geq 0$ is some tolerance dependent on the magnitudes of the diagonal elements of $\boldsymbol{R}_1$ and $\boldsymbol{R}_3$ and the machine precision. That is, we assume that $\boldsymbol{R}_1$ and $\boldsymbol{R}_3$ are nonsingular, but $\boldsymbol{R}_2$ may be rank deficient. Note that the offdiagonal nonzeros in $\boldsymbol{R}_2$ and the nonzeros in $\boldsymbol{S}_{23}$ need not be small, and hence the rank-deficiency of $\boldsymbol{C}$ may be less than the order of $\boldsymbol{R}_2$.

In practice, the structure of $\boldsymbol{R}$ would not have the form given in (3). However, for efficient implementation of the algorithm, all one needs is the ability to distinguish those rows in $\boldsymbol{R}_2$ from the remaining rows in $\boldsymbol{R}$. This can be achieved by maintaining a flag for each row so that the flag is set if the corresponding row has a small diagonal element. The cost of determining δ and searching for small diagonal elements is relatively small since the diagonal elements of $\boldsymbol{R}$ can be stored in a one-dimensional floating-point array, as suggested in [3] and [4].

To compute the minimum-norm solution to (1) using the orthogonal decomposition given by (2) and (3), we proceed as follows. First note that

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 = \left\| \begin{pmatrix} \boldsymbol{C} \\ \boldsymbol{D} \end{pmatrix} \boldsymbol{x} - \begin{pmatrix} \boldsymbol{e} \\ \boldsymbol{f} \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} \boldsymbol{R}_1 & \boldsymbol{S}_{12} & \boldsymbol{S}_{13} \\ \boldsymbol{O} & \boldsymbol{R}_2 & \boldsymbol{S}_{23} \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{R}_3 \\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{O} \\ \boldsymbol{D}_1 & \boldsymbol{D}_2 & \boldsymbol{D}_3 \end{pmatrix} \boldsymbol{x} - \begin{pmatrix} \boldsymbol{Q}^T\boldsymbol{e} \\ \boldsymbol{f} \end{pmatrix} \right\|_2.$$

Let $\boldsymbol{x} = \begin{pmatrix} \boldsymbol{u}^T & \boldsymbol{v}^T & \boldsymbol{w}^T \end{pmatrix}^T$ and $\boldsymbol{Q}^T\boldsymbol{e} = \begin{pmatrix} \boldsymbol{e}_1^T & \boldsymbol{e}_2^T & \boldsymbol{e}_3^T & \boldsymbol{e}_4^T \end{pmatrix}^T$ be partitioned accordingly. Then

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 = \left\| \begin{pmatrix} \boldsymbol{R}_1\boldsymbol{u} + \boldsymbol{S}_{12}\boldsymbol{v} + \boldsymbol{S}_{13}\boldsymbol{w} - \boldsymbol{e}_1 \\ \boldsymbol{R}_2\boldsymbol{v} + \boldsymbol{S}_{23}\boldsymbol{w} - \boldsymbol{e}_2 \\ \boldsymbol{R}_3\boldsymbol{w} - \boldsymbol{e}_3 \\ -\boldsymbol{e}_4 \\ \boldsymbol{D}_1\boldsymbol{u} + \boldsymbol{D}_2\boldsymbol{v} + \boldsymbol{D}_3\boldsymbol{w} - \boldsymbol{f} \end{pmatrix} \right\|_2.$$

Hence, $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$ is minimized when $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$ are chosen to minimize

$$\begin{aligned} &\|\boldsymbol{R}_1\boldsymbol{u} + \boldsymbol{S}_{12}\boldsymbol{v} + \boldsymbol{S}_{13}\boldsymbol{w} - \boldsymbol{e}_1\|_2^2 + \|\boldsymbol{R}_2\boldsymbol{v} + \boldsymbol{S}_{23}\boldsymbol{w} - \boldsymbol{e}_2\|_2^2 \\ &\quad + \|\boldsymbol{R}_3\boldsymbol{w} - \boldsymbol{e}_3\|_2^2 + \|\boldsymbol{D}_1\boldsymbol{u} + \boldsymbol{D}_2\boldsymbol{v} + \boldsymbol{D}_3\boldsymbol{w} - \boldsymbol{f}\|_2^2. \end{aligned} \tag{4}$$

If a unique solution does not exist (in the case when $\boldsymbol{A}$ is rank-deficient), we will choose $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$ so that (4) and $\|\boldsymbol{x}\|_2$ are both minimized.

We define the residuals $\boldsymbol{r}_1$, $\boldsymbol{r}_2$, $\boldsymbol{r}_3$, and $\boldsymbol{r}_D$ by

$$\boldsymbol{r}_1 = \boldsymbol{R}_1\boldsymbol{u} + \boldsymbol{S}_{12}\boldsymbol{v} + \boldsymbol{S}_{13}\boldsymbol{w} - \boldsymbol{e}_1, \tag{5}$$

$$\boldsymbol{r}_2 = \boldsymbol{R}_2\boldsymbol{v} + \boldsymbol{S}_{23}\boldsymbol{w} - \boldsymbol{e}_2, \tag{6}$$

$$\boldsymbol{r}_3 = \boldsymbol{R}_3\boldsymbol{w} - \boldsymbol{e}_3, \tag{7}$$

$$\boldsymbol{r}_D = \boldsymbol{D}_1\boldsymbol{u} + \boldsymbol{D}_2\boldsymbol{v} + \boldsymbol{D}_3\boldsymbol{w} - \boldsymbol{f}. \tag{8}$$

We use (7) and (5) to obtain formulas for $\boldsymbol{w}$ and $\boldsymbol{u}$ in terms of $\boldsymbol{v}$ and the residual vectors:

$$\boldsymbol{w} = \boldsymbol{R}_3^{-1}\boldsymbol{r}_3 + \boldsymbol{R}_3^{-1}\boldsymbol{e}_3 = \boldsymbol{R}_3^{-1}\boldsymbol{r}_3 + \hat{\boldsymbol{e}}_3. \tag{9}$$

$$\begin{aligned}\boldsymbol{u} &= \boldsymbol{R}_1^{-1}(\boldsymbol{r}_1 - \boldsymbol{S}_{12}\boldsymbol{v} - \boldsymbol{S}_{13}\boldsymbol{w} + \boldsymbol{e}_1)\\ &= \boldsymbol{R}_1^{-1}\boldsymbol{r}_1 - \boldsymbol{R}_1^{-1}\boldsymbol{S}_{13}\boldsymbol{R}_3^{-1}\boldsymbol{r}_3 - \boldsymbol{R}_1^{-1}\boldsymbol{S}_{12}\boldsymbol{v} + \boldsymbol{R}_1^{-1}(\boldsymbol{e}_1 - \boldsymbol{S}_{13}\hat{\boldsymbol{e}}_3)\\ &= \boldsymbol{R}_1^{-1}\boldsymbol{r}_1 - \boldsymbol{R}_1^{-1}\boldsymbol{S}_{13}\boldsymbol{R}_3^{-1}\boldsymbol{r}_3 - \boldsymbol{R}_1^{-1}\boldsymbol{S}_{12}\boldsymbol{v} + \hat{\boldsymbol{e}}_1.\end{aligned} \tag{10}$$

Now we substitute these formulas into (6) and (8) to obtain n_2+p equations involving the $n+p$ residual variables plus the unknown vector $\boldsymbol{v}$.

$$\begin{aligned}\boldsymbol{r}_2 &= \boldsymbol{S}_{23}\boldsymbol{R}_3^{-1}\boldsymbol{r}_3 + \boldsymbol{R}_2\boldsymbol{v} - (\boldsymbol{e}_2 - \boldsymbol{S}_{23}\hat{\boldsymbol{e}}_3)\\ &= \boldsymbol{S}_{23}\boldsymbol{R}_3^{-1}\boldsymbol{r}_3 + \boldsymbol{R}_2\boldsymbol{v} - \hat{\boldsymbol{e}}_2,\end{aligned} \tag{11}$$

$$\begin{aligned}\boldsymbol{r}_D &= \boldsymbol{D}_1\boldsymbol{R}_1^{-1}\boldsymbol{r}_1 + (\boldsymbol{D}_3\boldsymbol{R}_3^{-1} - \boldsymbol{D}_1\boldsymbol{R}_1^{-1}\boldsymbol{S}_{13}\boldsymbol{R}_3^{-1})\boldsymbol{r}_3\\ &\quad + (\boldsymbol{D}_2 - \boldsymbol{D}_1\boldsymbol{R}_1^{-1}\boldsymbol{S}_{12})\boldsymbol{v} - (\boldsymbol{f} - \boldsymbol{D}_1\hat{\boldsymbol{e}}_1 - \boldsymbol{D}_3\hat{\boldsymbol{e}}_3)\\ &= \boldsymbol{D}_1\boldsymbol{R}_1^{-1}\boldsymbol{r}_1 + \hat{\boldsymbol{D}}_3\boldsymbol{r}_3 + \hat{\boldsymbol{D}}_2\boldsymbol{v} - \hat{\boldsymbol{f}}.\end{aligned} \tag{12}$$

We rewrite this in matrix form:

$$\boldsymbol{M}\boldsymbol{r} = \boldsymbol{K}\boldsymbol{v} - \boldsymbol{g}, \tag{13}$$

where $\boldsymbol{M}$, $\boldsymbol{K}$, $\boldsymbol{r}$, and $\boldsymbol{g}$ are defined below.

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{O} & \boldsymbol{I} & -\boldsymbol{S}_{23}\boldsymbol{R}_3^{-1} & \boldsymbol{O} \\ -\boldsymbol{D}_1\boldsymbol{R}_1^{-1} & \boldsymbol{O} & -\hat{\boldsymbol{D}}_3 & \boldsymbol{I} \end{pmatrix}, \qquad \boldsymbol{K} = \begin{pmatrix} \boldsymbol{R}_2 \\ \hat{\boldsymbol{D}}_2 \end{pmatrix},$$

$$\boldsymbol{r} = \begin{pmatrix} \boldsymbol{r}_1 \\ \boldsymbol{r}_2 \\ \boldsymbol{r}_3 \\ \boldsymbol{r}_D \end{pmatrix}, \qquad \boldsymbol{g} = \begin{pmatrix} \hat{\boldsymbol{e}}_2 \\ \hat{\boldsymbol{f}} \end{pmatrix}.$$

Since (13) is a full-rank underdetermined system, an orthogonal factorization can be used to obtain a formula for $\boldsymbol{v}$, as the following discussion shows.

Denote an orthogonal decomposition of $\boldsymbol{M}$ by

$$\boldsymbol{M}^T = \boldsymbol{Q}_L^T \begin{pmatrix} \boldsymbol{L}^T \\ \boldsymbol{O} \end{pmatrix},$$

where $\boldsymbol{Q}_L^T$ is orthogonal and $\boldsymbol{L}^T$ is upper triangular and nonsingular. Using the orthogonal decomposition of $\boldsymbol{M}$ and writing $\boldsymbol{r}$ as $\boldsymbol{Q}_L^T\begin{pmatrix}\boldsymbol{s}\\ \boldsymbol{t}\end{pmatrix}$, (13) becomes

$$\begin{pmatrix}\boldsymbol{L} & \boldsymbol{O}\end{pmatrix}\begin{pmatrix}\boldsymbol{s}\\ \boldsymbol{t}\end{pmatrix} = \boldsymbol{K}\boldsymbol{v} - \boldsymbol{g}.$$

Thus $\boldsymbol{s} = \boldsymbol{L}^{-1}\boldsymbol{K}\boldsymbol{v} - \boldsymbol{L}^{-1}\boldsymbol{g}$ and $\boldsymbol{t}$ is arbitrary. The minimum-norm residual vector $\boldsymbol{r}$ is obtained by setting $\boldsymbol{t} = \boldsymbol{0}$ and by choosing $\boldsymbol{v}$ to be a solution to the following dense linear least squares problem

$$\min_{\boldsymbol{v}} \left\|\boldsymbol{L}^{-1}\boldsymbol{K}\boldsymbol{v} - \boldsymbol{L}^{-1}\boldsymbol{g}\right\|_2. \tag{14}$$

The coefficient matrix $\boldsymbol{L}^{-1}\boldsymbol{K}$ is $(n_2 + p) \times n_2$ and may be rank-deficient. A reliable way to solve (14) is to use the singular value decomposition of $\boldsymbol{L}^{-1}\boldsymbol{K}$. This determines $\boldsymbol{r}$ and, through (9), $\boldsymbol{w}$. It should be noted that this is the only place in the algorithm where rank-deficiency has to be determined.

Let $\bar{\boldsymbol{v}}$ be any solution to (14). Furthermore, suppose the columns of $\boldsymbol{N}$ form a basis of the null space of $\boldsymbol{L}^{-1}\boldsymbol{K}$. The matrix $\boldsymbol{N}$ can be obtained easily from the singular value decomposition of $\boldsymbol{L}^{-1}\boldsymbol{K}$. Thus the general solution to (14) is given by $\boldsymbol{v} = \bar{\boldsymbol{v}} + \boldsymbol{N}\boldsymbol{z}$, where $\boldsymbol{z}$ is arbitrary. We now obtain a formula for $\boldsymbol{u}$ in terms of $\boldsymbol{z}$ by substituting $\boldsymbol{v}$, $\boldsymbol{w}$, and $\boldsymbol{r}_1$ into (5):

$$\begin{aligned}\boldsymbol{u} &= \boldsymbol{R}_1^{-1}(\boldsymbol{r}_1 + \boldsymbol{e}_1 - \boldsymbol{S}_{12}\boldsymbol{v} - \boldsymbol{S}_{13}\boldsymbol{w})\\ &= -\boldsymbol{R}_1^{-1}\boldsymbol{S}_{12}\boldsymbol{N}\boldsymbol{z} + \boldsymbol{R}_1^{-1}(\boldsymbol{e}_1 - \boldsymbol{S}_{12}\bar{\boldsymbol{v}} - \boldsymbol{S}_{13}\boldsymbol{w} + \boldsymbol{r}_1)\\ &= -\boldsymbol{R}_1^{-1}\boldsymbol{S}_{12}\boldsymbol{N}\boldsymbol{z} + \bar{\boldsymbol{e}}_1.\end{aligned}$$

Since $\boldsymbol{u}$ and $\boldsymbol{v}$ both depend on $\boldsymbol{z}$, we can choose $\boldsymbol{z}$ to minimize $\left\|\begin{pmatrix}\boldsymbol{u}\\ \boldsymbol{v}\end{pmatrix}\right\|_2$:

$$\min_{\boldsymbol{z}} \left\|\begin{pmatrix}-\boldsymbol{R}_1^{-1}\boldsymbol{S}_{12}\boldsymbol{N}\\ \boldsymbol{N}\end{pmatrix}\boldsymbol{z} + \begin{pmatrix}\bar{\boldsymbol{e}}_1\\ \bar{\boldsymbol{v}}\end{pmatrix}\right\|_2. \tag{15}$$

This gives us a minimum-norm solution to our original least squares problem.

3. Implementation details. In the algorithm described in the previous section, we have to deal with different portions of the upper triangular factor $\boldsymbol{R}$ obtained in the orthogonal factorization of $\boldsymbol{C}$ at various stages. Since $\boldsymbol{R}$ is normally stored in some compact data structure, the implementation of the algorithm appears to be complicated. We will demonstrate below that the implementation can actually be done cleanly.

We first define $\hat{\boldsymbol{R}}$ to be the upper triangular matrix

$$\hat{\boldsymbol{R}} = \begin{pmatrix}\boldsymbol{R}_1 & \boldsymbol{S}_{12} & \boldsymbol{S}_{13}\\ \boldsymbol{O} & \boldsymbol{I} & \boldsymbol{S}_{23}\\ \boldsymbol{O} & \boldsymbol{O} & \boldsymbol{R}_3\end{pmatrix},$$

which can be obtained easily by scanning the data structure for the upper triangular factor $\boldsymbol{R}$ and replacing $\boldsymbol{R}_2$ by the identity matrix. We will assume that $\boldsymbol{R}_2$ is saved elsewhere. In the following discussion, we will make use of partitioned vectors and matrices frequently and we assume that the partitionings are consistent with the row partitionings in $\boldsymbol{R}$ and $\hat{\boldsymbol{R}}$.

The first three quantities we need are

$$\hat{\boldsymbol{e}}_3 = \boldsymbol{R}_3^{-1}\boldsymbol{e}_3, \qquad \hat{\boldsymbol{e}}_2 = \boldsymbol{e}_2 - \boldsymbol{S}_{23}\hat{\boldsymbol{e}}_3, \qquad \hat{\boldsymbol{e}}_1 = \boldsymbol{R}_1^{-1}(\boldsymbol{e}_1 - \boldsymbol{S}_{13}\hat{\boldsymbol{e}}_3).$$

Define $\boldsymbol{c} = \begin{pmatrix} \boldsymbol{e}_1^T & \boldsymbol{e}_2^T & \boldsymbol{e}_3^T \end{pmatrix}^T$. Then $\hat{\boldsymbol{e}}_1$, $\hat{\boldsymbol{e}}_2$, and $\hat{\boldsymbol{e}}_3$ can be obtained by solving $\hat{\boldsymbol{R}}\boldsymbol{d} = \boldsymbol{c}$ using a modified backward substitution algorithm, which is described below. Let $\boldsymbol{d} = \begin{pmatrix} \boldsymbol{d}_1^T & \boldsymbol{d}_2^T & \boldsymbol{d}_3^T \end{pmatrix}^T$. Solving the last $n_2 + n_3$ equations of $\hat{\boldsymbol{R}}\boldsymbol{d} = \boldsymbol{c}$ gives

$$\begin{aligned} \boldsymbol{d}_3 &= \boldsymbol{R}_3^{-1}\boldsymbol{e}_3 = \hat{\boldsymbol{e}}_3, \\ \boldsymbol{d}_2 &= \boldsymbol{e}_2 - \boldsymbol{S}_{23}\boldsymbol{d}_3 = \boldsymbol{e}_2 - \boldsymbol{S}_{23}\hat{\boldsymbol{e}}_3 = \hat{\boldsymbol{e}}_2. \end{aligned}$$

Now we save $\boldsymbol{d}_2$ elsewhere and set $\boldsymbol{d}_2 = \boldsymbol{0}$. The solution of the first n_1 equations would then produce

$$\boldsymbol{d}_1 = \boldsymbol{R}_1^{-1}(\boldsymbol{e}_1 - \boldsymbol{S}_{13}\boldsymbol{d}_3) = \boldsymbol{R}_1^{-1}(\boldsymbol{e}_1 - \boldsymbol{S}_{13}\hat{\boldsymbol{e}}_3) = \hat{\boldsymbol{e}}_1.$$

Thus $\boldsymbol{d} = \begin{pmatrix} \hat{\boldsymbol{e}}_1^T & \boldsymbol{0} & \hat{\boldsymbol{e}}_3^T \end{pmatrix}^T$.

Next we need the matrices

$$\boldsymbol{D}_1\boldsymbol{R}_1^{-1}, \qquad \hat{\boldsymbol{D}}_2 = \boldsymbol{D}_2 - \boldsymbol{D}_1\boldsymbol{R}_1^{-1}\boldsymbol{S}_{12}, \qquad \hat{\boldsymbol{D}}_3 = \boldsymbol{D}_3\boldsymbol{R}_3^{-1} - \boldsymbol{D}_1\boldsymbol{R}_1^{-1}\boldsymbol{S}_{13}\boldsymbol{R}_3^{-1}.$$

These matrices can be computed using an approach similar to the modified backward substitution algorithm described above. Consider the p lower triangular systems $\hat{\boldsymbol{R}}^T\boldsymbol{E} = \boldsymbol{D}^T$. Let $\boldsymbol{E} = \begin{pmatrix} \boldsymbol{E}_1^T & \boldsymbol{E}_2^T & \boldsymbol{E}_3^T \end{pmatrix}^T$. Solving the first $n_1 + n_2$ equations yields

$$\begin{aligned} \boldsymbol{E}_1 &= \boldsymbol{R}_1^{-T}\boldsymbol{D}_1^T, \\ \boldsymbol{E}_2 &= \boldsymbol{D}_2^T - \boldsymbol{S}_{12}^T\boldsymbol{E}_1 = \boldsymbol{D}_2^T - \boldsymbol{S}_{12}^T\boldsymbol{R}_1^{-T}\boldsymbol{D}_1^T = \hat{\boldsymbol{D}}_2^T. \end{aligned}$$

Now we save $\boldsymbol{E}_2$, set $\boldsymbol{E}_2 = \boldsymbol{O}$, and continue the forward substitution. Then

$$\boldsymbol{E}_3 = \boldsymbol{R}_3^{-T}(\boldsymbol{D}_3^T - \boldsymbol{S}_{13}^T\boldsymbol{E}_1) = \boldsymbol{R}_3^{-T}(\boldsymbol{D}_3^T - \boldsymbol{S}_{13}^T\boldsymbol{R}_1^{-T}\boldsymbol{D}_1^T) = \hat{\boldsymbol{D}}_3^T.$$

Thus, $\boldsymbol{E} = \begin{pmatrix} \boldsymbol{D}_1\boldsymbol{R}_1^{-1} & \boldsymbol{O} & \hat{\boldsymbol{D}}_3 \end{pmatrix}^T$ at the end of the modified forward substitution process.

The vector $\hat{\boldsymbol{f}}$ in (12) is obtained by computing $\boldsymbol{f} - \boldsymbol{D}\boldsymbol{d}$. To form $\boldsymbol{M}$ in (13), we need the matrix $-\boldsymbol{S}_{23}\boldsymbol{R}_3^{-1}$, which can be obtained by solving n_2 lower triangular systems $\hat{\boldsymbol{R}}^T\boldsymbol{F} = \hat{\boldsymbol{I}}$, with $\hat{\boldsymbol{I}} = \begin{pmatrix} \boldsymbol{O} & \boldsymbol{I} & \boldsymbol{O} \end{pmatrix}^T$. If $\boldsymbol{F} = \begin{pmatrix} \boldsymbol{F}_1^T & \boldsymbol{F}_2^T & \boldsymbol{F}_3^T \end{pmatrix}^T$, then it is easy to see that $\boldsymbol{F}_3^T = -\boldsymbol{S}_{23}\boldsymbol{R}_3^{-1}$. The matrix $\boldsymbol{K}$ is easy to construct since both $\boldsymbol{R}_2$ and $\hat{\boldsymbol{D}}_2$ have been saved.

Finally, we have to compute the matrix

$$\begin{pmatrix} -\boldsymbol{R}_1^{-1}\boldsymbol{S}_{12}\boldsymbol{N} \\ \boldsymbol{N} \end{pmatrix}$$

and the vector

$$\begin{pmatrix} \bar{\boldsymbol{e}}_1 \\ \bar{\boldsymbol{v}} \end{pmatrix}$$

in order to solve (15). The matrix can be obtained by solving the upper triangular systems $\hat{\boldsymbol{R}}\boldsymbol{G} = \hat{\boldsymbol{N}}$, where $\hat{\boldsymbol{N}} = \begin{pmatrix} \boldsymbol{O} & \boldsymbol{N}^T & \boldsymbol{O} \end{pmatrix}^T$. Note that the solution is

$$\boldsymbol{G} = \begin{pmatrix} -(\boldsymbol{R}_1^{-1}\boldsymbol{S}_{12}\boldsymbol{N})^T & \boldsymbol{N}^T & \boldsymbol{O} \end{pmatrix}^T.$$

To compute

$$\begin{pmatrix} \bar{e}_1 \\ \bar{v} \end{pmatrix},$$

we proceed as follows. After the solution of (13), we construct the vector $\hat{r} = \begin{pmatrix} r_1^T & \mathbf{o} & r_3^T \end{pmatrix}^T$ and solve the upper triangular system $\hat{R}h = \hat{r}$ using a strategy similar to the modified backward substitution described above. Let $h = \begin{pmatrix} h_1^T & h_2^T & h_2^T \end{pmatrix}^T$. The solution of the last $n_2 + n_3$ equations gives $h_3 = R_3^{-1} r_3$ and $h_2 = -S_{23} h_3$. Now we replace h_2 by $\bar{v}$, which is a solution to (14). Then the solution of the first n_1 equations will give

$$h_1 = R_1^{-1}(r_1 - S_{12}\bar{v} - S_{13}R_3^{-1}r_3).$$

Note that

$$h_3 + d_3 = R_3^{-1} r_3 + R_3^{-1} e_3 = w.$$

Also, $h_2 + d_2 = \bar{v}$. Furthermore,

$$\begin{aligned} h_1 + d_1 &= R_1^{-1}(r_1 - S_{12}\bar{v} - S_{13}R_3^{-1}r_3) + \hat{e}_1 \\ &= R_1^{-1}(r_1 - S_{12}\bar{v} - S_{13}R_3^{-1}r_3) + R_1^{-1}(e_1 - S_{13}R_3^{-1}e_3), \end{aligned}$$

which is the same as $\bar{e}_1$.

We now summarize the algorithm step by step below. We also indicate the time complexity at each step (in terms of the number of multiplicative operations required); the operation counts for some standard matrix operations are obtained from [9] and only high-order terms are retained.

1. Compute a sparse orthogonal decomposition of C.

$$C = Q \begin{pmatrix} R_1 & S_{12} & S_{13} \\ O & R_2 & S_{23} \\ O & O & R_3 \\ O & O & O \end{pmatrix}.$$

   It is generally impossible to give the time complexity for this step, since it is problem-dependent. Thus, we will use τ_C to denote the number of operations required to compute the orthogonal factorization.
2. Save R_2 and construct $\hat{R}$. Denote by σ the number of nonzeros (including the diagonal) in $\hat{R}$.
3. Compute $F_3^T = -S_{23}R_3^{-1}$ by solving $\hat{R}^T F = \hat{I}$, where $\hat{I} = \begin{pmatrix} O & I & O \end{pmatrix}^T$. The time complexity for this step is σn_2, since there are n_2 triangular systems to solve.
4. Use the modified forward substitution algorithm described above to "solve" $\hat{R}^T E = D^T$ to obtain $E = \begin{pmatrix} D_1 R_1^{-1} & O & \hat{D}_3 \end{pmatrix}^T$ and $\hat{D}_2$. The time complexity for this step is σp, since D is $p \times n$.
5. Construct $K = \begin{pmatrix} R_2 \\ \hat{D}_2 \end{pmatrix}$ and

$$M = \begin{pmatrix} O & I & -S_{23}R_3^{-1} & O \\ -D_1 R_1^{-1} & O & -\hat{D}_3 & I \end{pmatrix} = \begin{pmatrix} F^T & O \\ -E^T & I \end{pmatrix}.$$

Compute an orthogonal decomposition of $\boldsymbol{M}^T = \boldsymbol{Q}_L^T \begin{pmatrix} \boldsymbol{L}^T \\ \boldsymbol{O} \end{pmatrix}$. Since $\boldsymbol{M}^T$ is $(n+p)\times(n_2+p)$, the number of operations required to compute the orthogonal decomposition is $(n+p)(n_2+p)^2 - \frac{1}{3}(n_2+p)^3$.

6. Form the $(n_2+p)\times n_2$ matrix $\boldsymbol{L}^{-1}\boldsymbol{K}$ and compute the singular value decomposition of $\boldsymbol{L}^{-1}\boldsymbol{K}$. Determine $\boldsymbol{N}$, whose columns form a basis of the null space of $\boldsymbol{L}^{-1}\boldsymbol{K}$. The time complexity for the triangular solutions and the singular value decomposition is approximately
$$\left[\frac{1}{2}(n_2+p)^2 n_2\right] + \left[2(n_2+p)^2 n_2 + 4(n_2+p)n_2^2 + \frac{14}{3}n_2^3\right].$$
7. Solve $\hat{\boldsymbol{R}}\boldsymbol{G} = \hat{\boldsymbol{N}}$, where $\hat{\boldsymbol{N}} = \begin{pmatrix} \boldsymbol{O} & \boldsymbol{N}^T & \boldsymbol{O} \end{pmatrix}^T$, to obtain
$$\boldsymbol{G} = \begin{pmatrix} -(\boldsymbol{R}_1^{-1}\boldsymbol{S}_{12}\boldsymbol{N})^T & \boldsymbol{N}^T & \boldsymbol{O} \end{pmatrix}^T.$$
Compute an orthogonal decomposition of $\boldsymbol{G}$. Since $\boldsymbol{N}$ is at most $n_2 \times n_2$, computing $\boldsymbol{G}$ requires at most σn_2 operations. Similarly, the orthogonal decomposition of $\boldsymbol{G}$ requires at most $nn_2^2 - \frac{1}{3}n_2^3$ operations. Thus, the time complexity for this step is bounded by $\sigma n_2 + nn_2^2 - \frac{1}{3}n_2^3$.
8. Form $\boldsymbol{Q}^T\boldsymbol{e}$. Let $\boldsymbol{c}$ be the first n components of $\boldsymbol{Q}^T\boldsymbol{e}$. Here $\boldsymbol{Q}$ is the orthogonal matrix we obtain at step 1, which is in factored form. As in step 1, it is difficult to say how many operations are required in forming $\boldsymbol{Q}^T\boldsymbol{e}$. Hence, we will use τ_Q to denote the time complexity at this step.
9. "Solve" $\hat{\boldsymbol{R}}\boldsymbol{d} = \boldsymbol{c}$ in the way described above to obtain $\boldsymbol{d} = \begin{pmatrix} \hat{\boldsymbol{e}}_1^T & \mathbf{o} & \hat{\boldsymbol{e}}_3^T \end{pmatrix}^T$ and $\hat{\boldsymbol{e}}_2$. The time complexity at this step is σ.
10. Compute $\hat{\boldsymbol{f}} = \boldsymbol{f} - \boldsymbol{D}\boldsymbol{d}$. The number of operations required to form $\hat{\boldsymbol{f}}$ is pn.
11. Compute
$$\boldsymbol{L}^{-1}\boldsymbol{g} = \boldsymbol{L}^{-1}\begin{pmatrix} \hat{\boldsymbol{e}}_2 \\ \hat{\boldsymbol{f}} \end{pmatrix}.$$
Since $\boldsymbol{L}$ is $(n_2+p)\times(n_2+p)$, the time complexity for this step is $\frac{1}{2}(n_2+p)^2$.
12. Solve the linear least squares problem $\min_{\boldsymbol{v}} \|\boldsymbol{L}^{-1}\boldsymbol{K}\boldsymbol{v} - \boldsymbol{L}^{-1}\boldsymbol{g}\|_2$ using the singular value decomposition computed at step 6. Let $\bar{\boldsymbol{v}}$ be any solution. Determine the minimum-norm residual vector $\boldsymbol{s}$. The number of operations required at this step is approximately $2(n_2+p)^2 + n_2^2$.
13. Compute the minimum-norm residual vector of the original problem $\boldsymbol{r} = \boldsymbol{Q}_L^T \begin{pmatrix} \boldsymbol{s} \\ \mathbf{o} \end{pmatrix}$. Let $\boldsymbol{r} = \begin{pmatrix} \boldsymbol{r}_1^T & \boldsymbol{r}_2^T & \boldsymbol{r}_3^T & \boldsymbol{r}_D^T \end{pmatrix}^T$. The time complexity for this step is $[2(n+p) - (n_2+p)](n_2+p)$.
14. Construct $\hat{\boldsymbol{r}} = \begin{pmatrix} \boldsymbol{r}_1^T & \mathbf{o} & \boldsymbol{r}_3^T \end{pmatrix}^T$ and "solve" $\hat{\boldsymbol{R}}\boldsymbol{h} = \hat{\boldsymbol{r}}$ as described above. The number of operations required in the solution of the triangular system is σ.
15. Compute $\boldsymbol{h} + \boldsymbol{d}$. At this point, $(\boldsymbol{h}+\boldsymbol{d}) = \begin{pmatrix} \bar{\boldsymbol{e}}_1^T & \bar{\boldsymbol{v}}^T & \boldsymbol{w}^T \end{pmatrix}^T$.
16. Solve the linear least squares problem $\min_{\boldsymbol{z}} \|\boldsymbol{G}\boldsymbol{z} - (\boldsymbol{h}+\boldsymbol{d})\|_2$ using the orthogonal decomposition computed at step 7. The minimum-norm residual vector is the minimum-norm solution to $\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$. The time complexity for solving the least squares problem is at most $2(2n - n_2)n_2$.

In the case when several vectors $\boldsymbol{b}$ are to be solved, steps 1–7 have to be performed only once. Steps 8–16 have to be repeated for each different $\boldsymbol{b}$. When new equations are added, the matrix $\boldsymbol{D}$ changes. In this case, steps 4–16 have to be repeated.

Note that computing the orthogonal decomposition of $\boldsymbol{C}$ in step 1 is usually the most expensive step. Thus, if n is large and p and n_2 are small, and if a sparse orthogonal factorization of $\boldsymbol{C}$ has been computed, then the computation basically involves the solutions of a few sparse triangular systems and a few small dense problems. When a single least squares problem is solved, the time complexity of the entire algorithm is given by

$$\tau_C + \tau_Q + \sigma(2n_2 + p + 2) + n\left(2n_2^2 + p^2 + 2n_2p + 6n_2 + 3p\right) + O(n_2^3 + p^3 + n_2^2p + n_2p^2). \tag{16}$$

The first three terms are, respectively, the times for computing the orthogonal decomposition of $\boldsymbol{C}$ in step 1, for applying the transformations to $\boldsymbol{e}$ in step 8, and for solving various linear systems using the upper triangular matrix $\hat{\boldsymbol{R}}$. The fourth term in (16) is essentially the time for computing the orthogonal decompositions of $\boldsymbol{M}$ and $\boldsymbol{G}$.

Assuming p and n_2 are small, the storage requirement is dominated by the space for $\boldsymbol{Q}$ and $\boldsymbol{R}$, as the following discussion shows. As in the time complexity analysis, it is generally difficult to say how much storage is required by $\boldsymbol{Q}$ and $\boldsymbol{R}$. Thus, we simply denote it by θ_C. In the updating algorithm, we need space for the matrices $\boldsymbol{K}$ and $\boldsymbol{M}$, the dimensions of which are, respectively, $(n_2+p)\times n_2$ and $(n_2+p)\times(n+p)$. We also need space for $\boldsymbol{G}$, the dimension of which is at most $n\times n_2$. Moreover, we have to store the singular value decomposition of $\boldsymbol{K}$ and various vectors throughout the algorithm. It is easy to see that the total storage requirement is given approximately by

$$\theta_C + n(2n_2 + p) + O(n + n_2^2 + p^2 + n_2p). \tag{17}$$

4. Conclusion. In this paper, we have derived an algorithm for solving a rank-deficient sparse least squares problem using an orthogonal factorization of the least squares matrix that is computed without any column interchanges. We have also presented techniques for implementing the algorithm. Once the initial sparse orthogonal factorization has been computed, the solution can be obtained by solving a few sparse triangular systems of equations and some dense matrix problems. The algorithm described in this paper will be effective if the order of $\boldsymbol{R}_2$ (i.e., n_2) and the number of rows in $\boldsymbol{D}$ (i.e., p) are small. An efficient implementation of the algorithm will be available in a future release of SPARSPAK [5], [6].

If the approach described in [4] or [8] is employed to compute the sparse orthogonal decomposition, then enough information can be saved easily and explicitly in a static data structure so that multiple problems with the same least squares matrix can be handled effectively. This feature is not shared by the approach described in [1].

The new algorithm is very similar to that described in [1]. One important difference between the two is in the determination and handling of rank-deficiency. In [1], rank-deficiency is determined using a scheme proposed by Heath [10], in which the rank-deficiency is obtained by counting the number of diagonal elements of $\boldsymbol{R}$ that are less than δ in magnitude (after some postprocessing of $\boldsymbol{R}$). The choice of a suitable δ is important and is a delicate issue. In our algorithm, rows of the original upper triangular factor $\boldsymbol{R}$ with diagonal elements less than δ in magnitude are treated as row updates and the determination of rank-deficiency is delayed until the singular value decomposition of $\boldsymbol{L}^{-1}\boldsymbol{K}$ is computed. Thus the criterion for choosing δ is not as crucial in our algorithm as in Heath's approach, and the choice can be quite lib-

eral. Another important difference is that the initial sparse orthogonal factorization is unchanged throughout the computation in our approach.

Under the assumption that n_2 is small, our scheme and Björck's approach perform approximately the same amount of floating-point arithmetic. It is worthy to note that $\boldsymbol{R}_2$ and $\boldsymbol{S}_{23}$ are both zero in Björck's scheme. It is hard to compare storage requirements. When $\boldsymbol{R}_2$ is small, both our new algorithm and Björck's algorithm require roughly the same amount of storage for handling the dense subproblems. However, our sparse data structure will take up more space than that in [1], since we assume that $\boldsymbol{Q}$ is saved.

REFERENCES

[1] A. BJÖRCK, *A general updating algorithm for constrained linear least squares problems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 394–402.

[2] J. GEORGE AND M. HEATH, *Solution of sparse linear least squares problems using Givens rotations*, Linear Algebra Appl., 34 (1980), pp. 69–83.

[3] J. GEORGE AND J. W.-H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice–Hall, Englewood Cliffs, NJ, 1981.

[4] J. GEORGE, J. W.-H. LIU, AND E. G.-Y. NG, *A data structure for sparse* QR *and* LU *factors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 100–121.

[5] J. GEORGE AND E. G.-Y. NG, *A brief description of* SPARSPAK, *Waterloo sparse linear equation package*, ACM SIGNUM Newsletter, 16 (1982), pp. 17–20.

[6] ———, *A new release of* SPARSPAK—*The Waterloo sparse matrix package*, ACM SIGNUM Newsletter, 19 (1984), pp. 9–13.

[7] ———, *Orthogonal reduction of sparse matrices to upper triangular form using Householder transformations*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 460–472.

[8] ———, *Symbolic factorization for sparse Gaussian elimination with partial pivoting*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 877–898.

[9] G. GOLUB AND C. V. LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[10] M. HEATH, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 223–237.

[11] ———, *Numerical methods for large sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 26 (1984), pp. 497–513.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1184–1197, September 1991

LIMITING COMMUNICATION IN PARALLEL SPARSE CHOLESKY FACTORIZATION*

LAURIE HULBERT† AND EARL ZMIJEWSKI†

Abstract. A new parallel algorithm for computing the Cholesky factorization of a large sparse positive-definite matrix on a message-passing multiprocessor is developed. The algorithm attempts to reduce the communication overhead by redistributing the computational load and by repeatedly combining the effect of many messages into a single message. It is demonstrated experimentally that, for problems ordered and partitioned among the processors using nested dissection, the new algorithm communicates significantly fewer messages than a more straightforward approach. Because of this reduction in communication, for the test problems on an Intel iPSC/2 hypercube, the new algorithm is typically at least 20 percent faster. Theoretically, it is shown that in factoring a $k \times k$ grid on p processors, the new algorithm sends $\Theta(pk \log_2 p)$ messages, compared to $\Theta(pk \log_2 k)$ messages for the straightforward algorithm.

Key words. sparse Cholesky, parallel factorization, message-passing multiprocessors, hypercubes

AMS(MOS) subject classifications. 65F05, 65F50, 65W05

1. Introduction.

In this paper, we introduce a new column-oriented parallel algorithm for factoring an $n \times n$ large sparse symmetric positive-definite matrix A on a message-passing multiprocessor. Our algorithm is designed for a class of message-passing multiprocessors typified by the currently available hypercube machines of Ametek, Intel, and NCUBE. These machines consist of several identical processors, each containing a large local memory; there is no global memory. The processors coordinate their activities by passing messages along a network of communication links. Our only assumption about the topology of the communication network is that any processor can communicate efficiently with any other processor.

All of the message-passing multiprocessors currently on the market have large communication-to-computation ratios, i.e., it takes significantly longer to pass a byte of data between two processors than it does to compute a floating point operation on that byte. As these machines mature, this ratio is likely to *increase* rather than decrease. The reason for this is simple: it is a lot easier and faster to compute a floating point operation on a chip than it is to send a message down a wire connecting two processors that may be as much as several feet apart. For example, Intel's iPSC/1 hypercube that was introduced several years ago has a communication-to-computation ratio of about 26; for Intel's newly introduced iPSC/2 this ratio has increased to 80. (Although communication on the iPSC/2 is about twice as fast as the iPSC/1, computation is five to six times faster [20].) Without a radical change in architecture (e.g., using optical message passing), it is likely that this ratio will continue to increase before eventually leveling off at some high value. To avoid having the communication costs dominate the total running time, we must structure our computations carefully. In addition, the higher the message-passing frequency, the larger the message buffer each processor requires. These buffers consume space that could be used to store data, and hence, reducing communication may not only reduce execution time, but may also increase the size of the problems that can be solved.

* Received by the editors June 26, 1989; accepted for publication (in revised form) September 10, 1990.

† James Madison University, Department of Mathematics and Computer Science, Harrisonburg, Virginia 22807.

For sparse matrix problems, limiting communication is even more critical than for dense ones. For each column of a sparse matrix, there are typically only a few nonzeros and thus very few floating point operations are required per column. In a column-oriented factorization, each message will contain a single sparse column. Thus, with a high communication-to-computation ratio, good speedups can be obtained only if a small fraction of the columns are passed between processors.

Factoring the matrix A is one step in the solution of a linear system of equations of the form $Ax = b$. To solve for x, we can employ the following steps. First, reorder the initial matrix A to allow for parallelism and low fill. Next, set up the data structures needed to efficiently compute the Cholesky factor L of A. Typically this involves computing the elimination tree of A and allocating storage for L by symbolically factoring A. Using these data structures, numerically factor A and then find x by solving the triangular systems $Ly = b$ and $L^T x = y$. (For an introduction to sequential sparse matrix factorization, see Duff, Erisman, and Reid [3]; George and Liu [11]; or Pissanetzky [21]. George et al. [9] and Gilbert and Zmijewski [14] discuss parallel algorithms that order the matrix so that the subsequent stages of the computation can also be performed in parallel. Liu [18] provides an extensive discussion of elimination trees.)

For message-passing multiprocessors, there are at least two codes that perform each of these steps in parallel. Gilbert and Zmijewski [14], [22], [23] developed one and implemented it on Intel's first hypercube, the iPSC/1. At Oak Ridge National Laboratories, Geist and Ng [5] and George et al. [9] developed a different code and their current version runs on Intel's newest hypercube, the iPSC/2. We used this more up-to-date code in our experiments. In what follows, we will assume that we have already ordered and partitioned the matrix by columns among the p processors of a message-passing multiprocessor and that we have computed all the data structures needed to efficiently factor L by columns. In computing the factorization, the Oak Ridge code sends messages that consist of single columns of L to the processors that need them. We have modified this algorithm to reduce this communication overhead by repeatedly combining the effect of many messages into a single message; i.e., instead of sending single columns, we send appropriate linear combinations of them. (Ashcraft et al. [1], and Ashcraft, Eisenstat, and Liu [2] have independently proposed what they call *fan-in* algorithms that employ this same basic idea in a somewhat different way.) For simplicity, we will refer to the Oak Ridge factorization algorithm as the *old algorithm* and our modified version of it as the *new algorithm*.

George, Liu, and Ng [12] have analyzed the old factorization algorithm on a $k \times k$ grid graph. Using a nested dissection ordering to partition the subgrids onto subcubes of a p processor hypercube, they have shown that the processors will communicate $O(pk^2)$ nonzeros during the numeric factorization. Assuming that the old algorithm is used and the load is balanced, they further proved that this is asymptotically the minimum *volume* of communication that must occur for any ordering of the grid and any assignment of nonzeros to processors. However, the communication volume alone does not determine the total communication time. On message-passing machines such as the Intel hypercube, there is a significant startup cost associated with each message. Thus, the time required for the communication depends on both the message volume and the total number of messages. In fact, since each message contains only a single sparse column, the communication time depends primarily on the number of messages passed. In this paper, we present a new numeric factorization algorithm that will in general pass fewer messages than the old algorithm. For the $k \times k$ grid, we show that

the old algorithm sends $\Theta(pk \log k)$ messages, compared to $\Theta(pk \log p)$ messages for the new algorithm.[1] Furthermore, the new algorithm results in the same volume of communication.

Our paper is organized as follows. Section 2 reviews parallel column Cholesky factorization and the old factorization algorithm. Section 3 introduces the new factorization algorithm and discusses its implementation. Section 4 provides some analysis of both algorithms for square grids. Section 5 contains our numerical results for square grids and L-shaped triangular meshes (graded L's) on an Intel iPSC/2 hypercube. Section 6 contains conclusions and discusses some possible improvements to the new algorithm. Throughout this paper, we assume the reader is familiar with basic graph theoretic terms [16] and with the graph theoretic model of Cholesky factorization [3], [11], [21].

2. Parallel column Cholesky factorization. There are many ways of computing the Cholesky factorization of a *dense* symmetric matrix [11]. One method is known as *column* or *inner product Cholesky.* In this scheme, the jth column is determined by first subtracting multiples of columns 1 through $j-1$ of L from the jth column of A. The resulting column is then divided by the square root of its diagonal element to obtain the jth column of L. Note that once column j has been computed, appropriate multiples of it can be subtracted from columns $j+1$ through n of A *in parallel.* To exploit this idea on a message-passing multiprocessor, first assign each column of A to a processor. Then, the processor assigned column j of A waits to receive columns 1 through $j-1$ of L, modifying column j of A as it receives each one. Once this processor has completed computing column j of L, it sends this column to the other processors that need it, namely, those assigned columns $j+1$ through n. Geist and Heath [6], [7] and George et al. [9] used this idea to develop parallel dense Cholesky solvers for both shared-memory and message-passing multiprocessors and implemented their algorithms on a Denelcor HEP and an Intel iPSC/1 hypercube. Experimental results from both machines indicate that the column Cholesky approach is a reasonable way to exploit the parallelism inherent in the computation, provided that the number of columns n is much larger than the number of processors p. Since we can expect that multiprocessor systems will be used to factor large matrices, this is a reasonable assumption.

As in the dense case, we can use a column-oriented approach when A is sparse [5], [9], [22]. However, the sparsity of the matrix provides an opportunity for increasing the parallelism and reducing the message-passing overhead. To compute a column j of L, we only need those earlier columns that have nonzeros in row j. Hence, once we have computed column j, its nonzero structure determines where it must be sent; it will have to be sent to a particular processor if and only if this processor is assigned a column $k > j$ and $l_{kj} \neq 0$. However, without some preprocessing, the processor responsible for computing column j will not know how many earlier columns for which to wait. That is, it will not know how many nonzeros will appear in row j of L. The processors can easily compute the row structure of L from the elimination tree for A [18] and Gilbert, Hafsteinsson, and Zmijewski [13], [15], [22], [23] have developed parallel algorithms for computing this tree on shared-memory and local-memory machines.

[1] In this paper, all logarithms are base 2. Furthermore, if f and g are nonnegative functions of nonnegative integers, we say $f(n) = O(g(n))$ if there is a constant c such that $f(n) \leq cg(n)$ for all but finitely many n. We say $f(n) = \Omega(g(n))$ if $g(n) = O(f(n))$ and $f(n) = \Theta(g(n))$ if both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

```
initialize ready_queue;
while not done do
    if message_buffer ≠ ∅ then
      read in a column k from message_buffer;
      perform the relevant cmod(*, k) and update ready_queue;
    else if ready_queue ≠ ∅ then
      select a column k from ready_queue;
      perform cdiv(k);
      for each nonzero l_jk in column k with j > k do
          if column j is assigned to this processor then
            perform cmod(j, k) and update ready_queue
          else
            send column k to the processor assigned column j
          fi
      od
    fi
od
```

FIG. 1. *The old numeric factorization algorithm.*

In describing the old parallel numeric factorization algorithm, we assume that the processors have already determined the dependencies between columns. We use the following two subroutines in describing the algorithm.

- cmod(j, i) : subtracts the appropriate multiple of column i, namely, l_{ji}, from column j where $i < j$.
- cdiv(j) : divides column j by the square root of its diagonal element.

Figure 1 provides an outline of the old numeric factorization routine that each processor uses to compute its assigned columns. Each processor maintains a queue of columns *ready_queue*. Every column in the queue has had all of its cmod operations completed and is ready to have its cdiv performed. Each processor repeatedly performs one of two actions: it either selects a column from *ready_queue* to complete or it reads in a message (a column) and uses this column to update its assigned columns. Once a processor performs a cdiv on a column from *ready_queue*, it modifies its assigned columns using this newly computed column and sends this column to the processors that need it. In the actual implementation, a column will not be sent more than once to any given processor.

To avoid overflowing the message buffer, the old algorithm places a priority on reading messages and only processes a column from *ready_queue* when there are no pending messages. Note also that, once a column is completed, its processor may use the column for cmod operations before sending it off to other processors that need it. Because of this, it might be suspected that the old algorithm would run significantly slower than an algorithm that tried to send completed columns out as quickly as possible. However, in our experiments on an iPSC/2, such an algorithm was only sightly faster than the old algorithm on small problems, and quickly overflowed the message buffers on larger problems. We conclude that with a good initial ordering and a sufficient number of columns per processor, there is no need to send out columns as quickly as possible, since the processors will generally have unfinished work available. Although our new algorithm, described in the next section, generates many fewer messages than the old algorithm, we still needed to employ this approach to avoid overflowing the message buffers when solving large problems.

```
initialize ready_queue and col_mod_queue;
send_col := false;
while not done do
    if message_buffer ≠ ∅ then
      read in a column k or a column modification vector from message_buffer;
      perform the relevant cmod(*, k) or subtract the column modification vector
          from the relevant column and update ready_queue;
    else if ready_queue ≠ ∅ then
      select a column k from ready_queue;
      perform cdiv(k);
      for each nonzero l_jk in column k (in increasing order of j, j > k) do
          if column j is assigned to this processor then
            perform cmod(j, k) and update ready_queue
          else if send_col then
            send column k to the processor assigned column j
          else
            add column k to col_mod_queue and exit for-loop
          fi
      od;
    else if send_col then
      { Compute and send a column-modification vector. }
      select a row and add multiples of those columns in col_mod_queue with
          nonzeros in that row into a single sparse vector;
      send this vector to the relevant processor and update col_mod_queue;
    else
      { No other columns can be computed, so start sending messages. }
      send_col := true;
    fi
od
```

FIG. 2. *A new numeric factorization algorithm.*

3. Reorganizing column Cholesky. In this section, we describe the new factorization algorithm. In the old algorithm, each time a processor computes a column of L, the processor sends the column to those processors that need it to modify one or more of their assigned columns. Now, suppose that columns $j_1, \cdots, j_k$, assigned to processor P_1, must all modify column i, assigned to processor P_2. As each column j_r is computed, P_1 will send column j_r to P_2 and P_2 will then subtract the appropriate multiple of it from column i. Alternatively, P_1 can wait until columns $j_1, \cdots, j_k$ have all been computed and then add the appropriate multiples up into a *single* sparse vector. Processor P_1 can then send this single *column-modification vector* to processor P_2. Using this strategy, a processor can compute one of its assigned columns after receiving at most one column-modification vector from each of the other processors. Figure 2 describes a new factorization algorithm that uses this simple idea.

In the new algorithm, each processor initially computes as many of its assigned columns as possible without sending any messages. We control message passing with a Boolean flag *send_col*, initially set to *false*. As a processor computes each column, it places the column in a queue *col_mod_queue*. Then, when it exhausts its supply of columns that can be computed independently, it sets *send_col* to *true* and begins communicating with other processors. At this point, the processor starts computing column-modification vectors by adding multiples of columns in *col_mod_queue* with nonzeros in the same row. For each row i, a processor *scatters* into a dense vector

the appropriate multiple of each column in *col_mod_queue* with a nonzero in row i. Then, the processor *gathers* the nonzeros of the dense vector and sends them to the processor assigned column i. Note that once *send_col* becomes *true* no other columns are added to *col_mod_queue*. Instead, as in the old algorithm, the processors send off newly computed columns immediately.

The new algorithm performs the exact same arithmetic as the old algorithm. It simply redistributes the work in an attempt to reduce communication. The storage requirements for both algorithms are the same except that the new algorithm requires two additional $O(n)$ vectors: one to form column-modification vectors and another to maintain *col_mod_queue*. The queue *col_mod_queue* actually consists of n queues, one for each row in the matrix. Initially, column j is placed on row queue i if i is the first nonzero position in column j where column i is *not* assigned to the same processor as column j. When column j is eventually added into a column-modification vector, it is moved from row queue i to row queue k, where k is the first nonzero in column j below i. Since each column is on at most one row queue at a time, *col_mod_queue* requires only $O(n)$ storage. (The technique of using a single vector to represent n queues is used frequently in sequential sparse matrix factorizations, e.g., see George and Liu [11, §5.5].)

The number of messages passed by both the new and old algorithms obviously depends on the structure of L and the assignment of columns to processors. Note that the new approach will not necessarily decrease the total volume communication or the total number of messages. For example, suppose columns $i_1, \cdots, i_k$ on processor P_2 must all be modified by column j on processor P_1. If column j is added to *col_mod_queue*, processor P_1 will send one column modification vector to P_2 for each of k columns $i_1, \cdots, i_k$, rather than just send column j to P_2 and let it perform the k modifications. Of course, if these k column-modification vectors involve more than just column j, the new approach may still result in less communication. The new approach will succeed in reducing the communication overhead only if the column-modification vectors consist of contributions from several columns.

Using nested dissection to both order the matrix and partition the columns among the p processors will ensure that this happens. Given a graph G and a set of processors $\mathcal{P}$, this commonly used ordering strategy for parallel factorization works as follows. First, find a small separator S of G that partitions G into two roughly equal-sized pieces G_1 and G_2. Order the vertices of S last and wrap the corresponding columns onto all the processors in $\mathcal{P}$. Since a separator usually corresponds to a dense block in L, wrapping the columns in this way will balance the work required to compute this block among all the processors in $\mathcal{P}$. Finally, divide the processors in $\mathcal{P}$ into two roughly equal-sized groups $\mathcal{P}_1$ and $\mathcal{P}_2$ and recursively apply this nested dissection algorithm to the pairs $(G_1, \mathcal{P}_1)$ and $(G_2, \mathcal{P}_2)$. If $\mathcal{P}$ ever contains just a single processor, let $\mathcal{P}_1 = \mathcal{P}_2 = \mathcal{P}$.

We will say that all separators found at the same depth of the recursion belong to the same *level* and we will number levels beginning at 0. After finding $\log p$ levels of separators, the resulting $2^{\log p} = p$ subgraphs will each be assigned to its own processor. Since each separator from the initial $\log p$ levels partitions a subgraph among a group of processors, we will refer to these separators as *interprocessor separators*. For both the new and old factorization algorithms, those columns *not* belonging to any interprocessor separator can be computed independently by their assigned processors. Only those belonging to interprocessor separators will require communication. For the new algorithm to compute a column belonging to a level-i interprocessor separa-

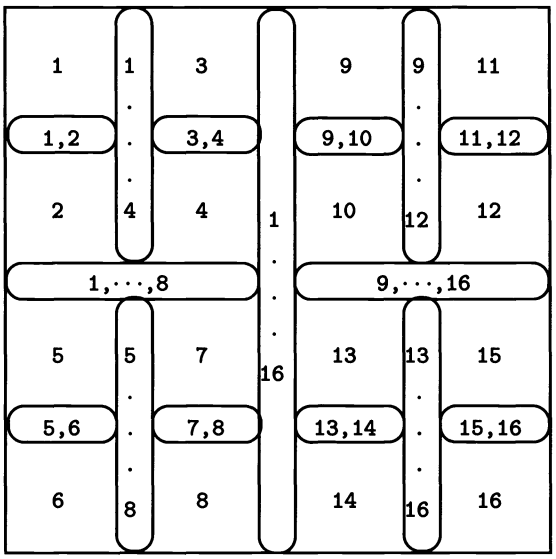


FIG. 3. *A nested dissection ordering on* 16 *processors.*

tor, it may require columns belonging to interprocessor separators at levels i through $\log p - 1$ plus at most $p/2^i$ column-modification vectors. Thus, the number of messages communicated by the new algorithm depends on the sizes of the interprocessor separators and the number of processors to which each is assigned. This is not the case for the old algorithm. For the old algorithm, a fill edge between two columns on different processors implies a communication, since the two columns depend on one another. Thus, the number of messages depends not only on the sizes of the interprocessor separators but also on the number of fill edges incident on these separators. Although both the new and old algorithms will work for any ordering and assignment of columns to processors, we obviously designed the new algorithm with nested dissection orderings in mind.

4. Communication complexity results for grid graphs. In this section, we analyze the communication requirements for factoring a $k \times k$ grid graph, using both the old and new algorithms. We assume the grid is ordered and assigned to processors using the nested dissection algorithm of §3. As the separator of a subgrid, we select all the vertices on a single mesh line (horizontal or vertical) that divides the subgrid into two nearly equal pieces. The mesh line is selected from those with the minimum number of vertices. Figure 3 provides an example of a grid that is ordered and partitioned among 16 processors using this approach. Each part of the grid is labeled with the processor(s) to which it is assigned. George [8] proved that for such an ordering of the $k \times k$ grid, the Cholesky factor will have $O(k^2 \log k)$ nonzeros and its computation will require $O(k^3)$ arithmetic operations. Hoffman, Martin, and Rose [17] proved that these bounds are optimal (in an order of magnitude sense).

As we will see below, given this ordering and assignment of columns to processors, both the old and new factorization algorithms will communicate the same volume of nonzeros, namely, $\Theta(pk^2)$. However, the old algorithm sends $\Theta(pk \log k)$ messages, compared to $\Theta(pk \log p)$ messages for the new algorithm. To simplify the derivation of these bounds, we assume that p is a power of 2 and that $k \gg p$.

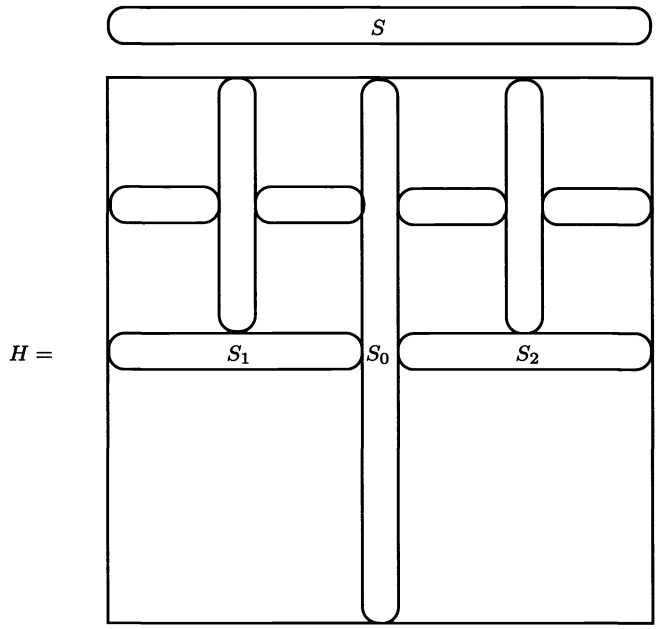


FIG. 4. *The separators of H with fill to S after four levels of dissection.*

THEOREM 4.1. *While factoring a $k \times k$ grid partitioned among p processors using nested dissection, the new factorization algorithm sends $\Theta(pk \log p)$ messages.*

Proof. The processors do not start communicating until they begin to compute columns belonging to the interprocessors separators. To compute such a column, a processor needs to receive both column-modification vectors and columns of L belonging to other interprocessor separators. There are $\log p$ levels of interprocessor separators and the ith level consists of 2^i separators. Since separator sizes decrease by roughly half every two levels, a level-i separator contains at most $k/2^{\lceil i/2 \rceil}$ vertices. Each level-i separator is wrapped onto $p/2^i$ processors and hence, each column in the separator requires at most one column-modification vector from each of these processors. Thus, the total number of messages containing column-modification vectors is bounded by

$$\sum_{i=0}^{\log p - 1} 2^i \cdot \frac{k}{2^{\lceil i/2 \rceil}} \cdot \frac{p}{2^i} = O(pk). \tag{1}$$

Now we must bound the number of messages containing columns of L. Let S be an arbitrary level-i interprocessor separator wrapped onto $q = p/2^i$ processors, where $0 \le i < \log p$. Thus,

$$r \stackrel{\text{def}}{=} |S| \le k/2^{\lceil i/2 \rceil}.$$

In computing a column in S, a processor will receive columns from some of the interprocessor separators in levels i through $\log p - 1$. Suppose that S divides a rectangular subgrid into two square subgrids and let H be one of these. Furthermore, suppose that S contains vertices in a horizontal mesh line (Fig. 4). We begin by bounding

the number of messages generated by fill between H and S. The level-$i+1$ separator in H, call it S_0, contains r vertices and there will be fill between each of these vertices and every vertex in S. Hence, the fill between S and S_0 produces at most rq messages. Now consider the two level-$i+2$ separators in H, call them S_1 and S_2 (Fig. 4). A vertex in either one will *not* be incident to all the vertices in S via fill edges. For example, each vertex in S_1 will only be incident to those vertices in the left half of S. Since $S_1 \cup S_2$ contains roughly r vertices, the fill between $S_1 \cup S_2$ and S will produce at most rq messages. If $r < 2q$ then the number of messages passed is bounded by $\frac{1}{2}r^2 < rq$. In general, for $i < j < \log p$, there are roughly r vertices belonging to level-j separators in H that have fill edges to S. Provided each level-j separator has fill edges to a piece of S that is assigned to at least q processors, the fill between all the level-j separators and S will produce roughly rq messages. This will happen whenever $j \leq c$, where $c = \min\{\log p - 1, i + 2\log(r/q)\}$. For j larger than c, the fill between the level-j separators of H and S will produce less than rq messages. More precisely, the total number of messages passed due to fill between H and S and fill within S is bounded by the following expression:

$$\sum_{j=i}^{c} rq \quad + \quad \sum_{j=c+1}^{\log p - 1} r \cdot \frac{q}{2^{j-c}}. \tag{2}$$

For $k \gg p$, the first term in this expression is the dominant one. In particular, for $k \geq p^{3/2}$, we have that $c = \log p - 1$ for all i and hence equation (2) is bounded by $O(rq \log p)$. Substituting in the value of q and the bound for r, we conclude that (2) is bounded by

$$O\left(\frac{k}{2^{\lceil i/2 \rceil}} \cdot \frac{p}{2^i} \log p\right).$$

Since this same bound applies if H is a rectangular rather than square subgrid or if S consists of vertices from a vertical mesh line, we can use this expression to bound the total number of messages containing columns of the interprocessor separators. Since there are 2^i level-i separators, each wrapped onto $p/2^i$ processors, and since each separator borders the two subgrids it separates, the following expression bounds the total number of messages containing columns of L:

$$\sum_{i=0}^{\log p - 1} 2^i \cdot 2 \cdot O\left(\frac{k}{2^{\lceil i/2 \rceil}} \cdot \frac{p}{2^i} \log p\right) = O(kp \log p).$$

Thus the bound on all messages communicated by the new algorithm is $O(kp \log p)$. A lower bound on just the first term in this sum is $\Omega(kp \log p)$ and hence, the proof is complete. □

THEOREM 4.2. *While factoring a $k \times k$ grid partitioned among p processors using nested dissection, the old factorization algorithm sends $\Theta(pk \log k)$ messages.*

Proof. The proof is almost identical to that of Theorem 4.1. As with the new algorithm, the old algorithm only needs to communicate in order to compute those columns belonging to interprocessor separators. However, a fill edge between a vertex at any level of the nested dissection ordering and an interprocessor separator may necessitate such a communication. Thus, the same analysis for bounding the number of columns of L passed by the new algorithm applies to the old algorithm, except we must consider all $O(\log k)$ levels of the dissection, rather than just the first $\log p$.

TABLE 1
Test problems.

Problem type	Equations	Nonzeros	Density (%)
grid	2500	12300	4.92
	5625	27825	4.95
	10000	49600	4.96
graded L	2614	17980	6.88
	5530	38248	6.92
	10297	71449	6.94

That is, in (2), the upper bound on the second summation is $2 \log k$ and c is no longer bounded above by $\log p - 1$. With these modifications, the same argument tells us that the old factorization algorithm sends $\Theta(pk \log k)$ messages. □

THEOREM 4.3 ([12]). *While factoring a $k \times k$ grid partitioned among p processors using nested dissection, the old factorization algorithm communicates $\Theta(pk^2)$ nonzeros.*

THEOREM 4.4. *While factoring a $k \times k$ grid partitioned among p processors using nested dissection, the new factorization algorithm communicates $\Theta(pk^2)$ nonzeros.*

Proof. Every message sent by the new algorithm that contains a column of L will also be sent by the old algorithm. Thus, from Theorem 4.3, in sending messages containing columns of L, the new algorithm communicates at most $O(pk^2)$ nonzeros.

Now we must bound the number of nonzeros communicated in column-modification vectors. After $\log p$ levels of nested dissection, the resulting p subgrids are assigned to single processors. Each subgrid is approximately $(k/\sqrt{p}) \times (k/\sqrt{p})$ and is bordered on at most four sides by interprocessor separators. All the fill edges that leave a subgrid must be incident on the vertices in the interprocessor separators that border it. Thus, each column-modification vector contains at most $O(k/\sqrt{p})$ nonzeros. Using the bound on the total number of column-modification vectors, equation (1), the total number of nonzeros contained in these vectors is bounded by $O(pk) \cdot O(k/\sqrt{p}) = O(p^{1/2}k^2)$.

Since the new algorithm communicates at least $\Omega(pk^2)$ nonzeros with respect to the level-0 interprocessor separator, it communicates a total of $\Theta(pk^2)$ nonzeros. □

5. Numerical results. At Oak Ridge National Laboratories, Geist and Ng [5] and George et al. [9] have developed a parallel sparse Cholesky factorization package in Fortran for the Intel iPSC/2 hypercube. This is a comprehensive experimental package that allows the user to select from several ordering and mapping strategies and performs each major phase of the factorization in parallel. This code incorporates the old numeric factorization algorithm discussed in §2. We implemented the new algorithm as a single Fortran routine that can be easily swapped with the old routine in the package. We then used the package to experiment with both routines. We also made three simple modifications to the package: we added statements to count the number of messages, we modified the code to allow the user to input arbitrary orderings and mappings of columns to processors, and we moved some data structure initialize code out of the factorization routine so it would not be counted in with the factorization time.

TABLE 2
Time and messages needed to factor $k \times k$ grids.

		Messages		Time (in seconds)	
		old	new	old	new
$k = 50$	$p = 4$	2092	429	6.433	4.979
	$p = 8$	4895	1603	4.348	3.196
	$p = 16$	9302	4319	2.878	2.104
$k = 75$	$p = 4$	3550	676	20.523	16.512
	$p = 8$	8666	2450	12.864	10.139
	$p = 16$	17345	6584	8.216	6.179
$k = 100$	$p = 4$	5517	872	49.421	40.227
	$p = 8$	13512	3281	29.571	23.708
	$p = 16$	27062	9030	17.837	13.385

We experimented with this code on the Cornell Theory Center's Intel iPSC/2 hypercube. We ran both algorithms on the six finite element problems listed in Table 1 using 4, 8, and 16 processors. The first three problems are $k \times k$ square grids for $k = 50$, 75, and 100; the last three are derived from very irregular L-shaped triangular meshes (graded L's) and are described by George and Liu [10]. (The Harwell–Boeing sparse matrix collection [4] contains graded L's with up to 3,466 equations.) As in §4, we ordered the grids and assigned columns to processors using nested dissection with horizontal or vertical mesh lines as separators. For the graded L's, we used the Oak Ridge package's nested dissection ordering and *subtree_to_subcube* mapping routines [5], [12]. These routines implement the nested dissection algorithm described in §3.

Table 2 contains our numerical results for the square grids and Table 3 contains those for the graded L's. Overall, the new algorithm communicates 51–84 percent fewer messages than the old algorithm. For the grids, the message-passing numbers agree with our analysis in §4. For example, since the new algorithm communicates $O(pk \log p)$ messages for a $k \times k$ grid, for fixed p, the number of messages should double as k increases from 50 to 100. This is, in fact, what happens. However, for the old algorithm applied to the same problems, the number of messages more than doubles. This reflects the $\log k$ factor in the message bound of the old algorithm. From Tables 2 and 3, we also observe that as the fraction n/p increases, so does the difference in messages communicated by the two algorithms. This is because as n/p increases, the number of columns per processor increases and hence, so does the number of columns that can be added into column-modification vectors.

Since the new algorithm communicates fewer messages while doing the same amount of work as the old algorithm, it requires less time to complete the factorizations. For the grid problems, the new algorithm requires 19–27 percent less time than the old algorithm. For the graded L problems, the new algorithm requires 23–27 percent less time when $p = 8$ or 16, but only 4–8 percent less time when $p = 4$. This anomalous behavior for $p = 4$ results from a load balancing problem with the new algorithm that we discuss in the next section.

6. Conclusions and future directions. We developed a new parallel algorithm for computing the Cholesky factorization of a large sparse positive-definite matrix on a message-passing multiprocessor. The algorithm attempts to reduce the communication overhead by redistributing the computational load and by repeatedly

TABLE 3
Time and messages needed to factor graded L*'s.*

		Messages		Time (in seconds)	
		old	new	old	new
$n = 2614$	$p = 4$	2353	529	13.149	12.099
	$p = 8$	5661	1917	7.078	5.152
	$p = 16$	11159	5442	4.767	3.599
$n = 5530$	$p = 4$	3936	844	38.920	36.492
	$p = 8$	9817	2941	20.474	15.421
	$p = 16$	19645	8120	13.390	10.125
$n = 10297$	$p = 4$	5950	1155	103.077	98.659
	$p = 8$	15468	4143	50.756	38.959
	$p = 16$	31607	11614	30.932	23.277

combining the effect of many messages into a single message. Experimentally, we demonstrated that for square grids and graded L's ordered and partitioned among the processors using nested dissection, the new algorithm communicates significantly fewer messages than the old algorithm. Because of this reduction in communication, the new algorithm is typically at least 20 percent faster on an Intel iPSC/2 hypercube. Theoretically, when factoring a $k \times k$ grid ordered by nested dissection on p processors, the new algorithm sends $\Theta(pk \log p)$ messages, compared to $\Theta(pk \log k)$ messages for the old algorithm. Both algorithms communicate the same volume of nonzeros, namely, $O(pk^2)$.

We can improve the performance of the new algorithm in several ways. Note that there is in general no value in adding a dense column to *col_mod_queue*, since such an addition may substantially increase the number of messages required. Thus, we may want to avoid adding columns to the queue of more than a specified density. Since later columns tend to be denser than earlier ones, one easy way to implement this idea is for each processor to terminate the queueing of columns before computing all eligible columns (i.e., set *send_col* equal to *true* after computing some fixed number of columns). Using this idea, we have been able to slightly lower both the number of messages communicated and the factorization times.

An improvement that we have not yet implemented is to send off the column-modification vectors as soon as possible. Namely, for each row, each processor could send the appropriate column-modification vector as soon as the last column with a nonzero in that row has been computed. Note that this would complicate the maintenance of *col_mod_queue* and would increase the storage required for this queue. If we use this idea and add *every* column to *col_mod_queue* as it is computed, then only column-modification vectors will be communicated. If we use this approach to factor a $k \times k$ grid, equation (1) tells us that we will pass a total of $O(pk)$ messages. Assuming that the level-0 separator is wrapped onto all p processors, this is the asymptotically minimum number of messages that can be communicated in factoring a $k \times k$ grid. In general, using these ideas it may be possible to reduce the communication overhead further at the expense of additional overhead. See Ashcraft et al. [1], Ashcraft, Eisenstat, and Liu [2] for algorithms that communicate only column-modification vectors. (See Mu and Rice [19] for an ordering and column-to-processor mapping scheme that is designed *exclusively* for grids. They prove that when this scheme is employed on a $k \times k$ grid the *old* algorithm sends $O(pk)$ messages.)

Another possible way we can improve the new factorization algorithm is to tailor

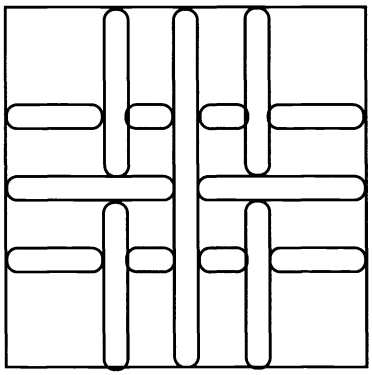


FIG. 5. *Using unbalanced separators to balance the load.*

the ordering of the matrix to the factorization. Since the new algorithm redistributes the work, the computational load might not be balanced among the p processors if we use the nested dissection ordering algorithm described in §3. For example, consider using this algorithm to order the columns of a $k \times k$ grid and assign them to p processors (Fig. 3). After $\log p$ levels of dissection, each processor ends up with a subgrid of approximately the same size, $k/\sqrt{p} \times k/\sqrt{p}$. Thus, the work needed to factor these subgrids is roughly evenly distributed. However, the work needed to compute the columns corresponding to the interprocessor separators will not, in general, be evenly distributed. To see this, consider the old factorization algorithm. For any column, its assigned processor computes all of that column's cmod operations. Thus, given the way the interprocessor separators are wrapped onto subsets of processors, the work corresponding to them is evenly distributed among all the processors. This is not the case for the new algorithm. Here, the processors assigned columns with nonzeros in row i will perform the cmod operations for column i. Thus, for square grids, the processors assigned subgrids adjacent to a particular interprocessor separator will compute the majority of the cmod operations pertaining to that separator. For $p > 4$, not all subgrids will be adjacent to each interprocessor separator and hence, the work pertaining to these separators will not be evenly distributed. We have observed this load imbalance experimentally on the Intel hypercube. To balance the computational load, we could select interprocessor separators in an *unbalanced* manner. That is, in finding a separator for a subgraph, select one that divides the subgraph *unevenly* into two pieces, where the smaller piece is the one adjacent to more previously found interprocessor separators. The degree of unevenness will depend on both the size of the graph and the number of interprocessor separators adjacent to each piece. Figure 5 illustrates this idea for the square grid.

Note also that, for both the new and old factorization algorithms, the communication overhead might not balance. For example, consider again the $k \times k$ grid of Fig. 3. Of the 16 subgrids assigned to individual processors, only 8 of them are adjacent to the initial separator, namely, 3, 4, 7, 8, 9, 10, 13, and 14. Only these subgrids will have fill edges to the initial separator and hence, their processors will do the bulk of the communication required to factor the columns in this separator. The lack of balance is unavoidable with this partitioning of columns to processors. The interprocessor separators are not all assigned to the same number of processors; the processors that are adjacent to those separators assigned to the most processors will

be involved in the most communication. As with the computational load, unbalanced separators could be used here to balance the communication overhead.

In conclusion we note that the success of the new algorithm depends on the ability to compute the multiple of column i needed by column j *entirely* from column i. Thus, the new algorithm cannot be adapted to Gaussian elimination or any other algorithm requiring pivoting for numerical stability without using additional communication.

Acknowledgments. We thank Esmond Ng for sending us the Oak Ridge National Laboratories' sparse parallel Cholesky factorization codes and the graded L problems used in the experiments. We also express our appreciation to Cornell University's Theory Center for the use of their Intel iPSC/2 hypercube computer.

REFERENCES

[1] C. ASHCRAFT, S. C. EISENSTAT, J. W. H. LIU, B. W. PEYTON, AND A. H. SHERMAN, *A compute-ahead implementation of the fan-in sparse distributed factorization scheme*, Tech. Report CS–90–03, Department of Computer Science, York University, North York, Ontario, Canada, 1990.

[2] C. ASHCRAFT, S. C. EISENSTAT, AND J. W. H. LIU, *A fan-in algorithm for distributed sparse numerical factorization,* SIAM J. Sci. Statist. Comput., 11 (1990), pp. 593–599.

[3] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1986.

[4] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[5] G. A. GEIST AND E. NG, *A partitioning strategy of parallel sparse Cholesky factorization*, Tech. Report ORNL–10937, Oak Ridge National Laboratory, Oak Ridge, TN, 1988.

[6] G. A. GEIST AND M. T. HEATH, *Parallel Cholesky factorization on a hypercube multiprocessor*, Tech. Report ORNL/TM–6190, Oak Ridge National Laboratory, Oak Ridge, TN, 1985.

[7] ———, *Matrix factorization on a hypercube multiprocessor*, in Proc. Conference on Hypercube Multiprocessors, Michael T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1986, pp. 161–180.

[8] A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[9] A. GEORGE, M. HEATH, J. LIU, AND E. NG, *Solution of sparse positive definite systems on a hypercube*, Tech. Report ORNL/TM–10865, Oak Ridge National Laboratory, Oak Ridge, TN, 1988.

[10] A. GEORGE AND J. W. H. LIU, *An automatic nested dissection algorithm for irregular finite element problems*, SIAM J. Numer. Anal., 15 (1978), pp. 1053–1069.

[11] ———, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[12] A. GEORGE, J. W. H. LIU, AND E. NG, *Communication results for parallel sparse Cholesky factorization on a hypercube*, Parallel Comput., 10 (1989), pp. 287–298.

[13] J. R. GILBERT AND H. HAFSTEINSSON, *Parallel symbolic factorization of sparse linear systems*, Parallel Comput., 14 (1990), pp. 151–162.

[14] J. R. GILBERT AND E. ZMIJEWSKI, *A parallel graph partitioning algorithm for a message-passing multiprocessor*, Internat. J. Parallel Programming, 16 (1987), pp. 427–449.

[15] H HAFSTEINSSON, *Parallel Sparse Cholesky Factorization.* Ph.D. thesis, Cornell University, Ithaca, NY, 1988.

[16] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[17] A. J. HOFFMAN, M. S. MARTIN, AND D. J. ROSE, *Complexity bounds for regular finite difference and finite element grids*, SIAM J. Numer. Anal., 10 (1973), pp. 364–369.

[18] J. W. H. LIU, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.

[19] M. MU AND J. R. RICE, *A grid based subtree-subcube assignment for solving* PDEs *on hypercubes*, Tech. Report CSD–TR–869, Purdue University, West Lafayette, IN, 1989.

[20] E. NG, personal communication, 1988.

[21] S. PISSANETZKY, *Sparse Matrix Technology*, Academic Press, New York, 1984.

[22] E. ZMIJEWSKI, *Sparse Cholesky factorization on a multiprocessor*, Ph.D. thesis, Cornell University, Ithaca, NY, 1987.

[23] E. ZMIJEWSKI AND J. R. GILBERT, *A parallel algorithm for sparse symbolic Cholesky factorization on a multiprocessor*, Parallel Comput., 7 (1988), pp. 199–210.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1198–1206, September 1991
©1991 Society for Industrial and Applied Mathematics
012

THE EQUATIONS OF MOTION OF PARTICLES IN SMOOTHED PARTICLE HYDRODYNAMICS*

G. V. BICKNELL†

Abstract. Detailed attention is given to the derivation of the equations of motion in smoothed particle hydrodynamics (SPH). It is rigorously shown that the four-dimensional divergence of the momentum density can be replaced by the total derivative with respect to time of the unsmoothed particle velocity. No approximations are required other than the standard SPH ansatz of approximating integrals by sums. The result is independent of the form of the kernel and provides a sound basis for the consideration of variable smoothing length kernels in SPH. It is noted that kernels with a variable smoothing length do not conserve momentum and energy exactly and explicit expressions for the rate of change of total momentum and energy are given.

Key words. particle methods, fluid dynamics

AMS(MOS) subject classifications. 65M99, 70F10, 76N10

1. Introduction.

Smoothed particle hydrodynamics (SPH; Lucy [8], Gingold and Monaghan [4], Monaghan [10]) has evolved into a popular technique, especially among astrophysicists, for three-dimensional fluid dynamic simulations which may or may not involve self gravity. The first basic element of SPH is the notion of smoothing. The smoothed value of a function ρf is given by

$$[\rho f]_s(x^\lambda, t) = \int W(x^\lambda - \bar{x}^\lambda)\, f(\bar{x}^\lambda, t)\, \rho(\bar{x}^\lambda, t)\, d^3\bar{x} \tag{1.1}$$

where ρ is the density, W is the smoothing kernel, and x^λ are the three-dimensional space coordinates. (The smoothed value ρ_s of the density is given by the above equation when $f = 1$.) The second basic element of SPH is the replacement of the above integral by a sum:

$$[\rho f]_s(x^\lambda, t) \approx \sum_{i=1}^{N} m_i\, f_i\, W(x^\lambda - x_i^\lambda) \tag{1.2}$$

where $x_i^\lambda(t)$ are the coordinates of N "particles" of "mass" m_i, distributed according to the density and $f_i = f(x_i^\lambda, t)$. Monaghan [9] and Monaghan and Lattanzio [12] have shown that there are advantages in considering the particles as moving interpolating points and that view is adopted here.

In this paper the treatment of Monaghan [9] is refined by using a new method for deriving the particle equations of motion directly from the fluid equations. The analysis is not restricted to any particular form of kernel and the treatment is therefore relevant to variable smoothing length kernels, an example of which is

$$W = W\left(\frac{x^\lambda - \bar{x}^\lambda}{h_s(x^\lambda, t)}\right) \tag{1.3}$$

*Received by the editors August 9, 1989; accepted for publication (in revised form) August 8, 1990.

†Mount Stromlo and Siding Spring Observatories, Institute of Advanced Studies, Australian National University, Private Bag, Weston Creek P O, ACT 2611, Canberra, Australia.

where $h_s(x^\lambda, t)$ is a spatially and temporally variable smoothing length. In the case of finite regions, other "edge sensitive" kernels may be required (see Coleman and Bicknell [1]) so that in this paper we assume the most general possible form

$$W = W(x^\lambda, \bar{x}^\lambda, t) \tag{1.4}$$

for which

$$\begin{aligned}[\rho f]_s\,(x^\lambda, t) &= \int W(x^\lambda, \bar{x}^\lambda, t)\, f(\bar{x}^\lambda, t)\, \rho(\bar{x}^\lambda, t)\, d^3\bar{x} \\ &\approx \sum_{i=1}^{N} m_i\, f_i\, W(x^\lambda, x_i^\lambda, t).\end{aligned} \tag{1.5}$$

We should state from the outset that kernels of the form (1.4) may cause difficulties with energy and momentum conservation and this is given some attention in the following sections. However, the main purpose of this paper is to put the derivation of particle equations on a more rigorous footing and a more detailed treatment of the questions associated with variable smoothing length is postponed to later work.

The plan of this paper is as follows. In § 2 equations of motion for the particles/interpolating points are derived which are consistent with both equation (1.5) and the fluid equations. The estimation of the force term and the related question of conservation of momentum are discussed in § 3. The energy equation and conservation of energy are discussed in § 4. A discussion follows in § 5.

2. Particle equations of motion in SPH. In this section we consider the particles as moving interpolating points and derive the equations of motion for those points which are consistent with both the Euler equations and the SPH equation (1.5) for the smoothed value of any function.

The following treatment is unified if, with both the continuity and momentum equations in mind, we consider the effect of applying the smoothing kernel to the general four-dimensional divergence:

$$D(\rho f) = \frac{\partial}{\partial t}(\rho f) + \frac{\partial}{\partial x^\alpha}\,(\rho f v^\alpha) \tag{2.1}$$

where f is any tensor (with indices suppressed). Note that the summation convention is observed with respect to repeated Greek tensorial indices.

The smoothed divergence is

$$\begin{aligned}[D\,(\rho f)]_s\,(x^\lambda, t) &= \int W(x^\lambda, \bar{x}^\lambda, t) \frac{\partial}{\partial t}\,[\rho(\bar{x}^\lambda) f(\bar{x}^\lambda)]\; d^3\bar{x} \\ &\quad + \int W(x^\lambda, \bar{x}^\lambda, t) \frac{\partial}{\partial \bar{x}^\alpha}\,[\rho(\bar{x}^\lambda) f(\bar{x}^\lambda) v^\alpha(\bar{x}^\lambda)]\; d^3\bar{x}.\end{aligned} \tag{2.2}$$

Now, these integrals can be rearranged by integration by parts, the motivation for this being to force all integrals to contain $\rho\, d^3\bar{x}$ and thus be easily discretized. Hence,

$$[D(\rho f)]_s\,(x^\lambda, t) = T_1 + T_2 + T_3 + T_4 \tag{2.3}$$

where

$$
\begin{aligned}
T_1 &= \frac{\partial}{\partial t}\int W(x^\lambda,\bar{x}^\lambda,t)\rho(\bar{x}^\lambda)f(\bar{x}^\lambda)\,d^3\bar{x},\\
T_2 &= -\int \frac{\partial W(x^\lambda,\bar{x}^\lambda,t)}{\partial t}\rho(\bar{x}^\lambda)f(\bar{x}^\lambda)\,d^3\bar{x},\\
T_3 &= \int_B W(x^\lambda,\bar{x}^\lambda,t)\rho(\bar{x}^\lambda)f(\bar{x}^\lambda)v^\alpha(\bar{x}^\lambda)n^\alpha\,d^2\bar{S},\\
T_4 &= -\int \frac{\partial W(x^\lambda,\bar{x}^\lambda,t)}{\partial \bar{x}^\alpha}\rho(\bar{x}^\lambda)f(\bar{x}^\lambda)v^\alpha(\bar{x}^\lambda)\,d^3\bar{x}.
\end{aligned}
\tag{2.4}
$$

The surface integral T_3 is over the boundary B (with unit normal n^α) of the three-dimensional volume. When the volume is infinite, it is assumed that W vanishes rapidly enough such that $T_3 = 0$. In finite volumes, T_3 is also zero if $W = 0$ on B. However, kernels constructed to satisfy this extra constraint are usually computationally awkward and for finite volumes it may be necessary to take the surface integral into account. In this paper, an infinite volume is assumed so that $T_3 = 0$.

Using the standard SPH method of replacing integrals by sums (i.e., equation (1.5)), remembering that $x_i^\lambda = x_i^\lambda(t)$ and also that $f_i = f(x_i^\lambda(t))$, gives for the remaining terms T_1, T_2, and T_4:

$$
\begin{aligned}
T_1 &\approx \frac{\partial}{\partial t}\sum_{i=1}^{N} m_i W(x^\lambda,x_i^\lambda,t)f_i\\
&= \sum_{i=1}^{N} m_i\left[\frac{\partial W(x^\lambda,x_i^\lambda,t)}{\partial t}f_i + \frac{\partial W(x^\lambda,x_i^\lambda,t)}{\partial x_i^\mu}\frac{dx_i^\mu}{dt}f_i + W(x^\lambda,x_i^\lambda,t)\frac{df_i}{dt}\right],\\
T_2 &\approx -\sum_{i=1}^{N} m_i\frac{\partial W(x^\lambda,x_i^\lambda,t)}{\partial t}f_i,\\
T_4 &\approx -\sum_{i=1}^{N} m_i\frac{\partial W(x^\lambda,x_i^\lambda,t)}{\partial \bar{x}_i^\alpha}f_i v_i^\alpha.
\end{aligned}
\tag{2.5}
$$

In the expressions for T_1 and T_2 above

$$
\frac{\partial W(x^\lambda,x_i^\lambda,t)}{\partial t} = \left[\frac{\partial W(x^\lambda,\bar{x}^\lambda,t)}{\partial t}\right]_{\bar{x}^\lambda = x_i^\lambda}.
\tag{2.6}
$$

The first of the terms in T_1 cancels T_2. Noting that μ and α appear as dummy indices in the expressions for T_1 and T_4, respectively, we obtain

$$
\begin{aligned}
[D(\rho f)]_s\,(x^\lambda,t) \approx &\sum_i m_i\frac{df_i}{dt}W(x^\lambda,x_i^\lambda,t)\\
&+\sum_i m_i\,f_i\left(\frac{dx_i^\alpha}{dt}-v_i^\alpha\right)\frac{\partial}{\partial x_i^\alpha}W(x^\lambda,x_i^\alpha,t).
\end{aligned}
\tag{2.7}
$$

Now, for the equation of continuity, $f = 1$, $D(\rho f) = 0$ and since the kernel is arbitrary, we recover a result which has been the intuitive basis of all particle methods to date, namely,

$$
\frac{dx_i^\alpha}{dt} = v_i^\alpha,
\tag{2.8}
$$

that is, the velocities of the interpolating points are equal to the fluid velocity at those points (when the particle masses are constant). This generalizes the result obtained by Monaghan [9] for the usual form of kernel $W = W(x^\alpha - \bar{x}^\alpha)$. (Monaghan also considered variable mass particles which are not considered here.) In view of (2.8) one now has for an arbitrary tensor f,

$$[D(\rho f)]_s(x^\lambda, t) \approx \sum_i m_i \frac{df_i}{dt} W(x^\lambda, x_i^\lambda, t). \tag{2.9}$$

We can now take the argument further and consider the implications of the momentum equations for the motion of the interpolating points by taking $f = v^\alpha$ in (2.9). The smoothed form of the momentum equations is

$$[D(\rho v^\alpha)]_s(x^\lambda, t) = -\int W(x^\lambda, \bar{x}^\lambda, t) \frac{\partial(P(\bar{x}^\lambda) + q(\bar{x}^\lambda))}{\partial \bar{x}^\alpha}\, d^3\bar{x} \tag{2.10}$$

where P is the gas pressure and q is an artificial viscous pressure used for shocks. Various forms can be used for q (see Gingold and Monaghan [4] and Monaghan and Lattanzio [12]) but the form used is unimportant in the following analysis. Using (2.9) above we obtain

(2.11)

$$\begin{aligned}\sum_i m_i W(x^\lambda, x_i^\lambda, t) \frac{dv_i^\alpha}{dt} &\approx -\int W(x^\lambda, \bar{x}^\lambda, t) \frac{\partial(P(\bar{x}^\lambda) + q(\bar{x}^\lambda))}{\partial \bar{x}^\alpha} d^3\bar{x} \\ &\approx -\sum_i m_i W(x^\lambda, x_i^\lambda, t) \left[\frac{1}{\rho(\bar{x}^\lambda)} \frac{\partial(P(\bar{x}^\lambda) + q(\bar{x}^\lambda))}{\partial \bar{x}^\alpha}\right]_{\bar{x}^\lambda = x_i^\lambda}.\end{aligned}$$

Again, since the kernel is arbitrary,

$$\frac{dv_i^\alpha}{dt} = -\left[\frac{1}{\rho(x^\lambda)} \frac{\partial(P(x^\lambda) + q(x^\lambda))}{\partial x^\alpha}\right]_{x^\lambda = x_i^\lambda}. \tag{2.12}$$

The right-hand side cannot be estimated directly and it is necessary to approximate the pressure gradient at particle i by its smoothed value. Such approximations are generally $\mathcal{O}(h_s^2)$ or better for appropriately designed kernels.

In the following section we consider the approximation of the pressure gradient and the associated issue of the conservation of momentum.

3. Representation of the pressure gradient and the conservation of momentum. A proper discussion of the representation of the pressure gradient necessitates a brief consideration of the smoothing of derivatives. Hence, for an arbitrary function F, the smoothed gradient of F is

$$\left[\frac{\partial F}{\partial x^\alpha}\right]_s (x^\lambda, t) = \int \frac{\partial F(\bar{x}^\lambda, t)}{\partial \bar{x}^\alpha} W(x^\lambda, \bar{x}^\lambda, t)\, d^3\bar{x}. \tag{3.1}$$

Integrating by parts and assuming that W is zero on the boundary of the volume, gives

$$\left[\frac{\partial F}{\partial x^\alpha}\right]_s (x^\lambda, t) = -\int F(x^\lambda, t) \frac{\partial W(x^\lambda, \bar{x}^\lambda, t)}{\partial \bar{x}^\alpha}\, d^3\bar{x}. \tag{3.2}$$

Note that, in general, differentiation and smoothing do not commute. In fact,

(3.3)

$$\frac{\partial F_s(x^\lambda,t)}{\partial x^\alpha} - \left[\frac{\partial F}{\partial x^\alpha}\right]_s (x^\lambda,t) = \int F(\bar{x}^\lambda,t)\left(\frac{\partial W(x^\lambda,\bar{x}^\lambda,t)}{\partial x^\alpha} + \frac{\partial W(x^\lambda,\bar{x}^\lambda,t)}{\partial \bar{x}^\alpha}\right) d^3\bar{x}.$$

There are several possible expressions on which to base the smoothed value of the pressure gradient (Monaghan [9], [10]). For convenience, let us choose the form:

$$\left[\frac{1}{\rho}\frac{\partial P}{\partial x^\alpha}\right]_{x^\lambda=x_i^\lambda} = \left[\frac{\partial(\rho P/\rho^2)}{\partial x^\alpha}\right]_{x^\lambda=x_i^\lambda} + \frac{P_i}{\rho_i^2}\left[\frac{\partial \rho}{\partial x^\alpha}\right]_{x^\lambda=x_i^\lambda}, \tag{3.4}$$

which it is known leads to a momentum conserving form of the equations of motion for kernels of the form $W(x^\alpha - \bar{x}^\alpha)$. Replacing the derivatives in equation (3.4) by their smoothed counterparts, replacing integrals by sums, and substituting in (2.12), leads to the particle equations of motion

$$\frac{dv_i^\alpha}{dt} = \sum_{j=1}^{N} m_j \left(\frac{P_i+q_i}{\rho_i^2} + \frac{P_j+q_j}{\rho_j^2}\right)\frac{\partial W(x_i^\lambda, x_j^\lambda, t)}{\partial x_j^\alpha}. \tag{3.5}$$

Note that the differentiation of the kernel is with respect to x_j^α, *not* x_i^α as in the more usual formulation involving a kernel $W(x^\lambda - \bar{x}^\lambda)$.

Let us now examine the conditions under which this set of equations conserves total momentum. The time rate of change of momentum of the system is

$$\sum_{i=1}^{N} m_i \frac{dv_i^\alpha}{dt} = \sum_{i=1}^{N}\sum_{j=1}^{N} m_i m_j \left(\frac{P_i+q_i}{\rho_i^2} + \frac{P_j+q_j}{\rho_j^2}\right)\frac{\partial W(x_i^\lambda, x_j^\lambda, t)}{\partial x_j^\alpha}. \tag{3.6}$$

Since the i and j indices in the two groups of terms are dummy indices, the above equation can be cast in the following form

$$\sum_{i=1}^{N} m_i \frac{dv_i^\alpha}{dt} = \sum_{i=1}^{N}\sum_{j=1}^{N} m_i m_j \left(\frac{P_i+q_i}{\rho_i^2}\right)\left(\frac{\partial W(x_i^\lambda, x_j^\lambda, t)}{\partial x_j^\alpha} + \frac{\partial W(x_j^\lambda, x_i^\lambda, t)}{\partial x_i^\alpha}\right). \tag{3.7}$$

Thus (denoting $W(x_i^\lambda, \bar{x}_j^\lambda, t)$ by W_{ij}) the condition for conservation of momentum is

$$\frac{\partial W_{ij}}{\partial x_j^\alpha} + \frac{\partial W_{ji}}{\partial x_i^\alpha} = 0 \tag{3.8}$$

and, in general, total momentum is not exactly conserved. Moreover, symmetry of the kernel does not guarantee conservation of momentum, for if $W_{ij} = W_{ji}$, then equation (2.19) implies the form:

$$W_{ij} = W(x_i^\alpha - x_j^\alpha, t) \tag{3.9}$$

Thus, a symmetric kernel with a spatially variable smoothing length does not exactly conserve momentum. Nevertheless, for the important case in which the smoothing length varies with time and the kernel is of the form (1.3) then momentum *is* conserved.

There are other ways of symmetrizing the force terms on the right-hand side of the momentum equations. Various approaches have been advocated by Evrard [3] and Hernquist and Katz [6], for example. However, a detailed discussion of these and other possible approaches is beyond the scope of this paper.

In order to complete this analysis of the particle equations, it is necessary to consider the energy equation. This is the subject of the following section.

4. The energy equation. In the previous sections I derived the form of the particle equations implied by the momentum equations for the fluid. This needs to be supplemented by consideration of the energy equations. I do this here, taking into account the contribution to the energy balance from the artificial viscous term in the pressure.

Following Landau and Lifshitz [7, p. 192 ff], the artificial viscous term implies a rate of change of entropy per unit mass (s) given by

$$\rho T \frac{ds}{dt} = -q \frac{\partial v^\alpha}{\partial x^\alpha}. \tag{4.1}$$

Thus the internal energy density (ϵ), pressure(P) and density (ρ) are related by

$$\frac{d\epsilon}{dt} - \frac{\epsilon + P}{\rho} \frac{d\rho}{dt} = -q \frac{\partial v^\alpha}{\partial x^\alpha}. \tag{4.2}$$

Now this equation applies along a streamline. Since, from (2.8) the particles move along streamlines, we can deduce that

$$\frac{d\epsilon_i}{dt} - \frac{\epsilon_i + P_i}{\rho_i} \frac{d\rho_i}{dt} = -q_i \left[\frac{\partial v^\alpha}{\partial x^\alpha} \right]_{x^\lambda = x_i^\lambda} \tag{4.3}$$

where "t" now refers to the time along a particle trajectory.

For a polytropic gas (in which $\epsilon = (\gamma - 1)^{-1} P$, with γ the polytropic index) equation (4.3) becomes

$$\frac{dK_i}{dt} = -(\gamma - 1) \rho_i^{-\gamma} q_i \left[\frac{\partial v^\alpha}{\partial x^\alpha} \right]_{x^\mu = x_i^\mu} \tag{4.4}$$

where $K_i(s_i) = P_i \rho_i^{-\gamma}$ is the pseudoentropy of the ith particle. Obviously, in the adiabatic case, $P_i = K_i \rho_i^\gamma$, a result which practitioners of SPH have used from the outset. Nevertheless, it is worth noting that this result also depends upon the use of unsmoothed particle velocities and is independent of the form of the kernel.

Let us now consider the case where an artificial viscous term is included in the pressure. Following Monaghan and Gingold [11], we express the dissipative term in the form:

$$\frac{q}{\rho} \frac{\partial v^\alpha}{\partial x^\alpha} = \frac{1}{2} \left[\frac{q}{\rho^2} \frac{\partial (\rho v^\alpha)}{\partial x^\alpha} - \frac{q}{\rho^2} v^\alpha \frac{\partial \rho}{\partial x^\alpha} + \frac{\partial (\rho q v^\alpha / \rho^2)}{\partial x^\alpha} - v^\alpha \frac{\partial (\rho q / \rho^2)}{\partial x^\alpha} \right]. \tag{4.5}$$

Replacing derivatives by their smoothed values and approximating integrals by sums, results in

$$T_i \frac{ds_i}{dt} = -\frac{1}{2} \sum_{j=1}^{N} m_j \left(\frac{q_i}{\rho_i^2} + \frac{q_j}{\rho_j^2} \right) (v_i^\alpha - v_j^\alpha) \frac{\partial W_{ij}}{\partial x_j^\alpha} \tag{4.6}$$

and

$$\frac{dK_i}{dt} = \frac{1}{2}(\gamma-1)\rho_i^{-(\gamma-1)} \sum_{j=1}^{N} m_j \left(\frac{q_i}{\rho_i^2} + \frac{q_j}{\rho_j^2}\right)(v_i^\alpha - v_j^\alpha)\frac{\partial W_{ij}}{\partial x_j^\alpha}. \tag{4.7}$$

The main difference between these expressions and those given by Monaghan and Gingold [11] is the differentiation with respect to x_j^α rather than x_i^α which also accounts for the differences in sign.

The time variation of the total energy of the system is

$$\begin{aligned}\frac{dE}{dt} &= \frac{d}{dt}\sum_{i=1}^{N} m_i \left(\frac{1}{2}v_i^2 + \frac{\epsilon_i}{\rho_i}\right) \\ &= \sum_{i=1}^{N} m_i \left(v_i^\alpha \frac{dv_i^\alpha}{dt} - \left[\frac{(P+q)}{\rho}\frac{\partial v^\alpha}{\partial x^\alpha}\right]_{x^\lambda = x_i^\lambda}\right)\end{aligned} \tag{4.8}$$

where the last term follows from (4.3) and equating $\rho^{-1}d\rho/dt$ to $-\partial v^\alpha/\partial x^\alpha$.

The term $\rho^{-1}(P+q)\partial v^\alpha/\partial x^\alpha$ in (4.7) can be manipulated in a similar way to $\rho^{-1}q\partial v^\alpha/\partial x^\alpha$ in equation (4.5) above. The resultant expression for this term combined with the scalar product of the momentum equations (3.5) with v_i^α gives

$$\frac{dE}{dt} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} m_i m_j \frac{(P_i+q_i)}{\rho_i^2}(v_i^\alpha + v_j^\alpha)\left(\frac{\partial W_{ij}}{\partial x_j^\alpha} + \frac{\partial W_{ji}}{\partial x_i^\alpha}\right). \tag{4.9}$$

Thus the condition for exact conservation of energy is the same as that for exact conservation of momentum, namely,

$$\frac{\partial W_{ij}}{\partial x_j^\alpha} + \frac{\partial W_{ji}}{\partial x_i^\alpha} = 0 \tag{4.10}$$

and clearly, a variable smoothing length causes problems with energy conservation as well.

We note, for the sake of completeness, that mass *is* conserved, since

$$\frac{d}{dt}\sum_{i=1}^{N} m_i = 0. \tag{4.11}$$

5. Discussion. It has been shown in this paper how the equations of motion of particles in SPH may be rigorously derived from the fluid equations, refining Monaghan's original approach [9] to the justification for particle methods. The main result is that the advective derivatives in the Euler or Navier–Stokes equations can be replaced by the total derivative with respect to time of the (unsmoothed) particle velocity, the only approximation involved being the standard SPH one of replacing integrals by sums. Prior to this, it had been thought that a sequence of two approximations was required to arrive at a similar result. The first involved approximating the smoothed value of $v^\alpha v^\beta$ by $v_s^\alpha v_s^\beta$ and the second involved approximating v_s^α by the particle velocity v_i^α. The demonstration that these approximations are unnecessary is quite satisfying and shows that numerical inaccuracies in the SPH method can be

entirely attributed to the approximation of the pressure gradient at particle locations by the smoothed value and the inaccuracy involved in replacing integrals by sums over interpolating points. Thus, future attempts at improving the accuracy of SPH should be directed at these aspects.

This result also provides further motivation for the use of the unsmoothed particle velocity in the equations of motion rather than the smoothed velocity. (Other arguments, based upon the convergence of particle methods and the motion of point vortices, may be found in the papers by Cottet and Raviart [2], Greengard and Anderson [5], and Raviart [13].) We expect the distinction between smoothed and particle velocity to be quite important in situations where there is a strong velocity gradient such as in cloud–cloud collisions or in regions of strong shear such as in the classical plane shear layer. I have shown here that the proper choice is unambiguous.

The conclusion that the advective terms in the equations of motion are also independent of the form of the kernel is also satisfying. The general form of the kernel used here accommodates a wide class of variable smoothing length kernels. However, it is clear from the discussions in §§ 3 and 4 that a spatially variable smoothing length generally has the unsatisfactory feature of not conserving momentum or energy. Just how unsatisfactory this feature is obviously depends upon how rapidly the smoothing length varies. Indeed it may be the case, for quite reasonable prescriptions of smoothing length behaviour, that the errors involved are no worse than the errors involved in estimating the pressure gradient. Another possibility is that there may be better ways of estimating derivatives that lead to automatic energy and momentum conservation. As was stated at the outset, such issues are beyond the scope of this paper. Nevertheless, the results derived in §§ 3 and 4 form the basis for a further discussion of variable smoothing length. The expressions given there should be useful for estimation of the effects of a variable smoothing length on momentum and energy conservation. Another useful result derived in those sections is that a smoothing length which varies only with time conserves energy and momentum exactly. Such a smoothing length prescription is useful in studying the evolution of fluids whose expansion or contraction is quasi-homologous. This is the case, for instance, in supernova explosions or gravitational collapse.

The rigorous derivation of the SPH equations of motion given in this paper also provides the basis for a more comprehensive analysis of the errors involved in SPH. One source of error, common to all particle codes, is the diffusion of particles leading to an unphysical diffusion of momentum and angular momentum. Inclusion of higher order corrections to the SPH evaluation of integrals will be necessary for the development of Fokker–Planck diffusion equations which will enable an assessment to be made of the importance of this effect. Future work will also involve an investigation of the relationship between SPH and Large Eddy Simulations (LES; Rogallo and Moin [14]). There is already a formal correspondence between Large Eddy Simulations and SPH in that the LES concept of filtering corresponds to the SPH concept of smoothing. Additional terms in the equations of motion are required to model the effect of a turbulent cascade unresolved by the smoothing length and to provide SPH terms analogous to the "Leonard terms" which are important in LES. Again, the source of the additional terms is higher order corrections to the SPH equations.

Finally, I note that the results obtained here greatly strengthen the general interpretation of particle methods (SPH in particular) in terms of moving interpolating points (Monaghan [9]). The "mass" attached to each particle can be interpreted as an interpolation weight. The historical interpretation of particles as "smeared out" elements of fluid, whilst forming the intuitive basis of particle methods to date, can no

longer be regarded as appealing and is prohibitive of further analytical development.

Acknowledgment. I thank the referees of the original version of this paper for a number of constructive comments.

REFERENCES

[1] C. S. COLEMAN AND G. V. BICKNELL, *Jets with entrained clouds* I. *Hydrodynamic simulations*, Mon. Not. R. Astr. Soc., 214 (1985), pp. 337–355.

[2] G. H. COTTET AND P. A. RAVIART, *Particle methods for the one-dimensional Vlasov–Poisson equations*, SIAM J. Numer. Anal., 21 (1984), pp. 52–76.

[3] A. EVRARD, *Beyond N-body:* 3D *cosmological gas dynamics*, Mon. Not. R. Astr. Soc., 235 (1988), pp. 911–934.

[4] R. A. GINGOLD AND J. J. MONAGHAN, *Kernel estimates as a basis for general particle methods in hydrodynamics*, J. Comput. Phys., 46 (1982), pp. 429–453.

[5] C. GREENGARD AND C. ANDERSON, *On vortex methods*, SIAM J. Numer. Anal., 22 (1985), pp. 413–440.

[6] L. HERNQUIST AND N. KATZ, TREESPH: *A unification of* SPH *with the hierarchical tree method*, Astrophys. J. Supp. Series, 70 (1989), pp. 419–446.

[7] L. D. LANDAU AND E. M. LIFSHITZ, *Fluid Mechanics*, 2nd ed., Pergamon Press, Oxford, 1987.

[8] L. B. LUCY, *A numerical approach to the testing of the fission hypothesis*, Astron. J., 82 (1977), pp. 1013–1024.

[9] J. J. MONAGHAN, *Why particle methods work*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 421–433.

[10] ———, *Particle methods for hydrodynamics*, Computer Physics Reports, 3 (1985), pp. 71–124.

[11] J. J. MONAGHAN AND R. A. GINGOLD, *Shock simulation by the particle method* SPH, J. Comput. Phys., 52 (1983), pp. 374–389.

[12] J. J. MONAGHAN AND J. LATTANZIO, *A refined particle method for astrophysical problems*, Astron. & Astrophys., 149 (1985), pp. 135.

[13] P. A. RAVIART, *An analysis of particle methods*, Numerical Methods in Fluid Dynamics, Lecture Notes in Mathematics, Vol. 1127, Springer-Verlag, Berlin, 1983, p. 243–262.

[14] R. S. ROGALLO AND P. MOIN, *Numerical simulation of turbulent flows*, Ann. Rev. Fluid Mech., 16 (1984), pp. 99–137.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1207–1244, September 1991
©1991 Society for Industrial and Applied Mathematics
013

AN ADAPTIVE FINITE ELEMENT METHOD FOR TWO-PHASE STEFAN PROBLEMS IN TWO SPACE DIMENSIONS. PART II: IMPLEMENTATION AND NUMERICAL EXPERIMENTS*

R. H. NOCHETTO†, M. PAOLINI‡, AND C. VERDI§

Abstract. An adaptive local mesh refinement strategy for two-phase Stefan problems is discussed in light of its efficiency and computational complexity. Three local parameters are used to equidistribute interpolation errors in maximum norm for temperature and a fourth one, in the event of mushy regions, to equidistribute L^1-interpolation errors for enthalpy within the mush. If certain quality mesh tests fail, then the current mesh is discarded and a new one completely regenerated by an efficient mesh generator, which in turn is briefly described. A typical triangulation is strongly graded to become very fine near computed interfaces and coarse away from them. Consecutive meshes are not compatible. The use of quadtree data structures is discussed as a means to reach a nearly optimal computational complexity in various tasks to be performed, mainly in generating a mesh and interpolating. Various implementation details are given so as to derive the computational complexity of each relevant subroutine. The approximation of both solutions and interfaces is drastically improved. The proposed method is robust in that it can handle the formation of cusps and mushy regions as well as the spontaneous appearance of phases. It is also superior in terms of computing time for a desired accuracy. Several numerical experiments illustrate these facts and provide quantitative information about each task complexity.

Key words. free boundaries, finite elements, adaptivity, mesh generation, quadtrees, computational complexity

AMS(MOS) subject classifications. 65N30, 65N50, 35R35

1. Introduction.

The two-phase Stefan problem models a phase change process which is governed by Fourier's law. The two phases, typically solid and liquid, are separated by an interface (or free boundary) that is not known in advance. Its motion is governed by the so-called Stefan condition, at least whenever the interface does not exhibit singularities. The solution is singular, however, on the interface no matter how smooth the latter is. This simple feature is responsible for the global numerical pollution observed in finite element solutions of the so-called enthalpy formulation. In such a formulation, the physical variables u (enthalpy) and θ (temperature) satisfy the partial differential equation (PDE)

$$u_t - \Delta\theta = f(\theta), \qquad \text{in } \Omega \times (0,T), \tag{1.1}$$

subject to the *strongly nonlinear* constitutive relation $\theta = \beta(u)$, where β vanishes on [0,1]; here 0 represents the melting temperature and 1 the latent heat. In particular,

*Received by the editors February 14, 1990; accepted for publication (in revised form) September 24, 1990. This work was partially supported by National Science Foundation grants, DMS-8805218 and DMS-9008999, and by Ministero della Pubblica Instruzione (Fondi per la Ricerca Scientifica 40%) and Consiglio Nazionale delle Ricerche (Istituto di Analisi Numerica, contract 880032601 and Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo," Sottoprogetto "Calcolo Scientifico per Grandi Sistemi") of Italy.

†Department of Mathematics and Institute for Physical Science and Technology, University of Maryland, College Park, Maryland 20742.

‡Istituto di Analisi Numerica del Consiglio Nazionale delle Ricerche, 27100 Pavia, Italy.

§ Dipartimento di Meccanica Strutturale, Università di Pavia and Istituto di Analisi Numerica del Consiglio Nazionale delle Ricerche, 27100 Pavia, Italy.

piecewise linear finite element approximations of (1.1) perform worse than expected according to the interpolation theory; the computational rate of convergence in the L^2-norm is never better than linear [15], [17]. A possible remedy to such an undesirable situation may be expressed in terms of suitably designed adaptive algorithms, such as that developed in [14]. The aim of this paper is thus to complement the basic mathematical setting of [14] with several crucial implementation issues that make our adaptive method (AM) competitive, as well as with numerous numerical experiments. We refer to [1] and [11] for an account of the state of the art on adaptive methods for parabolic PDEs and to [13] for a summary of AM; [12] is a preliminary report on AM.

As described in §3.1, we use three local parameters to equidistribute interpolation errors for temperature in *maximum norm*. If, in addition, mushy regions occur, we then use an extra local parameter to equidistribute L^1-interpolation errors for enthalpy within the mush. A typical triangulation is coarse away from the discrete interface, where discretization parameters satisfy a parabolic relation, whereas it is locally refined in the vicinity of the free boundary for the relation to become hyperbolic. Whenever the (nondegenerate) discrete interface escapes from the so-called *refined region* or any of the various quality tests performed on the computed solution fail, then the current mesh must be modified (see §3.2). Following the modern trend for transient problems, we do prefer remeshing to enrichment/coarsening strategies. This leads, though, to *noncompatible* consecutive meshes and raises several questions as to how to efficiently remesh the domain and transfer information among meshes. The first task is accomplished by the automatic mesh generator of [18], whereas the second one is a simple interpolation process. Their implementation, together with the effective resolution of the resulting strongly nonlinear algebraic problems, dictates the overall computational complexity (see §6.8). To reach a nearly optimal computational complexity in several search operations, we resort to *quadtree* data structures [2], [10], [13], [23]. They are fully discussed in §4 and next applied to the advancing front algorithm in §5 and the interpolation process in §6.5. Thus, compared to the standard fixed mesh method (FMM) [4], [15], [17], AM exhibits a superior performance as expressed in terms of computing time for a desired accuracy. This fact is illustrated in §7 in light of several numerical experiments. Moreover, the method is robust in that it can automatically detect and handle singular situations such as the formation of mushy regions and cusps as well as the spontaneous occurrence or disappearance of phases (see §§7.3 and 7.4). The approximation of interfaces is also drastically improved with the self-adaptive approach.

The main advantage of our remeshing strategy is its computational simplicity and efficiency, as CPU time for mesh generation reveals (see §7). In particular, a mesh is eliminated as soon as the interpolation process has been carried out, thus releasing the corresponding memory allocation. On the contrary, enrichment/coarsening procedures need a quite delicate data structure to keep track of successive mesh changes, which seems to be very sensitive to the number of refinement levels. For strongly graded meshes, such as those required by our local refinement strategy, this becomes a critical issue. In fact, as a refinement level refers only to its predecessor and not to a coarse triangle of the original mesh, the elimination of triangles of much smaller size than that of the desired final element (strong derefinement operation) may encounter serious computational as well as topological difficulties. Such difficulties are completely circumvented in our approach.

The local refinement strategy is based on four basic heuristic observations, is stable in various Sobolev spaces, and is convergent with a rate of essentially $O(\tau^{1/2})$, as

explained in [13], [14]; here τ stands for the time step. Even though this order coincides with previous ones for fixed meshes, we stress that, for well-behaved interfaces, the present method requires $O(\tau^{-3/2})$ degrees of freedom as opposed to $O(\tau^{-2})$ for previous practical schemes [5], [15], [22].

The code was written in C-language on a VAX/VMS. The choice of the C-language depends mainly on the possibility of defining new data structures, such as the quadtrees, on the recursion capability and memory allocation possibility, which is essential to avoid very large static memory space allocations. The graphic interface is based on the TEKTRONIX/STI library with a TEK-4107 graphic terminal.

The paper is organized as follows. Both the theoretical and numerical settings are described in §2. The adaptive local refinement strategy is recalled in §3; details can be found in [14]. The quadtree data structure is outlined in §4, whereas §5 deals with the automatic mesh generator. The code structure, along with the main subroutines involved and their complexity, is described in §6; the overall computational complexity is then derived in §6.8. A number of numerical experiments illustrate the superior performance of the proposed finite element adaptive method in §7. Comparisons with FMM are presented for most examples as well. Concluding remarks about AM and what lies ahead are finally presented in §8.

2. The setting. Let $\Omega \subset \mathbf{R}^2$ be a bounded convex polygon and $T > 0$ be fixed. Set $Q := \Omega \times (0,T)$. Let $\beta : \mathbf{R} \to \mathbf{R}$ be continuous, strictly increasing on $\mathbf{R}\backslash[0,1]$, and zero on [0,1]. The simplest case ($s^- := \max(-s,0)$, $s^+ := \max(s,0)$),

$$\beta(s) := c_1 s^- + c_2(1-s)^+, \quad \forall\, s \in \mathbf{R}, \tag{2.1}$$

where $c_1 < 0 < c_2$ are fixed constants, corresponds to an ideal material with constant thermal properties. Let u_0 indicate the initial enthalpy. Let $\theta_0 := \beta(u_0)$ denote the initial temperature, $I_0 := \{x \in \Omega : \theta_0(x) = 0\}$ the initial interface or mushy region, and set $\Omega^+ := \{x \in \Omega : \theta_0(x) > 0\}, \Omega^- := \{x \in \Omega : \theta_0(x) < 0\}$. Let $f : \mathbf{R} \to \mathbf{R}$ denote a heating source. All these quantities satisfy suitable regularity properties, as stated in [14].

The continuous two-phase Stefan problem consists of finding $\{\theta, u\}$ such that

$$u_t - \Delta_x \theta = f(\theta), \quad \text{in } Q, \tag{2.2}$$

subject to a variational boundary condition for θ, an initial condition for u, and the *strongly nonlinear* constitutive relation

$$\theta = \beta(u). \tag{2.3}$$

Let $I(t) := \{x \in \Omega : \theta(x,t) = 0\}$ indicate the interface at time t. If $\partial I(t)$ is smooth at x, then the following *Stefan condition* holds:

$$[\![\nabla_x \theta(x,t)]\!] \cdot \nu_x = [\![u(x,t)]\!]\, V(x), \tag{2.4}$$

where ν_x is the unit vector normal to $\partial I(t)$ and $V(x)$ is the normal velocity of $\partial I(t)$, both at point x. Hereafter, $[\![\cdot]\!]$ stands for the jump operator. Note that (2.4) includes the classical condition corresponding to a smooth curve $I(t)$ as well as that for a mushy region with smooth boundary. However, (2.4) does not hold at singular points (cusps) of $I(t)$.

Let $\tau = T/\mathbf{N}$ be the time step. Let $\{\mathcal{S}^n\}_{n=1}^{\mathbf{N}}$ denote a set of graded partitions of Ω into triangles that are shape regular [3, p. 132] and weakly acute uniformly with respect to $1 \le n \le \mathbf{N}$, and set $\mathbf{S}^n := \text{card}(\mathcal{S}^n)$. The second condition means that for any pair of adjacent triangles the sum of the opposite angles relative to the

common side does not exceed π. Given a triangle $S \in \mathcal{S}^n$, h_S stands for its size and verifies $\lambda\tau \le h_S \le \Lambda\tau^{1/2}$, where $0 < \lambda, \Lambda$ are fixed constants. Let $\mathcal{E}^n$ denote the set of interelement boundaries, $\mathcal{R}^n := \cup\{S \in \mathcal{S}^n : h_S = O(\tau)\}$ the refined (or hyperbolic-like) region, and $\mathcal{P}^n := \Omega\backslash\mathcal{R}^n$ the parabolic region. Let $\mathbf{V}^n \subset H_0^1(\Omega)$ indicate the usual piecewise linear finite element space over $\mathcal{S}^n$ and $\Pi^n : C^0(\bar{\Omega}) \to \mathbf{V}^n$ the Lagrange interpolation operator. Let $\mathcal{J}^n := \{x_j^n\}_{j=1}^{\mathbf{J}^n}$ denote the nodes of $\mathcal{S}^n$ and $\{\chi_j^n\}_{j=1}^{\mathbf{J}^n}$ the canonical basis of $\mathbf{V}^n$; thus $\mathbf{J}^n := \text{card}(\mathcal{J}^n)$. The discrete initial enthalpy $U^0 \in \mathbf{V}^1$ is defined at a generic node x_j^1 of $\mathcal{S}^0 := \mathcal{S}^1$ to be

(2.5)
$$U^0(x_j^1) := u_0(x_j^1) \quad \forall\, x_j^1 \in \Omega\backslash\partial I_0, \qquad U^0(x_j^1) := 1\ (0) \quad \forall\, x_j^1 \in \partial\Omega^+ \cap \Omega\ ((\partial\Omega^- \cap \Omega)\backslash\partial\Omega^+).$$

Hence U^0 is easy to evaluate in practice. The discrete scheme then reads as follows: *for any* $1 \le n \le N$, *select* $\mathcal{S}^n$ *and find* $U^n, \Theta^n \in \mathbf{V}^n$ *such that*

$$\Theta^n = \Pi^n \beta(U^n), \tag{2.6}$$

$$\tau^{-1}\langle U^n - U^{n-1}, \chi\rangle^n + \langle \nabla\Theta^n, \nabla\chi\rangle = \langle f(\beta(U^{n-1})), \chi\rangle^n \quad \forall\, \chi \in \mathbf{V}^n, \tag{2.7}$$

where $\langle\cdot,\cdot\rangle^n$ is defined by

$$\langle \varphi, \chi\rangle^n := \int_\Omega \Pi^n(\varphi\chi)dx \quad \forall\, \varphi, \chi \in C^0(\bar{\Omega}). \tag{2.8}$$

Note that the integral in (2.8) can be easily evaluated elementwise via the vertex quadrature rule, which is exact for piecewise linear functions [3, p. 182]. Let $\mathbf{M}^n := (\langle \chi_i^n, \chi_j^n\rangle^n)_{i,j=1}^{\mathbf{J}^n}$ denote the mass matrix and $\mathbf{K}^n := (\langle \nabla\chi_i^n, \nabla\chi_j^n\rangle)_{i,j=1}^{\mathbf{J}^n}$ the stiffness matrix, and set

$$\hat{U}^{n-1} := \Pi^n U^{n-1}, \qquad \hat{\Theta}^{n-1} := \Pi^n[\beta(U^{n-1})]. \tag{2.9}$$

Equation (2.7) can then be written in matrix form as

$$\mathbf{M}^n U^n + \tau\mathbf{K}^n\Theta^n = \mathbf{M}^n(\hat{U}^{n-1} + \tau f(\hat{\Theta}^{n-1})), \tag{2.10}$$

where we have identified piecewise linear functions with the vector of their nodal values.

The interpolation process (2.9) transfers information between two consecutive noncompatible meshes $\mathcal{S}^{n-1}$ and $\mathcal{S}^n$ solely through the nodes of $\mathcal{S}^n$, thus incorporating the interpolation error $U^{n-1} - \hat{U}^{n-1}$, which eventually accumulates in time. The choice of $\mathcal{S}^n$, as described in §3, is dictated by the need to prevent such an effect from compromising accuracy. It is necessary, however, to restrict the number of admissible mesh changes to $O(\tau^{-1/2})$, which in turn is compatible with the mesh selection algorithm of §3.2.

Since (2.6) enforces the constitutive relation (2.3) at the discrete level, the algebraic system (2.10) is (strongly) nonlinear and strictly monotone. Its unique solution can be easily and efficiently computed via a nonlinear (successive overrelaxation) method [15], [16], [17], [21] (see §6.6).

3. Adaptive mesh modification. Our objective now is to recall the adaptive local refinement algorithm; it is presented without justification, for which we refer to [14]. The basic principles of mesh adaptation reflect the physical property that the Stefan problem behaves as parabolic away from the interface but possesses a first-order hyperbolic-like structure in the vicinity of the interface. Let $\mathcal{S}^{n-1}$ be the current mesh and $\mathcal{S}^n$ a mesh to be constructed with information extracted from Θ^{n-1} and U^{n-1}. The quality criteria for mesh selection, based on equidistributing pointwise

interpolation errors for Θ^{n-1} and L^1-interpolation errors for U^{n-1} within the mush, allow interpolation errors of $O(\tau)$ at every single mesh change. The number of remeshings is limited to $O(\tau^{-1/2})$ to prevent those errors from compromising accuracy, but the method is stable regardless of this constraint [14]. The meshes $\mathcal{S}^{n-1}$ and $\mathcal{S}^n$ are *noncompatible*, which in turn reduces overhead required by enrichment-coarsening procedures and simplifies programming, but entails the use of an efficient automatic mesh generator [10], [18].

Since no confusion is possible, we remove the superscripts and simply use the following notation: $\mathcal{S} := \mathcal{S}^{n-1}$, $\hat{\mathcal{S}} := \mathcal{S}^n$, $\mathcal{E} := \mathcal{E}^{n-1}$, $\mathcal{R} := \mathcal{R}^{n-1}$, $\hat{\mathcal{R}} := \mathcal{R}^n$, $U := U^{n-1}$, $\Theta := \Theta^{n-1}$, and $F := F^{n-1} := \{x \in \Omega : \Theta^{n-1}(x) = 0\}$ (*discrete interface* or *mushy region*), for $2 \le n \le \mathbf{N}$.

Set $\mathcal{S}_W := \{S \in \mathcal{S} : S \cap W \neq \emptyset\}$, $\mathcal{E}_W := \{e \in \mathcal{E} : e \subset \partial S, S \in \mathcal{S}_W\}$, $\mathcal{W} := \cup_{S \in \mathcal{S}_W} S$, and $\mathbf{S}_W := \mathrm{card}(\mathcal{S}_W)$ for any given set $W \subset \bar{\Omega}$.

3.1. Local mesh parameters. Note that $\nabla\Theta|_S \in [P^0(S)]^2$ for all $S \in \mathcal{S}$. Set $d_S := |\nabla\Theta|_S|$ and define, for all $e \in \mathcal{E}$, $h_e = \mathrm{length}(e)$ and $D_e := |[\![\nabla\Theta]\!]_e|/h_e$. As usual, $|\cdot|$ stands for any $\mathbf{R}^2$-norm. Note that these quantities are easy to evaluate in practice. We then introduce the following local parameters away from the interface F:

$$\hat{h}_e := \mu_1 \frac{\tau^{1/2}}{D_e^{1/2}}, \quad \forall\, e \in \mathcal{E}\backslash\mathcal{E}_F; \qquad \hat{h}_S := \mu_2 \frac{\tau^{1/2}}{d_S}, \quad \forall\, S \in \mathcal{S}\backslash\mathcal{S}_F. \tag{3.1}$$

Set $B := \mathrm{Interior}(\cup\{S \in \mathcal{S}\backslash\mathcal{S}_F : \min_{e \in \mathcal{E}\backslash\mathcal{E}_F : e \subset \partial S}(\hat{h}_S, \hat{h}_e) < \lambda\tau\})$. Note that $\mathcal{B}$, the closure of B, represents the set where the discrete derivatives d_S, D_e are badly behaved.

At this stage $\mu_1, \mu_2 > 0$ are arbitrary constants which, in practice, result from computational considerations as well as specific properties of the problem at hand (see §§6.2.1, 6.2.2). The same comment applies to λ, Λ.

In order to determine a local parameter near the interface, in the absence of mush, we proceed as follows. First, for each piece $F_S := F \cap S$ where $S \in \mathcal{S}_F$, we use a suitable discretization of (2.4) to approximate the velocity V_S of F_S, namely,

$$V_S := (\nabla\Theta|_{S_1} - \nabla\Theta|_{S_2}) \cdot \nu, \tag{3.2}$$

where $S_1, S_2 \in \mathcal{S}$ belong to each phase, are displayed on the direction ν (unit vector normal to F), and satisfy $\mathrm{dist}(S_i, S) \ge h_S$ $(i = 1, 2)$ (see [14, §§3,4]). We next consider a cone $\mathcal{C}_S$ of axis ν, vertex at S, opening $\pi/2$, and height $\mu_3 V_S \tau^{1/2}$ as being the region most likely to contain the evolution of $F_S := S \cap F$ for at least $O(\tau^{-1/2})$ time steps. This is so because $\hat{\mathcal{S}}$ must meet the quality tests of §3.2 during that number of time steps. The local parameter associated with the interface is thus defined by (see [14, §§3,4])

$$\hat{h}_{F_S} := \tau \min\{\max(\lambda, V_S), M\}. \tag{3.3}$$

The above two new constants $\mu_3 > 0$ and $M \ge \lambda$ are arbitrary at this stage. The same rules of selection as for the previous four constants apply; μ_3 may depend on n (see §6.2.4).

We consider now the case of discrete mushy regions, which are a striking manifestation of degeneracy in that the underlying PDE becomes $u_t = f(0)$, as least formally. They may also be produced by the numerical method, as in §7.3. A *discrete mushy region* $\mathcal{M}$ is defined as the union of all triangles $S \in \mathcal{S}$ such that $\Theta|_S$ vanishes. Set $\Gamma := \partial\mathcal{M}$. We say that $\mathcal{M}$ is *degenerate* on a side $e \subset \Gamma$ if

$$\mathrm{osc}_e(U) < 1 \quad \text{and} \quad d_S \le \mu_4 h_e \quad \forall\, S \in \mathcal{S}_e, \tag{3.4}$$

where $\mu_4 > 0$ is a suitable constant and $\mathrm{osc}_e(U)$ is the oscillation of U in the set $\cup_{S\in\mathcal{S}_e} S$, which is easy to calculate because U is piecewise linear. Let Γ_D denote the union of all those e's and set $\Gamma_N := \Gamma\backslash\Gamma_D$, which stands for the nondegenerate part of Γ. The condition (3.4) implies that discrete second derivatives are uniformly bounded across Γ_D.

Set $I := (F\backslash\mathcal{M}) \cup \Gamma_N$, which represents the *nondegenerate interface*, namely, the union of the sharp interface and the nondegenerate part Γ_N of Γ.

Note that $\nabla U|_S \in [P^0(S)]^2$ and set $D_S := |\nabla U|_S|$ for all $S \in \mathcal{S}$. We then define the following local parameter to determine the local meshsize within the mush:

$$\hat{h}_S := \tau \max\left[\lambda, \min\left(\Lambda^\star, \frac{\mu_5}{D_S^{1/3}}\right)\right] \quad \forall\, S \in \mathcal{S}_\mathcal{M}\backslash\mathcal{S}_{\Gamma_N}, \tag{3.5}$$

where $\mu_5 > 0$ and $\Lambda^\star \geq \lambda$ are suitable constants (see [14, §S4]).

We also need a substitute for (3.2) for all sides $e \subset \Gamma_N$. In light of (2.4), we then consider $S_1, S_2 \in \mathcal{S}$ as in (3.2) and propose the quantity

$$V_e := \frac{(\nabla\Theta|_{S_1} - \nabla\Theta|_{S_2})\cdot\nu}{\mathrm{osc}_e(U)}. \tag{3.6}$$

3.2. Mesh selection algorithm. The initial mesh $\mathcal{S}^1$ is built via (3.1), (3.3), and (3.5), with the required pointwise information extracted from θ_0 and u_0 (see §§6.2.3, 6.2.5). Assuming now that we have a mesh $\mathcal{S}$, we would like to discuss the various tests to be performed on the computed solution $\{\Theta, U\}$ to either accept or discard $\mathcal{S}$ (see §6.7).

The first test consists of checking whether or not the discrete interface I is within the refined region $\mathcal{R}$. In the event I escapes from $\mathcal{R}$, we say that the test has failed. Note that only the nondegenerate boundary Γ_N of $\mathcal{M}$ must belong to $\mathcal{R}$, whereas $\mathcal{M}$ may exit $\mathcal{R}$. Rejection of $\mathcal{S}$ is mostly dictated by failure of this test (see §7).

The second test ascertains that interpolation errors are still equidistributed correctly:

$$h_e \leq \mu_1^\star\, \hat{h}_e \quad \forall\, e \in \mathcal{E}\backslash(\mathcal{E}_F \cup \mathcal{E}_B), \qquad h_S \leq \mu_2^\star\, \hat{h}_S \quad \forall\, S \in \mathcal{S}\backslash(\mathcal{S}_F \cup \mathcal{S}_B); \tag{3.7}$$

here $\mu_1^\star, \mu_2^\star > 1$ are suitable constants. This rules out the possibility of an excessive refinement induced by large discrete derivatives. However, the new local meshsize might be much smaller than the current one, if influenced by the refined region $\hat{\mathcal{R}}$. This test fails at least once in all examples of §7.

Sometimes the interface velocity may vary substantially during an $O(\tau^{-1/2})$ period of time so as to make (3.3) inadequate. This is avoided by simply enforcing the following third test (see [14, §§3,4]):

$$\mu_3^-\, \hat{h}_{F_S} \leq h_S \leq \mu_3^+\, \hat{h}_{F_S} \quad \forall\, S \in \mathcal{S}_I; \tag{3.8}$$

here $\mu_3^- < 1 < \mu_3^+$ are suitable constants. Example III of §7.3, for instance, makes (3.8) fail.

If any one of the above tests fails, then the current mesh $\mathcal{S}$ is rejected, as is the solution $\{\Theta, U\}$, which is overwritten with the previously computed solution. A new *graded* mesh $\hat{\mathcal{S}}$ with the following properties is then generated. To preserve the constraint $h_{\hat{S}} \geq \lambda\tau$, we must keep $\mathcal{S}_B$ fixed because discrete derivatives are badly behaved. In addition, we must not modify $\mathcal{S}_I$, because this would cause an inadmissible interpolation error for U [14, §§3,4]. Hence,

$$S \in \hat{\mathcal{S}} \quad \forall\, S \in \mathcal{S}_B \cup \mathcal{S}_I \tag{3.9}$$

is the first restriction on $\hat{\mathcal{S}}$. The second one reads

$$\lambda\tau \le h_{\hat{S}} \le \min_{\substack{S'\in\mathcal{S}_I:C_{S'}\cap\hat{S}\neq\emptyset \\ e\in\mathcal{E}\setminus(\mathcal{E}_B\cup\mathcal{E}_F):e\cap\hat{S}\neq\emptyset \\ S\in\mathcal{S}\setminus(\mathcal{S}_B\cup\mathcal{S}_I):S\cap\hat{S}\neq\emptyset}} \left(\Lambda\tau^{1/2}, \hat{h}_{F_{S'}}, \hat{h}_e, \hat{h}_S\right) \quad \forall\, \hat{S}\in\hat{\mathcal{S}}. \tag{3.10}$$

This accounts for the equidistribution of interpolation errors for both Θ and U, as well as the definition of refined region $\hat{\mathcal{R}}$, which satisfies $F\subset\hat{\mathcal{R}}$. The effective implementation of (3.10) will be discussed in §6.2, where an auxiliary meshsize function $\mathbf{h}:\bar{\Omega}\to\mathbf{R}^+$ is defined.

3.3. Eliminating fixed triangles. All triangles in $\mathcal{S}_B\cup\mathcal{S}_I$ were left fixed, which is nearly the best we can do in a general setting. For computational purposes, however, it is always preferable to remove the contraint (3.9), which is feasible whenever I is smooth [14, §S4.1]. The entire mesh can thus be regenerated, on replacing (3.1) by

$$\hat{h}_e := \max\left(\lambda\tau, \mu_1 \frac{\tau^{1/2}}{D_e^{1/2}}\right) \ \forall\, e\in\mathcal{E}\setminus\mathcal{E}_F, \qquad \hat{h}_S := \max\left(\lambda\tau, \mu_2 \frac{\tau^{1/2}}{d_S}\right) \ \forall\, S\in\mathcal{S}\setminus\mathcal{S}_F. \tag{3.11}$$

Both strategies are virtually equivalent for well-behaved interfaces, as illustrated in §7.

4. Quadtree data structures. To reach a nearly optimal computational complexity in several crucial routines of our finite-element code, it is essential to devise a suitable data structure to organize a finite set of points of the euclidean plane. Such a structure is called *quadtree* [2], [10], [13], [23].

4.1. Definition of quadtrees. The following operations should be performed as fast as possible: finding a point in the structure; inserting a new point in the structure; removing a point from the structure; finding a point in the structure nearest to a given point; finding all points in the structure lying inside a given set. The basic idea behind quadtrees is similar to the binary tree organization of totally ordered sets [7], [19]. The main difference is that we deal with *couples* of coordinates; thus it is not possible to define a *total* order without losing the information of "neighborhood" between two points.

Let $\mathcal{P}$ denote the set of points of $\bar{\Omega}$ which is to be organized in a quadtree structure and let R be a rectangle (cartesian product of two intervals of the real line) containing all points of $\mathcal{P}$. We can choose, for instance, the smallest rectangle containing Ω. Let us suppose for simplicity that $R=[0,1)\times[0,1)$. Then R is divided into four equal quadrants R_i $(i=1,\cdots,4)$, which denote the upper-right, upper-left, lower-right, and lower-left quadrant, respectively. Hence, $R_1=[\frac{1}{2},1)\times[\frac{1}{2},1)$, $R_2=[0,\frac{1}{2})\times[\frac{1}{2},1)$, $R_3=[\frac{1}{2},1)\times[0,\frac{1}{2})$, and $R_4=[0,\frac{1}{2})\times[0,\frac{1}{2})$. Each of these rectangles is partitioned again into four pieces in the same manner as before, giving rise to R_{ij} $(i,j=1,\cdots,4)$. Such a subdivision process is sequentially repeated for each rectangle $R_{i_1,\cdots,i_n}$ so created until it contains no more than one point of $\mathcal{P}$. Hence, if $R_{i_1,\cdots,i_n}$ contains at most one element of $\mathcal{P}$, then the predecessor rectangle $R_{i_1,\cdots,i_{n-1}}$ must contain at least two points of $\mathcal{P}$. In this way we end up with a spatially organized tree structure (Fig. 4.1). It can be easily proved that this iterative process eventually terminates (if the set $\mathcal{P}$ is finite and does not contain a couple of coincident points). More precisely, the depth d of the tree can be bounded from above as follows:

$$d \le -\log_2\left(\min_{P,Q\in\mathcal{P}} |P-Q|_\infty\right).$$

Each node of the quadtree represents a rectangle $R_{i_1,\cdots,i_n}$ which can be empty, can contain one node, or can be further divided into four subquadrants (Fig. 4.2). Given the initial rectangle R, any subquadrant $R_{i_1,\cdots,i_n}$ (i.e., the coordinates of its vertices) can be uniquely determined by the sequence $i_1,\cdots,i_n$, which is recursively obtained during the inspection of the quadtree starting from its root.

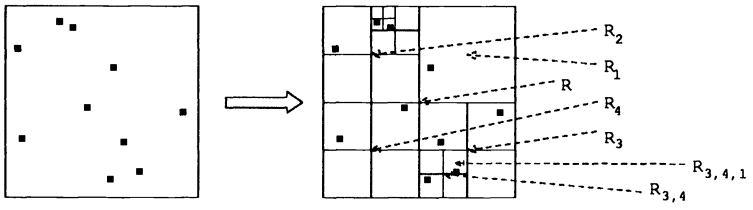


FIG. 4.1. *Points organized in a quadtree structure.*

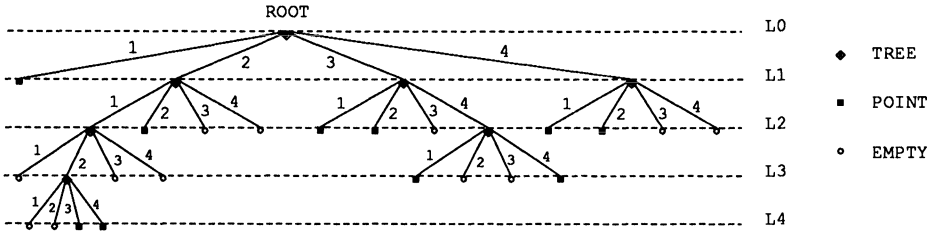


FIG. 4.2. *The quadtree structure corresponding to Fig.* 4.1.

More precisely, the quadtree structure is a tree of QUADRANT structures, each of which consists of four QUAD fields (Fig. 4.3). Any QUADRANT structure corresponds to the four subquadrants of a given level whereas a QUAD structure corresponds to a single rectangle $R_{i_1,\cdots,i_n}$. **ROOT** is the name of the isolated QUAD structure corresponding to the initial rectangle R and thus addressing the whole quadtree. QUAD is defined as a couple (FLAG, POINTER), where FLAG is an integer number and POINTER is a memory address. FLAG can have the following values:

— **EMPTY** means that there are no points in the rectangle; POINTER is **NULL**.
— **POINT** means that a single point of $\mathcal{P}$ belongs to the rectangle; POINTER addresses the data structure of such a point (coordinates, flags, and so on).
— **TREE** means that more than one point of $\mathcal{P}$ belongs to the rectangle; POINTER addresses the QUADRANT containing the information on the four subquadrants in which the rectangle is subdivided.

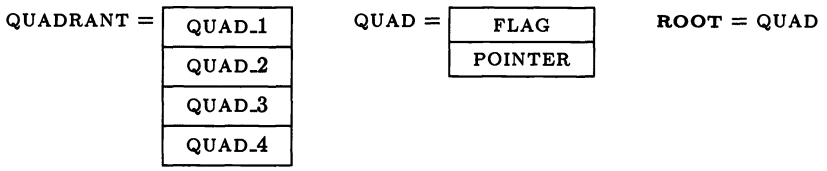


FIG. 4.3. *Definition of the quadtree structure.*

The whole quadtree is addressed by **ROOT**. In fact, if FLAG[**ROOT**]=**EMPTY**, then $\mathcal{P} = \emptyset$; whereas if FLAG[**ROOT**]=**POINT**, then the quadtree contains only one point. Otherwise, POINTER[**ROOT**] addresses the first-level QUADRANT and, as already said, this information, together with the coordinates of the upper-right and lower-left corners of the initial rectangle R, makes it possible to derive the definition of each rectangle of the quadtree structure. It is evident that any QUAD may be considered as the root of the subtree generated from it.

Let us now consider the five aforementioned tasks in light of the quadtree structure.

4.2. Finding or inserting a point in the quadtree. Both the search and the inserting operations are quite easy to implement: it is sufficient to inspect the quadtree, descending from a quadrant in one of its subquadrants in the direction suggested by the coordinates of the desired point, say P, until a point-QUAD or an empty-QUAD is reached. More precisely, denoting by Q the center of a quadrant **Q** containing P, the descent direction i ($1 \leq i \leq 4$) is simply given by the formula $i := 2.5 - 0.5\ \mathrm{sign}(x_P - x_Q) - \mathrm{sign}(y_P - y_Q)$. Let us call STEP(Q,P) the function exiting with that index. If the binary search ended up with a point-QUAD, a coordinate test will ensure that we have found the right point. If we end up with an empty-QUAD, or if the coordinate test fails, then the searched point is not contained in the quadtree structure. We can thus insert it into the structure in a simple way, namely, either by placing it into the last rectangle (setting FLAG to **POINT**), if it was empty, or by sequentially partitioning the rectangle containing two points into four quadrants and descending one level down following the direction given by STEP until the underlying points belong to two distinct quadrants (Fig. 4.4).

FIG. 4.4. *Inserting a new point* ○ *in the quadtree.*

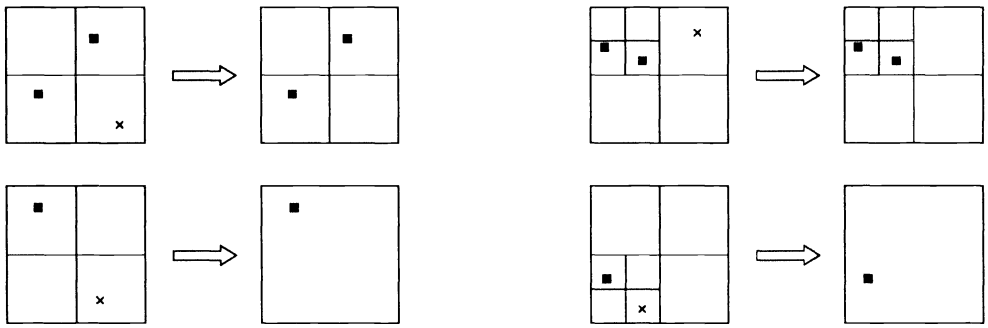


FIG. 4.5. *Removing a point* × *from the quadtree.*

4.3. Removing a point from the quadtree. The deletion operation consists of setting to **EMPTY** the point-QUAD containing the point P to be deleted, followed by a packing of the quadtree structure if a tree-QUAD containing only one point has

been generated by the emptying operation (Fig. 4.5). As a matter of fact, removing a point-QUAD **P** from the quadtree is done by the recursive function DROP(P,Q), first called with **Q=ROOT**, which removes **P** from the quadtree addressed by the QUAD **Q** and exits addressing the modified quadtree. Specifically, if FLAG[**Q**]=**EMPTY** or FLAG[**Q**]=**POINT** and POINTER[**Q**]≠**P**, DROP simply exits because **P** is not present in the quadtree **Q**; if POINTER[**Q**]=**P**, then FLAG[**Q**]=**EMPTY** is returned; lastly, if FLAG[**Q**]=**TREE**, the recursive call DROP(P,QP) is invoked, where the QUAD **QP** represents the unique subquadrant of **Q** containing the point P. After this recursive call, if POINTER[**Q**] contains one point-QUAD, say **QPOINT**, and three empty-QUADs, then the QUADRANT structure addressed by POINTER[**Q**] is removed (in order to free the memory) and the point-QUAD **Q** is overwritten by **QPOINT** on return; otherwise DROP simply exits (see Flow chart 4.1).

```
DROP(P,Q)
switch(FLAG[Q])
 EMPTY:return
 POINT: if(POINTER[Q]≠P) return
        else FLAG[Q]=EMPTY; POINTER[Q]=NULL; return
 TREE:  QP=POINTER[Q](STEP(Q,P))
        DROP(P,QP)
        EMPTY_CNT=0; POINT_CNT=0
        for i = 1,··· ,4
          QI=POINTER[Q](I)
          if (FLAG[QI]=EMPTY) EMPTY_CNT=EMPTY_CNT+1
          if (FLAG[QI]=POINT) POINT_CNT=POINT_CNT+1; QPOINT=POINTER[QI]
        endfor
        if (POINT_CNT=1 & EMPTY_CNT=3)
          FREE_QUADRANT(POINTER[Q])
          FLAG[Q]=POINT; POINTER[Q]=QPOINT
        endif
        return
```

FLOW CHART 4.1. *Recursive procedure* DROP *for removing a point from the quadtree.*

4.4. Finding a point in the quadtree nearest to a given point $P_0 \in \bar{\Omega}$. We define the recursive function NEAREST(P_0,Q,P,D), first called with **Q=ROOT**, **P=NULL**, and **D**= $+\infty$, which finds the point **P** belonging to the quadtree **Q**, if any, subject to the constraint $|P - P_0|_\infty <$ **D**. If successful, NEAREST updates **P** and sets **D**= $|P - P_0|_\infty$. Note the use of the *max* distance, which is preferable in terms of performance of the algorithm. We now explain how NEAREST works. Assume that FLAG[**Q**]=**TREE**; otherwise it is obvious. Denote by $\mathbf{D}_i$ the distance between P_0 and the subquadrant $\mathbf{Q}_i$ of QUAD **Q** and suppose $\mathbf{D}_1 \leq \mathbf{D}_2 \leq \mathbf{D}_3 \leq \mathbf{D}_4$. Recursively for all $1 \leq i \leq 4$, if $\mathbf{D}_i \geq$**D**, then NEAREST simply exits because no point can be found in **Q**; otherwise a recursive call NEAREST(P_0,Q_i,P,D) is done, which updates **P** and decreases **D** on return. The result of such a recursive process is a point **P** of the quadtree nearest to P_0, at a distance **D**.

4.5. Finding all points in the quadtree lying inside a given set $E \subset \bar{\Omega}$. Denoting by F the smallest rectangle containing E, a recursive function INSIDE simply descends down the quadtree, starting from the root and selecting a subquadrant whenever it intersects F. At this stage, the overlapping test is inexpensive. When a point-QUAD **Q** containing a point P is reached, a test is performed to check whether or not P belongs to E.

4.6. Computational complexity. It is easy to see that the tasks (i)–(iv) described above require a number of operations proportional to the quadtree depth d, namely, $O(d)$. On the other hand, the computational complexity of task (v) is $O(md)$, where m is the number of points of the quadtree lying within the background rectangle F. This complexity is optimal if the ratio between m and the number of points belonging to E, the given set, is bounded above. Let us now assume that $\mathcal{P}$ is the set of vertices $\mathcal{J}$ of the mesh $\mathcal{S}$. The constraint $h_S \geq \lambda\tau$ for all $S \in \mathcal{S}$, together with the regularity of the mesh, results in $|P-Q| \geq C\tau$ for all $P, Q \in \mathcal{J}$. Hence the quadtree depth d verifies $d = O(\log \tau^{-1}) = O(\log \mathbf{J})$. As a result, the above binary search algorithms have a nearly optimal complexity.

5. Automatic mesh generator. A detailed description of the advancing front mesh generator ADVFRONT can be found in [18]. ADVFRONT performs nonuniform triangular decompositions of general planar domains which, in turn, may be neither polygonal nor simply connected. Since several mesh changes are to be carried out during the entire process, efficiency of the mesh generator plays a crucial role for AM to be competitive. Its computational complexity is nearly *optimal*, namely, $O(\mathbf{J} \log \mathbf{J})$. Such an efficiency is reached through the use of binary search techniques on quadtree structured data in performing several crucial consistency tests. Similar ideas can also be found in [10].

5.1. The advancing front. The user supplies the boundaries of the connected components of Ω, which may consist either of straight lines or of curve tracts. The information about curve pieces is retained in subsequent refinements. To create a mesh, this program uses a predefined meshsize function $\mathbf{h} : \Omega \to \mathbf{R}^+$ such as that mentioned in §6.2. It first generates all nodes on the boundaries, which form the initial advancing front; this front is intended to invade Ω. Selecting a proper front segment, where a new triangle will be created, is essential to keep the front length as short as possible. To minimize the computational complexity, the front nodes are stored in a quadtree. After such a crucial step, the algorithm generates an optimal triangle and next makes use of the quadtree structure to efficiently perform suitable consistency tests, as described below.

There is, however, a slight difference with the quadtree structure presented in §4, in that there is a chance of having "twin" nodes. It may happen that two or more topologically distinct nodes of the front are physically coincident, so that they have the same coordinates and are treated as a single entity: the FLAG field of the QUAD will have the value **TWIN** and the POINTER will address the list of "twin" nodes sharing the same coordinates.

Since no angle control is enforced, very large obtuse triangles might be generated at this stage. A proper remedy to this inconvenience consists of swapping the common side of two adjacent triangles whenever the sum of the opposite angles, relative to that side, exceeds π. Such an iterative postprocess, called FLIPDIAG, terminates in finite steps and guarantees that the resulting mesh is weakly acute [6], [8], [9], [20].

We now describe in more detail how ADVFRONT works. The mesh is generated in two successive steps. First, the program generates nodes on the boundary, say P_i, in such a way that segments on the polygonal so obtained have a length bounded by $\mathbf{h}(P_i)$. Second, each subdomain is triangulated independently. The first step is obviously inexpensive compared with the second one. In order to incorporate ADVFRONT in our local refinement strategy, information concerning triangles to be kept fixed, namely those in $\mathcal{I} \cup \mathcal{B}$, is preprocessed for boundary and subdomain gridding.

The polygonals $\partial\Omega$ and $\partial(\mathcal{I} \cup \mathcal{B})$ define the initial advancing front (**FRONT**), which in turn might consist of more than one connected component. **FRONT** will then be moving inwards until it becomes empty. To create a new element, a recursive function CHOOSE is called. It selects a front node simply by descending down the tree from the root in its *first* subquadrant and so on until a point-QUAD or an empty-QUAD **Q** is reached. If FLAG[**Q**]=**POINT**, this point is selected; otherwise the algorithm examines the *second* quadrant, and so on. This seems to be the fastest systematic algorithm to select a front node. The agreement between the selection order of the nodes and their relative position in Ω ensures that the front length does not increase excessively. Once a point P_1 has been chosen, we actually have a segment P_1P_2 on the front, P_2 being the next node of P_1 along **FRONT**. A third node Q is next added so as to form an isosceles triangle P_1P_2Q having base P_1P_2 and height $\mathbf{h}(B)$, where B designates the barycenter of the equilateral triangle associated with P_1P_2 (Fig. 5.1(a)). If the distance from Q to the nearest front node $\bar{Q}$ is less than a fixed tolerance, then the program decides to build the triangle $P_1P_2\bar{Q}$ instead of P_1P_2Q (Fig. 5.1(b)).

5.2. Consistency tests. The following consistency tests are now made on triangle P_1P_2P, where either $P = Q$ or $P = \bar{Q}$, to ensure the correct position of the new point P.

TESTANG: Denote by P_1' the previous node of P_1 and by P_2' the next one of P_2 along **FRONT**. Check whether $P_1'P_1 \cap P_1P_2P = \emptyset$ or $P_2P_2' \cap P_1P_2P = \emptyset$ (by means of angle tests). If negative, set either $P = P_1'$ or $P = P_2'$ (Fig. 5.2).

TESTGEN: Check that no front segment R_1R_2 intersects the triangle P_1P_2P. If negative, either a front node is internal to the triangle (then replace P by that node—Fig. 5.3(a)) or the segment R_1R_2 intersects both P_1P and P_2P (then set $P = R_i$ with R_i lying in the correct halfplane—Fig. 5.3(b)).

TESTTWIN: If P is a "twin" front node, select the copy of P that avoids self-intersections of the front after inserting the new triangle.

Once these consistency tests have been performed, the new triangle P_1P_2P is incorporated to the mesh, namely, is appended to the list structure defining the mesh. The advancing front is then updated as shown in Fig. 5.4: (a) P does not belong to the present front (Fig. 5.4(a)); (b) P is either the previous of P_1 or the next of P_2 along the present front (Fig. 5.4(b)); (c) P belongs to the present front, but it is contiguous neither to P_1 nor to P_2. In the last case, the number of connected components of the front increases or reduces by one, and two twin copies of P are made (Fig. 5.4(c)).

5.3. Computational complexity. It is dictated by the routines which search through the advancing front to create a new element. They are:

CHOOSE: selects the segment P_1P_2 for the new triangle P_1P_2Q;

NEAREST: finds $\bar{Q}$, the front node nearest to Q;

TESTGEN: finds any front segment intersecting the new element P_1P_2P.

A naïve implementation of those routines could have complexity $O(l)$, where $l = O(\mathbf{J}^{1/2})$ stands for the number of front nodes. Instead, for the first two functions, it becomes $O(\log l) = O(\log \mathbf{J})$ (thus nearly *optimal*) if the nodes of the advancing front are stored in a quadtree structure and the binary search algorithms of §4 are used (see §4.6). To perform the test TESTGEN on the current triangle S, it is sufficient to detect all front nodes lying within a ball centered at the barycenter of S and radius Ch_S. Let m_S indicate the corresponding number of front nodes detected by the function INSIDE of §4.5. Thus the computational complexity of TESTGEN is of order $O(m_S \log l)$ for each single S, whereas the total one becomes $\sum_{S \in \mathcal{S}} m_S \log l = O(\mathbf{J} \log \mathbf{J})$.

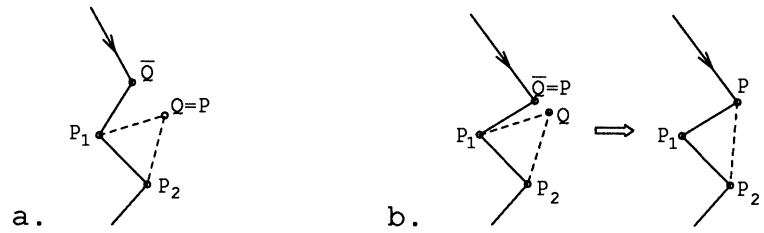


FIG. 5.1. *Generation of a new triangle.*

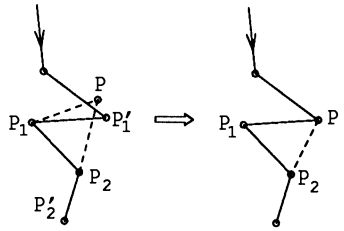


FIG. 5.2. TESTANG *test.*

FIG. 5.3. TESTGEN *test.*

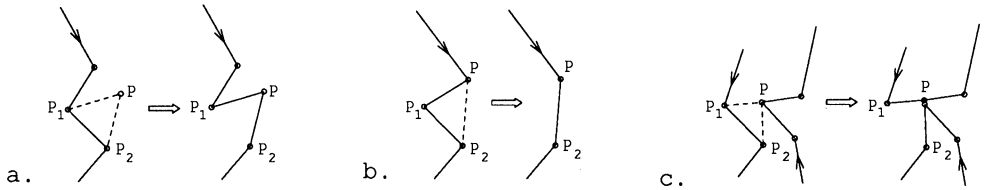


FIG. 5.4. *Updating of the advancing front.*

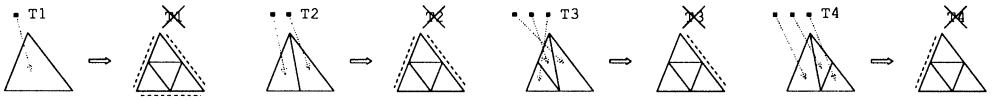


FIG. 5.5. *Elementary refinement operation types and* REFINE *actions.*

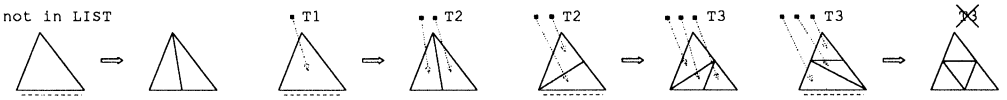


FIG. 5.6. SIDE_EFFECT *actions.*

We stress that the quadtree structure requires much less memory than the final mesh data, because the average length of the advancing front is expected to be the square root of the number of elements **S**. Hence the performance of the code is largely improved by the binary search techniques employed, even for small **S**.

5.4. Refinement techniques. Given a collection $\mathcal{S}_D$ of triangles, a local mesh refinement in $\mathcal{D} := \cup_{S\in\mathcal{S}_D} S$ is obtained simply by dividing each triangle of $\mathcal{S}_D$ into four subtriangles (by connecting the midpoints of the three edges). Note that the new nodes generated on $\partial\mathcal{D}$ produce a collateral effect on the triangles $S \in \mathcal{S}\backslash\mathcal{S}_D$ intersecting $\partial\mathcal{D}$. We now explain briefly how the procedures REFINE and SIDE_EFFECT work. A list (**OPLIST**) of *elementary refinement operations* (OP) is created, by setting TYPE[OP] = **T1** and POINTER[OP] = S, for all $S \in \mathcal{S}_D$. During the refinement algorithm, three more types of operations might be produced: TYPE[OP] = **T2** means that POINTER[OP] = S_1, S_2 (where $S_1 \cup S_2 = S \in \mathcal{S}_D$); TYPE[OP] = **T3** or **T4** means that POINTER[OP] = S_1, S_2, S_3 (where $S_1 \cup S_2 \cup S_3 = S \in \mathcal{S}_D$) (see Fig. 5.5). The routine REFINE pops one OP from **OPLIST**, makes the local modification of the mesh shown in Fig. 5.5 and then deletes OP from **OPLIST**. The triangles contiguous to S are dashed in Fig. 5.5. Next, for any contiguous triangle, say S_c, the routine SIDE–EFFECT scans **OPLIST** in order to find an operation concerning S_c (note that there might not exist), then modifies the mesh and either updates or deletes the operation concerning S_c in **OPLIST**, as shown in Fig. 5.6. The procedure ends when **OPLIST** is empty. Note that a consistent mesh is generated after executing any elementary operation.

It may happen that the diameter of a generated triangle is much bigger than the current value of the **h** matrix, because of the consistency tests. In this case, the refinement algorithm just explained may be used as a postprocess to recover the correct size. On the other hand, in order to create a mesh, the global refinement of a preliminary coarse grid is more convenient than the generation of the desired mesh. In fact, the refinement operation is inexpensive and the use of a doubled meshsize saves three quarters of the computational CPU time. This simple idea is embodied in our adaptive method and next followed by the use of FLIPDIAG to guarantee the weak acuteness of $\mathcal{S}$.

6. The finite element code. Our present goal is to describe the main program and the various subroutines involved along with their computational complexity. This section concludes with a discussion about the overall computational complexity of the adaptive algorithm. We will stick with the notation of §§2 and 3.

6.1. Main program. A schematic flow chart of our adaptive code is as follows:

$n = 1$; do (while $n \le \mathbf{N}$)	
{H_DEFINE(n)	(defines the meshsize function **h**)
MESH(n)	(generates the triangulation $\mathcal{S}^n$)
MATRIX	(computes $\mathbf{M}^n$ and $\mathbf{K}^n$)
INTERPOLATION(n)	(computes U^0 or $\hat{U}^{n-1} = \Pi^n U^{n-1}$)
do {STEFAN; $n = n + 1$}	(computes U^n and Θ^n)
while (TEST=**OK** & $n \le \mathbf{N}$)	
if (TEST=**FAIL**) {$n = n - 1$; SOL_UPDATE}}	(retrieves the previous solution)

In the sequel, we shall comment on the various subroutines above (in small capital face) and also derive their computational complexity, denoted by CC, noting that $\mathbf{S}^n \approx 2\mathbf{J}^n$.

6.2. Subroutine H_DEFINE (n). Tests on Θ^{n-1} and U^{n-1} described in §3 provide an expected local meshsize for the mesh $\mathcal{S}^n$ to be generated. This information is stored in a piecewise constant function $\check{\mathbf{h}}$, which is defined on an auxiliary

(or background) uniform square mesh $\mathcal{Q}$ of size $O(\tau^{1/2})$. Such a mesh, created and kept fixed from the beginning, satisfies $\bar{\Omega} \subset \cup_{R\in\mathcal{Q}} R$ and $\mathrm{card}(\mathcal{Q}) = O(\mathbf{N})$ because $r := [\mathrm{meas}(R)]^{1/2} = O(\tau^{1/2})$ for all $R \in \mathcal{Q}$. The essential difference between the cases $n=1$ and $n>1$ lies in the fact that, for $n = 1$, H_DEFINE works on the square grid $\mathcal{Q}$ with the initial data $\{\theta_0, u_0\}$, whereas it deals with $\mathcal{S}^{n-1}$ and $\{\Theta^{n-1}, U^{n-1}\}$ for $n>1$. The sequence of instructions reads:

if ($n = 1$) CONSTANTS_COMPUTE	(calculates the constants μ_1 and μ_2)
H_INIT	(initializes the matrix $\check{\mathbf{h}}$)
DERIVATIVE_WORK(n)	(updates $\check{\mathbf{h}}$ with $\hat{h}_e$ and $\hat{h}_S$)
VELOCITY_COMPUTE(n)	(calculates the constant μ_3^n)
FB_WORK(n)	(determines the predicted refined region $\check{\mathcal{R}}^n$)
AVERAGE	(postprocesses $\check{\mathbf{h}}$ to calculate $\mathbf{h}$)

6.2.1. Subroutine CONSTANTS_COMPUTE. Using the initial temperature θ_0 this subroutine calculates the constants μ_1 and μ_2 in (3.1). The user supplies two constants, namely n_1 and n_2, pretending that the number of elements created due to restrictions on second and first derivatives will be $N_1 := n_1/\tau$ and $N_2 := n_2/\tau$, respectively. This ensures the asymptotic parabolic constraint away from the initial interface I_0. To determine the constants we argue heuristically as follows. Suppose that first derivatives dominate everywhere; thus $\check{\mathbf{h}}|_R := \mu_2\tau^{1/2}/|\nabla\theta_0(x_R)|$ for all $R \in \mathcal{Q}$, where x_R denotes the center of R. Then $N_2 \approx \sum_{R\in\mathcal{Q}} r^2/\check{\mathbf{h}}^2|_R = (r^2/\mu_2^2\tau)\sum_{R\in\mathcal{Q}} |\nabla\theta_0(x_R)|^2$, whence $\mu_2 := (r/\sqrt{n_2})(\sum_{R\in\mathcal{Q}} |\nabla\theta_0(x_R)|^2)^{1/2}$. Similarly, this time assuming that second derivatives prevail, we easily find $\mu_1 := (r/\sqrt{n_1})(\sum_{R\in\mathcal{Q}} |D^2\theta_0(x_R)|)^{1/2}$. Note that both μ_1 and μ_2 are independent of τ. The computational complexity of this subroutine is given by $\mathrm{CC}_1 = O(\mathbf{N})$.

6.2.2. Subroutine H_INIT. This subroutine initializes $\check{\mathbf{h}}$ by setting $\check{\mathbf{h}}|_R := \Lambda\tau^{1/2}$ for all $R \in \mathcal{Q}$. Any further operation on $\check{\mathbf{h}}$ simply consists of updating its current value. In §§6.2.3 and 6.2.5 we use "$\check{\mathbf{h}}|_R \leftarrow s$" as an abbreviation for "$\check{\mathbf{h}}|_R := \min[\check{\mathbf{h}}|_R, \max(\lambda\tau, s)]$."

Remark 6.1. Avoiding time step loops. It may happen that the computed solution must be rejected right after the current mesh has been created. This is a typical situation when a new phase occurs spontaneously (see Remark 6.2). The subroutine H_INIT is then skipped, which implies that the entries of $\check{\mathbf{h}}$ can only decrease. The new mesh is thus a sort of refinement of the preceding one. Such a drastic but effective cure to avoiding time step loops, in turn makes the code more robust.

6.2.3. Subroutine DERIVATIVE_WORK (n). The initial matrix $\check{\mathbf{h}}$ is calculated, for all $R \in \mathcal{Q}$, as follows:

$$\check{\mathbf{h}}|_R \leftarrow \begin{cases} \tau^{1/2}\min(\mu_2/|\nabla\theta_0(x_R)|,\ \mu_1/|D^2\theta_0(x_R)|^{1/2}) & \text{if } R\cap I_0 = \emptyset, \\ \mu_5\tau/|\nabla u_0(x_R)|^{1/3} & \text{if } R\cap I_0 \neq \emptyset. \end{cases}$$

For $n > 1$, $\check{\mathbf{h}}$ is computed with the information extracted from $\{\Theta^{n-1}, U^{n-1}\}$. In fact, recursively for all $S \in \mathcal{S}^{n-1}\backslash(\mathcal{S}_F^{n-1} \cup \mathcal{S}_B^{n-1})$ and $e \in \mathcal{E}^{n-1}\backslash(\mathcal{E}_F^{n-1} \cup \mathcal{E}_B^{n-1})$, we set (Fig. 6.1)

$$\check{\mathbf{h}}|_R \leftarrow \mu_1\tau^{1/2}/D_e^{1/2} \quad \forall\, R\in\mathcal{Q} : R\cap e \neq \emptyset, \quad \check{\mathbf{h}}|_R \leftarrow \mu_2\tau^{1/2}/d_S \quad \forall\, R\in\mathcal{Q} : R\cap S \neq \emptyset,$$

and, for all $S \in \mathcal{S}_{\mathcal{M}}^{n-1}\backslash\mathcal{S}_{\Gamma_N}^{n-1}$,

$$\check{\mathbf{h}}|_R \leftarrow \min(\Lambda^\star\tau, \mu_5\tau/D_S^{1/3}) \quad \forall\, R \in \mathcal{Q} :\ R\cap S \neq \emptyset.$$

This definition is easier to implement and slightly weaker than that of [14, §4] because here the sides e are required to intersect R. This condition suffices also for theoretical purposes. The triangles of $\mathcal{S}_B^{n-1}$, so-called bad triangles, are identified at this stage as well. Since $h_S \leq \Lambda\tau^{1/2}$ for all $S \in \mathcal{S}^{n-1}$ and $r = O(\tau^{1/2})$, then $\text{CC}_1 = O(\mathbf{N})$ and $\text{CC}_n = O(\mathbf{J}^{n-1})$.

6.2.4. Subroutine VELOCITY_COMPUTE **(n).** Using the initial temperature θ_0 this subroutine calculates the constant μ_3^1, which determines the width of the initial predicted refined region $\check{\mathcal{R}}^1$; $\check{\mathcal{R}}^1$ is defined as the union of all cones $\mathcal{C}_P^1$ of axis perpendicular to ∂I_0 at $P \in \mathcal{Q}_{I_0}$, opening $\pi/2$ and $(v_P\tau^{1/2}\mu_3^1)$-high, and the initial mush. To this end, the user supplies a constant n_3 pretending that the number of triangles within $\check{\mathcal{R}}^1$ is $N_3 := n_3\tau^{-3/2}$. Set $\mathcal{Q}_{I_0} := \{R \in \mathcal{Q} : R \cap \partial I_0 \neq \emptyset\}$ and define v_R to be the velocity of ∂I_0 at R and $\tilde{v}_R := \min[\max(\lambda, v_R), M]$, where λ and M are given constants. A heuristic argument then leads to

$$N_3 = \sum_{R \in \mathcal{Q}_{I_0}} (r v_R \tau^{1/2} \mu_3^1)/(\tilde{v}_R^2 \tau^2)$$

and, as a result, to

$$\mu_3^1 := n_3/(r \sum_{R \in \mathcal{Q}_{I_0}} v_R/\tilde{v}_R^2).$$

The quantity r above is intended to approximate the length of ∂I_0 within a single square.

For $n > 1$, this subroutine calculates the constant μ_3^n by simply extracting information about the interface velocity from Θ^{n-1}. Given $S \in \mathcal{S}_I^{n-1}$ (triangle intersecting the nondegenerate interface), whose size h_S is expected to be comparable with $V_S\tau$ (V_S = velocity of I^{n-1} at S), the algorithm finds a couple of triangles displayed on the direction normal to I^{n-1} at S and located at a distance $O(h_S)$ from S. Then either (3.2) or (3.6) provides an expression for V_S. Finally, set $\tilde{V}_S := \min[\max(\lambda, V_S), M]$ and define $\mu_3^n := n_3/(\sum_{S \in \mathcal{S}_I^n} h_S V_S/\tilde{V}_S^2)$. As before, the quantity h_S above approximates the length of I^{n-1} in S. It is easily seen that $\text{CC}_1 = O(\mathbf{N}^{1/2})$ and $\text{CC}_n = O(\mathbf{S}_I^{n-1})$.

6.2.5. Subroutine FB_WORK **(n).** For $n = 1$, this subroutine determines $\check{\mathcal{R}}^1$ and updates $\check{\mathbf{h}}|_R$ for all $R \cap \check{\mathcal{R}}^1 \neq \emptyset$ with the corresponding local meshsize as follows: for all $P \in \mathcal{Q}_{I_0}$, set (Fig. 6.2(a))

$$\check{\mathbf{h}}|_R \leftarrow \tilde{v}_P\tau \quad \forall\, R \in \mathcal{Q}:\ R \cap \mathcal{C}_P^1 \neq \emptyset.$$

For $n > 1$, the predicted refined region $\check{\mathcal{R}}^n$ is defined as the union of cones $\mathcal{C}_S^{n-1}$, for all $S \in \mathcal{S}_I^{n-1}$, and the discrete mush $\mathcal{M}^{n-1}$. Then, $\check{\mathcal{R}}^n$ and $\check{\mathbf{h}}$ are determined in the same fashion as for $n = 1$, now using Θ^{n-1} and $\mathcal{S}^{n-1}$ to compute $\tilde{V}_S$ instead of $\tilde{v}_P$; namely, for all $S \in \mathcal{S}_I^{n-1}$, set (Fig. 6.2(a))

$$\check{\mathbf{h}}|_R \leftarrow \tilde{V}_S\tau \quad \forall\, R \in \mathcal{Q}:\ R \cap \mathcal{C}_S^{n-1} \neq \emptyset.$$

In the event that I^{n-1} touches the fixed boundary, the boundary condition itself furnishes further information about the interface velocity. It is worth noting that, for a Dirichlet condition, the more tangential I^{n-1} is to $\partial\Omega$, the worse is the interface velocity prediction; the opposite situation occurs for a Neumann condition. If the scalar product between the outward normal to $\partial\Omega$ and the interface velocity is negative, then the cone $\mathcal{C}_S^{n-1}$ above is replaced by a convenient rectangle (Fig. 6.2(b)). This trick suffices for computational purposes. It is easily seen that $\text{CC}_1 = O(\mathbf{N}^{1/2})$ and $\text{CC}_n = O(\mathbf{S}_I^{n-1})$, because the number of squares of $\mathcal{Q}$ intersecting each cone is $O(1)$.

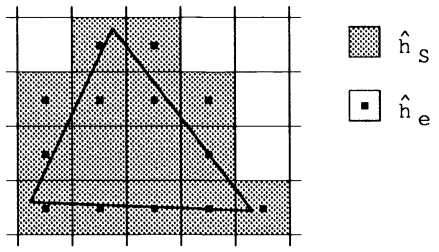


FIG. 6.1. DERIVATIVE_WORK.

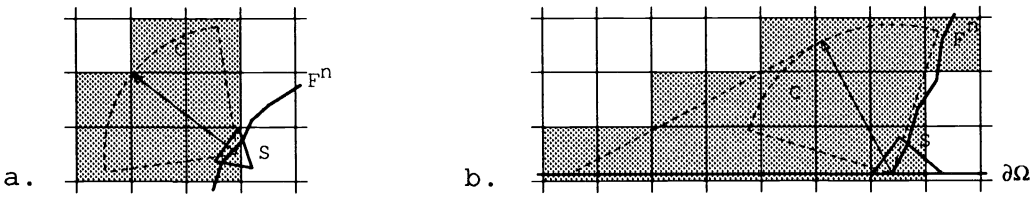


FIG. 6.2. FB_WORK.

6.2.6. Subroutine AVERAGE. This subroutine performs an averaging postprocess of $\check{\mathbf{h}}$ to determine the matrix $\mathbf{h}$ to be used by the automatic generator described in §5. This postprocess enforces the following compatibility constraint:

$$\forall\, R \in \mathcal{Q}, \quad \mathbf{h}|_R \le \check{\mathbf{h}}|_P \quad \forall\, P \in \mathcal{Q}:\ \operatorname{dist}(P,R) \le \mathbf{h}|_R,$$

which is essential for the mesh generator to produce an adequate triangulation $\mathcal{S}^n$, namely,

$$\forall\, S \in \mathcal{S}^n, \qquad h_S \le \check{\mathbf{h}}(x) \qquad \forall\, x \in S.$$

This, in view of the definition of $\check{\mathbf{h}}$, enforces the constraint (3.10). Note that, otherwise, there might be some $S \in \mathcal{S}^n$ intersecting squares $R \in \mathcal{Q}$ for which the corresponding meshsize verifies $\check{\mathbf{h}}|_R < h_S$! The matrix $\mathbf{h}$ is obtained from the following minimization process:

$$\mathbf{h}|_R := \min_{P \in \mathcal{Q}} \left[\max\left(\check{\mathbf{h}}|_P, \operatorname{dist}(P,R) - r\right)\right] \quad \forall\, R \in \mathcal{Q},$$

where dist stands for the l^∞-distance in $\mathbf{R}^2$. This choice makes the implementation easier.

Since $\mathbf{h}|_R \le \check{\mathbf{h}}|_R$, if $\operatorname{dist}(P,R) - r \ge \check{\mathbf{h}}|_R$ for some $P \in \mathcal{Q}$, then P has no effect in the definition of $\mathbf{h}|_R$. This leads to considering only the set of all squares $P \in \mathcal{Q}$ so that $\operatorname{dist}(P,R) - r < \check{\mathbf{h}}|_R$, whose cardinal is $O(1)$ because $\check{\mathbf{h}}|_R \le C\tau^{1/2}$. As a result, the computational complexity is CC $= O(\mathbf{N})$.

To determine $\mathbf{h}|_R$ efficiently, we resort to the following iterative process for all $R \in \mathcal{Q}$:

$$h_K := \min_{k \le K} \left\{ \min_{P \in \mathcal{Q}:\operatorname{dist}(P,R) = kr} \left[\max\left(\check{\mathbf{h}}|_P, (k-1)r\right)\right] \right\}, \qquad K \ge 0.$$

It is easily seen that $\{h_K\}_K$ is nonincreasing, $\lim_{K\to\infty} h_K = \mathbf{h}|_R$, and $h_K = \mathbf{h}|_R$ for all K such that $Kr \ge h_K$. We thus end up with the stopping criterion: $h_K \le Kr$.

6.3. Subroutine MESH (n). The sequence of instructions reads as follows:

```
if (n > 1) FIXED_TRIANGLES_INSERT (inserts triangles of S^{n-1}_{I∪B} in S^n)
ADVFRONT                          (performs the advancing front algorithm)
```

FLIPDIAG	(produces a weakly acute triangulation)
RED_ZONE_DEFINE	(determines the refined region $\mathcal{R}^n$)

6.3.1. Subroutine FIXED_TRIANGLES_INSERT. In the generation of $\mathcal{S}^n$, all triangles belonging to $\mathcal{I}^{n-1} \cup \mathcal{B}^{n-1}$, as well as their adjacent elements, must be inserted straight to $\mathcal{S}^n$, as required in (3.9). Adding adjacent triangles helps the postprocess FLIPDIAG produce a weakly acute triangulation, as observed in practice.

This subroutine first inserts the nodes of the above triangles which belong to $\partial\Omega$ for the automatic triangulation of $\partial\Omega$ to be performed. Next, it incorporates the triangles themselves whose boundaries will then give rise to the initial advancing front for the mesh generator ADVFRONT to be executed. It is easily seen that CC $= O(\mathbf{S}_I^{n-1}) + O(\mathbf{S}_B^{n-1})$.

If the nondegenerate interface is smooth, then the constraints (3.9) can be removed, namely, all the elements of $\mathcal{S}^{n-1}$ become free, which is always preferable for computational efficiency. The routine FIXED_TRIANGLES_INSERT is simply bypassed after checking the validity of (see [14, §S4.1])

$$
\begin{aligned}
&D_e \le C_1 \tau^{-1} \quad \forall\, e \in \mathcal{E}_I^{n-1} \cup \mathcal{E}_B^{n-1}, \qquad d_S \le C_2 \quad \forall\, S \in \mathcal{S}_I^{n-1} \cup \mathcal{S}_B^{n-1}, \\
&\qquad\qquad \text{length}(I^{n-1}) \le C_3, \qquad \text{meas}(\mathcal{B}^{n-1}) \le C_4 \tau.
\end{aligned} \tag{6.1}
$$

Here C_1, C_2, C_3, C_4 are given positive constants. In §7 we present relevant examples that fulfil such stronger properties and compare both strategies. The accuracy of AM does not seem to be affected by the interpolation errors introduced on triangles of $\mathcal{I}^{n-1} \cup \mathcal{B}^{n-1}$.

6.3.2. Subroutines ADVFRONT **and** FLIPDIAG. Making use of $\mathbf{h}$, the program ADVFRONT generates automatically a triangulation of Ω which, in principle, may not be weakly acute. The postprocess FLIPDIAG is next called to remedy this situation. The elements so created, however, may have a size that roughly oscillates between 0.5 and 2 times the expected value. As already mentioned in §5, CC $= O(\mathbf{J}^n \log \mathbf{J}^n)$.

6.3.3. Subroutine RED_ZONE_DEFINE. This subroutine classifies the elements of $\mathcal{S}^n$ into three classes according to whether they belong to the interior of the effective refined region $\mathcal{R}^n$, to its boundary or to the rest of the domain. These regions are labeled, respectively, SECURITY_ZONE, RED_ZONE, and PARABOLIC_ZONE (see Figs. 7.3, 7.7, 7.13, and 7.16, where the RED_ZONE is blackened). To this end, the squares of $\mathcal{Q}$ are previously "coloured" to distinguish between those belonging to the predicted refined region $\check{\mathcal{R}}^n$, those for which $\check{\mathbf{h}} = O(\tau)$ as a result of AVERAGE, and the rest. If all the elements of $\mathcal{S}^n$ are free, then we can always allow the computed interface to reach triangles of the PARABOLIC_ZONE adjacent to the RED_ZONE. If the elements of $\mathcal{S}_I^{n-1}$ were fixed, instead, the much bigger size of those adjacent elements that belong to $\mathcal{S}_I^{n-1}$ would create serious computational difficulties in specifying the new SECURITY_ZONE. Such a trick improves the robustness of our code in that the interface can stay longer without touching interior elements of the PARABOLIC_ZONE (see §7.1 and Fig. 7.3(a)). It is easily seen that CC $= O(\mathbf{S}^n)$.

6.4. Subroutine MATRIX. Boundary conditions of any variational type can be specified during the description of domain geometry. The elementary and global mass and stiffness matrices $\mathbf{M}^n$ and $\mathbf{K}^n$ are then constructed and stored, with CC $= O(\mathbf{J}^n)$.

6.5. Subroutine INTERPOLATION (n). For $n = 1$, this subroutine determines U^0 according to (2.5). For $n > 1$, instead, it interpolates $U^{n-1} \in \mathbf{V}^{n-1}$ over the new mesh $\mathcal{S}^n$, namely, it calculates $\hat{U}^{n-1} = \Pi^n U^{n-1}$. The interpolation algorithm has to

find the nodes of $\mathcal{S}^n$ belonging to each element of $\mathcal{S}^{n-1}$. It turns out that the quadtree structure of §4 is an appropriate tool for *fast interpolation.* In fact, we organize the set $\mathcal{J}^n$ into a quadtree and next use the function INSIDE of §4.5 to detect the elements of $\mathcal{J}^n$ that belong to each triangle $S \in \mathcal{S}^{n-1}$, whose cardinal is m_S. This search operation requires a computational labor of order $d^n(\mathbf{S}^{n-1}+\sum_{S\in\mathcal{S}^{n-1}} m_S) = d^n(\mathbf{S}^{n-1}+\mathbf{J}^n) = O((\mathbf{S}^{n-1}+\mathbf{J}^n)\log \mathbf{J}^n)$, where $d^n = O(\log \mathbf{J}^n)$ is the quadtree depth. Note that $\mathbf{S}^{n-1}$ above accounts for the case $m_S = 0$ that still requires a search operation. Since both $\mathbf{S}^{n-1}$ and $\mathbf{J}^n$ are expected to be of the same order, the overall complexity is $O(\mathbf{J}^n \log \mathbf{J}^n)$ as opposed to $O(\mathbf{S}^{n-1}\mathbf{J}^n)$ for a naïve implementation. This order compares quite favorably with the optimal amount $O(\mathbf{J}^n)$.

6.6. Subroutine STEFAN. It solves the nonlinear system (2.10) with the following nonlinear (successive overrelaxation) method [15]–[17], [21]. The system (2.10) can be written componentwise as

$$U_i^n + \sum_{j=1}^{\mathbf{J}^n} a_{ij}^n \beta(U_j^n) = b_i^n := U_i^{n-1} + \tau f(\beta(U_i^{n-1})) \quad \forall\, 1 \le i \le \mathbf{J}^n,$$

or equivalently, using the maximal monotone graph $\gamma := \beta^{-1}$ inverse of β, as follows:

$$\gamma(\Theta_i^n) + \sum_{j=1}^{\mathbf{J}^n} a_{ij}^n \Theta_j^n \ni b_i^n \quad \forall\, 1 \le i \le \mathbf{J}^n,$$

where $\mathbf{A}^n := \tau(\mathbf{M}^n)^{-1}\mathbf{K}^n = \{a_{ij}^n\}_{i,j=1}^{\mathbf{J}^n}$. The nonlinear (successive overrelaxation) method for computing the discrete temperature then, for all $m \ge 1$ and $1 \le i \le \mathbf{J}^n$, reads

$$\Theta_{i,m+1}^n := \Theta_{i,m}^n - \omega^n \left[\Theta_{i,m}^n - \zeta_i^n \left(b_i^n - \sum_{j=1}^{i-1} a_{ij}^n \Theta_{j,m+1}^n - \sum_{j=i+1}^{\mathbf{J}^n} a_{ij}^n \Theta_{j,m}^n \right) \right],$$

where $\Theta_1^n := \hat{\Theta}^{n-1}$, $\zeta_i^n := (\gamma + a_{ii}^n I)^{-1}$ is a nondecreasing function and $1 < \omega^n < 2$ is a suitably chosen relaxation parameter. The iterative procedure stops when

$$|\Theta_{i,m+1}^n - \Theta_{i,m}^n| \le \varepsilon \max(1, |\Theta_{i,m}^n|) \quad \forall\, 1 \le i \le \mathbf{J}^n,$$

where ε is a given tolerance. Next, the discrete enthalpy U^n is computed using one extra Gauss–Seidel iteration, namely,

$$U_i^n := \eta_i^n \left(b_i^n - \sum_{j=1}^{i-1} a_{ij}^n \Theta_j^n - \sum_{j=i+1}^{\mathbf{J}^n} a_{ij}^n \bar{\Theta}_j^n \right), \qquad \Theta_i^n := \beta(U_i^n) \quad \forall\, 1 \le i \le \mathbf{J}^n,$$

where $\eta_i^n := (I + a_{ii}^n \beta)^{-1}$ and $\bar{\Theta}^n$ is the discrete temperature previously computed. Note that the relation $\Theta^n = \Pi^n \beta(U^n)$ is not automatically satisfied by $\bar{\Theta}^n$, because $\bar{\Theta}^n$ is not the exact solution of the nonlinear system (2.10).

Since the mesh $\mathcal{S}^n$ is weakly acute, the previous nonlinear (successive overrelaxation) iterations converge as fast as the linear (successive overrelaxation) method for the M-matrix $\mathbf{B}^n := L_\beta^{-1}\mathbf{I} + \mathbf{A}^n$, where L_β is the Lipschitz constant of β [21]. Hence, the optimal value of the relaxation parameter ω_{opt}^n for the associated linear system can be efficiently approximated from the well-known relation

$$\omega_{\text{opt}}^n = \frac{2}{1 + \sqrt{1 - (\rho_{\text{Jac}}^n)^2}}.$$

Indeed, the spectral radius ρ^n_{Jac} of the Jacobi iteration matrix corresponding to $\mathbf{B}^n$ can be estimated accurately via very few iterations of the Rayleigh ratio method. The relation $\rho^n_{\text{opt}} = \omega^n_{\text{opt}} - 1$ gives the optimal rate of convergence of the (successive overrelaxation) iterations. The condition number of $\mathbf{B}^n$ plays an essential role in determining ρ^n_{opt}. It thus dictates the number of iterations $\mathbf{I}^n$ needed in reducing the initial error by a factor δ. In order to evaluate $\mathbf{I}^n$ we can argue heuristically as follows. Assuming that $\rho^n_{\text{opt}} = \exp(-(\mathbf{J}^n)^{-\alpha})$ with $\alpha > 0$ and $\delta = O(\tau)$, we have

$$\mathbf{I}^n \approx \frac{\log \delta}{\log \rho^n_{\text{opt}}} = C(\mathbf{J}^n)^{\alpha} |\log \tau|;$$

α will be estimated empirically in §7. The tolerance ε in the stopping test above should then be $O(\tau^{2+3\alpha/2})$, but numerical evidence suggests that a linear dependence is satisfactory in practice (see §7). As a result, the computational complexity of the nonlinear solver for any time step is at most $\text{CC} = O((\mathbf{J}^n)^{1+\alpha} \log \mathbf{N})$.

6.7. Subroutines TEST and SOL_UPDATE. Once the discrete solution has been computed, the algorithm checks whether or not the current mesh has to be discarded. A remeshing would be necessary as soon as either I^n escapes from the SECURITY_ZONE or the interpolation errors are no longer equidistributed correctly: TEST $\neq$ **OK** is then returned. More precisely, in the event either the PARABOLIC_ZONE has been reached (i.e., $\mathcal{I}^n \not\subset \mathcal{R}^n$) or conditions (3.7) have been violated, TEST = **FAIL** is returned; thus SOL_UPDATE discards the computed solution and recovers the previous one. To prevent the program from performing a useless time step, TEST = **ALERT** alerts that an imminent remeshing must be done whenever either the discrete interface reaches the RED_ZONE or conditions (3.7) with $\mu_i^\sharp$ in place of μ_i^* are not valid. Here, $1 < \mu_i^\sharp < \mu_i^*$ are selected so as to make violation of (3.7) an unlikely event. This is actually possible because first and second derivatives may exhibit large oscillations solely near the nondegenerate discrete interface I^n, and thus within $\mathcal{R}^n$ where the current (local) meshsize is already $O(\tau)$. Hence, remeshing is mostly dictated by the free boundary location. The violation of conditions (3.8) also set TEST = **ALERT**. We stress that TEST = **ALERT** only warns that $\mathcal{S}$ cannot be kept longer, but does not entail an immediate remeshing if only a few steps have been performed since the last mesh change. This pathologic case corresponds to a wildly-behaved interface, for instance. The computational complexity of this subroutine is $\text{CC} = O(\mathbf{S}^n)$.

Remark 6.2. Appearance of phases. When a new phase appears, TEST = **FAIL** is returned in case the nondegenerate interface intersects the PARABOLIC_ZONE; so the current solution must be rejected and the previous one recovered. Since a new phase is not yet present in the solution just recovered, the algorithm keeps track of triangles crossed by the new phase in order to foresee the future free boundary location. In fact, the subroutine FB_WORK constructs the refined region relative to a new phase by taking the union of suitable balls centered at the barycenter of those triangles. It may happen that a large part of the domain reaches the melting temperature at the same time, originating a wide new phase. Since the numerical solution near singularities depends on mesh structure, the discrete solution recomputed after a phase appearance may present a new free boundary in a different part of the domain, thus giving rise to a time step loop. The simple trick mentioned in Remark 6.1 is a remedy to such a situation.

6.8. Overall computational complexity. Suppose that there is no discrete mush and any discrete interface F^n is a polygonal curve with finite length uniformly in τ. This situation may correspond to a well-behaved exact free boundary such

as that in Examples I—III of §§7.1–7.3. Under this assumption, the refined region $\mathcal{R}^n$ is just a strip $O(\tau^{1/2})$-wide around F^n. Since the local meshsize is $O(\tau)$, the number of triangles within $\mathcal{R}^n$ is $O(\tau^{-3/2})$. Except possibly for a small transition region, triangles outside of $\mathcal{R}^n$ are $O(\tau^{1/2})$-big. Consequently, the number of triangles outside of $\mathcal{R}^n$ becomes $O(\tau^{-1})$; so the required computational labor in $\mathcal{R}^n$ dominates. The number of spatial degrees of freedom can be expressed in terms of $\mathbf{N}$ as follows:

$$\mathbf{S}^n, \mathbf{J}^n = O(\mathbf{N}^{3/2}).$$

Hence, the total computational complexity (TCC) turns out to be

$$\text{TCC} = O(\mathbf{N}^{(3\alpha+5)/2} \log \mathbf{N}) \tag{6.2}$$

for a global accuracy $O(\mathbf{N}^{-1/2})$. Since the number of remeshings is $O(\mathbf{N}^{1/2})$, the total overhead of our code (work not related to solving (2.10)) is $O(\mathbf{N}^2 \log \mathbf{N})$, which is much smaller than TCC. In particular, the cost of mesh modification is asymptotically negligible compared to that of solving the nonlinear system (2.10).

The quantity TCC compares quite favorably with similar ones for practical methods involving a single quasi-uniform mesh. In fact, in that case $h_S = O(\tau)$ for all $S \in \mathcal{S}^n$ and so $\text{TCC} = O(\mathbf{N}^{3+2\alpha} \log \mathbf{N})$ for the same global accuracy [5], [15]. We stress that even with mushy regions, TCC for AM is expected to be less than that for the standard approach for a given accuracy. A relevant example is fully discussed in §7.4.

A final comment regarding fixed mesh methods with preliminary regularization is in order [15]. Assuming that a global nondegeneracy property is valid, a result similar to (6.2) can be obtained [15], but at the expense of a much worse local approximation quality as numerical evidence indicates [15], [17]. Such a local property is extremely important in determining interfaces and explains the much better resolution associated with the present method. We also underline that most examples of §7 do not satisfy a global nondegeneracy property; the trick of [15] is thus of no use.

7. Numerical experiments. Several numerical experiments were performed with both our Adaptive FEM (AM) and the standard one with a fixed mesh (FMM) [4], [5], [15], [17], [22] to illustrate the superior performance of AM. They were run on a VAX 8530, VMS 5.1. The CPU times are affected by a 5–10 percent oscillation depending on the computer load.

In presenting the numerical results, we have employed the following notation:

$\mathbf{N}$:= number of uniform time steps,
$h_{\min}$:= $(\sum_{n=1}^{\mathbf{N}} \min_{S \in \mathcal{S}^n} h_S)/\mathbf{N}$ (average of minimum triangle diameters), scaled $\times 10^2$,
$h_{\max}$:= $(\sum_{n=1}^{\mathbf{N}} \max_{S \in \mathcal{S}^n} h_S)/\mathbf{N}$ (average of maximum triangle diameters), scaled $\times 10^2$,
h := uniform meshsize for FMM, scaled $\times 10^2$,
$\mathbf{J}$:= $(\sum_{n=1}^{\mathbf{N}} \mathbf{J}^n)/\mathbf{N}$ (average number of nodes),
$\mathbf{S}$:= $(\sum_{n=1}^{\mathbf{N}} \mathbf{S}^n)/\mathbf{N}$ (average number of triangles),
$\mathbf{S}_R$:= $(\sum_{n=1}^{\mathbf{N}} \mathbf{S}_R^n)/\mathbf{N}$ (average number of triangles within the refined region),
$\mathbf{S}_P$:= $(\sum_{n=1}^{\mathbf{N}} \mathbf{S}_P^n)/\mathbf{N}$ (average number of triangles within the parabolic region),
$\mathbf{E}_\theta^2$:= $\sum_{n=1}^{\mathbf{N}} \tau \int_\Omega \Pi^n[(\theta(x, n\tau) - \Theta^n(x))^2]dx$ ($\approx \|\theta - \Theta\|_{L^2(Q)}$), scaled $\times 10^2$,
$\mathbf{E}_\theta^\infty$:= $\max_{\{1 \le n \le \mathbf{N}, 1 \le j \le \mathbf{J}^n\}} |\theta(x_j^n, n\tau) - \Theta_j^n|$ ($\approx \|\theta - \Theta\|_{L^\infty(Q)}$), scaled $\times 10^2$,
$\mathbf{E}_u^2$:= $\sum_{n=1}^{\mathbf{N}} \tau \int_\Omega \Pi^n[(u(x, n\tau) - U^n(x))^2]dx$ ($\approx \|u - U\|_{L^2(Q)}$), scaled $\times 10^2$,
$\mathbf{E}_I^\infty$:= $\max_{1 \le n \le \mathbf{N}} \text{dist}(I(n\tau), I^n)$, scaled $\times 10^2$,
$\mathbf{I}$:= $(\sum_{n=1}^{\mathbf{N}} \mathbf{I}^n)/\mathbf{N}$ (average number of successive overrelaxation iterations),
ρ_{opt} := $(\sum_{n=1}^{\mathbf{N}} \rho_{\text{opt}}^n)/\mathbf{N}$ (average optimal spectral radius of matrices $\mathbf{B}^n$),
$\mathbf{C}$:= number of mesh changes (and number of SOL_UPDATE calls, if any),

$\mathbf{t}$:= total CPU time in seconds,
$\mathbf{t}_{\mathrm{H}}$:= CPU time of H_DEFINE in seconds,
$\mathbf{t}_{\mathrm{MESH}}$:= CPU time of MESH in seconds,
$\mathbf{t}_{\mathrm{MAT}}$:= CPU time of MATRIX in seconds,
$\mathbf{t}_{\mathrm{TEST}}$:= CPU time of RED_ZONE_DEFINE and TEST in seconds,
$\mathbf{t}_{\mathrm{INT}}$:= CPU time of INTERPOLATION in seconds,
$\mathbf{t}_{\mathrm{SOR}}$:= CPU time of STEFAN in seconds.

The tables below provide enough information to evaluate the actual computational complexity of the various subroutines, as well as that of the entire code. In particular, assuming the relation $\mathbf{I} = C\mathbf{J}^{\alpha}$, with C independent of $\mathbf{N}$, and denoting by $\mathbf{I}_1$ and $\mathbf{I}_2$ the number of successive overrelaxation iterations corresponding to $\mathbf{N}_1$ and $\mathbf{N}_2$, we infer that $\alpha = \log(\mathbf{I}_1/\mathbf{I}_2)/\log(\mathbf{J}_1/\mathbf{J}_2)$. Similar reasoning applies to all remaining quantities of interest.

The value of ε in the stopping test of §6.6 varies linearly with τ. Even though this is not what theory predicts, the linear dependence appears to be quite satisfactory from a practical viewpoint. In addition, we have observed experimentally that halving ε increases the iterations $\mathbf{I}$ by at most 10 percent. Hence, the CPU time of the various tests is not very sensitive to perturbations on ε about the proposed value.

The values for $h_{\min}$ and $h_{\max}$ reported in the tables below may slightly violate the constraint $\lambda\tau \le h_S \le \Lambda\tau^{1/2}$ for all $S \in \mathcal{S}^n$. This is so because the mesh generator can only guarantee a local meshsize roughly between 0.5 and 2 times the expected one.

Note that the L^2-errors $\mathbf{E}_\theta^2$ and $\mathbf{E}_u^2$ are calculated via the quadrature rule (2.8). Since the corresponding integrands are convex, the local quadrature error is positive and $O(h_S^2)$. Hence, it is $O(\tau)$ for AM, thus affecting the final rate, whereas it is infinitesimal for FMM, having no influence on the final rate. Therefore, computational L^2-errors are not as suitable for comparison as pointwise ones are. Since our strategy is based on equidistributing L^∞-interpolation errors, we thus emphasize pointwise accuracy in our tests.

We use the simplest constitutive relation $\beta(s) := (s-1)^+ - s^-$ in all tests below. The first two experiments correspond to the evolution of a smooth interface. The third example illustrates the formation of a cusp, whereas the last one shows the evolution of a mushy region that eventually becomes a sharp interface.

7.1. Example I: Oscillating interface. This is a classical two-phase Stefan problem with an interface that moves up and down. Here $\Omega := (0,5) \times (0,5)$ and $T := \pi/1.25$. The exact temperature is given by the following expression:

$$\theta(x,y,t) := \begin{cases} 0.75(r^2-1), & r < 1, \\ (1.5 - \rho'(t)\sin\varphi)(r-1), & r \ge 1, \end{cases}$$

where $r := (x^2 + (y-\rho(t))^2)^{1/2}$, $\rho(t) := 0.5 + \sin(1.25t)$, and $\sin\varphi := (y-\rho(t))/r$. Moreover, $\|\theta\|_{L^2(Q)} \approx 33.16$, $\|\theta\|_{L^\infty(Q)} \approx 13.38$, and $\|u\|_{L^2(Q)} \approx 39.81$, with an error of one unit on the fourth digit. Dirichlet boundary conditions are imposed on the sides $y = 0$, $y = 5$, and $x = 5$, whereas a homogeneous Neumann condition is prescribed at $x = 0$. Since the exact interface $I(t)$ is a circle with center $(0, \rho(t))$ and radius 1, the velocity V normal to $I(t)$ exhibits a significant variation along the front which makes this example an extremely difficult test for our numerical method. Moreover, since V vanishes at both $t = T/2$ and $(1, \rho(t))$, and is thus very small nearby, this test constitutes a fair measure of robustness under degenerate situations.

The various constants introduced in §§3,6 are as follows:

$\lambda = 1.5$, $\Lambda = 5$, $M = 5$, $r = 0.75\sqrt{\tau}$,
$n_1 = 3.5$, $n_2 = 1$, $n_3 = 1.75 \rightarrow \mu_1 = 2.22$, $\mu_2 = 5.59$, average value of $\mu_3 \approx 1.2$,
$\mu_1^\star = \mu_2^\star = 3.5$, $\mu_1^\sharp = \mu_2^\sharp = 2.75$, $\mu_3^- = 0.5$, $\mu_3^+ = 2$,
$C_1 = \|D^2\theta_0\|_{L^\infty(\Omega)}$, $C_2 = 3\|\nabla\theta_0\|_{L^\infty(\Omega)}$, $C_3 = 2$ length(I_0), $C_4 = C_3\lambda$,
$\varepsilon = 128 \cdot 10^{-6}$ for $\mathbf{N} = 40$ $\quad (\varepsilon = 64 \cdot 10^{-6}$ for $\mathbf{N} = 50$ (FMM)).

The upper bound M for the interface velocity was never required by the various tests. The values of μ_1 and μ_2 exhibited variations of one unit in the third digit; they are thus independent of $\mathbf{N}$ as expected. Mesh regeneration was mainly dictated by the free boundary location because the first remeshing, after the interface recedes from the upmost position, was the only one due to failure of (3.7). We have found $\mathcal{B} = \emptyset$ for all tests. Since the interface is regular, the condition (6.1) is met for appropriate constants. It is then possible to utilize both strategies with and without fixed triangles. Their performance is fairly similar. Nevertheless, the strategy without fixed triangles never needs the computed solution to be discarded, a desirable fact, whereas the strategy with fixed triangles does.

Results for AM are reported in Tables 7.1 and 7.3 for the strategy without fixed triangles and in Tables 7.2 and 7.4 for that with fixed triangles. Table 7.5 contains results for FMM. The constant of proportionality between τ and the (uniform) meshsize was chosen so as to minimize $\mathbf{E}_\theta^2$ for a desired number of DOF. We are thus competing against the best possible standard results.

TABLE 7.1
Example I: *Adaptive method* ($S_F \cup S_B$ *free*).

N	$h_{\min}$	$h_{\max}$	J	$\mathbf{S}_R$	$\mathbf{S}_P$	$\mathbf{E}_\theta^2$	$\mathbf{E}_u^2$	$\mathbf{E}_\theta^\infty$	$\mathbf{E}_I^\infty$	t
40	9.52	151.	339	536	144	18.8	37.0	10.7	5.83	50
60	6.30	136.	592	957	231	11.3	28.8	7.42	5.01	126
80	3.63	115.	818	1306	334	7.59	26.1	5.17	3.26	235
120	2.91	100.	1406	2306	510	5.73	23.3	3.92	2.66	583
160	1.45	86.4	2110	3563	662	4.60	20.8	3.44	2.32	1068
240	0.96	72.3	3631	6245	1023	2.99	18.5	2.58	1.72	2916

TABLE 7.2
Example I: *Adaptive method* ($S_F \cup S_B$ *fixed*).

N	$h_{\min}$	$h_{\max}$	J	$\mathbf{S}_R$	$\mathbf{S}_P$	$\mathbf{E}_\theta^2$	$\mathbf{E}_u^2$	$\mathbf{E}_\theta^\infty$	$\mathbf{E}_I^\infty$	t
40	9.69	162.	319	496	148	16.1	35.4	11.3	5.37	58
60	4.45	138.	593	957	233	10.5	29.1	6.65	4.44	132
80	4.00	115.	875	1430	328	7.54	24.2	5.23	3.45	224
120	2.39	99.1	1413	2320	510	5.34	22.7	3.93	2.60	546
160	1.74	88.2	2089	3504	679	4.07	19.6	3.14	2.08	1111
240	1.03	70.4	3620	6201	1042	2.82	17.7	2.41	1.65	2998

TABLE 7.3
Example I: *Adaptive method* ($S_F \cup S_B$ *free*).

N	I	ρ_{opt}	C	t_H	t_{MESH}	t_{MAT}	t_{TEST}	t_{INT}	t_{SOR}
40	13.8	0.50	3	7.1	16.3	1.8	6.9	1.6	15.6
60	15.8	0.56	4	14.7	38.6	3.8	17.1	4.6	46.3
80	17.7	0.60	5	25.1	68.4	6.6	24.7	8.9	99.3
120	21.3	0.66	6	51.7	149.7	13.7	58.2	21.0	284.9
160	23.3	0.69	6	70.8	221.5	20.6	105.3	34.2	609.3
240	27.9	0.74	8	150.8	534.4	48.9	221.3	89.2	1855.9

TABLE 7.4
Example I: *Adaptive method* ($S_F \cup S_B$ *fixed*).

N	I	ρ_{opt}	C	t_H	t_{MESH}	t_{MAT}	t_{TEST}	t_{INT}	t_{SOR}
40	13.1	0.49	4(1)	9.2	22.3	2.2	6.9	2.2	14.8
60	16.0	0.56	4(1)	14.8	43.1	3.8	17.5	4.3	47.1
80	17.2	0.60	4	19.8	65.7	5.6	26.1	6.7	97.9
120	20.1	0.66	5	43.9	141.4	11.7	58.8	16.0	270.1
160	23.8	0.69	6(1)	69.7	259.0	19.7	103.8	31.1	621.2
240	28.4	0.75	8(1)	149.5	568.4	47.9	216.9	87.5	1883.3

TABLE 7.5
Example I: *Fixed mesh method.*

N	h	J	S	E_θ^2	E_u^2	E_θ^∞	E_I^∞	t
50	27.8	448	942	15.9	39.7	20.8	13.8	37
75	18.5	1017	2104	10.5	30.4	15.3	9.40	113
100	13.9	1812	3718	7.81	26.8	12.4	6.88	292
150	9.16	4107	8356	5.57	22.3	7.78	5.65	919
200	6.89	7361	14912	4.52	18.6	6.31	3.53	2264

TABLE 7.6
Example I: *Comparison of pointwise accuracy.*

Adaptive method ($S_F \cup S_B$ free)				Fixed mesh method			
N × J	E_θ^∞	E_I^∞	t	N × J	E_θ^∞	E_I^∞	t
40×339	10.7	5.83	50	100×1812	12.4	6.88	292
60×592	7.42	5.01	126	150×4107	7.78	5.65	919
80×818	5.17	3.26	235	200×7361	6.31	3.53	2264

Here are several observations extracted from the tables. Both the L^2 and L^∞-errors for temperature depend almost linearly on τ, whereas the L^2-error for enthalpy behaves almost like $O(\tau^{1/2})$. As expected, we have $\mathbf{S}_P = O(\mathbf{N})$ and $\mathbf{J}, \mathbf{S}_R = O(\mathbf{N}^{3/2})$. The number of mesh changes **C**, however, does not evolve as $O(\mathbf{N}^{1/2})$. This is so because the velocity of the interface decreases first to zero, and thus is small for a number of time steps, making the remeshing unnecessary for a long period of time. The total CPU time **t** grows slightly faster than $O(\mathbf{N}^{5/2})$, whereas $\alpha \approx 0.25$. These figures are compatible with theoretical prediction. Table 7.5 may be compared with the first five rows of Tables 7.1 and 7.2 as far as $\mathbf{E}_\theta^2$ is concerned. Note how much faster **t** increases for FMM than for AM. The superior local approximation of interfaces produced by AM is, however, mainly dictated by the pointwise approximation quality for temperature. We thus rely on the errors $\mathbf{E}_\theta^\infty$ and $\mathbf{E}_I^\infty$ in comparing the various results: in Table 7.6, each row roughly corresponds to the same local accuracy, but compare **t**. We stress that, to obtain $\mathbf{E}_\theta^\infty \approx 5 \cdot 10^{-2}$ with FMM, the number **J** becomes so large that the central memory of the computer is filled up. This forces the use of mass storage memory, making **t** inappropriate to measure complexity.

Figure 7.1 contains the initial meshes for different values of **N**. The various meshes for $\mathbf{N} = 80$ are depicted in Fig. 7.2. Observe the proper grading produced by the automatic mesh generator of [18]. We see how the refined region moves up and down accompanying the interface motion. Note that even in the upmost position, when the interface is motionless, the proposed strategy is successful. When the interface recedes the number of remeshings increases. This is justified by the fact that the

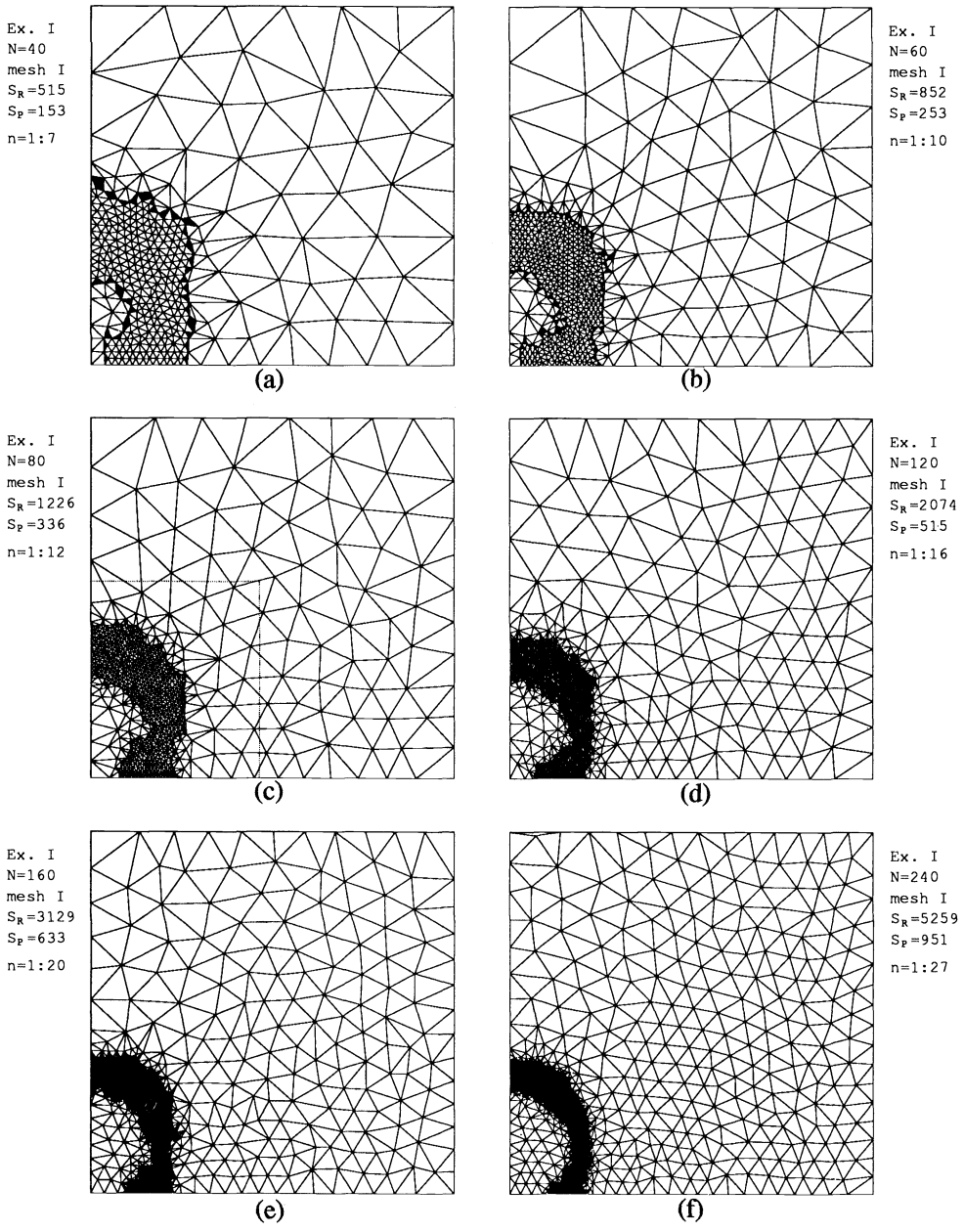


FIG. 7.1. *Example* I (AM, $\mathbf{N} = 40, 60, 80, 120, 160, 240$): *First mesh.*

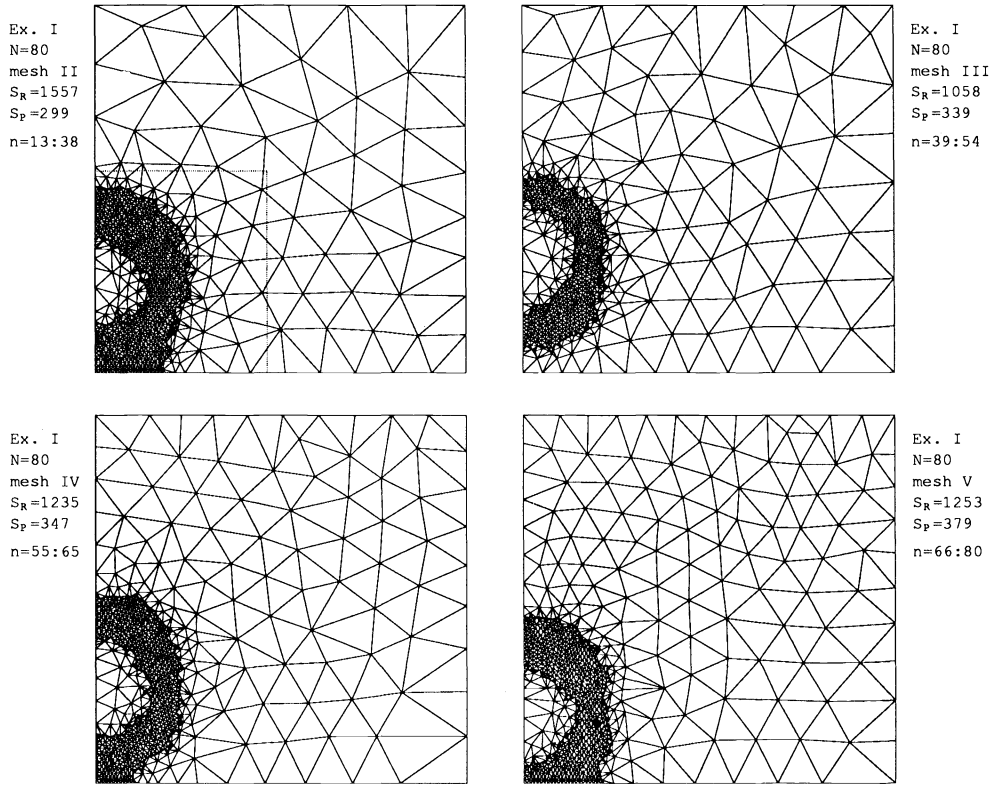


FIG. 7.2. *Example* I (AM, $\mathbf{N} = 80$, $S_F \cup S_B$ *free*): *Subsequent meshes.*

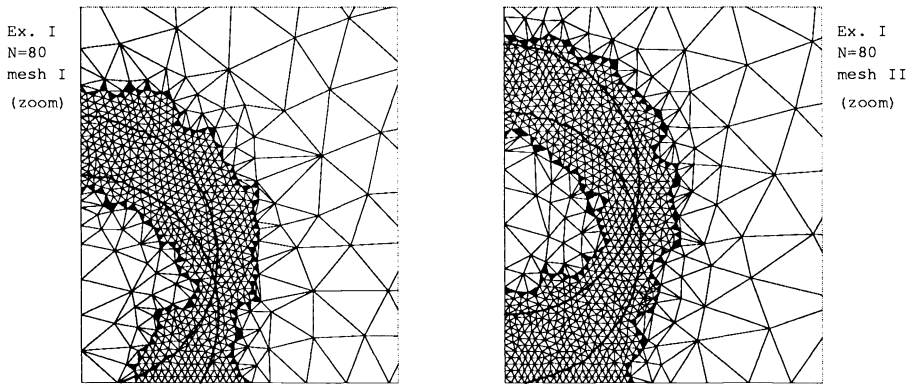


FIG. 7.3. *Example* I (*zoom of Figs* 7.1(c) *and* 7.2(a)): *Refined region,* RED_ZONE, *first and last interface.*

interface velocity is also increasing in time, which enables the free boundary to reach the RED_ZONE a bit faster than predicted. Figure 7.3 is a zoom of the refined region. The RED_ZONE appears blackened in the picture, which also shows the first and last

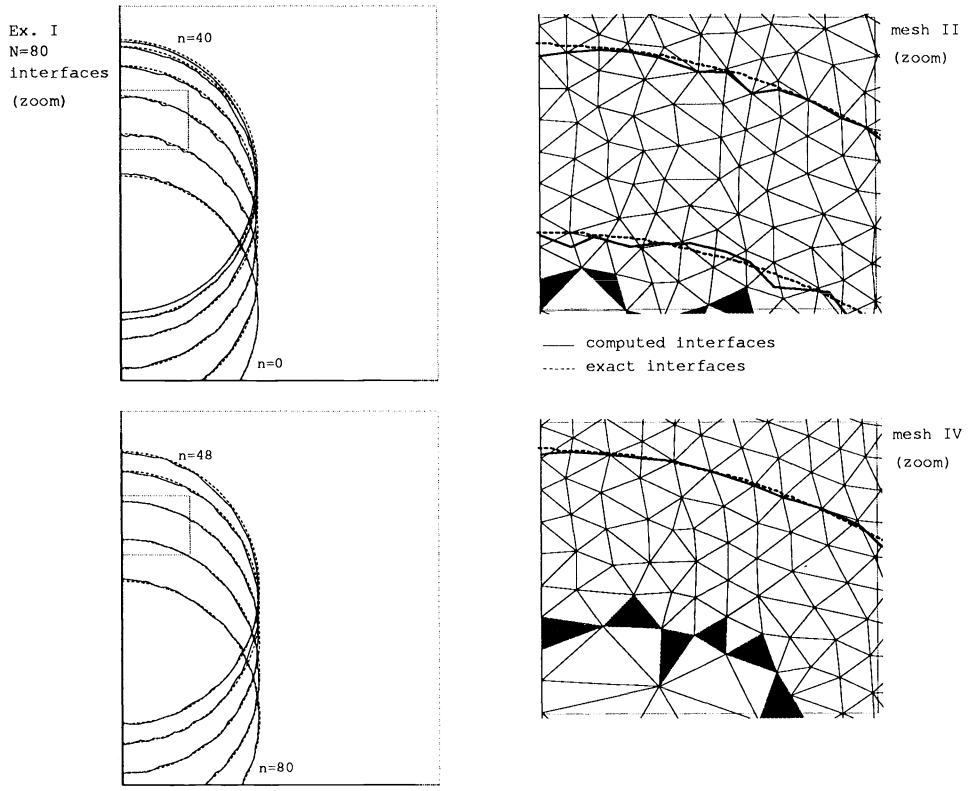


FIG. 7.4. *Example* I (AM, $\mathbf{N} = 80$, $S_F \cup S_B$ *free*): *Interfaces at* $n = 8k$ $(0 \leq k \leq 10)$; *zoom of meshes* II *and* IV.

discrete interfaces computed with the current mesh. Figures 7.4(a) and 7.4(c) are zooms of both the exact and discrete interfaces for a number of time steps obtained with AM and $\mathbf{N} = 80$. The agreement between these curves is quite remarkable, as can be observed in Figures 7.4(b) and 7.4(d). The local meshsize can be seen in those pictures as well as the computed interface touching elements in the PARABOLIC_ZONE adjacent to those blackened (see §6.3.3). The exact and discrete interfaces obtained with FMM for $\mathbf{N} = 100$ are depicted in Fig. 7.5. It is worth comparing this picture with Fig. 7.4: the interface location is drastically improved. An attempt at explanation follows. Since $\Theta^n = \beta(U^n)$ nodewise and β vanishes on [0,1], the free boundary F^n touches nodes very often, which in turn leads to loss of accuracy. Such a situation even worsens whenever the local meshsize h_S is much larger than $V_S\tau$, which is avoided by (3.3) and (3.10), but there is no counterpart for FMM.

7.2. Example II: Expanding interface. The second test is still a classical two-phase Stefan problem with an interface $r = \rho(t)$ expanding at increasing velocity.

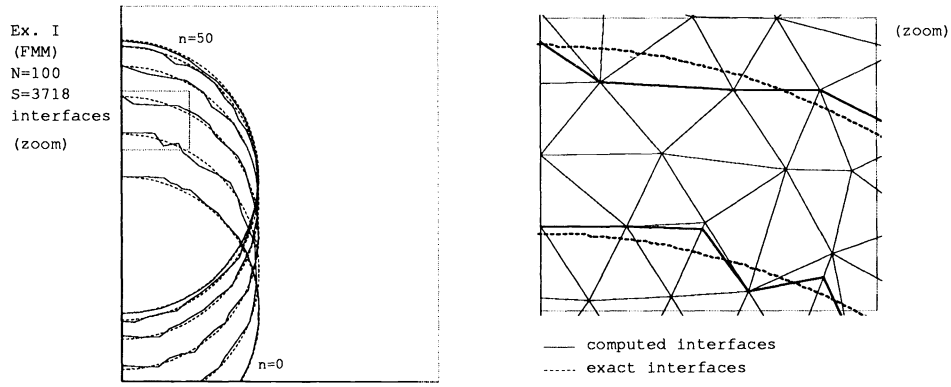


FIG. 7.5. *Example* I (FMM, $\mathbf{N} = 100$): *Interfaces at* $n = 10k$ $(0 \le k \le 5)$; *zoom of the mesh.*

Here $\Omega := (-1,5) \times (0,5)$ and $T := 2.5$. The exact temperature is given by the following expression

$$\theta(x,y,t) := \begin{cases} 0.25(r^2 - \rho(t)^2)(1+\rho(t)^{-1}), & r < \rho(t), \\ 0.5(r - \rho(t)), & r \ge \rho(t), \end{cases}$$

where $r := (x^2+y^2)^{1/2}$ and $\rho(t) := \exp(0.5(t-1))$. Moreover, $\|\theta\|_{L^2(Q)} \approx 12.52$, $\|\theta\|_{L^\infty(Q)} \approx 3.23$, and $\|u\|_{L^2(Q)} \approx 20.27$, with an error of one figure in the fourth digit. Dirichlet boundary conditions are imposed on the sides $x = -1$, $x = 5$, and $y = 5$, whereas a homogeneous Neumann boundary condition is assigned at $y = 0$.

The relevant constants used in this test are the following:

$\lambda = 1.5$, $\Lambda = 5$, $M = 5$, $r = 0.75\sqrt{\tau}$,
$n_1 = 4$, $n_2 = 1$, $n_3 = 2 \to \mu_1 = 0.84$, $\mu_2 = 2.76$, average value of $\mu_3 \approx 1.2$,
$\mu_1^\star = \mu_2^\star = 4$, $\mu_1^\sharp = \mu_2^\sharp = 2.5$, $\mu_3^- = 0.5$, $\mu_3^+ = 2$,
$C_1 = \|D^2\theta_0\|_{L^\infty(\Omega)}$, $C_2 = 4\|\nabla\theta_0\|_{L^\infty(\Omega)}$, $C_3 = 2$ length(I_0), $C_4 = C_3\lambda$,
$\varepsilon = 3 \cdot 10^{-5}$ for $\mathbf{N} = 60$ $\quad (\varepsilon = 2 \cdot 10^{-5}$ for $\mathbf{N} = 50$ (FMM)).

The values of μ_1 and μ_2 for the various $\mathbf{N}$'s differ only by one unit in the third digit; hence, they do not depend on $\mathbf{N}$, as expected. The remeshings were always dictated by the interface location, except for $\mathbf{N} = 180$. In that case, the first mesh change was caused by failure of test (3.7). Both strategies, with and without fixed triangles, were used because the smoothness of the free boundary makes (6.1) hold. We stress that $\mathcal{B} = \emptyset$ again.

Results are reported in Tables 7.7 and 7.9 for the strategy without fixed triangles and in Tables 7.8 and 7.10 for that with fixed triangles. Table 7.11 contains the results with FMM. Table 7.12 compares AM and FMM performances with regard to pointwise approximation. Compare $\mathbf{t}$ again for both methods.

TABLE 7.7
Example II: *Adaptive method* ($S_F \cup S_B$ *free*).

N	$h_{\min}$	$h_{\max}$	J	S_R	S_P	E_θ^2	E_u^2	E_θ^∞	E_I^∞	t
60	5.41	125.	638	936	345	7.28	26.9	6.44	4.25	156
90	3.51	114.	1086	1635	546	5.14	22.2	4.70	3.07	363
120	2.63	95.0	1630	2535	733	4.27	19.6	3.75	2.38	713
180	1.56	75.4	2876	4618	1148	3.65	18.1	2.77	1.79	1852

TABLE 7.8
Example II: *Adaptive method* ($S_F \cup S_B$ *fixed*).

N	$h_{\min}$	$h_{\max}$	J	S_R	S_P	E_θ^2	E_u^2	E_θ^∞	E_I^∞	t
60	4.07	131.	634	930	343	6.60	26.7	6.20	4.06	139
90	3.34	108.	1076	1617	543	5.17	22.8	4.63	3.02	379
120	2.28	92.5	1642	2567	726	4.44	20.3	3.73	2.42	742
180	1.40	75.1	2888	4642	1146	3.58	18.0	2.71	1.72	1919

TABLE 7.9
Example II: *Adaptive method* ($S_F \cup S_B$ *free*).

N	I	ρ_{opt}	C	t_H	t_{MESH}	t_{MAT}	t_{TEST}	t_{INT}	t_{SOR}
60	16.0	0.56	5	21.1	59.2	6.1	12.7	8.6	47.0
90	18.8	0.62	6	40.3	121.2	12.1	26.5	19.2	140.9
120	21.2	0.66	7	63.5	219.1	21.4	54.3	37.8	311.5
180	25.4	0.71	9	125.2	476.3	45.6	126.1	92.9	974.8

TABLE 7.10
Example II: *Adaptive method* ($S_F \cup S_B$ *fixed*).

N	I	ρ_{opt}	C	t_H	t_{MESH}	t_{MAT}	t_{TEST}	t_{INT}	t_{SOR}
60	15.8	0.56	4	16.8	51.1	4.7	13.1	5.6	46.8
90	18.4	0.62	6	43.5	138.7	12.3	26.6	18.4	135.9
120	20.9	0.66	7	63.9	253.4	21.4	52.5	35.9	309.6
180	25.0	0.71	9	126.2	554.6	46.0	122.8	86.6	970.4

TABLE 7.11
Example II: *Fixed mesh method.*

N	h	J	S	E_θ^2	E_u^2	E_θ^∞	E_I^∞	t
50	21.2	908	1878	6.59	39.1	11.6	7.59	74
75	14.1	2087	4266	4.02	29.0	8.01	5.86	254
100	10.6	3693	7508	3.10	25.8	6.48	4.23	621
150	7.01	8418	17020	2.23	21.4	4.74	3.13	2228

TABLE 7.12
Example II: *Comparison of pointwise accuracy.*

Adaptive method ($S_F \cup S_B$ fixed)				Fixed mesh method			
N × J	E_θ^∞	E_I^∞	t	N × J	E_θ^∞	E_I^∞	t
60×634	6.20	4.06	139	100×3693	6.48	4.23	621
90×1076	4.63	3.02	379	150×8418	4.74	3.13	2228

Figure 7.6 contains the initial meshes for different values of **N**. The various meshes for **N** = 60 are depicted in Fig. 7.7. Figure 7.8(a) is a zoom of both the exact and discrete interfaces for a number of time steps corresponding to **N** = 90

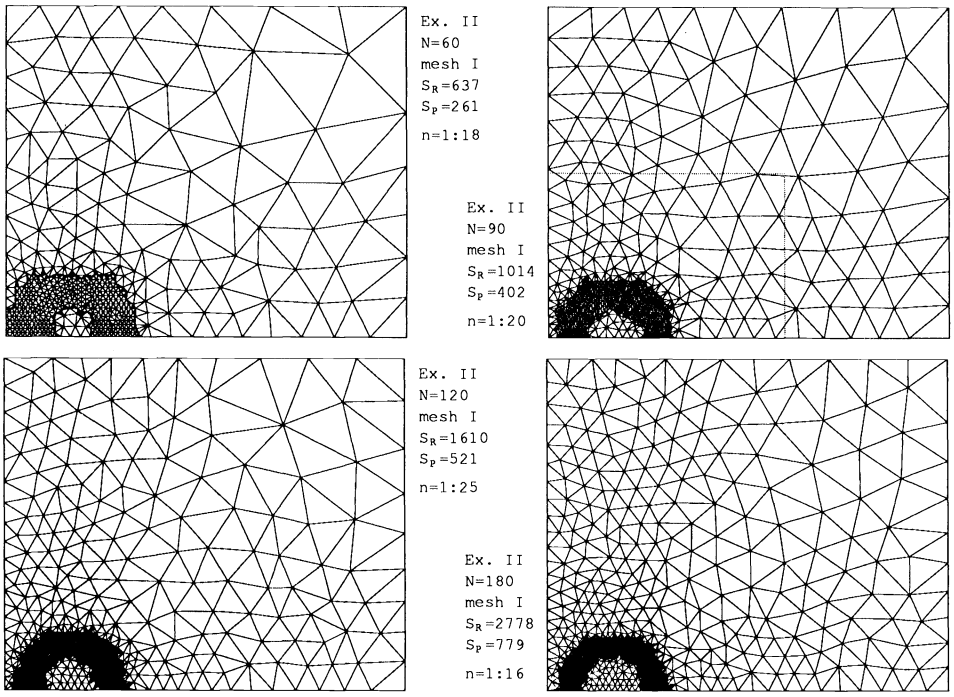


FIG. 7.6. *Example* II (AM, $\mathbf{N} = 60, 90, 120, 180$): *First mesh.*

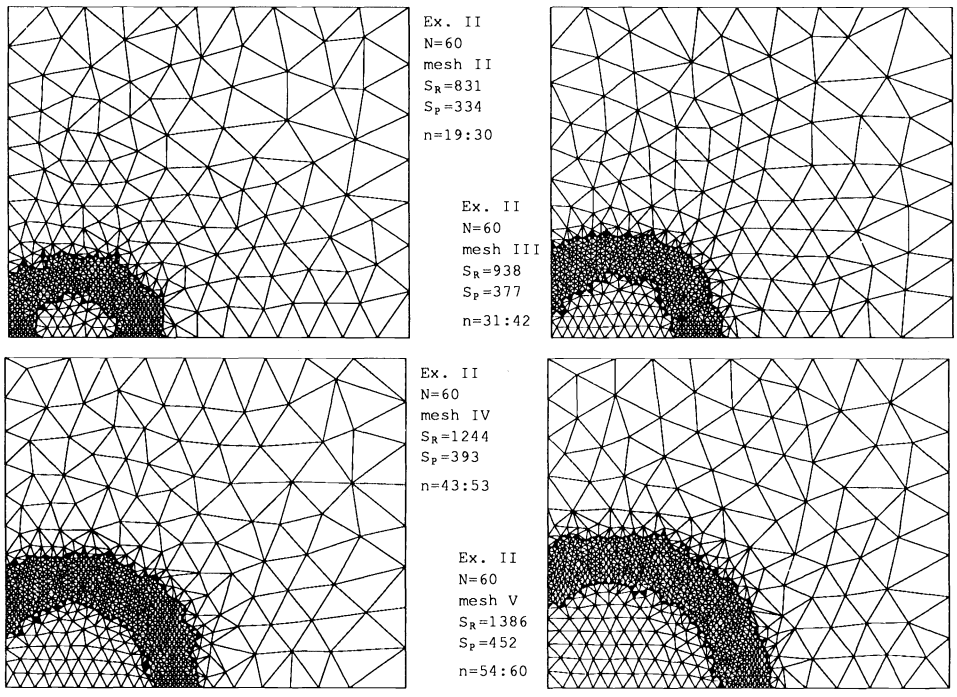


FIG. 7.7. *Example* II (AM, $\mathbf{N} = 60$, $S_F \cup S_B$ *free*): *Subsequent meshes.*

and fixed triangles. The agreement between these curves is again quite remarkable, as can be observed in Fig. 7.8(b). The local meshsize can be seen in that picture

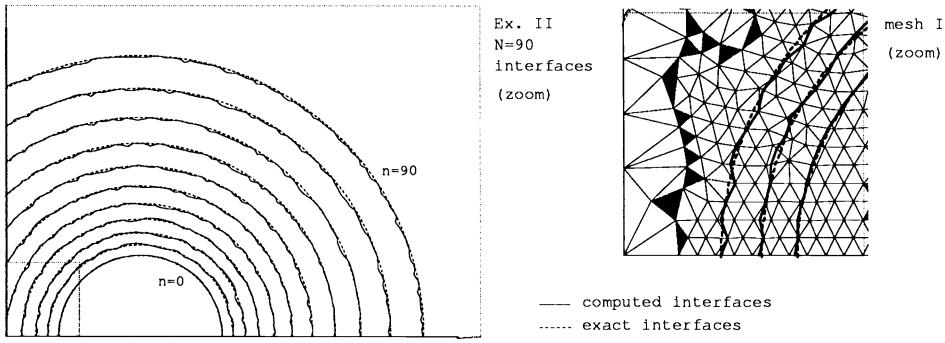


FIG. 7.8. *Example* II (AM, $\mathbf{N} = 90$, $S_F \cup S_B$ *fixed*): *Interfaces at* $n = 9k$ $(0 \leq k \leq 10)$; *zoom of mesh I.*

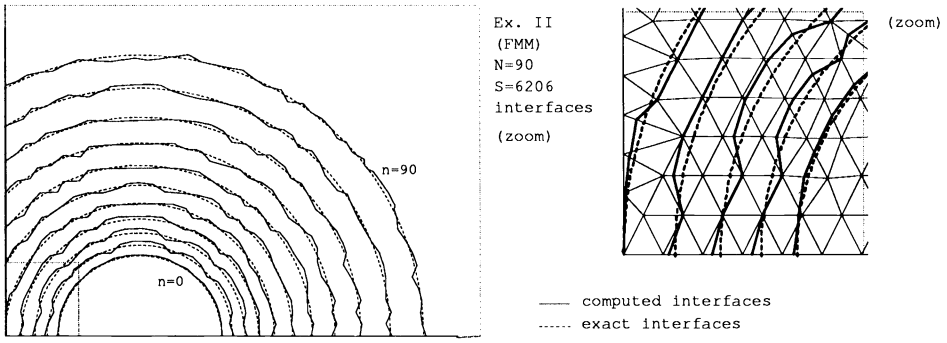


FIG. 7.9. *Example* II (FMM, $\mathbf{N} = 90$): *Interfaces at* $n = 9k$ $(0 \leq k \leq 10)$; *zoom of the mesh.*

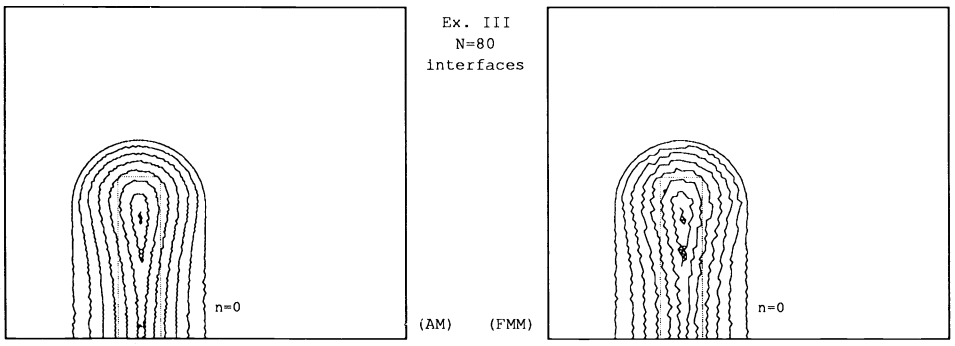


FIG. 7.10. *Example* III (AM (FMM), $\mathbf{N} = 80$): *Interfaces* at $n = 0, 10, 20, 30, 40, 50, 60, 67(68)$.

as well. Figure 7.9 is the analogue of Fig. 7.8 for FMM with $\mathbf{N} = 90$, $\mathbf{S} = 6206$, $\mathbf{J} = 3048$. This experiment is not reported in Table 7.11, but corresponds roughly to the same computational effort as for AM with $\mathbf{N} = 90$; however, $\mathbf{E}_\theta^\infty = 6.22 \cdot 10^{-2}$ and $\mathbf{E}_I^\infty = 4.34 \cdot 10^{-2}$ for FMM. Comments similar to those of Example I can certainly be made here.

7.3. Example III: Cusp formation. We consider a two-phase problem in $\Omega := (-2,4) \times (0,5)$, $T := 1$, with initial temperature given by the expression

$$\theta_0(x,y) := \begin{cases} 0.25(r^2-1), & r \le 1,\ y \ge 2; \\ (r-1), & r > 1,\ y \ge 2; \\ (|x|-1)(3-2\cos\pi(y-2)), & \end{cases} \qquad \begin{array}{ll} 0.25(x^2-1), & |x| < 1,\ y < 2; \\ 5(|x|-1), & |x| > 1,\ y < 1; \\ & |x| > 1,\ 1 \le y < 2. \end{array}$$

Here $r := (x^2 + (y-2)^2)^{1/2}$; a Dirichlet boundary condition $g_D := \theta_0(1+t)$ is imposed at $x = -2$, $x = 4$, $y = 5$; and a reflection condition is prescribed at $y = 0$. The exact solution is not known, but a cusp is expected to develop at $(0,0)$.

The problem was solved with $\mathbf{N} = 80$ and the following parameters:

$\lambda = 2.5$, $\Lambda = 6$, $M = 5$, $r = 0.65\sqrt{\tau}$,
$n_1 = 0.75$, $n_2 = 0.5$, $n_3 = 1.5 \to \mu_1 = 5.71$, $\mu_2 = 26.4$, average value of $\mu_3 \approx 1.1$,
$\mu_1^* = \mu_2^* = 3$, $\mu_1^\sharp = \mu_2^\sharp = 2$, $\mu_3^- = 0.67$, $\mu_3^+ = 1.5$,
$C_1 = \|D^2\theta_0\|_{L^\infty(\Omega)}$, $C_2 = 2\|\nabla\theta_0\|_{L^\infty(\Omega)}$, $C_3 = 2$ length(I_0), $C_4 = C_3\lambda$,
$\varepsilon = 5 \cdot 10^{-6}$ for $\mathbf{N} = 80$.

The first three mesh changes were due to a simultaneous failure of both tests (3.7) and (3.8). The interface location was then responsible for the remaining remeshings.

Figure 7.10 shows the discrete interface for a number of time steps for both AM and FMM, whereas the cusp formation is depicted in Figs. 7.11 and 7.12. It is worth noting the formation of a small numerical mush just when the cusp develops, that remains as long as the solid phase does. The annulus so created shrinks until it disappears, but exhibiting a sort of cusp for at least seven time steps. Note the superior local resolution associated with AM. A numerical mush is to be expected every time the temperature gradient is almost zero, because such a condition is close to a degenerate situation. This is precisely what happens in the solid phase, which contains so little energy that it cannot be numerically distinguished from a true mush. The number of triangles with zero temperature is, however, very small, making possible the use of the free triangle strategy. This is always preferable in terms of mesh regularity. Figure 7.13 illustrates the initial mesh, together with a zoom of the refined region. The subsequent meshes are depicted in Fig. 7.14.

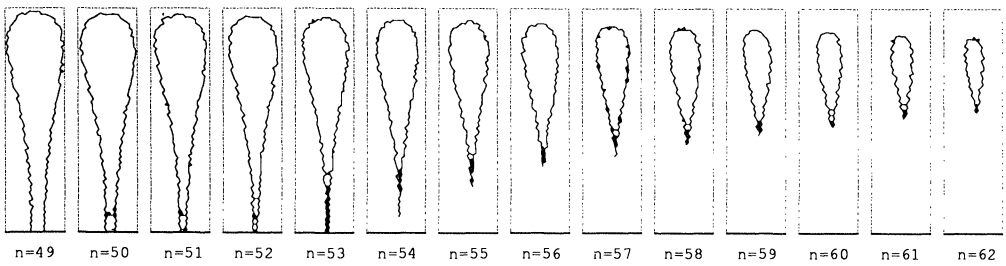


FIG. 7.11. *Example* III (AM, $\mathbf{N} = 80$, $S_F \cup S_B$ *free*): *Cusp formation* (*zoom*).

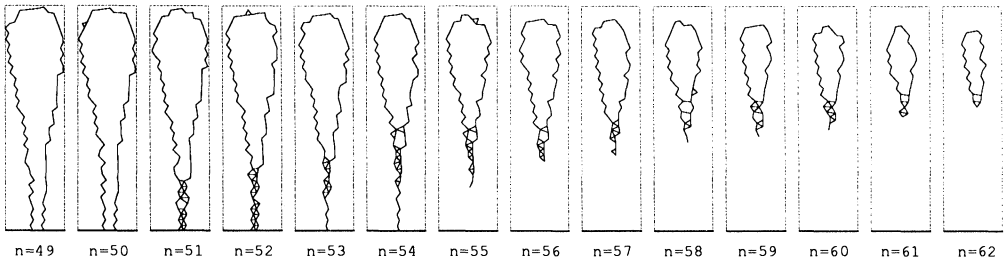


FIG. 7.12. *Example* III (FMM, **N** = 80): *Cusp formation* (zoom).

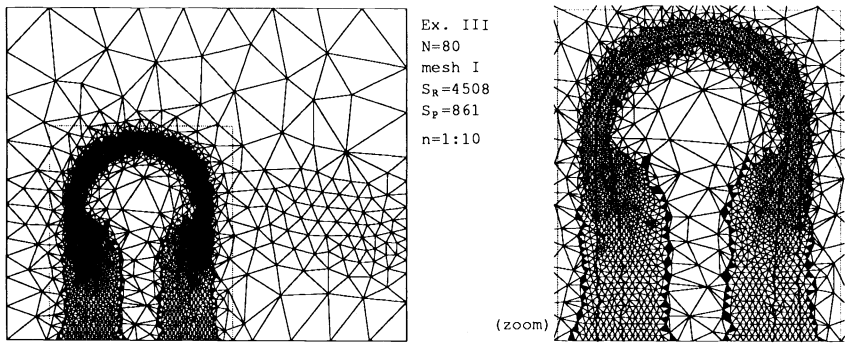


FIG. 7.13. *Example* III (AM, **N** = 80): *Mesh I and zoom of refined region,* RED_ZONE, *first and last interface.*

7.4. Example IV: Evolution of a mushy region. The last test is a two-phase Stefan problem with an initial mushy region expanding in the solid phase. The enthalpy in the mush increases so as to cause the spontaneous appearance of a liquid phase in its interior. Such a phase then expands faster than the mush, which eventually disappears for the interface to become sharp and the problem to become a classical one. The exact enthalpy is given by the following expression, with $(x, y) \in \Omega := (0,5)\times(0,1)$ and $0 < t < T := 1$:

$$u(x,y,t) := \begin{cases} \begin{cases} 4\Phi_2(x,y,t), & \Phi_2(x,y,t) \geq 0, \\ -\Phi_2(x,y,t)^2, & \Phi_2(x,y,t) < 0, \end{cases} & 0 < t \leq 0.25, \\ \begin{cases} \Psi_2(x,y,t) + 1, & \Phi_1(x,y,t) \geq 0, \\ 4\Phi_2(x,y,t), & \Phi_1 < 0,\ \Phi_2 \geq 0, \\ -\Phi_2(x,y,t)^2, & \Phi_2(x,y,t) < 0, \end{cases} & 0.25 < t \leq 0.75, \\ \begin{cases} \Psi_2(x,y,t) + 1, & \Phi_1(x,y,t) \geq 0, \\ \Psi_1(x,y,t), & \Phi_1(x,y,t) < 0, \end{cases} & 0.75 < t < 1, \end{cases}$$

where $s_1(t) := \max(t - 0.25, 0)$ and $s_2(t) := \max(0.5(t + 0.25), t - 0.25)$ determine the mushy region, namely, $\mathcal{M}(t) = (s_1(t), s_2(t))$, and $\Phi_1(x,y,t) := s_1(t) - x$, $\Phi_2(x,y,t) := s_2(t) - x$, $\Psi_1(x,y,t) := 2\Phi_1(x,y,t)(t - 0.75) - \Phi_1(x,y,t)^2$, $\Psi_2(x,y,t) := s_1(t)^2 - x^2$. A Dirichlet boundary condition is assigned at $x = 5$, whereas a reflection condition is applied at $x = 0$, $y = 0$, and $y = 1$. The exact norms are

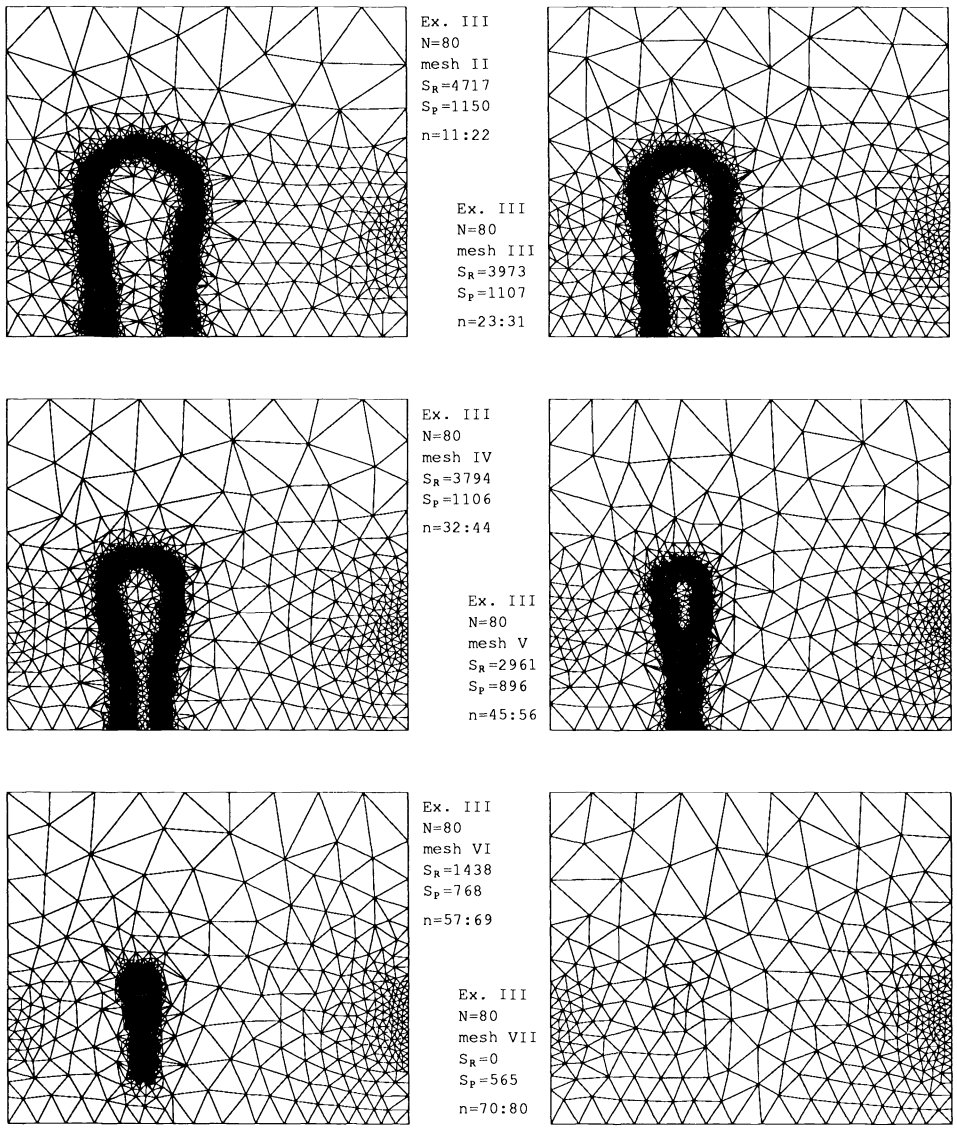


FIG. 7.14. *Example* III (AM, $\mathbf{N} = 80$, $S_F \cup S_B$ *free*): *Subsequent meshes.*

$\|\theta\|_{L^2(Q)} \approx 5.61$, $\|\theta\|_{L^\infty(Q)} \approx 6.13$, and $\|u\|_{L^2(Q)} \approx 5.65$. The problem is solved in two dimensions without taking any advantage of its one-dimensional structure. We can thus compare the true interfaces $x = s_1(t)$ and $x = s_2(t)$ with the computed ones, as illustrated in Fig. 7.15.

The parameters employed in our numerical simulation are $\mathbf{N} = 80, 160$, and

$\lambda = 2$, $\Lambda = 5$, $M = 3$, $r = 0.45\sqrt{\tau}$,
$n_1 = 1$, $n_2 = 1$, $n_3 = 3 \rightarrow \mu_1 = 0.83$, $\mu_2 = 2.17$, average value of $\mu_3 \approx 1.2$,
$\mu_4 = 3$, $\mu_5 = 2.5\lambda$, $\Lambda^\star = \mu_5$,
$\mu_1^\star = \mu_2^\star = 4$, $\mu_1^\sharp = \mu_2^\sharp = 2.4$, $\mu_3^- = 0.5$, $\mu_3^+ = 2$,
$C_1 = \|D^2\theta_0\|_{L^\infty(\Omega)}$, $C_2 = 2\|\nabla\theta_0\|_{L^\infty(\Omega)}$, $C_3 = 2$ length(I_0), $C_4 = C_3\lambda$,
$\varepsilon = 5 \cdot 10^{-6}$ for $\mathbf{N} = 80$.

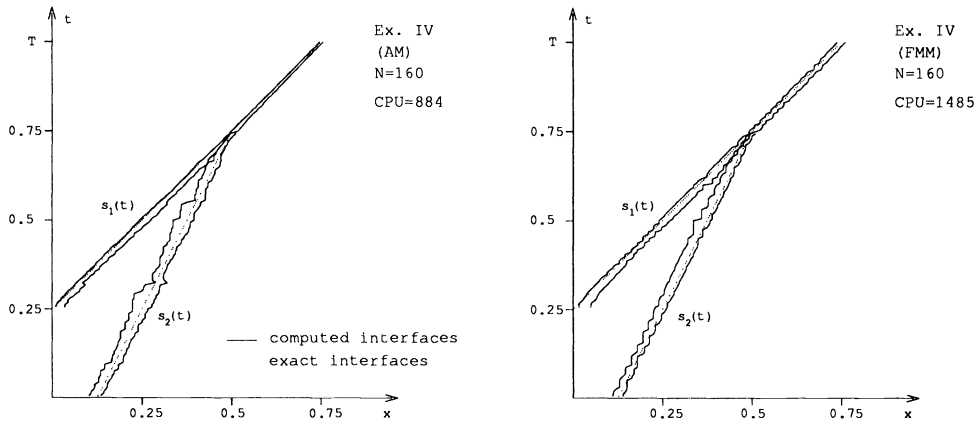


FIG. 7.15. *Example* IV (AM *and* FMM, N = 160) : *Interfaces* $x = s_1(t)$ *and* $x = s_2(t)$.

The various mesh changes were produced by either incorrect free boundary location or failure of (3.7). Table 7.13 shows a comparison of AM and FMM performances in terms of pointwise accuracy. The superior performance of AM is again apparent, even though the refined region is not small compared with the entire domain.

TABLE 7.13
Example IV: *Comparison of pointwise accuracy.*

Adaptive method ($S_I \cup S_B$ free)				Fixed mesh method			
N × J	E_θ^∞	E_I^∞	t	N × J	E_θ^∞	E_I^∞	t
80×506	9.09	5.47	173	80×1396	9.27	9.53	180
160×1385	1.15	4.36	884	160×5216	1.67	4.39	1485

Figure 7.15 contains the exact interfaces $x = s_1(t)$ and $x = s_2(t)$, together with the computed ones. The two lines near each curve $s_i(t)$ represent minimum and maximum abcissas of the approximate free boundaries for each time step. Pointwise accuracy is now comparable for both methods, AM and FMM, but AM is certainly faster and provides a better approximation of the nondegenerate interface $s_1(t)$. The approximation of $s_2(t)$ is, instead, less accurate because it crosses big triangles. This, however, does not destroy the pointwise approximation quality of AM simply because the physical variables are smooth across $s_2(t)$. Moreover, $s_2(t)$ happens to be properly identified as degenerate and, as so, can exit from the SECURITY_ZONE without causing a mesh modification (see §3.3). Figure 7.16 shows the various meshes for N = 80 as well as exact and computed interfaces for those time steps corresponding to mesh changes.

8. Conclusions. We finally summarize the leading features of our self-adaptive mesh modification strategy for Stefan problems. The method is based on several heuristic ideas that reflect the underlying degenerate parabolic structure, but possesses a rigorous mathematical foundation. Its efficiency is dictated by the nonlinear

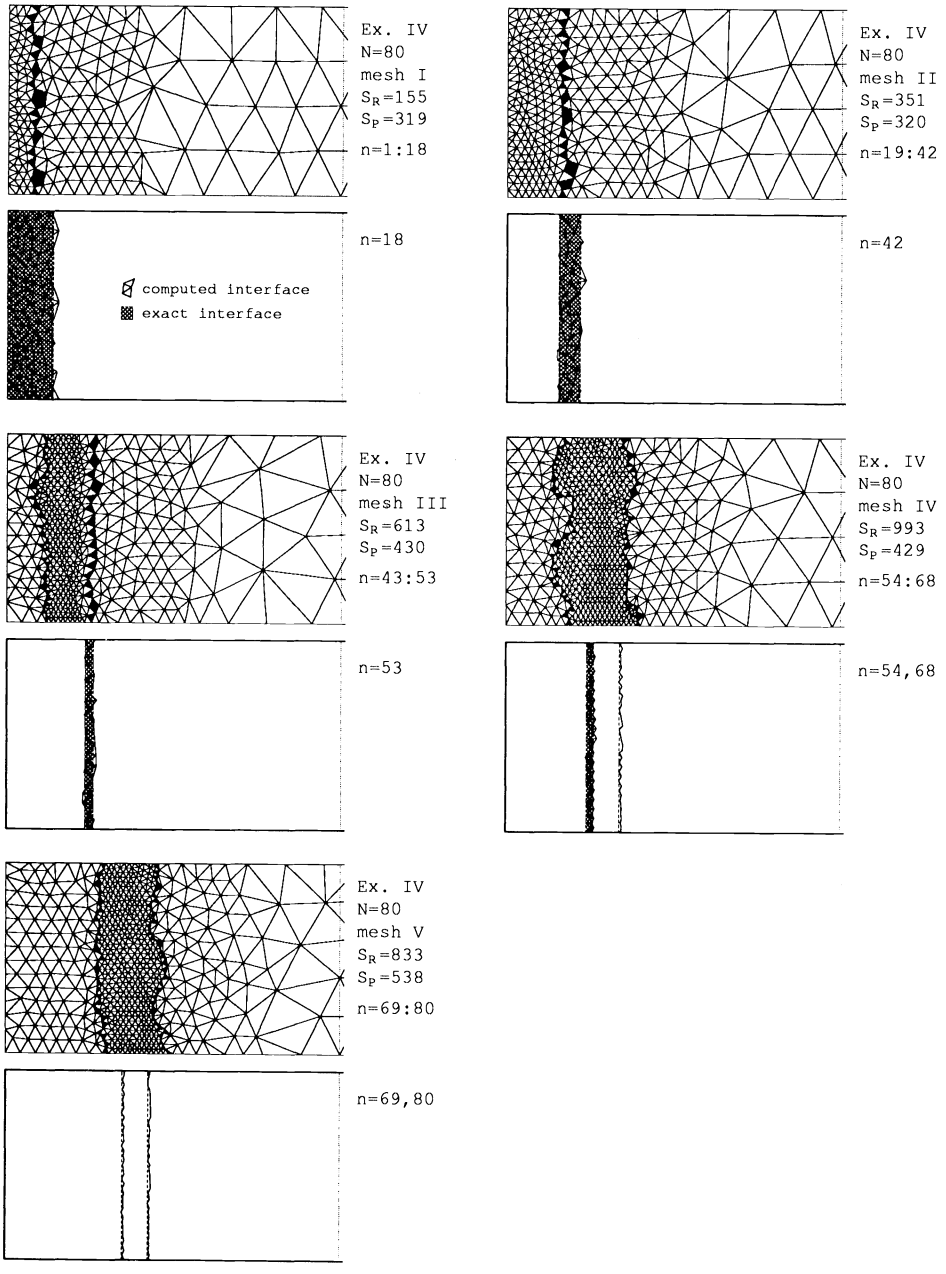


FIG. 7.16. *Example* IV (*AM*, $\mathbf{N} = 80$, $S_I \cup S_B$ *free*) : *Meshes and interfaces.*

(successive overrelaxation) solver, by several search operations to be performed essentially in remeshing the domain and interpolating the discrete enthalpy, and by the various quality tests. The use of quadtree structured data is thus crucial to achieve a quasi-optimal computational complexity in those search tasks. The adaptive method requires less computational effort for a desired pointwise accuracy than the fixed mesh method. Moreover, the two proposed local refinement strategies, with and without fixed triangles, perform quite similarly. Both the L^2 and the L^∞-errors for temperature exhibit a practical linear rate, which is not theoretically explainable; this topic warrants further investigation. The improvement gained in L^∞ is extremely pronounced, as expected for a pointwise interpolation strategy (see §3.1). Nondegenerate exact free boundaries are approximated within one single element, thus providing a practical $O(\tau)$-rate of convergence in distance for interfaces, the best one can hope for. In such cases, the "jump discontinuity" for discrete enthalpy is also located within one element without exhibiting spurious oscillations. The total number of degrees of freedom is not yet optimal as compared with $O(\mathbf{N}^2)$ for the heat equation. This issue, as well as the design of more efficient nonlinear solvers, deserves further analysis.

The two-phase Stefan problem has insufficient structure to model some important physical phenomena in solidification. Perhaps the most basic one is supercooling, which, together with surface tension effects, is responsible for pattern formation such as dendritic growth. It seems that our present approach provides a powerful numerical tool to be exploited, for instance, in investigating how well phase field models mimic such complex physical situations.

REFERENCES

[1] I. BABUŠKA, O.C. ZIENKIEWICZ, J. GAGO, AND E.R. DE A. OLIVEIRA, *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley, Chichester, U.K., 1986.

[2] G.F. CAREY, M. SHARMA, AND K.C. WANG, *A class of data structures for* 2-D *and* 3-D *adaptive mesh refinement*, Internat. J. Numer. Methods Engrg., 26 (1988), pp. 2607–2622.

[3] P.G. CIARLET, *The Finite Element Method for Elliptic Problems*, North–Holland, Amsterdam, 1978.

[4] J.F. CIAVALDINI, *Analyse numérique d'un problème de Stefan à deux phases par une méthode d'éléments finis*, SIAM J. Numer. Anal., 12 (1975), pp. 464–487.

[5] C.M. ELLIOTT, *Error analysis of the enthalpy method for the Stefan problem*, IMA J. Numer. Anal., 7 (1987), pp. 61–71.

[6] B. JOE, *Delaunay triangular meshes in convex polygons*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 514–539.

[7] D.N. KNUTH, *The Art of Computer Programming*, Vol. 3, Addison–Wesley, Reading, MA, 1973.

[8] C.L. LAWSON, *Transforming triangulations*, Discrete Math., 3 (1972), pp. 365–372.

[9] S.H. LO, *Delaunay triangulation of non-convex planar domains*, Internat. J. Numer. Methods Engrg., 28 (1989), pp. 2695–2707.

[10] R. LÖHNER, *Some useful data structures for the generation of unstructured meshes*, Comm. Appl. Numer. Methods, 4 (1988), pp. 123–135.

[11] R.H. NOCHETTO, *Numerical methods for free boundary problems*, in Free Boundary Problems: Theory and Applications II, K.H. Hoffmann and J. Sprekels, eds., Res. Notes Math. 186, Pitman, London, 1990, pp. 555–566.

[12] R.H. NOCHETTO, M. PAOLINI, AND C. VERDI, *Local mesh refinements for two-phase Stefan problems in two space variables*, in Mathematical Models for Phase Change Problems, J.F. Rodrigues, ed., Birkhäuser-Verlag, Basel, Switzerland, 1989, pp. 261–286.

[13] ———, *Selfadaptive mesh modification for parabolic* FBP*s: theory and computation*, in Free Boundary Problems, K.-H. Hoffmann and J. Sprekels, eds., Birkhäuser-Verlag, Basel, Switzerland, 1990, pp. 181–206.

[14] ———, *An adaptive finite elements method for two-phase Stefan problems in two space dimensions. Part* I: *Stability and error estimates*, Math. Comp. (1991, to appear).

[15] R.H. NOCHETTO AND C. VERDI, *Approximation of degenerate parabolic problems using numerical integration*, SIAM J. Numer. Anal., 25 (1988), pp. 784–814.

[16] J.M. ORTEGA AND W.C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[17] M. PAOLINI, G. SACCHI, AND C. VERDI, *Finite element approximations of singular parabolic problems*, Internat. J. Numer. Methods Engrg., 26 (1988), pp. 1989–2007.

[18] M. PAOLINI AND C. VERDI, *An automatic mesh generator for planar domains*, Riv. Inform., 20 (1990), pp. 251–267.

[19] R. SEDGEWICK, *Algorithms*, Addison–Wesley, Reading, MA, 1983.

[20] R. SIBSON, *Locally equiangular triangulations*, Comput. J., 21 (1978), pp. 243–245.

[21] C. VERDI, *On the numerical approach to a two-phase Stefan problem with non-linear flux*, Calcolo, 22 (1985), pp. 351–381.

[22] ———, *Optimal error estimates for an approximation of degenerate parabolic problems*, Numer. Funct. Anal. Optim., 9 (1987), pp. 657–670.

[23] M.A. YERRY AND M.S. SHEPARD, *A modified quadtree approach to finite element mesh generation*, IEEE Comput. Graphics Appl., 3 (1983), pp. 39–46.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 5, pp. 1245-1253, September 1991
© 1991 Society for Industrial and Applied Mathematics
014

TIMELY COMMUNICATION

Under the "timely communications" policy for the SIAM Journal on Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of the journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

BLOCK GAUSS REDUCTION TO HESSENBERG FORM*

AUGUSTIN A. DUBRULLE†

Abstract. The latest developments in matrix software generally confine the block methods based on stabilized elementary (Gauss) transformations to the solution of linear equations, in a concern for numerical stability. For example, LAPACK relies entirely on Householder methods for the solution of standard eigenvalue problems, including the reduction to Hessenberg form. There is, however, a wide range of problems in practice for which the Gauss reduction, more economical of computation, is safe and quite comparable to its Householder alternative for accuracy of the computed eigenvalues. In this paper, a block version of the Gauss reduction is derived from the serial method by bundling sequences of stabilized elementary matrices into single operators amenable to implementation with high-level BLAS. Results of experiments conducted with an IBM 3090 VF model *E* (one processor) for random matrices include details of FORTRAN implementation and a performance comparison with a Householder block algorithm.

Key words. Hessenberg form, stabilized elementary transformations, matrix block algorithms, matrix eigenvalue problem, numerical linear algebra

AMS(MOS) subject classification. 65F

1. Introduction.

Matrix block methods are generally associated with high-performance computers as a device to improve processing efficiency by paring memory references. Their simplest form is found in implementations of the BLAS (basic linear algebra subprograms [8], [9], [16]) tuned for specific machine architectures. In more sophisticated computations, they are based on the replacement of groups of elementary transformations by single operators that are constructed economically and can be implemented with high-level BLAS [2], [7]. Because of their demonstrated effectiveness in the solution of linear equations [2], [18] and eigenvalue problems [1], [7], [10], block methods pervade the design of LAPACK, a package currently under development for matrix computations with high-performance machines [3], [4].

In LAPACK, the solution of the eigenvalue problem for general matrices relies entirely on block Householder algorithms in order to guarantee unconditional stability. There is, however, a wide range of problems for which Householder transformations can be safely replaced with stabilized elementary transformations in the initial phase of the computation, that is, the similarity reduction of the matrix to Hessenberg form. Such a substitution has the advantage of cutting down by half the reduction time, but introduces a risk of numerical instability that fortunately appears to be remote in practice [13], [21], [23].

This paper describes a block transformation of a general dense matrix to Hessenberg form, based on elementary similarity transformations. It is a generalization of the "direct reduction" in [24], and is akin to the block versions of the LU decomposition.

* Received by the editors May 22, 1989; accepted for publication (in revised form) December 6, 1990.
† IBM Scientific Center, 1530 Page Mill Rd., Palo Alto, California 94304.

Section 2 introduces the notation used in the derivation of the algorithm, outlines the basic numerical method, and briefly discusses the properties of Gauss elementary transformations. The block form of products of stabilized elementary matrices is derived in § 3. The reduction algorithm is constructed in § 4. Section 5 reports experiments where the Gauss method and a Householder homologue are compared for performance on an IBM 3090 VF, a vector machine with cache.

2. Background. Let us first introduce a useful notation. For any vector $\mathbf{t} \in \mathbb{R}^n$, there is a "pivoting" matrix in $\mathbb{R}^{n\times n}$, not necessarily unique, defined by

$$\boldsymbol{\Theta}_j[\mathbf{t}] = \mathbf{I} - (\mathbf{e}_j - \mathbf{e}_{j'})(\mathbf{e}_j - \mathbf{e}_{j'})^T, \qquad |t_{j'}| = \max_{i \geqq j} |t_i|.$$

Such a matrix, obviously symmetric and involutory, is orthogonal ($\boldsymbol{\Theta}_j = \boldsymbol{\Theta}_j^T = \boldsymbol{\Theta}^{-1}$), and has the effect of exchanging t_j and some component $t_{j'}$ of largest magnitude in the set $\{t_j, \cdots, t_n\}$.

For $\mathbf{t} \in \mathbb{R}^n$, $t_j \neq 0$, the Gauss matrix

$$\mathbf{G}_j[\mathbf{t}] = \mathbf{I} + \mathbf{v}_j \mathbf{e}_j^T, \qquad (\mathbf{G}_j[\mathbf{t}])^{-1} = \mathbf{I} - \mathbf{v}_j \mathbf{e}_j^T,$$

$$v_{ij} = 0, \qquad i \leqq j, \tag{2.1}$$

$$v_{ij} = -\frac{t_i}{t_j}, \qquad i > j,$$

is designed to annihilate the components of $\mathbf{t}$ below t_j, and to leave invariant any vector in span $\{\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_j\}$. Such an operator is generally unstable, since the "Gauss vector" $\mathbf{v}_j[\mathbf{t}]$ is not bounded. Hence, it is usually paired with a pivoting matrix to form a stabilized elementary matrix [24], defined by

$$\mathbf{M}_j[\mathbf{t}] = \mathbf{G}_j[\boldsymbol{\Theta}_j[\mathbf{t}]]\boldsymbol{\Theta}_j[\mathbf{t}],$$

which also annihilates the components of $\mathbf{t}$ below t_j, may modify t_j, and leaves invariant any vector in span $\{\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_{j-1}\}$. Its ℓ_2 norm and condition number are the same as those of $\mathbf{G}_j[\boldsymbol{\Theta}_j[\mathbf{t}]]$:

$$\|\mathbf{M}_j\|_2 = \tfrac{1}{2}(\|\mathbf{v}_j\|_2 + \sqrt{\|\mathbf{v}_j\|_2^2 + 4}), \qquad \kappa_2(\mathbf{M}_j) = \|\mathbf{M}_j\|_2^2,$$

$$\mathbf{v}_j = \mathbf{v}_j[\boldsymbol{\Theta}_j[\mathbf{t}]], \qquad \|\mathbf{v}_j\|_\infty \leqq 1,$$

obtained by direct derivation of the extreme singular values of Gauss matrices. A bound immediately follows by setting to unity the nontrivial components of $\mathbf{v}_j$:

$$\kappa_2(\mathbf{M}_j) \leqq 1 + \frac{n-j}{2}\left[1 + \sqrt{1 + \frac{4}{(n-j)^2}}\right]. \tag{2.2}$$

Note that simpler bounds can be obtained for the ℓ_∞ norm:

$$\|\mathbf{M}_j\|_\infty \leqq 2, \qquad \|\mathbf{M}_j^{-1}\|_\infty \leqq 2, \qquad \kappa_\infty(\mathbf{M}_j) \leqq 4. \tag{2.3}$$

In the following, the notation $\boldsymbol{\Theta}_j[\mathbf{t}]$, $\mathbf{G}_j[\boldsymbol{\Theta}_j[\mathbf{t}]]$, and $\mathbf{v}_j[\boldsymbol{\Theta}_j[\mathbf{t}]]$, associated with a stabilized elementary matrix $\mathbf{M}_j[\mathbf{t}]$, will be abbreviated to $\boldsymbol{\Theta}_j$, $\mathbf{G}_j$, and $\mathbf{v}_j$, for simplicity.

We now turn to the similarity reduction of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ to Hessenberg form $\mathbf{H}$ by stabilized elementary transformations. The basic (serial) right-looking algorithm is represented by

$$\left.\begin{aligned} &\mathbf{A}^{(1)} = \mathbf{A}, \\ &\mathbf{M}_{k+1} = \mathbf{M}_{k+1}[\mathbf{a}_k^{(k)}], \\ &\mathbf{A}^{(k+1)} = \mathbf{M}_{k+1}\mathbf{A}^{(k)}\mathbf{M}_{k+1}^{-1}, \end{aligned}\right\} k = 1, 2, \cdots, n-2. \tag{2.4}$$

It is easy to verify that $\mathbf{a}_k^{(k+1)} = \mathbf{M}_{k+1}\mathbf{a}_k^{(k)}$ is by construction the kth column of a Hessenberg matrix. It remains invariant in the postmultiplication by $\mathbf{M}_{k+1}^{-1}$, and in the subsequent similarity transformations of the reduction. Hence, $\mathbf{A}^{(n-1)}$ is a Hessenberg form of $\mathbf{A}$, to which the *QR* algorithm can be applied for eigenvalue computations [24].

If bound (2.2) or (2.3) is taken as a stability indicator of the method, it is clear that there is a risk of exponential growth for the elements of the matrix [22], [24], with possible adverse effect on the computed eigenvalues. For that reason, a reduction that substitutes appropriate Householder reflectors for the stabilized elementary matrices in equations (2.4) is generally preferable for its unconditional stability, but at the cost of doubling the number of arithmetic operations. In practice, however, except for pathological cases often contrived, the Gauss method behaves much better than the bounds (2.2) or (2.3) would indicate, and delivers good results for eigenvalue computations [13], [21], [24]. The reasons for such consistent behavior are hard to analyze, and are further complicated by the fact that the method does not preserve the condition numbers of the eigenvalues. The empirical study of stabilized elementary transformations for the solution of linear systems in [23] provides some insight into the subject, based on statistical analyses. In any case, results from error analysis and perturbation theory [24] indicate that the use of Gauss reduction should be restricted to diagonalizable matrices with matrices of eigenvectors reasonably well conditioned, an occurrence seemingly not uncommon in practice. Some numerical examples presented in § 5 illustrate the results that can be expected from the method under these general (neccesary) conditions of applicability.

Because of its economy of computation, the Gauss algorithm is particularly attractive when the reduction to Hessenberg form consumes a sizeable portion of the time devoted to the solution of an eigenvalue problem (e.g., when only eigenvalues are computed). A FORTRAN implementation of algorithm (2.4) is found in subroutine *elmhes* of EISPACK [6], [21].

3. Products of stabilized elementary matrices. Algorithm (2.4) can be recast in terms of products of stabilized elementary matrices,

$$\mathbf{A}^{(p)} = \mathbf{\Phi}_{k+1:p}\mathbf{A}^{(k)}\mathbf{\Phi}_{k+1:p}^{-1}, \qquad p > k+1, \tag{3.1}$$

defined by

$$\begin{aligned}\mathbf{\Phi}_{k+1:p} &= \mathbf{M}_p\mathbf{M}_{p-1}\cdots\mathbf{M}_{k+1},\\ &= \mathbf{G}_p\mathbf{\Theta}_p\mathbf{G}_{p-1}\mathbf{\Theta}_{p-1}\cdots\mathbf{G}_{k+1}\mathbf{\Theta}_{k+1},\end{aligned} \tag{3.2}$$

and for which we shall seek equivalent block operators.

The first step of the reformulation consists of separating the Gauss and pivoting matrices in $\mathbf{\Theta}_{k+1:p}$. Consider the product

$$\mathbf{\Psi}_{i+1:j} = \mathbf{\Pi}_{i+1:j}\mathbf{G}_i, \qquad \mathbf{\Pi}_{i+1:j} = \mathbf{\Theta}_j\mathbf{\Theta}_{j-1}\cdots\mathbf{\Theta}_{i+1}, \qquad j > i.$$

Using the property that $\mathbf{e}_i^T\mathbf{\Theta}_j = \mathbf{0}$ for $j > i$, we have

$$\mathbf{\Psi}_{i+1:j} = \mathbf{G}_i^{(j)}\mathbf{\Pi}_{i+1:j}, \qquad \mathbf{G}_i^{(j)} = \mathbf{I} + \mathbf{v}_i^{(j)}\mathbf{e}_i^T, \qquad \mathbf{v}_i^{(j)} = \mathbf{\Pi}_{i+1:j}\mathbf{v}_i,$$

where $\mathbf{G}_i^{(j)}$ is a Gauss matrix. The repeated application of these formulas to the product (3.2) yields

$$\mathbf{\Phi}_{k+1:p} = (\mathbf{G}_p^{(p)}\mathbf{G}_{p-1}^{(p)}\cdots\mathbf{G}_{k+1}^{(p)})\mathbf{\Pi}_{i+1:p}, \qquad \mathbf{G}_p^{(p)} = \mathbf{G}_p. \tag{3.3}$$

The product of the Gauss matrices can now be expressed as a single operator, as outlined below. Using inverses,

$$\mathbf{G}_p^{(p)}\mathbf{G}_{p-1}^{(p)}\cdots\mathbf{G}_{k+1}^{(p)} = (\mathbf{G}_{k+1}^{(p)-1}\mathbf{G}_{k+2}^{(p)-1}\cdots\mathbf{G}_p^{(p)-1})^{-1},$$

with $\mathbf{G}_i^{(p)-1} = \mathbf{I} - \mathbf{v}_i^{(p)} \mathbf{e}_i^T$, $k+1 \leqq i \leqq p$, we easily derive the following identity:

$$\mathbf{G}_p^{(p)} \mathbf{G}_{p-1}^{(p)} \cdots \mathbf{G}_{k+1}^{(p)} = (\mathbf{I} - \mathbf{V}_{k+1:p}^{(p)})^{-1}, \qquad \mathbf{V}_{k+1:p}^{(p)} = \sum_{j=k+1}^{p} \mathbf{v}_j^{(p)} \mathbf{e}_j^T,$$

where $\mathbf{V}_{k+1:p}$ is subdiagonal in its nontrivial columns $k+1, \cdots, p$. Hence, the composition of stabilized elementary transformations (3.2) reduces to

$$\mathbf{\Phi}_{k+1:p} = (\mathbf{I} - \mathbf{V}_{k+1:p}^{(p)})^{-1} \mathbf{\Pi}_{k+1:p}, \qquad \mathbf{\Pi}_{k+1:p} = \mathbf{\Theta}_p \mathbf{\Theta}_{p-1} \cdots \mathbf{\Theta}_{k+1}.$$

The algorithmic implications of this formulation can be analyzed through the transformation of a vector:

$$\mathbf{z} = \mathbf{\Phi}_{k+1:p} \mathbf{x}, \qquad \mathbf{x} \in \mathbb{R}^n.$$

This operation consists of two steps, represented by

$$\mathbf{y} = \mathbf{\Pi}_{k+1:p} \mathbf{x}, \qquad \mathbf{z} = (\mathbf{I} - \mathbf{V}_{k+1:p}^{(p)})^{-1} \mathbf{y}. \tag{3.4}$$

The computation of $\mathbf{y}$ simply amounts to a succession of component exchanges in $\mathbf{x}$, and is trivial. The computation of $\mathbf{z}$ from $\mathbf{y}$ is more interesting. Based on the fact that $\mathbf{V}_{k+1:p}^{(p)}$ is generally nonzero only in columns $k+1, \cdots, p$, the second equation of (3.4) can be represented by the system

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{L} & \mathbf{0} \\ \mathbf{0} & -\mathbf{B} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{z}' \\ \mathbf{z}'' \\ \mathbf{z}''' \end{bmatrix} = \begin{bmatrix} \mathbf{y}' \\ \mathbf{y}'' \\ \mathbf{y}''' \end{bmatrix},$$

to be solved for $\mathbf{z}^T = (\mathbf{z}'^T, \mathbf{z}''^T, \mathbf{z}'''^T)$. The subspaces of the partition so defined have the dimensions k, $p-k$, and $n-p$, which accordingly determine $\mathbf{y}^T = (\mathbf{y}'^T, \mathbf{y}''^T, \mathbf{y}'''^T)$. $\mathbf{L}$ is a lower-triangular matrix with unit diagonal. The solution is given by

$$\mathbf{z}' = \mathbf{y}', \qquad \mathbf{z}'' = \mathbf{L}^{-1} \mathbf{y}'', \qquad \mathbf{z}''' = \mathbf{y}''' + \mathbf{B}\mathbf{z}''.$$

The nontrivial parts of this computation consist of a forward substitution proper for $\mathbf{z}''$ and a matrix multiplication for $\mathbf{z}'''$. The latter operation is especially interesting from a viewpoint of performance for allowing the use of level-two BLAS (or level-three BLAS when $\mathbf{y}$ and $\mathbf{z}$ are replaced by matrices). These considerations are pivotal to the efficient implementation of the block method derived in the next section.

4. Block Gauss reduction to Hessenberg form. The derivation of the block algorithm is based on some remarks below about the similarity transformation in algorithm (2.4),

$$\mathbf{M}_{k+1} = \mathbf{M}_{k+1}[\mathbf{a}_k^{(k)}],$$
$$\mathbf{A}^{(k+1)} = \mathbf{M}_{k+1} \mathbf{A}^{(k)} \mathbf{M}_{k+1}^{-1},$$

where the first $k-1$ columns of $\mathbf{A}^{(k)}$ are in the Hessenberg form.

We first consider the effect on $\mathbf{A}^{(k)}$ of the postmultiplication by $\mathbf{M}_{k+1}^{-1}$. The transformation involves columns $k+1, \cdots, n$ of $\mathbf{A}^{(k)}$ and alters at most two columns, $k+1$ and l, $l \geqq k+1$ (l is defined by the pivoting scheme). This operation, to which the partial Hessenberg form of the matrix is irrelevant, requires at most $2n$ storage updates. The part of the computation corresponding to the transformation by $\mathbf{G}_{k+1}^{-1}$ essentially reduces to a matrix-vector multiplication, an operation that can be performed efficiently with a level-two BLAS. Note that the transformation of column $k+1$ is essential to the determination of $\mathbf{M}_{k+2}$, the next stabilized elementary matrix of the reduction.

By contrast, the left-hand transformation of $\mathbf{A}^{(k)}$ by $\mathbf{M}_k$ affects all the elements of the trailing principal submatrix of order $(n-k+1)$ and requires $(n-k+1)^2$ storage updates.

These remarks suggest an algorithm where the right-hand transformations are applied sequentially and the left-hand transformations are implemented in block form. From the expression

$$\mathbf{A}^{(p)} = \mathbf{M}_p \mathbf{M}_{p-1} \cdots \mathbf{M}_{k+1} \mathbf{A}^{(k)} \mathbf{M}_{k+1}^{-1} \cdots \mathbf{M}_{p-1}^{-1} \mathbf{M}_p^{-1}, \qquad k < p,$$

and the formulations of the previous section, we get

$$\mathbf{A}^{(p)} = (\mathbf{I} - \mathbf{V}_{k+1:p}^{(p)})^{-1} \mathbf{\Pi}_{k+1:p} \hat{\mathbf{A}}^{(p)}, \qquad \hat{\mathbf{A}}^{(j)} = \hat{\mathbf{A}}^{(j-1)} \mathbf{M}_j^{-1}, \qquad k < j \leqq p, \qquad \hat{\mathbf{A}}^{(k)} = \mathbf{A}^{(k)},$$

where the stabilized elementary transformations are defined by

$$\mathbf{a}_j^{(j)} = (\mathbf{I} - \mathbf{V}_{k+1:j}^{(j)})^{-1} \mathbf{\Pi}_{k+1:j} \hat{\mathbf{a}}_j^{(j)}, \qquad \mathbf{M}_{j+1} = \mathbf{M}_{j+1}[\mathbf{a}_j^{(j)}].$$

These equations provide the basis for the block algorithm described by the formulas below, where r is the blocking parameter, and $k = 1 + sr$ for $s = 0, 1, \cdots, [(n-3)/r]$:

$$\hat{\mathbf{A}}^{(k)} = \mathbf{A}^{(k)}, \qquad \mathbf{\Pi}_{k+1:k} = \mathbf{I}, \qquad \mathbf{V}_{k+1:k}^{(k)} = \mathbf{0},$$

$$\left.\begin{aligned} &\mathbf{a}_j^{(j)} = (\mathbf{I} - \mathbf{V}_{k+1:j}^{(j)})^{-1} \mathbf{\Pi}_{k+1:j} \hat{\mathbf{a}}_j^{(j)}, \\ &\mathbf{M}_{j+1} = \mathbf{M}_{j+1}[\mathbf{a}_j^{(j)}], \\ &\mathbf{a}_j^{(j+1)} = \mathbf{M}_{j+1} \mathbf{a}_j^{(j)}, \\ &\mathbf{\Pi}_{k+1:j+1} = \mathbf{\Theta}_{j+1} \mathbf{\Pi}_{k+1:j}, \\ &\mathbf{V}_{k+1:j+1}^{(j+1)} = \mathbf{\Theta}_{j+1} \mathbf{V}_{k+1:j}^{(j)} + \mathbf{v}_{j+1} \mathbf{e}_{j+1}^T, \\ &\hat{\mathbf{A}}^{(j+1)} = \hat{\mathbf{A}}^{(j)} \mathbf{M}_{j+1}^{-1}, \end{aligned}\right\} \quad j = k, \cdots, p, \qquad p = \min(k + r - 1, n - 2),$$

$$\mathbf{A}^{(p+1)} = (\mathbf{I} - \mathbf{V}_{k+1:p+1}^{(p+1)})^{-1} \mathbf{\Pi}_{k+1:p+1} \hat{\mathbf{A}}^{(p+1)}.$$

The set of equations delimited by the brace describes the computation of a block of r columns $\mathbf{a}_j^{(j+1)}$ of the Hessenberg matrix (perhaps fewer columns for the last block), and the construction of the successive right-transformed matrices $\hat{\mathbf{A}}^{(j+1)}$. Its most significant operations are the forward substitution for $\mathbf{a}_j^{(j)}$ and the computation of $\hat{\mathbf{A}}^{(j+1)}$, which involve matrix-vector multiplications (level-two BLAS). The definition of $\mathbf{\Pi}_{k+1:j+1}$ is purely formal, and represents the recording of the pivoting operations. The last equation (matrix update), used only for columns $p+1, \cdots, n$ of $\mathbf{A}^{(p+1)}$, represents a transformation where the main computation consists of a matrix-matrix multiplication, and is thus amenable to the use of a level-three BLAS.

The block method generates algorithm (2.4) when $r = 1$, and Wilkinson's "direct" reduction [24] when $r = n$. Numerically, it preserves the properties of the basic algorithm from which it differs only in the left-hand transformations: there, for each group of r steps, a Gaussian elimination scheme is replaced by one akin to the corresponding LU decomposition, numerically equivalent if carried out in the same arithmetic (see, for example, [24, p. 223]). The only overhead at each block step consists of $r(r-1)/2$ element exchanges in the construction of $\mathbf{V}_{k+1:p+1}^{(p+1)}$, for a total of the order of $nr/2$ such operations. Consequently, from a comparison of operation counts, we should still expect the Gauss block method to perform the reduction in about half the time taken by its Householder homologues [7], [10].

The in situ implementation of the method has the columns of the Hessenberg matrix coexisting without overlap with the necessary trailing columns of the right-transformed iterates $\hat{\mathbf{A}}^{(j)}$ in the array of the original matrix. The Gauss vectors $\mathbf{v}_i^{(j)}$ are stored in the sub-Hessenberg cells, as is customary for the basic algorithm. Since the matrix updates essentially amount to forward substitutions performed in place, there is no need for ancillary storage. By contrast, Householder block reductions require an array of nr cells to hold intermediate results in the matrix updates. In addition, the WY representation of products of reflectors needs nr cells for its full form, or $r^2/2$ cells for its compact form [20].

5. Experiments. The performance results reported in this section were obtained from runs using one processor on an IBM 3090 VF model E [12], [17] for double-precision floating-point computations (14 digits in base 16 with rounding in magnitude towards zero). The following summarizes the main characteristics of this machine. The central processors have vector and scalar arithmetic units fed by a 65,536-byte cache. The operands of vector floating-point instructions can be cache sections of 128 elements, vector registers of the same capacity, and scalar registers. The results are returned in vector registers. The cycle time is 17.2×10^{-9} second. Taking into account pipeline overhead, actual processing speed for full registers is about 47 Mflop/s for vector instructions with one operation per cycle, or 87 Mflop/s for instructions with two operations per cycle and data in the cache. Because of the hierarchical memory, best algorithm efficiency is achieved for unit-stride successive storage references. The threshold vector dimension for which vector operation is more efficient than scalar operation is approximately 16.

The VS FORTRAN 2.3 compiler used in the experiments is described in [14] and [19]. In addition to mapping the obvious DO groups for scalar products and elementary linear combinations into corresponding vector instructions, this compiler can also translate some higher-level constructs into code, achieving supervector speed (see, for example, the discussion of the matrix–vector multiplication in [10]). As desired, vector or scalar machine operation can be specified for selected computations through compiler directives.

For comparison, the performance measurements also cover a block Householder reduction [10], preferred to that in [7] for its better performance attributable to a higher content of operations with level-three BLAS. It is based on the straight application of a *WY* representation of products of Householder matrices [2].

Some details in the FORTRAN implementations *orthes*3 and *elmhes*3 of the block Householder and Gauss algorithms deserve a few comments. In the range of matrix orders covered by the measurements, $100 \leqq n \leqq 1200$, experimental timings showed that values of r producing near-optimum performance are very much like $\sqrt{n}$ for both reductions, and thereby generate level-two operations with "thin and tall" matrices. For those, in-line straightforward FORTRAN code left to compiler vectorization was preferred for efficiency to an implementation based on level-two BLAS: although the super-vector speed delivered by the compiled code is slightly inferior to that of the BLAS, that loss is offset by the overhead of the subroutine calls for the range of values of r considered here. Likewise, all level-one operations were implemented in FORTRAN. The question of effective cache usage is resolved in the FORTRAN code simply by adhering to a unit-stride pattern of reference to matrix elements. At each block step of *elmhes*3, the forward substitutions involving the top triangular part of $\mathbf{V}_{k+1:p}^{(p)}$ are combined with the exchange operations for scalar mode of execution, as warranted by the values of $r(r/2$ is the average vector dimension in the forward substitutions). The right-hand elementary transformations and the matrix updates are carried out with level-two and level-three BLAS of the ESSL library [15], especially tuned for vector machine operation.

Experiments not detailed here were conducted for matrix orders less than 100 in order to find the lower limit of applicability of the Gauss block method. A unit-stride version *elmhes*1 of the basic Gauss algorithm, exclusively based on level-one computations and used for both scalar and vector machine operations, provided the reference for comparison. $(n = 32)$ was found to be the threshold of better performance of *elmhes*3 for vector execution of the two routines (a similar experiment revealed a threshold $(n = 16)$ for *orthes*3).

Random matrices with elements uniformly distributed in $(-1, +1)$ and $(0, 1)$, and normally distributed with null mean and unit variance, were used for the measurements of CPU execution times summarized by their arithmetic means in Fig. 1. The times for *elmhes*1 correspond to machine vector operation.

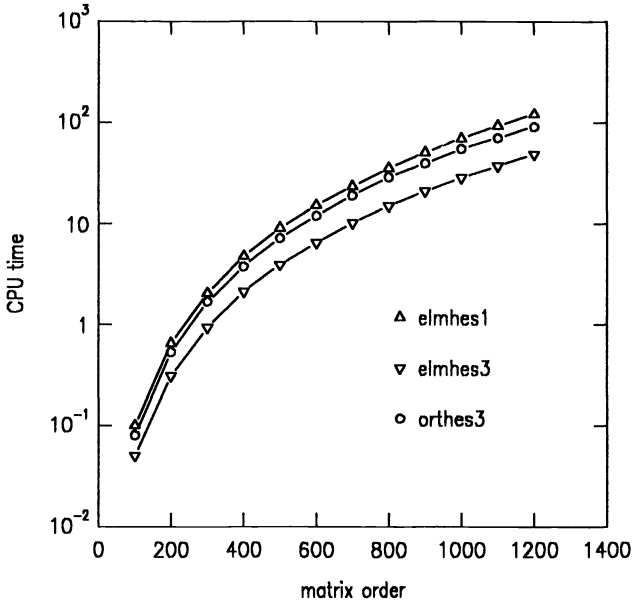


FIG. 1. *Average reduction times in seconds for* $r = [\sqrt{n}]$ (IBM 3090 VF *Model* E *using one processor*).

In all runs, the eigenvalues computed by EISPACK's routine *hqr* for Hessenberg matrices obtained from *elmhes*3 and *orthes*3 always had at least 10 decimal digits in common. Overall, the ratios of the execution times of *orthes*3 and *elmhes*3 are fairly close to 2, the ratio of the operation counts of the basic algorithms:

$$c_{orthes} \approx \frac{10}{3} n^3, \qquad c_{elmhes} \approx \frac{c_{orthes}}{2}.$$

The computation of Mflop/s rates shows that *orthes*3 uses the machine slightly more efficiently. This is not surprising: its overhead is more substantial (of the order of n^2 operations), but essentially consists of level-two operations performed with supervector speed. By contrast, *elmhes*3 cannot escape data movements for pivoting and scalar processing for the short vectors of the forward substitutions in $\mathbb{R}^r$. Still, its efficiency exceeds the rate of one operation per cycle (58 Mflop/s) for matrices of orders 900 and above—a commendable performance.

The estimation of near-optimum values for r can be obtained from approximate system-dependent time models [11] that need not be very accurate, as the performance of block methods is rarely sensitive in the neighborhood of the optimum.

Numerous experiments were performed on diagonalizable matrices with specified spectra, using a scheme of data generation modeled after [5], and in which the double-precision test matrices were built in quadruple precision. The effect of the reduction to Hessenberg form on small eigenvalues was assessed by using a quadruple-precision version of *hqr*. As usual, the eigenvalues of the matrices resulting from

*elmhes*3 and *orthes*3 did not differ markedly. For example, from samples of 50 matrices of order $n = 500$ specified to have the distribution of eigenvalues

$$|\lambda_i| = \alpha|\lambda_{i-1}|, \qquad i = 2, \cdots, n, \qquad |\lambda_1| = \rho^{-1}, \qquad |\lambda_n| = 1,$$

and matrices of eigenvectors with singular values uniformly distributed in $[\kappa^{-1}, 1]$, the following results were obtained. For $\kappa = 10^4$ and $\rho = 10^8$, *elmhes*3 could deliver at least two digits of accuracy in the smallest pair of eigenvalues, against three digits for *orthes*3. For $\kappa = 10^4$ and $\rho = 10^{10}$, *orthes*3 produced one digit for the smallest pair of eigenvalues, while *elmhes*3 only gave the order of magnitude. Overall, the observed worst effect of the Gauss reduction was an inflation of the order of $\sqrt{n}$ in the relative error on the eigenvalues.

6. Conclusion. For those eigenvalue problems where one can safely use the Gauss reduction to Hessenberg form, the block method presented in this paper constitutes an economical alternative to its Householder homologues. Its advantages include low overhead, absence of requirement for ancillary storage, and a substantial reduction of computation time close to 50 percent. The numerical properties of the serial algorithm are entirely preserved. Experiments with the IBM 3090 VF show that its efficiency of machine usage is satisfactory, although slightly inferior to that of a block Householder method.

REFERENCES

[1] Z. Bai and J. Demmel, *On a block implementation of Hessenberg multishift* QR *iteration*, LAPACK working note #8, TM No. 127, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1989.

[2] C. Bischof and C. Van Loan, *The WY representation for products of Householder matrices*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s2-s13.

[3] C. Bischof, J. Demmel, J. J. Dongarra, J. Ducroz, A. Greenbaum, S. Hammarling, and D. Sorensen, LAPACK *provisional contents*, LAPACK working note #5, TM No. 38, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1988.

[4] J. Demmel, J. J. Dongarra, J. Ducroz, A. Greenbaum, S. Hammarling, and D. Sorensen, *Prospectus for the development of a linear-algebra library for high-performance computers*, TM No. 97, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1987.

[5] J. Demmel and A. McKenney, *A test matrix generation suite*, TM MCS-P69-0389, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1989.

[6] J. J. Dongarra and C. B. Moler, EISPACK: *A package for solving matrix eigenvalue problems*, in Sources and Development of Mathematical Software, W. R. Cowell, ed., Prentice-Hall, Englewood Cliffs, NJ, 1984.

[7] J. J. Dongarra, S. Hammarling, and D. Sorensen, *Block reduction of matrices to condensed forms for eigenvalue computations*, LAPACK working note #2, TM No. 99, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1987.

[8] J. J. Dongarra, J. Ducroz, S. Hammarling, and R. Hanson, *An extended set of* FORTRAN *basic linear algebra subprograms*, ACM Trans. Math. Software, 14 (1988), pp. 1-17.

[9] J. J. Dongarra, J. Ducroz, I. Duff, and S. Hammarling, *A set of level-3 basic linear algebra subprograms*, TM ANL-MCS-P1-0888, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1988.

[10] A. A. Dubrulle, *On block Householder algorithms for the reduction of a matrix to Hessenberg form*, in Supercomputing 88: Vol II, Science and Applications, J. L. Martin and S. F. Lundstrom, eds., IEEE Computer Society Press, Washington D.C., 1989.

[11] A. A. Dubrulle, *Matrix computations with* FORTRAN *and the* IBM 3090 VF, IBM Scientific Center Report G320-3534, Palo Alto, CA, 1989.

[12] D. H. Gibson, D. W. Rain, and H. F. Walsh, *Engineering and scientific processing on the* IBM 3090, IBM Sys. J., 25 (1986), pp. 36-50.

[13] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[14] IBM, VS FORTRAN *Version 2 Release 3, Programming Guide*, SC26-4222.3, 1988.

[15] IBM, *Engineering and Scientific Subroutine Library Release 3, Guide Reference*, SC23-0184-3, 1988.

[16] C. LAWSON, R. HANSON, D. KINCAID, AND F. KROGH, *Basic linear algebra subprograms for* FORTRAN *usage*, ACM Trans. Math. Software, 5 (1979), pp. 308–323.

[17] A. PADEGS, B. B. MOORE, R. M. SMITH, AND W. BUCHHOLZ, *The* IBM *System/370 vector architecture: Design considerations*, IEEE Trans. Comput., 37 (1988), pp. 509–520.

[18] Y. ROBERT AND P. SGUAZZERO, *The LU decomposition algorithm and its efficient* FORTRAN *implementation on the* IBM 3090 *vector multiprocessor*, in Scientific Computing on IBM Vector Multiprocessors, R. Benzi and P. Sguazzero, eds., IBM European Center for Scientific and Engineering Computing, Rome, Italy, 1987.

[19] R. G. SCARBOROUGH AND H. G. KOLSKY, *A vectorizing* FORTRAN *compiler*, IBM J. Res. Develop., 30 (1986), pp. 163–171.

[20] R. SCHREIBER AND C. VAN LOAN, *A storage-efficient WY representation for products of Householder transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 53–57.

[21] B. T. SMITH, J. M. BOYLE, Y. IKEBE, V. C. KLEMA, AND C. B. MOLER, *Matrix Eigensystem Routines:* EISPACK *Guide*, Springer-Verlag, New York, 1976.

[22] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, New York, 1980.

[23] L. N. TREFETHEN AND R. SCHREIBER, *Average-case stability of Gaussian elimination*, Numerical Analysis Report 88-3, Mathematics Department, Massachusetts Institute of Technology, Cambridge, MA, 1988.

[24] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, U.K., 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1255-1265, November 1991
© 1991 Society for Industrial and Applied Mathematics

STABILITY ANALYSIS OF A HOUSEHOLDER-BASED ALGORITHM FOR DOWNDATING THE CHOLESKY FACTORIZATION*

ADAM W. BOJANCZYK† AND ALLAN O. STEINHARDT†

Abstract. In this paper a new algorithm for downdating the Cholesky factorization is presented and analyzed. The algorithm is based on Householder transformations. It is less expensive than algorithms based on Givens rotations if more than one row is to be removed from the Cholesky product. Rounding error analysis shows that the new method is as stable as the rotation-based methods for solving this problem.

Key words. downdating, Cholesky factorization, Householder transformation, numerical stability

AMS(MOS) subject classifications. 65F05, 65F30

1. Introduction.

The problem considered in this paper is the following. Given an upper triangular $n \times n$ matrix S and an $m \times n$ matrix X such that $S^TS - X^TX$ is positive definite, find an $n \times n$ upper triangular matrix R such that

$$R^TR = S^TS - X^TX. \tag{1.1}$$

By our assumption of positive definiteness, R exists and is unique up to the signs of its diagonal elements. In what follows we will always assume that the diagonal elements of the Cholesky factors are positive.

The problem is related to that of downdating a QR decomposition of a given matrix A. There, we want to find the QR decomposition of a matrix obtained from A after a number of rows are removed from A.

We can treat the problem as a sequence of m rank-1 downdating problems and solve it by a standard algorithm described in [3]. This approach requires $2.5n^2m + 3.5nm + O(m)$ multiplications and is numerically stable [5]. Recently, a slightly faster algorithm has been proposed in [1]. The algorithm uses what might be called half-hyperbolic and half-orthogonal rotations. It is also numerically stable and requires $2n^2 + 3n + O(1)$ multiplications per rank 1 update for a total of $2n^2m + 3nm + O(m)$ multiplications. In addition, it is well suited for parallel implementation.

It is well known that Householder-like transformations are approximately twice as cost-effective as rotationlike (Givens) transformations. To exploit this computational savings hyperbolic Householder transformations have been recently proposed to solve the downdating problem (1.1) when several rows are removed simultaneously [4]. This approach yields a cost of $n^2(m+1) + nm + O(m+n)$ multiplications, which is always less than for rotation-based methods, and in fact yields a twofold savings for large n.

In this paper we propose a modification to the hyperbolic Householder transformation which preserves the cost savings and improves the numerical properties of the transformation. This modified scheme is again, in some sense, half-hyperbolic and half-orthogonal, and its derivation is analogous to that presented in [1]. Rounding error analysis of the modified method is also presented. The analysis shows that the modified method is stable in the same sense as the methods analyzed in [1] and [5].

* Received by the editors December 21, 1987; accepted for publication (in revised form) September 6, 1990.

† School of Electrical Engineering, Cornell University, Phillips Hall, Ithaca, New York 14850-5401. The research of the first author was supported by the National Science Foundation grant MIP-8710835. The research of the second author was supported by Air Force Office of Scientific Research grant AFOSR-89-0267.

In a companion paper [2] applications of the modified hyperbolic Householder transformations are presented together with numerical tests which demonstrate that the modified transformation scheme exhibits superior numerical behavior compared to approaches based on purely hyperbolic transformations.

2. Notation. In what follows let $\| \ \|$ denote the 2-norm of vectors and matrices.

Suppose we are given an m-vector x, $m > 1$. Consider the problem of constructing the Householder transformation H such that

$$Hx = \sigma e_1 \tag{2.1}$$

where $\sigma = \|x\|$ and $e_1^T = [1, 0, \cdots, 0]$. It is well known that $H = H(u)$ is given by

$$H = I - uu^T/K \tag{2.2}$$

where

$$u^T = [x_1 + \text{sign}\,(x_1)\sigma, x_2, \cdots, x_m] \tag{2.3}$$

and

$$K = \sigma^2 + \text{abs}\,(x_1)\sigma = \frac{\|u\|^2}{2}. \tag{2.4}$$

Without loss of generality, to make notation simpler, from now on we assume that x_1 is positive, and we will omit writing sign (x_1) in the expression (2.3), which defines the vector u.

3. Derivation of the method. The downdating problem of finding an upper triangular R such that

$$S^TS - X^TX = R^TR \tag{3.1}$$

can be rewritten as an equivalent implicit updating problem of finding R such that

$$R^TR + X^TX = S^TS. \tag{3.2}$$

Assume for a moment that we know R and we want to find S (while actually the opposite is the case). This is an updating problem and as such can be solved by applying a sequence of Householder transformations $F_1, F_2, \cdots, F_n$ to the augmented matrix

$$Z = \begin{bmatrix} X \\ R \end{bmatrix} \tag{3.3}$$

in such a way that all subdiagonal elements in Z are annihilated [3]. Then $F_n \cdots F_2F_1Z$ yields the required upper triangular matrix S,

$$F_n \cdots F_2F_1 \begin{bmatrix} X \\ R \end{bmatrix} = \begin{bmatrix} S \\ 0 \end{bmatrix}. \tag{3.4}$$

Note that because the lower triangular part of R is already zero, the triangularization process has to remove only m main subdiagonals of Z. Hence, the matrix F_i operates only on rows i through $i+m$, i.e., F_i has the following form:

$$F_i = \begin{bmatrix} I_{i-1} & 0 & 0 \\ 0 & H_i & 0 \\ 0 & 0 & I_{n-i} \end{bmatrix}, \tag{3.5}$$

where I_{i-1} and I_{n-i} are identity matrices of dimension $i-1$ and $n-i$, respectively, and H_i is a Householder matrix of dimension $(m+1)\times(m+1)$,

$$H_i = I - u_i u_i^T / K_i \tag{3.6}$$

where

$$u_i^T = [u_{1,i}, u_{2,i}, \cdots, u_{m+1,i}].$$

Define recursively sequences of intermediate matrices $X^{(1)}, X^{(2)}, \cdots, X^{(n)}$ and $Z^{(1)}, Z^{(2)}, \cdots, Z^{(n)}$,

$$Z^{(0)} = Z, \qquad X^{(0)} = X,$$

$$Z^{(i)} = F_i Z^{(i-1)}, \qquad \begin{bmatrix} e_i^T S \\ X^{(i)} \end{bmatrix} = H_i \begin{bmatrix} X^{(i-1)} \\ e_i^T R \end{bmatrix} \tag{3.7}$$

where e_j is a vector of conformal dimension in 1 in position j and zeros elsewhere; hence $e_j^T R$ and $e_j^T S$ are the jth rows of R and S, respectively. It is straightforward to see that

$$Z^{(i)} = \begin{bmatrix} e_1^T S \\ \vdots \\ e_i^T S \\ X^{(i)} \\ e_{i+1}^T R \\ \vdots \\ e_n^T R \end{bmatrix}. \tag{3.8}$$

Furthermore, let vectors s_j, r_j, and $x_j^{(i)}$, denote the jth columns of S, R, and $X^{(i)}$, namely,

$$s_j = Se_j = \begin{bmatrix} s_{1,j} \\ \vdots \\ s_{j,j} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad r_j = Re_j = \begin{bmatrix} r_{1,j} \\ \vdots \\ r_{j,j} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad x_j^{(i)} = X^{(i)} e_j = \begin{bmatrix} x_{1,j}^{(i)} \\ x_{2,j}^{(i)} \\ \vdots \\ x_{m,j}^{(i)} \end{bmatrix}. \tag{3.9}$$

Then

$$(z_j^{(i)})^T = (Z^{(i)} e_j)^T = [s_{1,j}, \cdots, s_{i,j}, x_j^{(i)}, r_{i+1,j}, \cdots, r_{j,j}, 0, \cdots, 0]. \tag{3.10}$$

The components of the vector u_i which defines the transformation H_i can now be expressed in terms of the components of $x_i^{(i-1)}$ and r_{ii} as follows:

$$u_i^T = [u_{1,i}, u_{2,i}, u_{m,i}, \cdots, u_{m+1,i}] = [x_{1,i}^{(i-1)} + \sigma_i, x_{2,i}^{(i-1)}, \cdots, x_{m,i}^{(i-1)}, r_{i,i}] \tag{3.11}$$

where

$$\sigma_i = \left\| \begin{bmatrix} x_i^{(i-1)} \\ r_{i,i} \end{bmatrix} \right\|. \tag{3.12}$$

We are ready to describe a recursive procedure for computing the matrix R from the matrices X and S. The procedure computes R row by row starting from the top row.

Assume that the matrix $X^{(i-1)}$ and the $(i-1)$st row of R have been determined in step $i-1$. In step i we want to determine row i of R.

Recall from (3.5) that H_i operates only on rows i through $i+m$ of $Z^{(i-1)}$. As H_i is orthogonal and hence preserves the two norm vectors, we have that

$$\left\| \begin{bmatrix} x_i^{(i-1)} \\ r_{i,i} \end{bmatrix} \right\| = \left\| \begin{bmatrix} s_{i,i} \\ 0 \end{bmatrix} \right\| = s_{i,i}. \tag{3.13}$$

In the downdating problem the transformation H_i cannot be expressed explicitly because $r_{i,i}$ is not known. However, if we assume inductively that the vector $x_i^{(i-1)}$ has already been computed, $r_{i,i}$ can be determined from the above relation as $s_{i,i}$ is known. By solving the above equation we get the following "hyperbolic" expression for $r_{i,i}$:

$$r_{i,i} = \left[s_{i,i}^2 - \sum_{j=1}^{m} (x_{j,i}^{(i-1)})^2 \right]^{1/2}. \tag{3.14}$$

Once $r_{i,i}$ has been found, H_i can easily be constructed. In order to apply H_i, all other elements of $e_i^T R$ must be determined. Note, however, that

$$e_i^T H_i \begin{bmatrix} X^{(i-1)} \\ e_i^T R \end{bmatrix} = e_i^T S, \tag{3.15}$$

or componentwise,

$$x_{i,j}^{(i-1)} - u_{1,i} \left[\sum_{k=1}^{m} u_{k,i} x_{k,j}^{(i-1)} + u_{m+1,i} r_{i,j} \right] \Big/ K_i = s_{i,j}. \tag{3.16}$$

The relation (3.16) is a scalar linear equation in r_{ij}. Hence,

$$r_{i,j} = \left[(x_{i,j}^{(i-1)} - s_{i,j}) K_i / u_{1,i} - \sum_{k=1}^{m} u_{k,i} x_{k,j}^{(i-1)} \right] \Big/ u_{m+1,i}. \tag{3.17}$$

Recall from (3.11) that $u_{1,i}$ and $u_{m+1,i}$ must be nonzero. The relation (3.17) is again a "hyperbolic" part of the modified Householder transformation.

Since $e_i^T R$ has been determined, the transformation H_i can be applied to $X^{(i-1)}$ and the ith row of R, producing $X^{(i)}$. This step constitutes the "orthogonal" part of the modified transformation.

The description of step i of the modified hyperbolic Householder downdating method is now complete.

The ith step of the procedure is defined as follows:

(1) compute $r_{i,i} = [s_{i,i}^2 - \sum_{j=1}^{m} (x_{j,i}^{(i-1)})^2]^{1/2}$,
(2) compute $u_{1,i} = x_{1,i}^{(i-1)} - s_{i,i}$,
(3) compute $K_i = s_{i,i}(s_{i,i} + x_{1,i}^{(i-1)})$,
(4) compute $r_{i,j} = [(x_{1,j}^{(i-1)} - s_{i,j}) K_i / u_{1,i} - \sum_{k=1}^{m} u_{k,i} x_{k,j}^{(i-1)}] / u_{m+1,i}$, for $i < j \leqq n$,
(5) compute $x_j^{(i)}$ by applying H_i, for $i < j \leqq n$. □

This ith step requires $2(m+1)(n-i)+2m$ multiplications for a total cost of $n^2(m+1)+nm+O(m+n)$, the same as for the full hyperbolic Householder approach. It is interesting to note that if u^T/K is not computed prior to applying $H(u)$, the total cost is then $n^2(m+3/2)$, which is less than indicated above whenever $m > n/2$.

4. Roundoff error analysis. We present a detailed analysis of step (1) of the procedure. Analysis of all subsequent steps is analogous.

We denote computed quantities with a bar or by prefixing them with fl. The relative machine precision is denoted by ε and terms of order $O(\varepsilon^2)$ are neglected.

The first operation is the determination of $r_{1,1}$. For $\bar{r}_{1,1}$, the computed value of $r_{1,1}$, we have

$$\bar{r}_{1,1} = \mathrm{fl}\,(r_{1,1}) = (1+\gamma)\left(s_{1,1}^2(1+\beta) - (1+\alpha)\sum_{k=1}^{m}(x_{k,1}^{(0)})^2\right)^{1/2}, \tag{4.1}$$

where α, β, γ are perturbations and

$$|\alpha|, |\beta| \leqq 2^{-t}, \qquad |\gamma| \leqq 1.5\varepsilon.$$

We can treat $\bar{r}_{1,1}$ as the true data and define a Householder transformation $\tilde{H}_1 \equiv H(\tilde{u}_1)$ by the following relation:

$$\tilde{H}_1 \begin{bmatrix} x_1^{(0)} \\ \bar{r}_{1,1} \end{bmatrix} = \begin{bmatrix} N_1 \\ 0 \end{bmatrix} \tag{4.2}$$

where

$$N_1 = \left\| \begin{bmatrix} x_1^{(0)} \\ \bar{r}_{1,1} \end{bmatrix} \right\|. \tag{4.3}$$

The vector $\tilde{u}_1$ and the constant $\tilde{K}_1$ that define $\tilde{H}_1$ are given, respectively, by

$$\tilde{u}_1^T = [x_{1,1}^{(0)} + N_1, x_{2,1}^{(0)}, \cdots, x_{m,1}^{(0)}, \bar{r}_{1,1}] \tag{4.4}$$

and

$$\tilde{K}_1 = N_1^2 + x_{11}^{(0)} N_1. \tag{4.5}$$

Note that, despite the fact that $\bar{r}_{1,1}$ may substantially differ from $r_{1,1}$ the transformation $\tilde{H}_1$ is well defined and is truly orthogonal.

LEMMA 4.1.

$$\tilde{H}_1 \begin{bmatrix} x_{1,1}^{(0)} + \Delta x_{1,1}^{(0)} \\ \vdots \\ x_{m,1}^{(0)} + \Delta x_{m,1}^{(0)} \\ \bar{r}_{1,1} + \Delta \bar{r}_{1,1} \end{bmatrix} = \begin{bmatrix} s_{1,1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.6}$$

where

$$\left\| \begin{bmatrix} \Delta x_{1,1}^{(0)} \\ \vdots \\ \Delta x_{m,1}^{(0)} \\ \Delta \bar{r}_{1,1} \end{bmatrix} \right\| \leqq 2.5\varepsilon \left\| \begin{bmatrix} x_{1,1}^{(0)} \\ \vdots \\ x_{m,1}^{(0)} \\ \bar{r}_{1,1} \end{bmatrix} \right\|.$$

Proof. From (4.1) and (4.3),

$$N_1^2 = \|x_1^{(0)}\|^2 + \bar{r}_{1,1}^2 = \sum_{k=1}^{m}(x_{k,1}^{(0)})^2 + \bar{r}_{1,1}^2 = s_{1,1}^2(1+\beta+2\gamma) - (\alpha+2\gamma)\sum_{k=1}^{m}(x_{k,1}^{(0)})^2,$$

so

$$|N_1^2 - s_{1,1}^2| \leqq s_{1,1}^2(|\alpha| + 2|\gamma| + |\beta|) = s_{1,1}^2|\delta|,$$

where $|\delta| \leqq 5\varepsilon$. Thus,

$$N_1 = s_{1,1}(1+\rho) \quad \text{where } |\rho| \leqq 2.5\varepsilon. \tag{4.7}$$

The relation (4.2) now becomes

$$\tilde{H}_1 \begin{bmatrix} x_1^{(0)} \\ \bar{r}_{1,1} \end{bmatrix} = \begin{bmatrix} s_{1,1}(1-\rho) \\ 0 \end{bmatrix}. \tag{4.8}$$

As $\tilde{H}_1$ is orthogonal, the relation (4.8) is equivalent to

$$\tilde{H}_1 \begin{bmatrix} x_{1,1}^{(0)} + \Delta x_{1,1}^{(0)} \\ \vdots \\ x_{m,1}^{(0)} + \Delta x_{m,1}^{(0)} \\ \bar{r}_{1,1} + \Delta \bar{r}_{1,1} \end{bmatrix} = \begin{bmatrix} s_{1,1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.9}$$

where

$$\begin{bmatrix} \Delta x_{1,1}^{(0)} \\ \vdots \\ \Delta x_{m,l}^{(0)} \\ \Delta \bar{r}_{1,1} \end{bmatrix} := \tilde{H}_1^T \begin{bmatrix} \rho s_{1,1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \text{and} \quad \left\| \begin{bmatrix} \Delta x_{1,1}^{(0)} \\ \vdots \\ \Delta x_{m,1}^{(0)} \\ \Delta \bar{r}_{1,1} \end{bmatrix} \right\| \leqq 2.5\varepsilon \left\| \begin{bmatrix} x_{1,1}^{(0)} \\ \vdots \\ x_{m,1}^{(0)} \\ \bar{r}_{1,1} \end{bmatrix} \right\|,$$

which completes the proof. □

In words, (4.9) states that there exists an orthogonal matrix $\tilde{H}_1$, which transforms an only "slightly" perturbed first column of $\bar{Z}^{(0)}$ into the desired first column of $\bar{Z}^{(1)}$ (which at this point happens to be equal to the first column of $Z^{(1)}$). We next show an analogous property for the other columns.

Consider $\bar{u}_1 = \mathrm{fl}(\tilde{u}_1)$. The only component of $\tilde{u}_1$ actually computed is $\bar{u}_{1,1}$. For it we have from (4.7)

$$\bar{u}_{1,1} = (1+\bar{\mu})(x_{1,1}^{(0)} + s_{1,1}) = (1+\mu)(x_{1,1}^{(0)} + N_1) = (1+\mu)\tilde{u}_{1,1} \tag{4.10}$$

where $|\mu| \leqq 3.5\varepsilon$.

For $\bar{K}_1$ we have from (4.7)

$$\bar{K}_1 = \mathrm{fl}\,(s_{1,1}(s_{1,1} + x_{1,1}^{(0)})) = (N_1^2 + N_1 x_{1,1}^{(0)})(1+\nu) = (1+\nu)\tilde{K}_1 \tag{4.11}$$

where $|\nu| \leqq 7\varepsilon$.

Given $\bar{u}_1$ and $\bar{K}_1$ the relation (3.17) determines the first row of R. For the computed value $\bar{r}_{1,j}$ we have

$$\bar{r}_{1j} = \left[(1+\xi_j)(x_{1,j}^{(0)} - s_{1,j})\tilde{K}_1 / \tilde{u}_{1,1} - \tilde{u}_{1,1} x_{1,j}(1+\xi_{1,j}) - \sum_{k=2}^{m} x_{k,1}^{(0)} x_{k,j}^{(0)} (1+\xi_{k,j}) \right] \Big/ \bar{r}_{1,1} \tag{4.12}$$

with the following bounds on perturbations ξ_j, $\xi_{i,j}$:

$$|\xi_j| \leqq 16\varepsilon, \quad |\xi_{1,j}| \leqq 7\varepsilon, \quad |\xi_{k,j}| \leqq (m-k+5)\varepsilon.$$

Knowing $\bar{r}_{i,j}$, the transformation $H(\tilde{u}_1)$ can be applied giving the computed vector $\bar{x}_j^{(1)}$,

$$\begin{bmatrix} s_{1,j} + \Delta s_{1,j} \\ \bar{x}_j^{(1)} \end{bmatrix} = \mathrm{fl}\left(\tilde{H}_1 \begin{bmatrix} x_j^{(0)} \\ \bar{r}_{1,j} \end{bmatrix} \right) = \mathrm{fl}\left(\begin{bmatrix} x_j^{(0)} \\ \bar{r}_{1,j} \end{bmatrix} - \bar{u}_1 \left(\bar{u}_1^T \begin{bmatrix} x_j^{(0)} \\ \bar{r}_{1,j} \end{bmatrix} \Big/ \bar{K}_1 \right) \right).$$

The order of operations here is as follows. First, the inner product is computed. Next, $\bar{u}_1$ is multiplied by a scalar, and finally the difference of two vectors is formed.

LEMMA 4.2.

$$\tilde{H}_1 \begin{bmatrix} x_j^{(0)} + \Delta x_j^{(0)} \\ \bar{r}_{1,j} + \Delta \bar{r}_{1,j} \end{bmatrix} = \begin{bmatrix} s_{1,j} \\ = \bar{x}_j^{(1)} \end{bmatrix}$$

where

$$\left\| \tilde{H}_1 \begin{bmatrix} x_j^{(0)} \\ \bar{r}_{1,j} \end{bmatrix} - \begin{bmatrix} s_{1,j} \\ \bar{x}_j^{(1)} \end{bmatrix} \right\| \leqq (m+20)\varepsilon \left\| \begin{bmatrix} x_j^{(0)} \\ \bar{r}_{1,j} \end{bmatrix} \right\|.$$

Proof. See Appendix A for the proof. □

By induction it is easy to show that Lemmas 4.1 and 4.2 hold for each step of the algorithm, i.e., we have the following result.

LEMMA 4.3. *For* $i = 1, 2, \cdots, n$, *and* $j = i, i+1, \cdots, n$,

$$\tilde{H}_i \begin{bmatrix} \bar{x}_j^{(i-1)} + \Delta\bar{x}_j^{(i-1)} \\ \bar{r}_{i,j} + \Delta\bar{r}_{i,j} \end{bmatrix} = \begin{bmatrix} s_{i,j} \\ \bar{x}_j^{(i)} \end{bmatrix}, \tag{4.13}$$

where

$$\left\| \begin{bmatrix} \Delta\bar{x}_j^{(i-1)} \\ \Delta\bar{r}_{i,j} \end{bmatrix} \right\| \leqq (m+20)\varepsilon \left\| \begin{bmatrix} \bar{x}_j^{(i-1)} \\ \bar{r}_{i,j} \end{bmatrix} \right\|. \tag{4.14}$$

□

Lemma 4.3 expresses the key property of the modified hyperbolic Householder downdating algorithm, namely, that there exists an orthogonal matrix $\tilde{H}_i$ that transforms a "slightly" perturbed jth column of $\bar{Z}^{(i-1)}$ into the desired ith column of $\bar{Z}^{(i)}$. Now we show how the stability of the algorithm follows from this key property.

Remark. Note that at this point $\bar{Z}^{(j)}$ may differ substantially from the true $Z^{(j)}$, which would have been obtained had exact arithmetic been used.

From Lemma 4.3, for step 2 we have

$$\tilde{H}_2 \begin{bmatrix} \bar{x}_j^{(1)} + \Delta\bar{x}_j^{(1)} \\ \bar{r}_{2,j} + \Delta\bar{r}_{2,j} \end{bmatrix} = \begin{bmatrix} s_{2,j} \\ \bar{x}_j^{(2)} \end{bmatrix}. \tag{4.15}$$

Let $[(\delta^{(1)}x_j^{(1)})^T, \delta^{(1)}r_{1,j}]^T$ be a vector such that

$$\tilde{H}_1 \begin{bmatrix} \delta^{(1)}x_j^{(1)} \\ \delta^{(1)}r_{1,j} \end{bmatrix} = \begin{bmatrix} 0 \\ \Delta\bar{x}_j^{(1)} \end{bmatrix}. \tag{4.16}$$

By combining the relation (4.13) with $i = 1$ and (4.15) we obtain

$$\begin{bmatrix} 1 & 0 \\ 0 & \tilde{H}_2 \end{bmatrix} \begin{bmatrix} \tilde{H}_1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_j^{(0)} + \Delta x_j^{(0)} + \delta^{(1)}x_j^{(1)} \\ \bar{r}_{1,j} + \Delta\bar{r}_{1,j} + \delta^{(1)}\bar{r}_{1,j} \\ \bar{r}_{2,j} + \Delta\bar{r}_{2,j} \end{bmatrix} = \begin{bmatrix} s_{1,j} \\ s_{2,j} \\ \bar{x}_j^{(2)} \end{bmatrix} \tag{4.17}$$

where

$$\left\| \begin{bmatrix} \Delta x_j^{(0)} \\ \Delta\bar{r}_{1,j} \end{bmatrix} \right\| \leqq (m+20)\varepsilon \left\| \begin{bmatrix} x_j^{(0)} \\ \bar{r}_{1,j} \end{bmatrix} \right\|, \quad \left\| \begin{bmatrix} \sigma^{(1)}x_j^{(1)} \\ \delta^{(1)}\bar{r}_{1,j} \\ \Delta\bar{r}_{2,j} \end{bmatrix} \right\| \leqq (m+20)\varepsilon \left\| \begin{bmatrix} \bar{x}_j^{(1)} \\ \bar{r}_{2,j} \end{bmatrix} \right\|.$$

By induction we finally have the following result.

LEMMA 4.4. *For* $j = 1, \cdots, n$

$$\tilde{F}_j \cdots \tilde{F}_1 \begin{bmatrix} x_j^{(0)} + \Delta x_j \\ \bar{R}e_j + \Delta r_j \end{bmatrix} = \begin{bmatrix} Se_j \\ 0 \end{bmatrix} \tag{4.18}$$

where F_i *is defined in* (3.5), $\tilde{F}_i$ *is the computed version of* F_i, *and*

$$\left\| \begin{bmatrix} \Delta x_j \\ \Delta r_j \end{bmatrix} \right\| \leqq j(m+20)\varepsilon \|Se_j\|. \tag{4.19}$$

Proof. See Appendix B for the proof. □

The perturbations in (4.18) are given column by column. For the matrices we have

$$\tilde{H} \begin{bmatrix} X + \Delta X \\ \bar{R} + \Delta R \end{bmatrix} = \begin{bmatrix} S \\ 0 \end{bmatrix} \tag{4.20}$$

where ΔR is an upper triangular matrix of perturbations.

The relation (4.20) states that the computed Cholesky factor $\bar{R}$ is very near the factor obtained by downdating with slightly perturbed data. This means that the algorithm is numerically stable in the sense discussed in [1] and [5].

This kind of stability, called mixed stability, is a more general concept than the classical backward stability where it is postulated that the computed result must be exact for slightly perturbed data. The mixed stability is a weaker result than the backward stability. (For a discussion of different concepts of stability in numerical linear algebra the reader may wish to refer to [6] and [7].) Here we accept the result of mixed stability because, as shown in [5], the perturbation ΔR is not important, since the major contributions to the error $\bar{R}-R$ come from the perturbation ΔX of the original data. Thus, the result guaranteed by the mixed stability analysis is almost as satisfactory as the one guaranteed by the backward stability analysis since in both cases the errors have comparable bounds.

Finally, as the downdating problem may be very ill conditioned, we cannot expect $\bar{R}$ to be close to the true downdated factor R. Moreover, there may be no matrix $\hat{X}$ close to X that leads exactly to the computed factor $\bar{R}$.

On the other hand, (4.21) guarantees that the relative error $\bar{R}^T\bar{R}$ is small. We have conducted numerical experiments with the modified scheme and found that it behaves better than the conventional hyperbolic Householder scheme discussed in [4]. We conjecture based on these experiments that hyperbolic Householder does not guarantee (4.21) with bounds comparable to those in (4.20).

Appendix A.

Proof of Lemma 4.2. We derive a componentwise bound on the difference vector h_j:

$$h_j = \tilde{H}_1 \begin{bmatrix} x_j^{(0)} \\ \bar{r}_{1,j} \end{bmatrix} - \begin{bmatrix} s_{1,j} \\ \bar{x}_j^{(1)} \end{bmatrix}. \tag{A1}$$

From (4.12) for the first component, we have

$$\begin{aligned} h_{1,j} &= \left[\tilde{H}_1 \begin{bmatrix} x_j^{(0)} \\ \bar{r}_{1,j} \end{bmatrix} \right]_1 - s_{1,j} = x_{1,j}^{(0)} - \frac{\tilde{u}_{1,1}}{\tilde{K}_1} \left(\sum_1^m \tilde{u}_{k,1} x_{k,j}^{(0)} + \tilde{u}_{m+1,1} \bar{r}_{1,j} \right) - s_{1,j} \\ &= (x_{1,j}^{(0)} - s_{1,j}) - \frac{\tilde{u}_{1,1}}{\tilde{K}_1} \left(\sum_{k=1}^m \tilde{u}_{k,1} x_{k,j}^{(0)} + (1+\xi_j)(x_{1,j}^{(0)} - s_{1,j}) \frac{\tilde{K}_1}{\tilde{u}_{1,1}} \right. \\ &\qquad \left. - \sum_{k=1}^m \tilde{u}_{k,1} x_{k,j}^{(0)} (1+\xi_{k,j}) \right) \\ &= \xi_j (x_{1,j}^{(0)} - s_{1,j}) + \frac{\tilde{u}_{1,1}}{\tilde{K}_1} \sum_{k=1}^m \tilde{u}_{k,1} x_{k,j}^{(0)} \xi_{k,j}. \end{aligned} \tag{A2}$$

From (4.12) we can also derive that

$$x_{1,j}^{(0)} - s_{1,j} = \frac{\tilde{u}_{1,1}}{\tilde{K}_1} \sum_{k=1}^{m+1} \tilde{u}_{k,1} x_{k,j}^{(0)} (1+\eta_{k,j})$$

where $|\eta_{k,j}| \leqq (m-k+2)\varepsilon$ and $\bar{x}_{m+1,j} = \bar{r}_{1,}j$. Thus

$$h_{1,j} = \left[\tilde{H}_1 \begin{bmatrix} x_j^{(0)} \\ \bar{r}_{1,j} \end{bmatrix} - s_{1,j} \right]_1 = \frac{\tilde{u}_{1,1}}{\tilde{K}_1} \sum_{k=1}^{m+1} \tilde{u}_{k,1} x_{k,j}^{(0)} \eta_{k,j}$$

where $|\eta_{k,j}| \leqq (m-k+2)\varepsilon$.

Similarly, for $1 < i \leqq m+1$,

$$
\begin{aligned}
h_{i,j} &= \bar{x}^{(1)}_{i-1,j} - \left[\tilde{H}_1 \begin{bmatrix} x^{(0)}_j \\ \bar{r}_{1,j} \end{bmatrix}\right] \\
&= (1+\sigma)\left[x^{(0)}_{i,j} - \frac{\bar{u}_{i,1}}{\tilde{K}_1}\left(\sum_{k=1}^{m+1} \bar{u}_{k,1} x^{(0)}_{k,j}(1+\eta_{k,i,j})\right)\right] - \left[x^{(0)}_{i,j} - \frac{\tilde{u}_{i,1}}{\tilde{K}_1}\sum_{k=1}^{m+1} \tilde{u}_{k,1} x^{(0)}_{k,j}\right] \qquad \text{(A3)} \\
&= \sigma_i x^{(0)}_{i,j} - \frac{\tilde{u}_{i,1}}{\tilde{K}_1}\sum_{k=1}^{m+1} u_{k,1} x^{(0)}_{k,j}\bar{\eta}_{k,i,j}
\end{aligned}
$$

where $|\sigma_i| \leqq \varepsilon$, $|\bar{\eta}_{k,i,j}| \leqq (20+m)\varepsilon$. The relations (A2) and (A3) together yield

$$
\begin{aligned}
h_j = \tilde{H}_1 \begin{bmatrix} x^{(0)}_j \\ \bar{r}_{1,j} \end{bmatrix} - \begin{bmatrix} s_{1,j} \\ \bar{x}^{(1)}_j \end{bmatrix} &= \operatorname{diag}(0, \operatorname{diag}(\sigma_i))[0, (x^{(0)}_j)^T]^T \qquad \text{(A4)} \\
&\quad + \operatorname{diag}\left(\sum_{k=1}^{m+1} \bar{u}_{k,1} x^{(0)}_{k,j}\bar{\eta}_{k,i,j}\right)\frac{\tilde{u}_1}{\tilde{K}_1}.
\end{aligned}
$$

Hence,

$$
\begin{aligned}
\|h_j\| &\leqq \max|\sigma_i| \left\| \begin{bmatrix} x^{(0)}_j \\ \bar{r}_{1j} \end{bmatrix} \right\| + \frac{\|\tilde{u}_1\|}{\tilde{K}_1} \max\left(\sum_{k=1}^{m+1} |\tilde{u}_{k,1}| \|x^{(0)}_{k,j}\| \eta_{k,i,j}|\right) \\
&\leqq \varepsilon \left\| \begin{bmatrix} x^{(0)}_j \\ \bar{r}_{1j} \end{bmatrix} \right\| + \max|\eta_{k,i,j}| \frac{\|\tilde{u}_1\|^2}{\tilde{K}_1} \left\| \begin{bmatrix} x^{(0)}_j \\ \bar{r}_{1,j} \end{bmatrix} \right\|,
\end{aligned}
$$

and finally

$$
\|h_j\| \leqq (m+20)\varepsilon \left\| \begin{bmatrix} x^{(0)}_j \\ \bar{r}_{1,j} \end{bmatrix} \right\|
$$

where we used (2.4), i.e., the fact that $\|\tilde{u}_1\|^2 = 2\tilde{K}_1$. Because $\tilde{H}_1$ is orthogonal, the rounding errors in (A4) can be mapped backwards without changing their norm onto $[(x^{(0)}_j)^T, \bar{r}_{1,j}]^T$. This now gives

$$
\tilde{H}_1 \begin{bmatrix} x^{(0)}_j + \Delta x^{(0)}_j \\ \bar{r}_{1,j} + \Delta \bar{r}_{1,j} \end{bmatrix} = \begin{bmatrix} s_{1,j} \\ \bar{x}^{(1)}_j \end{bmatrix}, \qquad \text{(A5)}
$$

which completes the proof. □

Appendix B.

Proof of Lemma 4.4. For $j = 2, \cdots, n$ and $k = 1, \cdots, j$, define perturbations $\delta^{(k-1)}x^{(1)}_j$ of $x^{(0)}_j \equiv Xe_j$, and $\delta^{(k-1)}r_j$ of Re_j by the following relations:

$$
\delta^{(k-1)}r_j \equiv \begin{bmatrix} \delta^{(k-1)}r_{1j} \\ \delta^{(k-2)}r_{2j} \\ \vdots \\ \delta^{(1)}r_{k-1j} \\ \Delta\bar{r}_{kj} \\ 0_{n-k} \end{bmatrix}, \qquad \tilde{F}_{k-1}\cdots\tilde{F}_1 \begin{bmatrix} \delta^{(k-1)}x^{(1)}_j \\ \delta^{(k-1)}r_j \\ 0_{n-k} \end{bmatrix} = \begin{bmatrix} 0_{k-1} \\ \Delta\bar{x}^{(k-1)}_j \\ \Delta\bar{r}_{kj} \\ 0_{n-k} \end{bmatrix} \qquad \text{(B1)}
$$

where $\Delta\bar{x}^{(k-1)}_j$ and $\Delta\bar{r}_{kj}$ are the same as in (4.13), and 0_p denotes a p-dimensional zero vector. From the orthogonality of the matrices $\tilde{F}_1, \cdots, \tilde{F}_k$, we have

$$
\left\| \begin{bmatrix} \delta^{(k-1)}x^{(1)}_j \\ \delta^{(k-1)}r_j \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} \Delta\bar{x}^{(k-1)}_j \\ \Delta\bar{r}_{kj} \end{bmatrix} \right\|_2. \qquad \text{(B2)}
$$

Note that for $k=2$, $i=1$, (B1) is equivalent to (4.16). Define

$$\Delta^{(k-1)}x_j \equiv \sum_{i=0}^{k-1} \delta^{(i)} x_j^{(1)}, \qquad \Delta^{(k-1)}r_j \equiv \sum_{i=0}^{k-1} \delta^{(i)} r_j. \tag{B3}$$

Then from (4.13), (B1), and (B3) it is easy to see that for $k=1,\cdots,j-1$,

$$\tilde{F}_k \cdots \tilde{F}_1 \begin{bmatrix} Xe_j+\Delta^{(j-1)}x_j \\ Re_j+\Delta^{(j+1)}r_j \end{bmatrix} = \begin{bmatrix} s_{1j} \\ s_{2j} \\ \vdots \\ s_{kj} \\ \bar{x}_j^{(k)} \\ 0 \end{bmatrix}$$

and

$$\tilde{F}_j \cdots \tilde{F}_1 \begin{bmatrix} Xe_j+\Delta^{(j-1)}x_j \\ Re_j+\Delta^{(j-1)}r_j \end{bmatrix} = \begin{bmatrix} Se_j \\ 0 \end{bmatrix}. \tag{B4}$$

We want to derive a bound on the perturbation vector $[(\Delta^{(j-1)}x_j)^T, (Re_j\Delta^{(j-1)}r_j)^T]^T$ in (B4). From (4.13) and (4.14) we can easily derive the following bounds:

$$\left\| \begin{bmatrix} \bar{x}_j^{(k-1)} \\ \bar{r}_{kj} \end{bmatrix} \right\|_2 \leqq \frac{1}{1-(m+20)\varepsilon} \left\| \begin{bmatrix} s_{kj} \\ \bar{x}_j^{(k)} \end{bmatrix} \right\|_2 \tag{B5}$$

and

$$\left\| \begin{bmatrix} \bar{x}_j^{(j-1)} \\ \bar{r}_{jj} \end{bmatrix} \right\|_2 \leqq \frac{1}{1-(m+20)\varepsilon} \left\| \begin{bmatrix} s_{jj} \\ 0 \end{bmatrix} \right\|_2. \tag{B6}$$

From (4.9), (4.14), (B5), and (B6) it follows that

$$\begin{aligned} \left\| \begin{bmatrix} \Delta\bar{x}_j^{(k-1)} \\ \Delta\bar{r}_{kj} \end{bmatrix} \right\|_2^2 &\leqq \left(\frac{(m+20)\varepsilon}{1-(m+20)\varepsilon} \right)^2 \left\| \begin{bmatrix} s_{kj} \\ \bar{x}_j^{(k)} \end{bmatrix} \right\|_2^2 \\ &\leqq \left(\frac{(m+20)\varepsilon}{1-(m+20)\varepsilon} \right)^2 \left(s_{kj}^2 + \left\| \begin{bmatrix} \bar{x}_j^{(k)} \\ \bar{r}_{k+1j} \end{bmatrix} \right\|_2^2 \right) \\ &\leqq \left(\frac{(m+20)\varepsilon}{(1-(m+20)\varepsilon)^{(j-k+1)}} \right)^2 \|Se_j\|_2^2. \end{aligned} \tag{B7}$$

After discarding higher-order terms in ε—as is customary—we can rewrite (B7) as

$$\left\| \begin{bmatrix} \Delta\bar{x}_j^{(k-1)} \\ \Delta\bar{r}_{kj} \end{bmatrix} \right\|_2 \leqq (m+20)\varepsilon \|Se_j\|_2. \tag{B8}$$

From (B2), (B3), and (B8) we finally have

$$\left\| \begin{bmatrix} \Delta^{(j-1)}x_j \\ \Delta^{(j-1)}r_j \end{bmatrix} \right\|_2^2 \leqq j(m+20)\varepsilon \|Se_j\|_2. \tag{B9}$$

Denoting

$$\begin{bmatrix} \Delta x_j \\ \Delta r_j \end{bmatrix} := \begin{bmatrix} \Delta^{(j-1)}x_j \\ \Delta^{(j-1)}r_j \end{bmatrix},$$

we get the relation (4.19). □

REFERENCES

[1] A. W. BOJANCZYK, R. P. BRENT, P. VAN DOOREN, AND F. R. DE HOOG, *A note on downdating the Cholesky factorization*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 210–221.

[2] A. W. BOJANCZYK AND A. O. STEINHARDT, *Stabilized hyperbolic Householder transformations*, IEEE Trans. Acoust. Speech Signal Process, 37 (1989), pp. 1286–1288.

[3] P. E. GILL, G. H. GOLUB, W. MURRAY, AND M. A. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505–535.

[4] C. M. RADER AND A. O. STEINHARDT, *Hyperbolic Householder transformations*, IEEE Trans. Acoust. Speech Signal Process., 34 (1986), pp. 1584–1602.

[5] G. W. STEWART, *The effect of rounding errors on an algorithm for downdating a Cholesky factorization*, J. Inst. Math. Appl., 23 (1979), pp. 203–213.

[6] M. JANKOWSKI AND H. WOZNIAKOWSKI, *Iterative refinement implies numerical stability*, BIT, 17 (1977), pp. 303–311.

[7] J. R. BUNCH, *The weak and strong stability of algorithms in numerical linear algebra*, Linear Algebra Appl., 88/89 (1987), pp. 49–66.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1266-1283, November 1991
© 1991 Society for Industrial and Applied Mathematics
002

SENSITIVITY ANALYSIS FOR REGULARIZED ESTIMATION IN SOME SYSTEM IDENTIFICATION PROBLEMS*

FINBARR O'SULLIVAN†

Abstract. Estimation of functional parameters in differential equations arises in a number of contexts including reservoir engineering and seismology. Various techniques for solving these ill-posed inverse problems have been proposed. Sensitivity analysis provides second-order information associated with a given solution that may be used for adaptation, interval estimation, and asymptotic resolution analysis. Computational aspects of sensitivity analysis are nontrivial. An approach to sensitivity calculation that may be used with regularization estimators is described. In a parallel computing environment almost linear speedup can be achieved. The techniques are illustrated with one-dimensional model problems.

Key words. constrained nonlinear regularization, cross validation, ill-posed inverse problems, interval estimates, parallelism, resolution analysis

AMS(MOS) subject classifications. primary 62-G05; secondary 62-J05, 41-A35, 41-A25, 47-A53, 45-L10, 45-M05

1. Introduction.

Estimation problems associated with the identification of functional parameters in partial differential equations arise in several fields including diffraction tomography, reservoir engineering and seismology. Several such problems are discussed in [7] and [9], for example. A generalized framework for a large class of these problems is outlined in [11] and [17]: U, F, and Θ are function spaces and there is a *system* operator A such that for each f in F and θ in a subset C of Θ, there is a locally unique u in U satisfying

$$A(u, \theta) = f. \tag{1.1}$$

The *inverse system identification* problem is to estimate θ from measured information about u (f is taken to be known). Let X be a linear mapping on U, $X: U \to R^n$; we are given a vector of measurements $\mathbf{z}$

$$\mathbf{z} = Xu + \boldsymbol{\varepsilon} \tag{1.2}$$

where $\boldsymbol{\varepsilon}$ is a vector of measurement or possible modeling errors (1.1). The estimation problem for θ can be put in the form of a nonlinear nonparametric regression. Equation (1.1) gives an expression for u, which in general is nonlinear in f and θ. Fixing f, the measurements in (1.2) can thus be regarded as nonlinear functionals of θ. The data are written as

$$z_i = \eta(\theta; x_i) + \varepsilon_i, \qquad i = 1, 2, 3, \cdots, n \tag{1.3}$$

where the design points x_i lie in some set I. The ε_i are the measurement/modeling errors. The situation is illustrated by the following simple model problems.

1.1. Model problems.

(i) Parabolic diffusion equation. The temperature at time t and position x on a one-dimensional rod with unknown heat conductance characteristics is denoted $u(x, t)$.

* Received by the editors March 1, 1987; accepted for publication (in revised form) April 22, 1990. This research was supported in part by the National Science Foundation under grant MCS-840-3239 and by the Department of Energy under grant DE-FG06-88ER25006. Some of the work was done during a visit to the Institute for Mathematics and Its Applications, University of Minnesota, Minneapolis, Minnesota 55455.

† Department of Statistics, University of Washington, Seattle, Washington 98195.

An initial heat distribution is placed on the rod and by observing what happens to the distribution of heat in time one hopes to gain information about heat conductance θ of the rod. The system is governed by the differential equation

$$\frac{\partial u}{\partial t}-\frac{\partial}{\partial x}\left\{\theta(x)\frac{\partial u}{\partial x}\right\}=0, \qquad 0\leqq x\leqq 1, \qquad 0\leqq t\leqq T \tag{1.4}$$

with initial/boundary conditions

$$u(x,0)=u_0(x),$$

$$\frac{\partial u(x,t)}{\partial x}=0 \quad \text{for } x=0,1,$$

u_0 is the initial heat distribution and the boundary condition says that no heat escapes from the system. Three-dimensional versions of this problem arise in reservoir engineering, see [5] for example.

(ii) Hyperbolic wave equation. Here $u(x, t)$ represents a local displacement at time t and position x along a rod with unknown sound transmission characteristics. An explosion set off at one end of the rod causes the rod to vibrate, and by observing the displacement pattern at various positions along the rod (seismograms), one hopes to get information about the sound transmission characteristics of the material of the rod. The system is modeled by an equation of the form

$$\frac{\partial^2 u}{\partial t^2}-\frac{\partial}{\partial x}\left\{\theta(x)\frac{\partial u}{\partial x}\right\}=0, \qquad 0\leqq x\leqq 1, \qquad 0\leqq t\leqq T \tag{1.5}$$

with initial/boundary conditions

$$u(x,0)=0, \qquad \frac{\partial u(x,0)}{\partial t}=0,$$

$$\frac{\partial u(0,t)}{\partial x}=f(t), \qquad u(1,t)=0.$$

The initial conditions say that the rod is initially at rest. The explosion is set off at $x=0$ and is described by $f(t)$; the right-hand boundary is held fixed. We are interested in the estimation of θ. The problem of reflection seismology is of this type [7]. Seismologists are interested in two- and three-dimensional versions of this problem.

Parameter values and initial conditions for the model problems are given in Figs. 1.1 and 1.2. These model problems are used to illustrate various points in the paper. The measured data in both problems are time series of the solution characteristics (or state variable) observed at a finite number of distinct locations along the rod. It is clear to see how the nonlinear nonparametric regression framework in (1.3) applies to these problems. The measurements are

$$z_{ij}=z(x_i,t_j)=u(x_i,t_j;\theta)+\varepsilon_{ij}, \qquad i=1,2,\cdots,n, \qquad j=1,2,\cdots,m. \tag{1.6}$$

Here we explicitly indicate the dependence of u on the underlying profile θ; $u(x_i, t; \theta)$ are nonlinear functionals of the functional parameter θ. Later on in the paper we do numerical studies in which data are generated according to (1.6) with the ε_{ij} being independent Gaussian random variables.

1.2. Solution methods and regularization. Since θ is high-dimensional and the data are discrete and noisy the estimation problem is ill posed and some form of regularization is required to obtain reliable solutions. Various solution methods have been

Parabolic Diffusion Equation

$$\frac{\partial u}{\partial t} - \frac{\partial}{\partial x}\left\{\theta(x)\frac{\partial u}{\partial x}\right\} = 0$$

$$0 \leqq x \leqq 1, \qquad 0 \leqq t \leqq T$$

with initial/boundary conditions

$$u(x, 0) = u_0(x)$$

$$\frac{\partial u(x, t)}{\partial x} = 0 \quad \text{for } x = 0, 1.$$

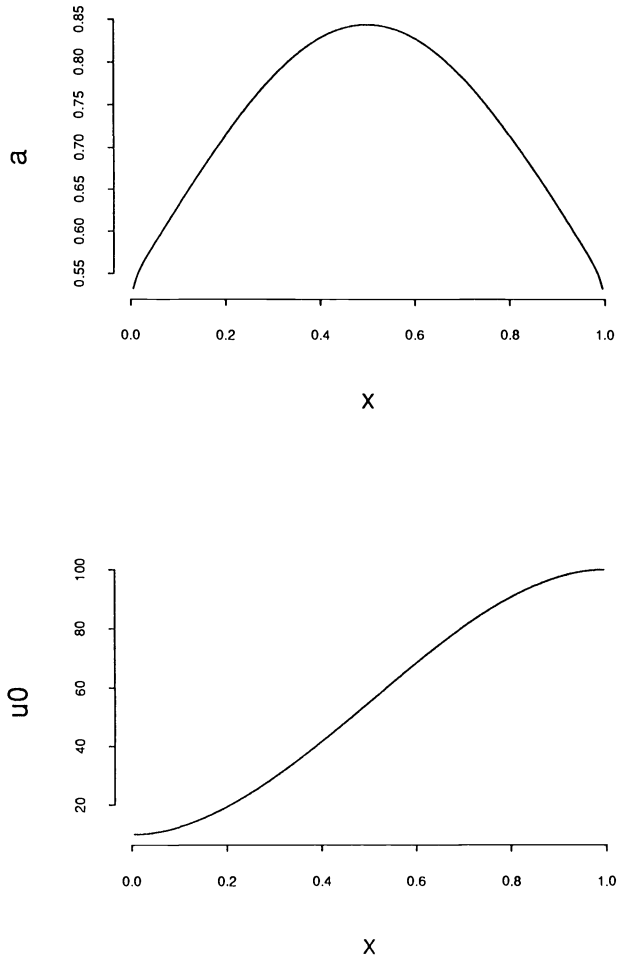


FIG. 1.1. *Parameters for the parabolic problem.*

proposed. The approach which we consider here is constrained nonlinear least squares regularization (see [3], [11], for example). The estimator is defined as

$$\hat{\theta}_\lambda = \arg\min_{\theta \in C} \left\{ \sum_{i=1}^{n} [z_i - \eta(\theta; x_i)]^2 + \lambda J(\theta) \right\}. \tag{1.7}$$

In general C is some subset of the parameter space Θ that might represent constraints such as positivity, monotonicity, or convexity. $\lambda > 0$ is the regularization parameter and J is a penalty functional (see [24] for a discussion of possible penalty functions). Quadratic J functions based on L_2 norms of derivatives are familiar in the statistical spline smoothing literature and Gaussian state-space representations have been used to motivate such functions from a Bayesian viewpoint [10]. Nonquadratic J functions involving L_1 norms of derivatives for example have also been proposed [19]; a Bayesian motivation using non-Gaussian state-space representations is available for these J functions also. In some circumstances it is appropriate to replace the residual sum of squares term in (1.7) by a more robust alternative.

lyperbolic Wave Equation

$$\frac{\partial^2 u}{t^2} - \frac{\partial}{\partial x}\left\{\theta(x)\frac{\partial u}{\partial x}\right\} = 0$$

$$\leqq x \leqq 1, \qquad 0 \leqq t \leqq T$$

ith initial/boundary conditions

$$(x, 0) = 0; \qquad \frac{\partial u(x, 0)}{\partial t} = 0$$

$$\frac{u(0, t)}{\partial x} = f(t); \qquad u(1, t) = 0.$$

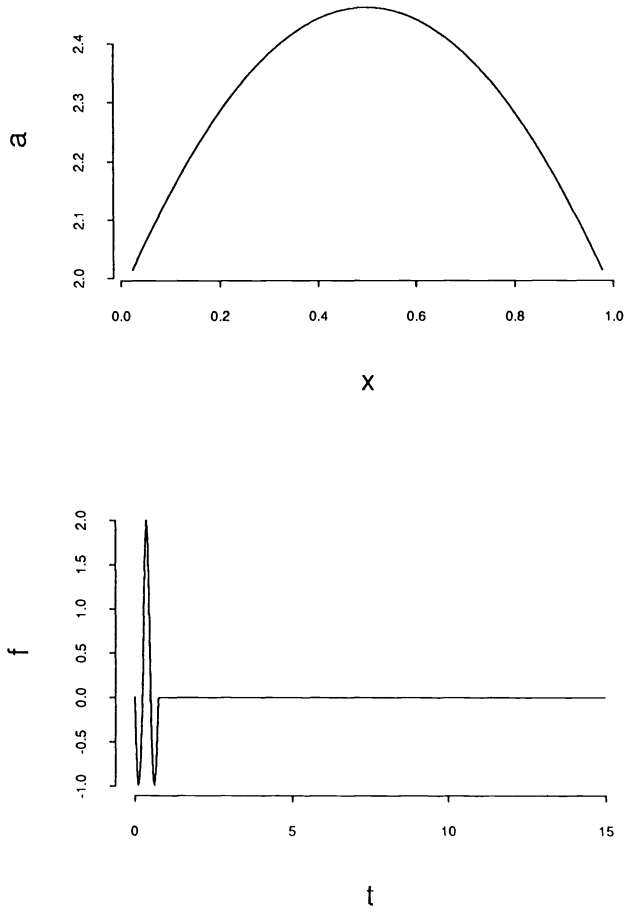


FIG. 1.2. *Parameters for the hyperbolic problem.*

Some theoretical convergence characteristics of constrained nonlinear least squares estimators with quadratic penalty functionals are studied in [17]. The identifiability of θ when λ is zero and the sampling is continuous plays an important role in this analysis. For λ greater than zero the identifiability issue is much less problematic since, intuitively, the data need only be relied upon to identify the component of θ lying in the null space of the penalty functional.

1.3. Sensitivity analysis. The model sensitivity is defined as the derivative of the observed data functionals ($\eta(\theta, x_i)$ in (1.3)) with respect to θ. In the case where the target parameter θ is represented in terms of a p-dimensional finite element approximation

$$\theta(x) = \sum_{k=1}^{p} \theta_k \phi_k(x) \tag{1.8}$$

where ϕ_k are a set of approximating elements (p maybe very large), the model sensitivity is a matrix defined as

$$\mathbf{X}_{ik} = \frac{\partial \eta(x_i; \boldsymbol{\theta})}{\partial \theta_k}, \qquad k = 1, 2, \cdots, p \tag{1.9}$$

for $i=1, 2, \cdots$. Here $\boldsymbol{\theta}=(\theta_1, \theta_2, \cdots, \theta_p)'$. For the model problems the sensitivity matrix becomes

$$\mathbf{X}_{ijk}=\frac{\partial u(x_i, t_j; \boldsymbol{\theta})}{\partial \theta_k}, \qquad k=1, 2, \cdots, p. \tag{1.10}$$

Consideration of the model sensitivity provides second-order information about the regularized solution method which has a number of useful applications.

1.4. Outline. The paper is organized as follows. Section 2 presents a computation method for evaluating the sensitivities. The method is naturally suited to parallel processing and some timing data are presented. Various applications of the sensitivity calculation are described in §§ 3–6.

The applications discussed in the following sections are as follows. Applications of sensitivity analysis: in § 3, implementation of the Gauss–Newton algorithm is shown; in § 4, adaptive selection of the regularization parameter λ, is shown; in § 5, confidence regions for the regularized solution are given; and in § 6, an analysis of resolution characteristics bias and variance is given. In each case the applications are illustrated in the context of the model problems.

2. Sensitivity computation. We begin by defining a numerical method for solving the forward problem, i.e., how values of $u(x_i, t_j; \theta)$ are obtained. The sensitivity calculation parallels the solution of the forward problem in a straightforward fashion. If θ is represented as a p-dimensional finite element approximation, it will be seen that the computational effort in the sensitivity calculation is p times that of the forward problem. This computation is necessarily intensive, however it is naturally suited to parallel processing, as is indicated below. For illustration purposes, we consider a simple implicit finite difference scheme for solving the forward problem. The procedure is presented for the one-dimensional model problems. Generalizations to more sophisticated forward problem solvers and to higher dimensions is indicated below.

2.1. A simple forward problem solver. Implicit finite difference schemes for solving parabolic and hyperbolic equations are described in [22], for example. The interval $0 \leqq x \leqq 1$ is divided into $N+1$ subintervals each of width $h=1/(N+1)$. The discretized approximation to the solution of the differential equation is obtained at time intervals of length Δ. The solution at time $j\Delta$ is denoted $\mathbf{u}^j$ for $j=0, 1, 2, \cdots$ with $u_i^j \approx u(ih, j\Delta; \theta)$. Throughout this section we assume for simplicity that measurement times are $t_j=j\Delta$ for $j=1, 2, \cdots, m$. When the measurement times are otherwise, a linear interpolation scheme can be used. Suppose the elliptic component of the differential equations is approximated by a difference quotient

$$\frac{\partial}{\partial x}\left(\theta(x)\frac{\partial u(ih, j\Delta; \theta)}{\partial x}\right) \approx \theta(ih)\frac{(u_{i+1}^j-2u_i^j+u_{i-1}^j)}{h^2}+\dot{\theta}(ih)\frac{(u_{i+1}^j-u_{i-1}^j)}{2h} \tag{2.1}$$

for $i=1, 2, \cdots, N$. $\dot{\theta}$ denotes differentiation with respect to x. The boundary conditions are used to eliminate u_0^j and u_{N+1}^j from these equations and we obtain

$$\frac{\partial}{\partial x}\left(\theta(x)\frac{\partial u(ih, j\Delta; \theta)}{\partial x}\right) \approx \mathbf{A}\mathbf{u}^j+\mathbf{q}^j \tag{2.2}$$

where $\mathbf{A}$ is a nonsymmetric tridiagonal matrix (independent of time) and $\mathbf{q}^j$ involves boundary and/or forcing conditions. With Dirichlet or zero Neuman boundary conditions $\mathbf{q}^j$ does not involve θ and so for the parabolic problem $\mathbf{q}^j=0$. Nonzero Neuman

boundary conditions for $\mathbf{q}^j$ to depend on θ; for the hyperbolic problem we have

$$q_i^j = \begin{cases} -2\theta(h)f(k\Delta)/h + \dot{\theta}(h)f(k\Delta), & i=1, \\ 0, & i>1. \end{cases} \tag{2.3}$$

Implicit difference schemes set time differences in u to be equal time averages of the elliptic component of the differential equation. By standard manipulation we have the following.

DIFFERENCE SCHEME FOR THE PARABOLIC EQUATION.

$$\mathbf{S}\mathbf{u}^{j+1} = \mathbf{B}\mathbf{u}^j + \tfrac{1}{2}(\mathbf{q}^{j+1} + \mathbf{q}^j), \qquad j = 0, 1, 2, \cdots \tag{2.4}$$

where $\mathbf{S} = \mathbf{I} - (\Delta/2)\mathbf{A}$ and $\mathbf{B} = \mathbf{I} + (\Delta/2)\mathbf{A}$. The method, which is recognizable as the Crank–Nicolson technique, is completely specified since $\mathbf{u}^0$ is given as initial data.

DIFFERENCE SCHEME FOR THE HYPERBOLIC EQUATION.

$$\mathbf{S}\mathbf{u}^{j+1} = \mathbf{B}\mathbf{u}^j - \mathbf{S}\mathbf{u}^{j-1} + \Delta^2(\tfrac{1}{4}\mathbf{q}^{j+1} + \tfrac{1}{2}\mathbf{q}^j + \tfrac{1}{4}\mathbf{q}^{j-1}), \qquad j = 0, 1, 2, \cdots \tag{2.5}$$

where $\mathbf{S} = \mathbf{I} - (\Delta^2/4)\mathbf{A}$ and $\mathbf{B} = 2\mathbf{I} + (\Delta^2/2)\mathbf{A}$. Here $\mathbf{u}^0$ is given but the initial conditions must be used to obtain $\mathbf{u}^1$. Since $(\partial/\partial t)u(x, 0) = 0$ we set $\mathbf{u}^1 - \mathbf{u}^{-1} = 0$ and obtain

$$2\mathbf{S}\mathbf{u}^1 = \mathbf{B}\mathbf{u}^0 + \Delta^2(\tfrac{1}{4}\mathbf{q}^1 + \tfrac{1}{2}\mathbf{q}^0). \tag{2.6}$$

The equations are easily solved by direct methods. The Cholesky decomposition of $\mathbf{S}$ is obtained once at $j = 0$ and the solutions at all subsequent times are computed by backsubstitution. In the case where θ depends on time (directly or via u which occurs with nonlinear diffusion) separate Cholesky decompositions are needed at every timestep.

2.2. Computation of the sensitivity matrix. The sensitivity matrix is defined as the gradient of the solution u with respect to the profile θ. By definition the sensitivity computation must track whatever method is used for evaluating u. In the present situation an implicit difference scheme is used to solve the forward problem. Differentiating the implicit difference equations we obtain a system of equations for the sensitivities; let $\partial_k \mathbf{u}^j = \partial \mathbf{u}^j / \partial \theta_k$.

DIFFERENCE SCHEME FOR $\partial_k \mathbf{u}$ IN THE PARABOLIC EQUATION. From (2.4)

$$\mathbf{S}\partial_k \mathbf{u}^{j+1} = \mathbf{B}\partial_k \mathbf{u}^j - \mathbf{S}_k \mathbf{u}^{j+1} + \mathbf{B}_k \mathbf{u}^j, \qquad j = 0, 1, 2, \cdots \tag{2.7}$$

where $\mathbf{S}_k = \partial \mathbf{S}/\partial \theta_k$ and $\mathbf{B}_k = \partial \mathbf{B}/\partial \theta_k$ (both of which are easy to evaluate since $\mathbf{S}$ and $\mathbf{B}$ are linear in θ, see equations (2.1) and (2.2)). The $\mathbf{q}$ term drops out because in the parabolic problem $\mathbf{q}$ is independent of θ. $\partial_k \mathbf{u}^0 = 0$ because the initial data does not depend on θ.

DIFFERENCE SCHEME FOR $\partial_u \mathbf{u}$ IN THE HYPERBOLIC EQUATION. Similar manipulations applied to (2.5) give

$$\begin{aligned} \mathbf{S}\partial_k \mathbf{u}^{j+1} = {} & \mathbf{B}\partial_k \mathbf{u}^j - \mathbf{S}\partial_k \mathbf{u}^{j-1} - \mathbf{S}_k \mathbf{u}^{j+1} + \mathbf{B}_k \mathbf{u}^j - \mathbf{S}_k \mathbf{u}^{j-1} \\ & + \Delta^2(\tfrac{1}{4}\partial_k \mathbf{q}^{j+1} + \tfrac{1}{2}\partial_k \mathbf{q}^j + \tfrac{1}{4}\partial_k \mathbf{q}^{j-1}), \qquad j = 1, 2, \cdots \end{aligned} \tag{2.8}$$

where $\mathbf{S}_k = \partial \mathbf{S}/\partial \theta_k$ and $\mathbf{B}_k = \partial \mathbf{B}/\partial \theta_k$. Since $\partial_k \mathbf{u}^0 = 0$ from (2.6), we obtain an expression for $\partial_k \mathbf{u}^1$

$$2\mathbf{S}\partial_k \mathbf{u}^1 = -2\mathbf{S}_k \mathbf{u}^1 + \Delta^2(\tfrac{1}{4}\partial_k \mathbf{q}^1 + \tfrac{1}{2}\partial_k \mathbf{q}^0). \tag{2.9}$$

2.3. Computation for X'r and X'X. Suppose we have n measurement sites x_i, $i = 1, 2, \cdots, n$. In the applications in §§ 3–6 it is of interest to compute $\mathbf{X}'\mathbf{r}$ and $\mathbf{X}'\mathbf{X}$, where $\mathbf{r}$ is an $m \cdot n$-vector. Note that these quantities can be computed without having to

store the matrix **X**—this is a substantial saving. Let $\partial_k \bar{\mathbf{u}}^j$ represent the n-vector resulting from linearly interpolating $\partial_k \mathbf{u}^j$ to the measurement sites. Let $\mathbf{r} = (\mathbf{r}(t_1), \mathbf{r}(t_2), \cdots, \mathbf{r}(t_m))$ with $\mathbf{r}(t_j)$ an n-vector. $\mathbf{X}'\mathbf{r}$ and $\mathbf{X}'\mathbf{X}$ are computed as

$$\begin{aligned} \mathbf{X}'\mathbf{r}_k &\approx \sum_{j=1}^{m} (\partial_k \bar{\mathbf{u}}^j)'\mathbf{r}(t_j), \\ \mathbf{X}'\mathbf{X}_{kl} &\approx \sum_{j=1}^{m} (\partial_k \bar{\mathbf{u}}^j)'\partial_l \bar{\mathbf{u}}^j, \qquad k, l = 1, 2, \cdots, p. \end{aligned} \tag{2.10}$$

At the jth timestep we first compute $\mathbf{u}^{j+1}$ and with this we then evaluate $\partial_k \mathbf{u}^{j+1}$ for $k = 1, 2, \cdots, p$. $\mathbf{X}'\mathbf{r}$ and $\mathbf{X}'\mathbf{X}$ are updated as

$$\begin{aligned} \mathbf{X}'\mathbf{r}_k^{j+1} &= \mathbf{X}'\mathbf{r}_k^j + (\partial_k \bar{\mathbf{u}}^{j+1})'\mathbf{r}(t_{j+1}), \\ \mathbf{X}'\mathbf{X}_{kl}^{j+1} &= \mathbf{X}'\mathbf{X}_{kl}^j + (\partial_k \bar{\mathbf{u}}^{j+1})'\partial_l \bar{\mathbf{u}}^{j+1}, \qquad k = 0, 1, \cdots, m-1. \end{aligned} \tag{2.11}$$

$\mathbf{X}'\mathbf{r}_k^0$ and $\mathbf{X}'\mathbf{X}_{kl}^0$ are both set to zero.

2.4. Generalizations. For more sophisticated forward problem solvers or higher-dimensional problems the linear systems for $u^{(j+1)}$ in (2.4)-(2.8) are modified (with new definitions for quantities analogous to **q**, **B** and **S**). Differentiating the linear system with respect to the kth coefficient for θ yields a linear system for $\partial_k u^{(j+1)}$. By the chain rule this system will be of the same form as the linear system for $u^{(j+1)}$ except that the right-hand side is modified. If an adaptive timestep is used in solving the forward problem then the matrices **B** and **S** will depend on the time index j. The mechanics of the sensitivity computation remain the same. With more complicated nonlinear problems analytic expressions for derivatives of quantities such as **B** may not be available and in this case divided difference approximations could be employed. The computational effort in the sensitivity calculation is still p-times that of the forward problem.

2.5. Parallelism. We can take advantage of parallel processing in computing sensitivities. In a parallel processing environment at the jth step the computation of $\partial_k \mathbf{u}^{j+1}$ for $k = 1, 2, \cdots, p$ is distributed over available processors (we are solving p linear systems involving the same coefficient matrix but different left-hand sides). An individual processor will first evaluate the right-hand side for $\partial_k \mathbf{u}^{j+1}$ and then obtain $\partial_k \mathbf{u}^{j+1}$ by back substitution. The shared data are $\mathbf{u}^{j+1}$, $\mathbf{u}^j$, ($\mathbf{u}^{j-1}$ in the hyperbolic case), and the **S** and **B** matrices.

To illustrate the speedup in a multiprocessor environment, an experiment was conducted on the Sequent computer in the Department of Computer Science at the University of Washington. For the parabolic model problem we chose a B-spline approximation for θ (see § 3.1) with $p = 256$ elements. The discretization in the implicit difference scheme was set at $N = 4 \cdot p$. Figure 2.1 shows the compute time per processor for one timestep of the sensitivity computation. The compute times are plotted against the reciprocal of the number of available processors which ranged from 1 to 18. There is a clear linear relation which demonstrates a linear speedup. Note the slight deviation from linearity as the number of processors increases. This is the synchronization overhead which is seen to be negligible in this experiment.

2.6. Further computational considerations. In one-dimension the computations effort per iteration is proportional to $N \cdot (p+1) \cdot m$. The N term comes from the time required to solve the linear systems in (2.4)-(2.8), $(p+1)$ because there are $(p+1)$ systems to solve at each timestep and m is the number of timesteps. For adaptive

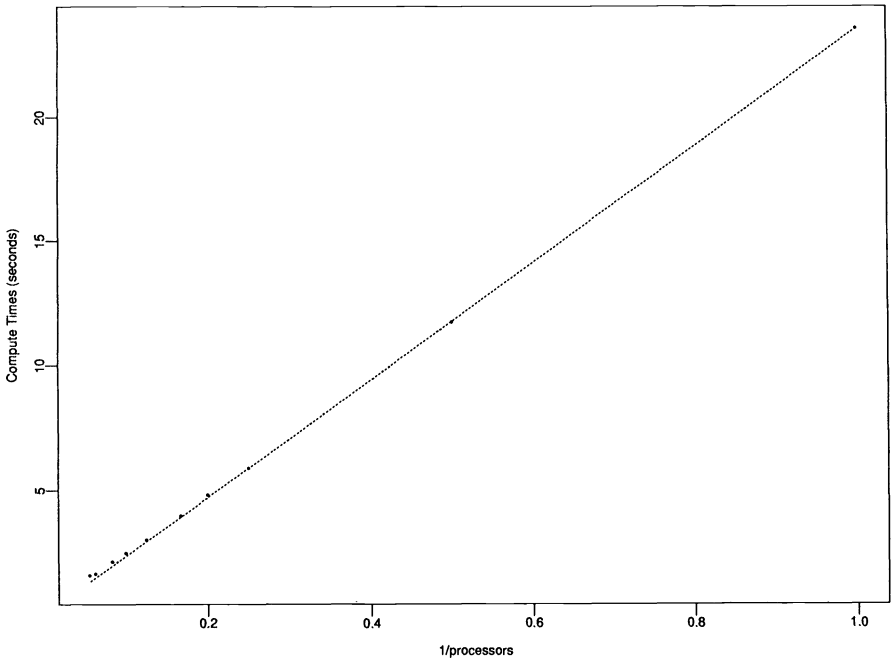


FIG. 2.1. *Parallel speedup in the sensitivity computation.*

methods m depends on θ and specified initial and boundary data. It is natural to relate the number of terms in the finite element representation p, and the order of discretization N, such as $N = \alpha \cdot p$. From these considerations the computing time for sensitivity analysis of a system involving one space variable should be

$$\text{time} \approx f_c p^2 m. \tag{2.12}$$

The factor f_c depends on the dimension of the problem, the definition of the forward problem solver and the particular computer. For the model we set $\alpha = 4$. The compute time factors f_c corresponding to the simple scheme in (2.4) and (2.7) were estimated to be $f_c = 6{\cdot}0 \times 10^{-5}$ seconds on an early generation Cray-1 X-MP and $f_c = 0.38 \times 10^{-5}$ seconds on a more recent Cray-2 Y-MP. (In both these computations the sensitivity calculation was not parallelized.) The later compute time makes the sensitivity computation realistic with p on the order of 1,000 on this computer. Newer generation Cray-3 supercomputers are projected to be between 2 and 4 times faster than the Cray-2 we used. Note that (2.12) assumes that the compute time for a single solution is a linear function of the order of discretization. In dimensions greater than one, (2.12) should reflect the additional time required to solve the system. For example, if Fourier methods were used, we might have $\log(p)p^2$ in place of p^2.

3. Implementation of the Gauss–Newton algorithm. We consider a penalty function $J(\theta)$ which is quadratic (positive semidefinite) in θ. Discretizing θ as in (1.8) and expressing the regularization functional in terms of the finite element coefficients $\boldsymbol{\theta}$ leads to the objective functional

$$l_\lambda(\boldsymbol{\theta}) = \sum_{j=1}^{m} \sum_{i=1}^{n} [z_{ij} - u(x_i, t_j; \boldsymbol{\theta})]^2 + \lambda \boldsymbol{\theta}' \boldsymbol{\Omega} \boldsymbol{\theta} \tag{3.1}$$

where the matrix $\boldsymbol{\Omega}$ comes from the quadratic penalty term. To compute the regularized estimator we minimize $l_\lambda(\boldsymbol{\theta})$ subject to the constraints.

The minimization could be approached either by first- or second-order gradient algorithms and it is likely that a combination of first- and second-order gradient algorithms would prove most efficient. First-order gradient methods have been proposed, for instance, in [2] and [11]. A powerful result which comes from optimal control theory is that the gradient for the regularization functional can be computed by solving the original differential equation (state equation) together with *one* associated *adjoint* equation (see [2], [11], for example). Second-order gradient algorithms use Hessian information and as a result the computational effort per iteration is much greater than for first-order methods. In practice, this must be balanced against available computing resources and the fact that second-order methods can achieve higher-order rates of convergence.

The damped Gauss–Newton algorithm leads to the iteration

$$\boldsymbol{\theta}^{(s+1)} = \boldsymbol{\theta}^{(s)} - \tau \cdot [\mathbf{X}'\mathbf{X} + 2\lambda\boldsymbol{\Omega}]^{-1}[-\mathbf{X}'\mathbf{r} + 2\lambda\boldsymbol{\Omega}\boldsymbol{\theta}^{(s)}] \tag{3.2}$$

for $s = 0, 1, 2, \cdots$. Here τ is a stepsize which is chosen so that l_λ is minimized subject to $\theta^{(s+1)}$ satisfying the constraints. In our implementation a simple stepsize halving strategy is used to determine τ. $\mathbf{X}$ is the sensitivity matrix computed at $\boldsymbol{\theta}^{(s)}$ and $\mathbf{r}$ is the current residual

$$r_{ij} = z_{ij} - u(x_i, t_j; \boldsymbol{\theta}^{(s)}) \tag{3.3}$$

for $i = 1, 2, \cdots, n$, $j = 1, 2, \cdots, m$, and $k = 1, 2, \cdots, p$. The computation of $\mathbf{X}'\mathbf{X}$ and $\mathbf{X}\mathbf{r}$ is described in § 2. Convergence is declared when successive iterates produce small relative changes in the objective function. At convergence $\hat{\theta}_\lambda$ is set to the converged iterate. Note that the Gauss–Newton iterate could also be evaluated in terms of the QR-decomposition of $\mathbf{X}$. However as we noted in § 2.3 the evaluation of $\mathbf{X}$ or its QR-decomposition requires excessive storage.

3.1. Application to the model problems. In the model problems we use cubic B-spline elements ϕ_k defined with respect to a uniform knot sequence on $(0, 1)$ and knots of multiplicity four at 0 and 1 (see [6]). The penalty is $J(\theta) = \int_0^1 [\ddot{\theta}(x)]^2\, dx$, so $\boldsymbol{\Omega}$ is given by

$$\boldsymbol{\Omega}_{kl} = \int_0^1 \ddot{\phi}_k(x)\ddot{\phi}_l(x)\, dx, \qquad k, l = 1, 2, \cdots, p. \tag{3.4}$$

Since the ϕ_k's are cubic B-splines, $\boldsymbol{\Omega}$ is a symmetric seven-banded matrix. Positivity is enforced by requiring that the coefficients be positive; since the individual B-splines are positive (see [6]) the positivity of the solution is guaranteed. In our investigations with the model problems we have not found it necessary to explicitly enforce the positivity constraint.

3.2. Numerical convergence characteristics. Theoretical results concerning the convergence characteristics of the Gauss–Newton algorithm are rather well documented in standard texts (see [18], for example). Newton methods can be shown to be locally quadratically convergent for well-behaved objective functions. Damped Newton methods are globally convergent for strictly convex objective functions with compact level sets. The objective function involved in (3.1) is complicated and very difficult to work with because there is no analytic expression for $u(x_i, t_j; \boldsymbol{\theta})$ as a function of $\boldsymbol{\theta}$. The regularization functional consists of two parts: a residual sum of squares term and a penalty term. If we knew that the residual sum of squares term was approximately convex then we should have a good deal of confidence in the convergence of the damped Gauss–Newton method. Let the residual sum of squares be denoted RSS $(\boldsymbol{\theta})$.

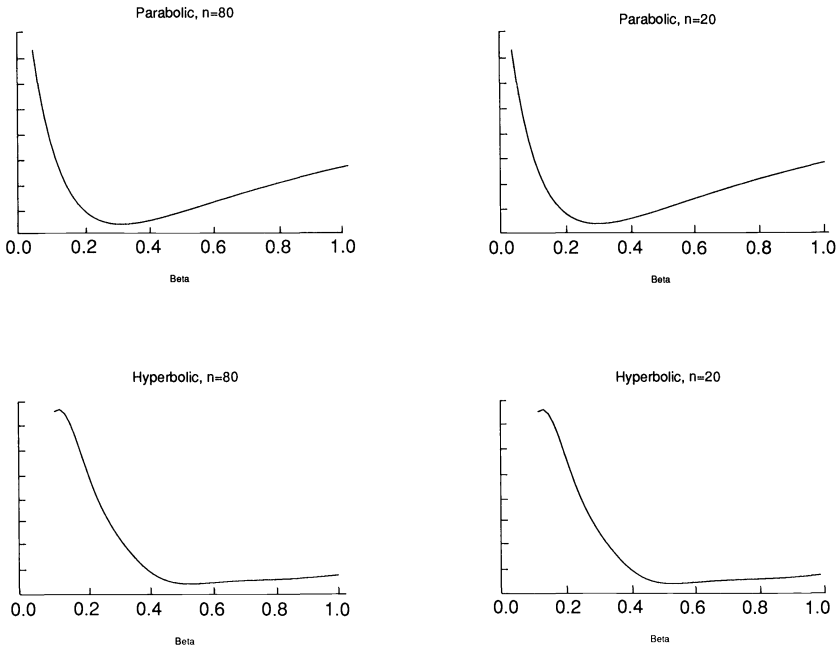


FIG. 3.1. *Random profiles of the residual sum of squares.*

Some random profiles of the residual sum of squares function were evaluated to investigate convexity. Two values for $\boldsymbol{\theta}$ were generated as follows: $\theta_j^{(1)} = \theta_j + 3 \cdot \alpha_j$ and $\theta_j^{(2)} = \boldsymbol{\theta}_j - 3 \cdot (1 - \alpha_j)$ for $j = 1, 2, 3, \cdots p$ where $\boldsymbol{\theta}$ is the true value and the α_j's are independent uniforms over the interval [0, 1]. The percent errors in Euclidean-norm between $\boldsymbol{\theta}^{(1)}$, $\boldsymbol{\theta}^{(2)}$, and $\boldsymbol{\theta}$ are about 50 percent for the hyperbolic and 240 percent for the parabolic. Figure 3.1 gives plots of the residual sum of squares component of the objective function, RSS $(\beta\boldsymbol{\theta}_{(1)} + (1-\beta)\boldsymbol{\theta}_{(2)})$ for $0 \leqq \beta \leqq 1$, for the parabolic and hyperbolic systems. Both systems give rise to very well behaved (strictly convex) behavior in this range. The hyperbolic problem shows some nonconvexity for the residual sum of squares component of the objective function in (3.1) on larger scales (this would not affect the local convergence rate of the method). From the plots we would guess that the Gauss-Newton algorithm would perform rather well at least provided the initial estimates are reasonable. We must realize that even a very large number of random profiles would not generate enough information to allow us to conduct a formal statistical test of a convexity hypothesis—the set of alternatives is just too large. However, given the appearance of random profiles it would be unreasonable to act as if the objective function were pathological.

3.3. Implementation. The algorithm is terminated when the relative change in the objective function is small; we test if

$$[l_\lambda(\boldsymbol{\theta}^{(s+1)}) - l_\lambda(\boldsymbol{\theta}^{(s)})] < \text{tol} \cdot l_\lambda(\boldsymbol{\theta}^{(s+1)}) \tag{3.5}$$

where tol = 0.001. We note that this criterion works well because the objective function is regularized. If $\lambda = 0$ then (3.8) would be highly unreliable as the estimation problem is ill posed.

Estimates for successively smaller values of λ were computed, and the best value is chosen using the method of generalized cross validation described in § 5. If $\lambda_1 > \lambda_2$

are successive values of λ at which solutions are computed $\hat{\boldsymbol{\theta}}_{\lambda_1}$ is used as a starting guess in solving for $\hat{\boldsymbol{\theta}}_{\lambda_2}$. For the examples computed in § 5, see Fig. 5.1. The initial solution took about six to seven iterations but subsequent solutions numerically converged in one or two iterations. The smoothest solution is essentially linear so that the computational effort with this solution could be reduced by explicitly fitting a line for $\boldsymbol{\theta}$.

4. Adaptive choice of the regularization parameter. A number of techniques have been proposed for choosing regularization parameters. For example, § 5 of [11] describes two techniques, one due to Miller [12] and another that is a variant of the discrepancy principle of Arcangeli [1] and Morozov [13]. In regularization problems it is well known that the optimal choice of the regularization parameter is a function of the unknown function θ and the unknown noise level. Miller's method requires an a priori upper bound on $J(\theta)$, the penalty function evaluated at the true parameter value, and also a specification of the noise level in the data. In practice such quantities are difficult to obtain. For optimal convergence in the linear regularization problems the upper bound has to increase with increasing sample size, yet it is unclear how the method can accommodate this. The discrepancy principle approach is more clearly defined. The method has been analyzed and shown to be suboptimal (see [8], [21]), basically because it produces solutions which are consistently too smooth.

The choice of regularization-like parameters is a problem which has received considerable attention in the statistics literature (see [15] for a review) because of the close similarity with the problem of choosing the number of variables to enter into a regression equation. A number of procedures have emerged which have been shown to have good theoretical properties, one of these methods is generalized cross validation (GCV) [5]. The GCV criterion chooses λ by minimizing the criterion

$$V(\lambda) = \text{RSS}(\lambda)/[mn - \mu_1(\lambda)]^2 \tag{4.1}$$

with

$$\text{RSS}(\lambda) = \sum_{j=1}^{m} \sum_{i=1}^{n} [z_{ij} - u(x_i, t_j; \hat{\boldsymbol{\theta}}_\lambda)]^2$$

and

$$\mu_1(\lambda) = \text{trace}\{[\mathbf{X}'\mathbf{X} + 2\lambda\boldsymbol{\Omega}]^{-1}\mathbf{X}'\mathbf{X}\}. \tag{4.2}$$

RSS (λ) is the residual sum of squares and $[mn - \mu_1(\lambda)]$ is referred to as the *effective* degrees of freedom for error (see [14] for another use of this sort of cross validation function in a nonlinear least squares regularization problem).

For linear regularization problems, the GCV estimate is known to produce estimates that perform well from the point of view of the predictive mean square error loss. In the present situation the predictive mean square error is

$$R(\lambda) = \frac{1}{nm} \sum_{j=1}^{m} \sum_{i=1}^{n} [u(x_i, t_j; \boldsymbol{\theta}) - u(x_i, t_j; \hat{\boldsymbol{\theta}}_\lambda)]^2 \tag{4.3}$$

where $\boldsymbol{\theta}$ is the true parameter and $\hat{\boldsymbol{\theta}}_\lambda$ is the regularization estimate. An asymptotic theory for GCV in nonlinear regularization models is not available at present but numerical experiments in [14], [16] have shown that the technique has definite promise even in nonlinear situations.

4.1. Application to the model problems. For the parabolic and hyperbolic problems data were generated according to (1.7) with the ε_{ij} being independent Gaussian random

variables with mean zero and standard deviation σ. One sample was generated with $n = 80$ equispaced measurement sites over the interval $[0, 1]$ and a subsample with $n = 20$ equispaced measurement sites was also taken. The temporal sampling in the parabolic case consisted of $m = 100$ equispaced points over the interval $[0, 0.35]$; in the hyperbolic case $m = 400$ equispaced points over the interval $[0, 15]$. The noise level σ was 3.0 in the parabolic case and 0.1 in the hyperbolic case; this resulted in the ratio of σ to the sample standard deviation of the true signal u being roughly 20 percent in both the parabolic and hyperbolic cases. For the $n = 80(20)$ sampling there are 8,000(2,000) observations in the parabolic case and 32,000(8,000) observations in the hyperbolic case. Regularization estimates with $p = 30$ and $N = 120$, were computed on each data set for a range of λ values and the GCV function evaluated. The starting guess was a constant profile $\theta(x) = 0.5$. Figure 4.1 shows the GCV function for each sample. The GCV function is plotted against $\mu_1(\lambda)$—the effective degrees of freedom for the model. $\mu_1(0) \approx p = 30$ (the order of discretization for θ) and $\mu_1(\lambda) \to 2$ as $\lambda \to \infty$. In all four cases the minimizer of the GCV criterion produces a value of λ that is very close to the minimizer of the predictive mean square error. The efficacy of the GCV function is measured by

$$\min_{\lambda} R(\lambda)/R(\hat{\lambda}) \tag{4.4}$$

where $\hat{\lambda}$ is the value of λ minimizing the GCV function. Efficacies, marked by "eff" on the graphs, are close to one in all cases, which means that the GCV criterion chooses a value of λ that performs well from the point of view of the predictive mean square error loss.

4.2. Implementation. We chose the scaling of λ on a degrees of freedom for model scale, i.e., given a $\mathbf{X}'\mathbf{X}$ matrix, λ is chosen so that the effective degrees of freedom for

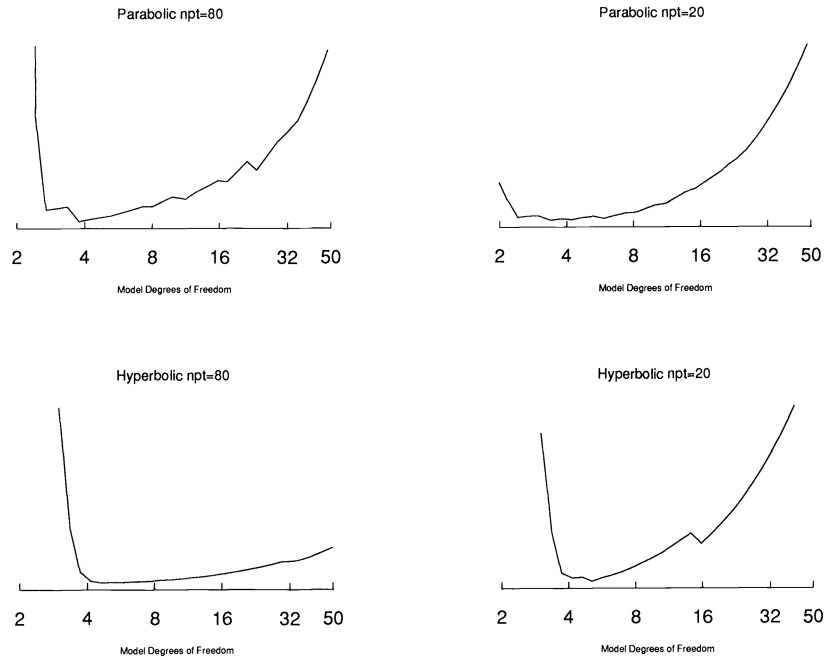


FIG. 4.1. *Cross validation functions plotted against the effective degrees of freedom for the model* $\mu_1(\lambda)$ *in* (4.2). *The efficacy relative to the predictive mean square error* (*see* (4.4)) *is indicated by* eff.

the model $\mu_1(\lambda)$ is some specified value. Since μ_1 is monotonically decreasing with increasing λ a zero finding routine was used to find the λ which achieves the desired model degrees of freedom. At present the optimal value is found by brute force but gradient methods might also be used provided care is taken to avoid local minima—cross validation functions can be noisy (see Fig. 4.1).

5. Simultaneous interval estimates for model parameters. Approximate Bayesian intervals estimates are obtained in a straightforward manner (see, for example, [16], [20], [23]). Second-order expansion of the regularization functional leads to a Gaussian approximation for the posterior distribution of $\boldsymbol{\theta}$ with a given value of λ

$$\pi(\boldsymbol{\theta}|\lambda) \approx N(\hat{\boldsymbol{\theta}}_\lambda, \sigma^2[\mathbf{X}'\mathbf{X}+2\lambda\boldsymbol{\Omega}]^{-1}). \tag{5.1}$$

σ is estimated by the residual standard deviation

$$\hat{\sigma} = \sqrt{\mathrm{RSS}(\lambda)/[mn-\mu_1(\lambda)]}. \tag{5.2}$$

As indicated by Silverman [20], approximate interval estimates for $\boldsymbol{\theta}$, or functionals thereof, can be generated by simulating from the posterior distribution. To obtain highest probability density (HPD) bands for the curve we generate realizations from the posterior distribution $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \cdots, \boldsymbol{\theta}_B$ with B large, say 1,000 or so. The 2.5th and 97.5th percentiles of the set of values $\{\theta_1(x), \theta_2(x) \cdots \theta_B(x)\}$ are the lower and upper

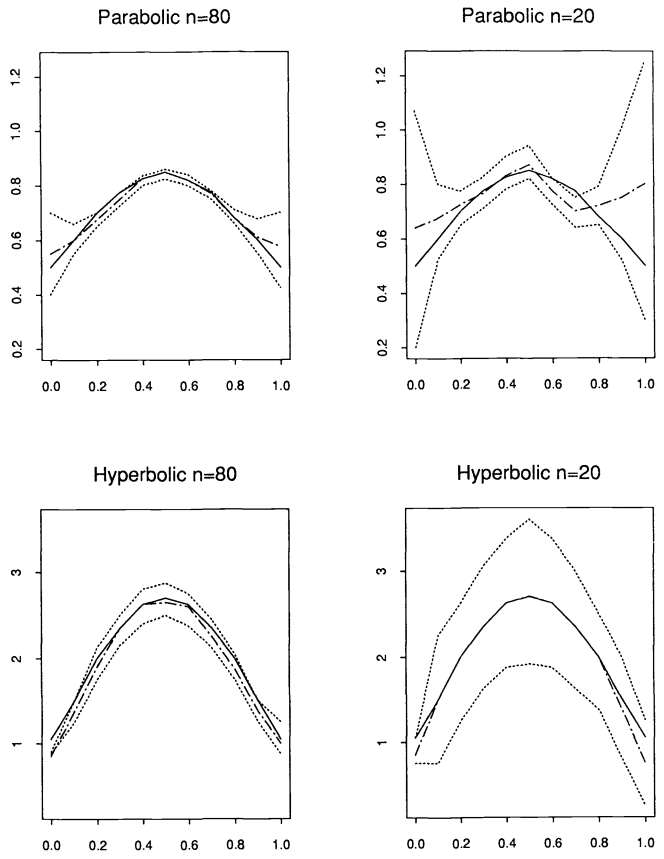


FIG. 5.1. *Estimate and 95 percent confidence bands. The plots show the true curve (solid line), the cross validated regularization estimate (dashed line) and the upper and lower confidence bands (dotted lines).*

limits for a 95 percent HPD interval at a point x. Since (5.1) is Gaussian, the interval has the form

$$\hat{\theta}_\lambda(x) \pm 1.96 \operatorname{se}(x) \tag{5.3}$$

where $\operatorname{se}(x)$ is the standard error in $\hat{\theta}_\lambda(x)$ and 1.96 is the appropriate percentile of the standard normal distribution.

For simultaneous intervals we replace the standard normal percentile value of 1.96 by a factor $\sqrt{f}$ with the property that

$$P\left[\max_{0 \leqq x \leqq 1} \left[\frac{\theta(x) - \hat{\theta}_\lambda(x)}{\operatorname{se}(x)}\right]^2 \leqq f\right] = 0.95 \tag{5.4}$$

P is evaluating the posterior probability of the event. The factor f is estimated by simulation, i.e., we take the 95th percentile of the set of values

$$\left\{\max_{0 \leqq x \leqq 1} \left[\frac{\theta_b(x) - \hat{\theta}_\lambda(x)}{\operatorname{se}(x)}\right]^2 ; b = 1, 2, \cdots, B\right\}. \tag{5.5}$$

The true curve and the 95 percent simultaneous HPD bands are plotted in Fig. 5.1 (the factor $\sqrt{f} \approx 3.0$ so the interval is about 33 percent wider than the pointwise interval). Here λ is set at the GCV estimate. The plot gives the true curve together with $\hat{\theta}_\lambda$ and

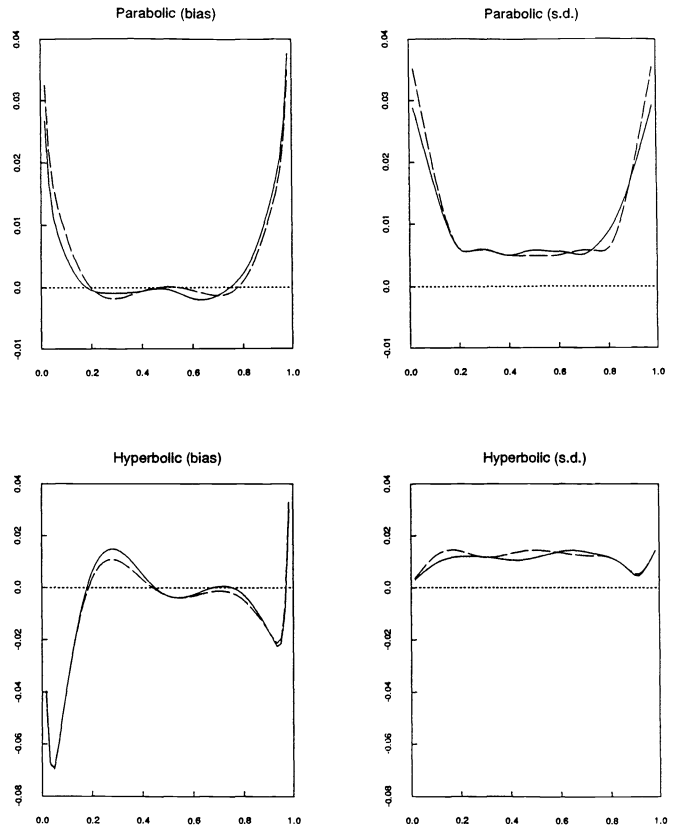


FIG. 6.1. *Pointwise error characteristics for λ giving a model degrees of freedom of* 4.5. *The plots show the true error characteristic estimated from the simulation (solid line) and the theoretical approximations based on* $\mathbf{X}'\mathbf{X}$ *(dashed line).*

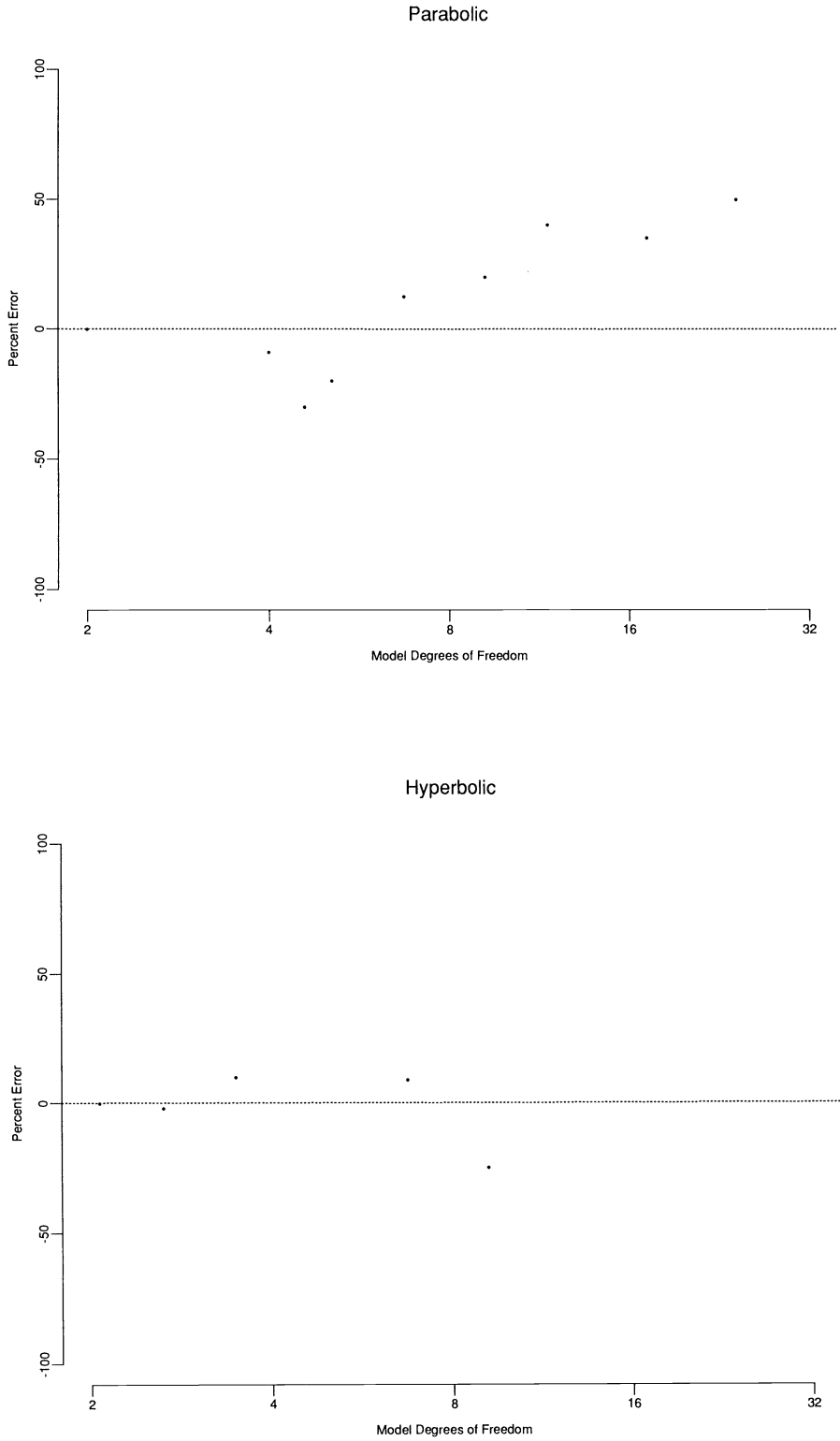


FIG. 6.2. *Percent errors in the mean integrated squared error approximation.*

the upper and lower HDP band limits. The initial estimate is also shown. In view of the approximation in (5.1), the intervals do a very good job of capturing the true curve. Intervals are wider in the case where the sampling is reduced from 80 to 20 measurement sites, reflecting a relative lack of confidence in these estimates. The intervals for the hyperbolic estimation are much wider than for the parabolic. This could be due to a difference in the temporal sampling rate. Looking back to (1.4) and (1.5), we might guess that while the parabolic must somehow involve an estimation of $\partial u/\partial t$, the hyperbolic must involve an estimation of $\partial^2 u/\partial t^2$—parameter estimation in the hyperbolic problem is more difficult than in the parabolic problem. Note that the interval estimates become wider at the endpoints of the interval [0, 1]. This is intuitive because the information is truncated at the ends. These effects are more severe in the parabolic case probably because the $u(x, t)$ has zero gradient at the boundary.

A theoretical discussion of Bayesian confidence intervals for linear regularization estimators can be found in [4]. It could be argued that the intervals should be adjusted to take into account the uncertainty in the determination of λ. An investigation of this issue is beyond the scope of this paper.

6. Approximation of error characteristics by sensitivity analysis. A straightforward procedure for the analysis of resolution characteristics that uses $\mathbf{X}'\mathbf{X}$ is described in [15]. The method is based on a one-step Gauss–Newton expansion; some asymptotic analysis of this is provided in [17]. Here we present a simulation study which shows that resolution analysis based on $\mathbf{X}'\mathbf{X}$ works rather well in finite samples.

6.1. Bias and variance. The error in the regularization estimator $\hat{\boldsymbol{\theta}}_\lambda$ may be expressed as

$$\hat{\boldsymbol{\theta}}_\lambda - \boldsymbol{\theta}_0 = [\boldsymbol{\theta}_\lambda - \boldsymbol{\theta}_0] + [\hat{\boldsymbol{\theta}}_\lambda - \boldsymbol{\theta}_\lambda] \tag{6.1}$$

where $\boldsymbol{\theta}_0$ is the true value of $\boldsymbol{\theta}$, and $\boldsymbol{\theta}_\lambda$ is the regularized solution with continuous error-free data. The first term is the systematic error or *bias* in statistical jargon and the second is the random error. The one-step Gauss–Newton approximations to the systematic and random errors are

$$\boldsymbol{\theta}_\lambda - \boldsymbol{\theta}_0 \approx [\mathbf{X}'\mathbf{X} + 2\lambda\boldsymbol{\Omega}]^{-1}\mathbf{X}'\mathbf{X}\boldsymbol{\theta}_0 - \boldsymbol{\theta}_0 \tag{6.2}$$

and

$$\hat{\boldsymbol{\theta}}_\lambda - \boldsymbol{\theta}_\lambda \approx [\mathbf{X}'\mathbf{X} + 2\lambda\boldsymbol{\Omega}]^{-1}\mathbf{X}'\boldsymbol{\varepsilon} \tag{6.3}$$

where $\boldsymbol{\varepsilon}$ is the vector of errors. In the first case the sensitivity matrix is computed at θ_0 while in the second case it is computed at θ_λ. These approximations follow after some manipulation of (3.2) with $\tau = 1$. If the errors have mean zero and are uncorrelated with constant variance σ^2, then from (6.3)

$$\operatorname{Var} \hat{\boldsymbol{\theta}}_\lambda \approx \sigma^2[\mathbf{X}'\mathbf{X} + 2\lambda\boldsymbol{\Omega}]^{-1}\mathbf{X}'\mathbf{X}[\mathbf{X}'\mathbf{X} + 2\lambda\boldsymbol{\Omega}]^{-1}. \tag{6.4}$$

Thus by the one-step linearization the systematic and random error in the regularization estimator can be approximated. The results in [17] study when the latter two formulas are asymptotically justified in a neighbourhood of $\boldsymbol{\theta}_0$. The bias formula becomes more accurate as λ tends to zero and the sample size increases, while the variance approximation should improve with increasing sample size. With these approximations asymptotic resolution characteristics can be studied by considering the simultaneous diagonalization of $\mathbf{X}'\mathbf{X}$ relative to $\boldsymbol{\Omega}$ (see [15], for example).

6.2. A simulation experiment. A simulation study was used to evaluate the accuracy of the $\mathbf{X}'\mathbf{X}$-based resolution analysis. Data sets were generated according to the parabolic and hyperbolic models:

$$z_{ij}^{(s)} = u(x_i, t_j; \theta) + \varepsilon_{ij}^{(s)}, \qquad s = 1, 2, \cdots \tag{6.5}$$

for $i = 1, 2, \cdots, n$ and $j = 1, 2, \cdots, m$, with $\varepsilon_{ij}^{(s)}$ being independently and identically distributed zero mean Gaussian random variables with standard deviation of σ. The setup was the same as in § 4 except that only the $n = 80$ spatial sampling was considered. For each of several λ-values the regularization estimate $\hat{\theta}_\lambda[s]$ was computed for 25 data sets generated according to (6.5). For fixed λ, the mean and variance of $\hat{\theta}_\lambda[s]$ over the simulation runs provided direct estimates of the pointwise bias and variance. The approximations in (6.3) and (6.4) were used to obtain $\mathbf{X}'\mathbf{X}$-based pointwise bias and variance characteristics. The linearized bias based on (6.2) is a poor predictor of the true bias except for small λ. For larger λ the bias is much better approximated by evaluating the regularization estimator on error free data (i.e., (6.5) without the $\varepsilon_{ij}^{(s)}$). Pointwise bias and standard deviation results for λ corresponding to a 4.5 degrees of freedom model (see Fig. 4.1) are presented in Fig. 6.1. The approximation matches the true remarkably well. We see that the regularization estimate overshoots the true at the edges for the parabolic case and undershoots in the hyperbolic case. It would be natural to try to use this information to do a crude bias correction akin to the idea of twicing in the context of time series detrending. Correcting the bias will inflate the variance so some caution would have to be exercised. The results for a range of λ-values are summarized in Fig. 6.2 which gives the percent error between the true and approximated mean integrated squared error (MISE) over the domain of θ as a function of the model degrees of freedom. The approximation breaks down for the hyperbolic problems when the model degrees of freedom is large. Even so, it is clear that the $\mathbf{X}'\mathbf{X}$-based analysis provides a remarkably good approximation for the true mean integrated squared error.

Acknowledgments. I am grateful to Michael Stewart and Tony Wong for a number of valuable comments. The experiment on the Sequent computer was programmed by Bill Dunlap. The machine was made available by Professor Ed Lazowska in the Department of Computer Science at the University of Washington.

I am grateful to a referee for highlighting the point in the first paragraph of § 2.4.

REFERENCES

[1] R. ARCANGELI, *Pseudo-solution de l'equation Ax = y*, C. R. Acad. Sci. Ser. A, 263 (1966), pp. 282–285.

[2] W. H. CHEN, G. R. GAVALAS, J. H. SEINFELD, AND M. L. WASSERMAN, *A new algorithm for automatic history matching*, Soc. Pet. Eng. J., 14 (1974), pp. 593–608.

[3] R. L. COOLEY, *Incorporation of prior information on parameters into nonlinear regression groundwater flow models, 1 Theory*, Water Resour. Res., 18 (1982), pp. 965–976.

[4] D. D. COX, *An analysis of Bayesian inference for non-parametric regression*, Department of Statistics, University of Washington, Seattle, WA, 1986.

[5] P. CRAVEN AND G. WAHBA, *Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation*, Numer. Math., 31 (1979), pp. 377–403.

[6] C. DE BOOR, *A Practical Guide to B-Splines*, Springer-Verlag, New York, 1978.

[7] P. DEUFLHARD AND E. HAIRER, *Numerical Treatment of Inverse Problems in Differential and Integral Equations*, Birkhäuser, Boston, 1983.

[8] H. W. ENGL AND A. NEUBAUER, *Optimal parameter choice for ordinary and iterated Tikhonov regularization*, in Inverse and Ill-Posed Problems, H. W. Engl and C. W. Groetsch, eds., Academic Press, New York, 1987, pp. 97–125.

[9] H. W. ENGL AND C. W. GROETSCH, *Inverse and Ill-posed Problems*, Academic Press, New York, 1987.

[10] G. S. KIMELDORF AND G. WAHBA, *A correspondence between Bayesian estimation in stochastic processes and smoothing by splines*, Ann. of Math. Statist., 41 (1970), pp. 495–562.

[11] C. KRAVARIS AND J. H. SEINFELD, *Identification of parameters in distributed parameter systems by regularization*, SIAM J. Control Optim., 23 (1985), pp. 217–241.

[12] K. MILLER, *Least-squares methods for ill-posed problems with a prescribed bound*, SIAM J. Math. Anal., 1 (1970), pp. 52–74.

[13] A. MOROZOV, *On the solution of functional equations by the method of regularization*, Soviet Math. Dokl., 7 (1966), pp. 414–417.

[14] F. O'SULLIVAN AND G. WAHBA, *A cross validated Bayesian retrieval algorithm for nonlinear remote sensing experiments*, J. Comp. Phys., 59 (1985), pp. 441–455.

[15] F. O'SULLIVAN, *A statistical perspective on ill-posed problems* (*with discussion*), J. Statist. Sci., 1 (1986), pp. 502–527.

[16] ———, *Fast computation of fully automated log-density and log-hazard estimators*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 363–379.

[17] ———, *Convergence characteristics of method of regularization estimators for nonlinear operator equations*, SIAM J. Numer. Anal., 27 (1990), pp. 1635–1649.

[18] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solutions of Non-linear Equations in Several Variables*, Academic Press, New York, 1970.

[19] F. SANTOZA AND W. W. SYMES, *Linear inversion of band-limited reflection seismograms*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1307–1330.

[20] B. W. SILVERMAN, *Some aspects of the spline smoothing approach to nonparametric regression curve fitting* (*with discussion*), J. R. Statist. Soc. Ser. B, 47 (1985), pp. 1–50.

[21] D. M. TITTERINGTON, *Common structure of smoothing techniques in statistics*, Internat. Statist. Rev., 53 (1985), pp. 141–170.

[22] E. H. TWIZELL, *Computational Methods for Partial Differential Equations*, Ellis Horwood, Chichester, 1984.

[23] G. WAHBA, *Bayesian confidence intervals for the cross-validated smoothing spline*, J. Roy. Statist. Soc., 45 (1983), pp. 133–150.

[24] ———, *Spline Models for Observational Data*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1284-1303, November 1991
© 1991 Society for Industrial and Applied Mathematics
003

SOME STABILITY ASPECTS OF SCHEMES FOR THE ADAPTIVE INTEGRATION OF STIFF INITIAL VALUE PROBLEMS*

LUCA DIECI† AND DONALD ESTEP†

Abstract. The phenomenon of "superstability" in the numerical solution of stiff initial value problems of ordinary differential equations is considered. Superstability occurs when many implicit schemes are used, and here the backward differentiation formulas and the discontinuous Galerkin method are concentrated on. In the latter case, some implementation aspects are discussed. These methods are implemented with "local" and "global" error control strategies, respectively. By comparison, the effect that the different strategies have on the occurrence of superstability in two model problems is studied, and the trapezoidal rule implemented with a local error control strategy is also considered. New insight is gained into the superstability phenomenon, and it becomes clear that the method used to solve the nonlinear algebraic system, usually a (quasi)Newton method, has a large bearing. This problem is also considered in the context of the numerical integration of parabolic partial differential equations by the method of lines. In particular, the consequences for the dynamical behavior of solutions of a reaction-diffusion problem are studied.

Key words. stability, superstability, stiff initial value problems, backward Euler, backward differentiation formulas, discontinuous Galerkin, trapezoidal rule, adaptive integration, local and global error control, method of lines

AMS(MOS) subject classifications. 65L05, 65L07, 65L20

1. Introduction.

The term "superstable" was introduced in [DESS] to describe integration formulas whose region of numerical stability encloses regions of physical instability of the solution. The authors remarked on the possible danger of automatic integration codes based on such formulas. Similar dangerous properties had already been observed by Lindberg [L], who also considered schemes that were not superstable in the sense of [DESS]. In this paper, we propose the following, broader, working definition: "*superstability* is that situation in which a numerical integrator does not detect that the underlying solution is physically unstable." The purpose of this paper is to examine this potentially dangerous phenomenon and to study the reasons for its occurrence.

Understanding superstability is important because it occurs in the adaptive integration of stiff (IVPs). Our experience indicates its occurrence in the solution of boundary value problems (BVPs) by initial value (IV) techniques, and in the integration of parabolic partial differential equations (PDEs) by the method of lines. In all cases, the situation is one in which the solution of the underlying IVP is (initially) very stable, and the danger is that a possible (later) change in stability is not discovered.

Recently, the superstability phenomenon has been observed and studied for the backward differentiation formulas (BDF) when solving differential Riccati equations [D1], [D2]. In [D2], by using the structure of Riccati equations, a method for avoiding the practical occurrence of superstability is given. In general, however, it is not clear how to avoid the phenomenon.

Superstability occurs also when using the method of lines to numerically integrate time-dependent PDEs. A physical example shows that this procedure can give startingly incorrect, yet believable, results.

* Received by the editors May 7, 1990; accepted for publication September 20, 1990.

† School of Mathematics, Georgia Institute of Technology, Atlanta, Georgia 30332. The work of the first author was partially supported by National Science Foundation grant DMS8802762. The work of the second author was partially supported by Defense Advanced Research Projects Agency grant #70NANB8HO860.

We consider the IVP

$$\text{(1.1)} \qquad \begin{cases} \dfrac{dy}{dt} = f(t, y), & 0 \leqq t \leqq T, \\ y(t_0) = y_0, & y \in \mathbb{R}^N. \end{cases}$$

A standard example of a superstable formula is given by the backward Euler method (BEM). In the scalar case ($N = 1$ in (1.1)), a step of the BEM with uniform steps is given by

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}),$$

where h denotes the timestep, t_n denotes the time $n \cdot h$, and $y_n \approx y(t_n)$. The numerical stability of this step is guaranteed if $|1 - hf_y| \geqq 1$. This is certainly true if $\operatorname{Re}(f_y) < 0$, which corresponds to physical stability, but it is also true if $\operatorname{Re}(f_y) \geqq 2/h$, which corresponds to a physically unstable solution. On the basis of some error estimate, an adaptive code can select large timesteps and in the process completely miss a region of change in stability in the underlying problem and proceed to solve for an incorrect solution. This is the explanation of superstability given in [DESS].

Clearly, using formulas with a fixed stepsize can also lead to superstability if the stepsize is large enough; if we force the steps to be small everywhere, then superstability would not be of concern. For the same reason, it does not seem to be a concern when using explicit integration schemes. But these remedies are not practical in general.

We consider the superstable (in the sense of [DESS]) BDF and discontinuous Galerkin (DG) formulas. We confirm that superstability occurs for these schemes. However, the problem is more complex than this. We observe that superstability can occur also when using formulas that are not superstable, and we show that the phenomenon occurs for the trapezoidal rule.

Based on our results, we think that the key factor that determines the occurrence of superstability is the iterative solution process for the nonlinear discretized problem. The iterative solution process has its own dynamics, which might conflict (as for Newton's method) with the dynamics of the problem. Superstability can arise because of this fact. The viewpoint that emerges from our work is that the numerical scheme is all one thing with solving the nonlinear system. Neglecting this latter aspect and basing stepsize selection purely on accuracy considerations leads to a faulty analysis.

It is natural to suspect that the practical occurrence of superstability is related to the error control strategy. Since existing practical codes attempt to control the local truncation error (LTE), we are to wonder whether a code based on a more conservative control of the global error (GE) could be susceptible to the superstability phenomenon. In this paper we show that it can. We consider implementations of the BDF with LTE control and the DG methods with GE control. Although both of these error control strategies are based on a priori error estimates, in our experience a posteriori error control exhibits superstability as well. In any case, useful a posteriori estimates, even for the simple BEM, have been proved so far only for linear constant coefficient problems that have a fixed stability character.

The DG methods are receiving particular attention for the integration of reaction-diffusion equations [EJ], [EL1], [EL2], and have also been proposed recently [J] for stiff ODEs. Since they have not been implemented in any generality yet, we also discuss some of the relevant aspects of their practical implementation.

We develop our viewpoint by analyzing particular problems. We consider in detail two scalar Riccati differential equations arising from well-conditioned BVPs (see [D1]),

and analyze the occurrence of superstability for the schemes mentioned above. For BDF and DG methods, we have limited our theoretical analysis to their first-order representatives, but we have observed the same behavior for the higher-order schemes. We then consider two parabolic PDEs and their discretization by the method of lines, and once more observe the occurrence of superstability.

A plane of the paper is as follows. In the next section, we briefly discuss the BDF and DG methods, and the trapezoidal rule. In § 3, we analyze the two scalar model problems. In § 4, we consider superstability for some reaction-diffusion PDEs. Section 5 contains our conclusions.

2. Numerical methods. The system (1.1) is stiff when some of the eigenvalues of the Jacobian f_y have large and negative real parts. The difficulties inherent in the treatment of stiff problems are well known [SG]. A-stable, stiffly stable, and stiffly A-stable schemes are considered to be the best choice in general.

A qualitative description of the local stability properties of a scheme is commonly provided by analyzing the stability of the scheme applied to the problem $\dot{y} = \lambda y$, $\lambda \in \mathbb{C}$. With a fixed step h, the regions of stability are plotted in the complex $h\lambda$ plane. Within a stability region, the scheme acts to reduce roundoff error and bring solution curves together. Superstable formulas are characterized by the presence of stability regions in the right-hand half-plane.

We first consider the BDF and DG methods. Both yield a first-order scheme that is essentially the BEM, but otherwise have little in common. The BDF (orders one to five are used) and the pictures of their stability regions are well known (see Gear [G]). The BDF are stable on essentially the whole left half-plane and away from the origin on the right half-plane (stiffly stable schemes). As implementation of the BDF with local error control, we use the well-known code LSODE [H]. In this code, as usual, the stepsize selection for these formulas is based upon an approximation of the local truncation error, computed by comparing predicted and corrected values.

Remark. We have obtained identical outcome, as far as superstability is concerned, with other codes implementing the same formulas, as well as with our own unsophisticated implementations and full Newton iteration at each timestep.

We recall the first-order scheme for reference. Let $h_n := t_{n+1} - t_n$, $n = 0, 1, \cdots$. Then the first-order BDF scheme for (1.1) is the BEM

$$y_{n+1} = y_n + h_n f(t_{n+1}, y_{n+1}), \tag{2.1}$$

and (2.1) is usually solved for y_{n+1} by Newton's method (or some of its variants) with initial guess provided by

$$y_{n+1}^{(0)} := y_n + h_n f(t_n, y_n). \tag{2.2}$$

The local truncation error for (2.1) is estimated by comparing (2.2) and (2.1) as below:

$$|\text{local truncation error}| = \left| y''(\eta) \frac{h_n^2}{2} \right| \simeq \left| \frac{1}{2}(y_{n+1} - y_{n+1}^{(0)}) \right|. \tag{2.3}$$

This is the criterion normally used and it is a simple application of the mean value theorem.

The DG method [DHT] is a finite element method that yields a piecewise polynomial approximant. The approximant can be discontinuous at nodal points since its nodal values will be more accurate than its interior values, in general, and this discontinuity enters explicitly into the formulations. More precisely, on $(t_n, t_{n+1}]$ the DG $(2q+1)$ approximant is the polynomial of degree q in $\mathbb{R}^N$, $\tilde{y} := y|_{In} \in$

$[\mathcal{P}_q((t_n, t_{n+1}])]^N$, which satisfies

$$\int_{In} (\dot{\tilde{y}} - f(t, \tilde{y}(t))) \cdot v \, dt + y_n^+ \cdot v_n^+ = y_n^- \cdot v_n^+ \quad \text{for all } v \in [\mathcal{P}_q((t_n, t_{n+1}])]^N,$$

given y_n^-, where $v_n^\pm = \lim_{s \to 0^\pm} v(t+s)$ and $y_0^- = y_0$. The DG method allows for global error estimates on a variety of PDEs and ODEs. See [EJ], [EL1], [EL2], and [J]. In Fig. 1, we present a plot of the stability regions for DG1, DG3, and DG5 (notation refers to order). These formulas are clearly superstable but are different from the BDF in that they are stable on the whole left-hand half-plane (stiffly A-stable).

When applied on (1.1), DG1 consists of finding a piecewise constant approximant y_{n+1} on $(t_n, t_{n+1}]$ via

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(t, y_{n+1}) \, dt. \tag{2.4}$$

DG3 yields a piecewise linear approximant

$$\tilde{y}|_{(t_n, t_{n+1}]} = \frac{t_{n+1} - t}{h_n} y_n^+ + \frac{t - t_n}{h_n} y_{n+1}^- \quad \text{on } (t_n, t_{n+1}]$$

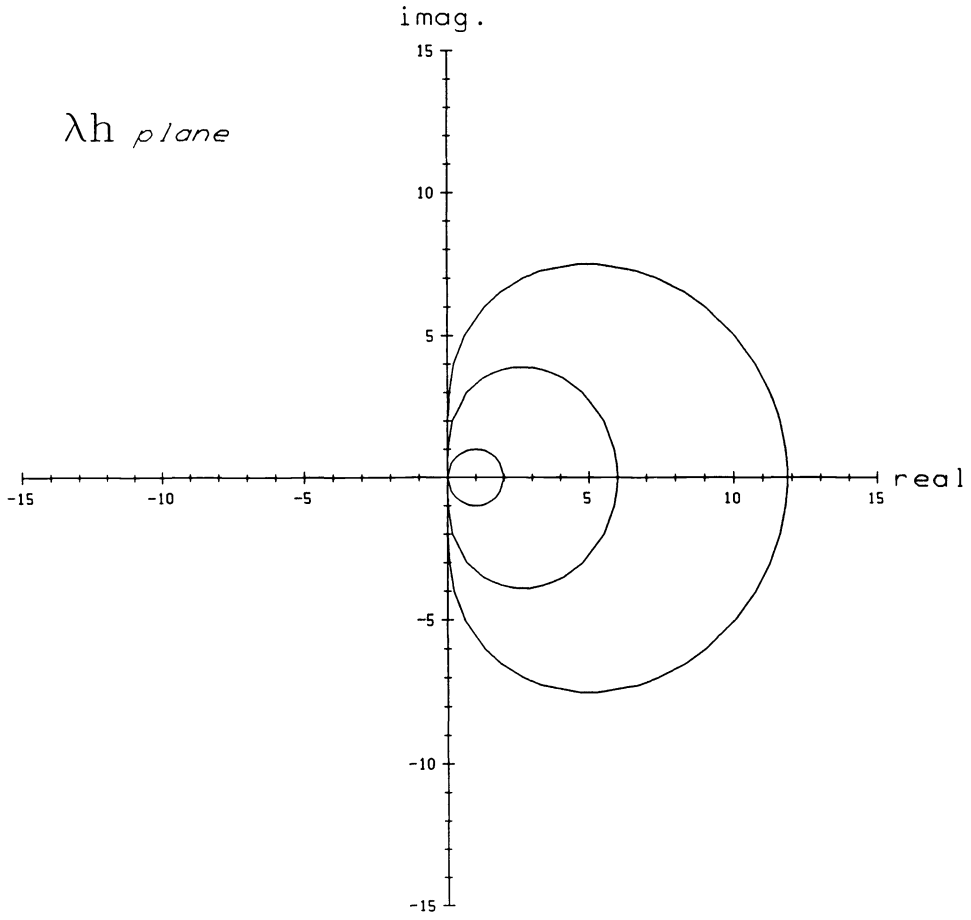


FIG. 1. *Stability regions for* DG *methods.* DG1, DG3, DG5 *are stable outside closed regions.*

via

$$
\begin{aligned}
y_{n+1}^{-} &= y_n^{-} + h_n \int_0^1 f(t_n + sh_n, (1-s)y_n^{+} + sy_{n+1}^{-})\, ds, \\
y_n^{+} &= y_{n+1}^{-} - 2h_n \int_0^1 f(t_n + sh_n, (1-s)y_n^{+} + sy_{n+1}^{-})\, ds.
\end{aligned} \tag{2.5}
$$

We have implemented the global error control for DG1 and DG3, based on a conservative version of the theoretical estimates in [J]. In [J], Johnson considers $f(t, y)$ of the form $f(t, y) := f(y(t)) + g(t)$, and makes a number of assumptions on the functions and the solution. Then, he shows a global error estimate for DG1 and DG3, assuming that the integrals in (2.4) and (2.5) are computed exactly. With $I_n = (t_n, t_{n+1}]$, and $\|y\|_n := \sup_{t \in I_n} \|y(t)\|$, where $\|\cdot\|$ denotes the Euclidean norm, these are

$$
\|y - \tilde{y}\|_{n+1} \leqq C \max_{l \leqq n+1} h_l^p \|y^{(p)}\|_l, \qquad p = 1, 2 \tag{2.6}
$$

with further order three for the nodal values y_{n+1}^{-} for (2.5):

$$
\|y(t_{n+1}) - y_{n+1}^{-}\| \leqq C \max_{l \leqq n+1} h_l^3 \|y^{(3)}\|_l. \tag{2.7}
$$

Remarks. (1) Our test problems in § 3 do not satisfy the assumptions in [J] on the form of f. While there is no problem in formulating the DG method for our problems, it is not clear that (2.6) and (2.7) still hold. On the other hand, our goal is to discover whether using a very conservative global error control would prevent the superstability phenomena. In this spirit, we use (2.6) and (2.7). For the parabolic equations in § 4, the analogous estimates do hold.

(2) The values of the constants in (2.6) and (2.7) have a large relevance for the error control. Considering that these are global error estimates, it is not surprising [SG] that these constants depend on the Jacobian f_y. More precisely (see [J]), they are given by taking appropriate norms on the solution of the linearized problem

$$
\begin{cases}
\dfrac{dx}{dt} - f_y(y(t))x = 0, & 0 < t \leqq t_n, \\
x(0) = Y_n - y(t_n).
\end{cases}
$$

Under certain assumptions on f_y, Johnson proves that these constants can be bounded globally. These assumptions do not hold for our test problems. Thus, we have estimated the constants in a rather crude way. We use $2\|f_y\|_n$. Then for (2.6), with $p = 1$, the estimate reads

$$
\|\text{global error}\| \leqq 2\|f_y\| \cdot \max_{i \leqq n} h_i \|y'\|_i. \tag{2.8}
$$

If we take the worst case, the bound becomes $ch\, \|f_y\|\, \|y'\|$. In comparison, by adding up local truncation errors, an estimate of the form $\|\text{global error}\| \leqq ch\, \|y''\|$ can be proven. Since $y'' = f_y \cdot y' + f_t$, we conclude that the two estimates for the BEM are not very different in a global sense. Local and global error control based on (2.3) and (2.8), respectively, will be different in implementation, however. For DG3, the analogous estimate does hold an advantage of one less derivative required of the solution over a typical local error estimate. However, we believe that this is lost when a quadrature is used to compute the integrals, though the form of the global estimate is preserved.

(3) In this generality, the DG method does not give a complete description of how to compute y_{n+1}, unless it happens to be possible to evaluate explicitly the integrals on the right-hand side of the equations. The choice of a suitable quadrature is a

nontrivial problem, since any quadrature rule gives de facto a different formula. A good choice should preserve the accuracy, of course, and it should allow for global error estimates similar to those obtained for the exact integration. Also, a choice should not alter the stability properties of the DG method. For DG1, the choice is easy since the approximant is constant on I_{n+1}, and replacing (2.4) by the BEM causes no difficulties analytically or numerically. For DG3, it is not clear now to proceed. For semilinear parabolic problems, Simpson's rule applied to the interpolant at (t_{n-1}, y_{n-1}^-), (t_n, y_n^-), and (t_{n+1}, y_{n+1}^-), where the approximant has its highest accuracy, allows for global error estimates and performs well in practice (see [EL2]). However, the resulting scheme is akin to a multistep method and when applied to (1.1), its stability properties are unclear. We have experimented with several choices, some involving nodal values at t_n and t_{n+1}, and some using values of $f(t, y)$ taken just in $(t_{n-1}, t_n]$. Three-point Radau quadrature computed on $(t_n, t_{n+1}]$ works particularly well; this quadrature leads to a scheme which preserves order and has good stability properties. However, the global a priori error estimate cannot be proven by the standard techniques.

Remark. On the test problems studied, we implemented the true DG method with exact integration of the nonlinearities, but we have obtained identical outcome by using the above integration rules.

The result of (2.5) is a $2N \times 2N$ system of nonlinear equations for y_n^+ and y_{n+1}^-. We use the full Newton iteration at each integration step. There are two natural choices for the initial guess. The first is to use an approximate Taylor expansion, i.e., for DG3, $y_{n+1}^{-(0)} := y_n + hy_n' + (h^2/2)y_n'' + (h^3/6)y_n'''$ and $y_n^{+(0)} = y_n^-$, where the "derivatives" used to compute $y_{n+1}^{-(0)}$ are differences computed at the nodal values of the approximant. We need to compute these quantities for the error estimation, so there is no added expense. An alternate method is to replace the integrals in (2.4), (2.5) by available approximations over previous intervals.

To show that the superstability phenomenon can occur also for nonsuperstable schemes, we consider the *trapezoidal rule.* For (1.1), this is

$$y_{n+1} = y_n + \frac{h_n}{2}(f(t_n, y_n) + f(t_{n+1}, y_{n+1})), \tag{2.9}$$

where (2.9) is usually solved by Newton's method (or some variant), as before. We consider (2.9) as an interpolatory formula at t_n and t_{n+1}, and then it is natural to consider as initial guess for the Newton iteration the one provided by

$$y_{n+1}^{(0)} = y_n + h_n\left(f(t_{n-1}, y_{n-1})\left(1 - \frac{h_n + 2h_{n-1}}{2h_{n-1}}\right) + f(t_n, y_n)\left(\frac{h_n + 2h_{n-1}}{2h_{n-1}}\right)\right), \tag{2.10}$$

that is, we look at (2.9)–(2.10) as an Adam's pair. In our implementation of this scheme, we attempt to control the local truncation error, which can be estimated by comparing (2.1) and (2.9) as below:

$$|\text{local truncation error}| = \left|\frac{h_n^3}{12} y'''(\xi)\right| \simeq \left|\frac{h_n}{3(h_n + h_{n-1})}(y_{n+1} - y_{n+1}^{(0)})\right|. \tag{2.11}$$

3. Two case studies. A typical manifestation of superstability in a nonlinear problem with several solution branches is a computation that remains on a branch, which is not the desired (physical) solution. This usually happens in adaptive integration mode when a branch passes from being stable to being unstable without these being reflected in the stepsize control.

We now present an analysis of the two case studies (3.1) and (3.11) below. The first is a simplified version of a test problem used in [DESS] and in [D1], and we call

it by the same nickname, the *knee problem*:

$$\begin{aligned} &\varepsilon \dot{y} = y(y-t), \qquad -1 < t \leqq 1, \\ &y(-1) = -1, \qquad 0 < \varepsilon \ll 1. \end{aligned} \tag{3.1}$$

This has the solution

$$y(t) = -\frac{\varepsilon\, e^{-t^2/2\varepsilon}}{\int_{-1}^{t} e^{-s^2/2\varepsilon}\, ds + \varepsilon\, e^{-1/2\varepsilon}}.$$

Making an asymptotic expansion in this formula, we find that $y(0) \approx -\sqrt{\varepsilon}/(\sqrt{\varepsilon}\, e^{-\varepsilon/2} + \sqrt{\pi/2})$ (note the minus sign) and that the solution has a corner layer in an $O(\sqrt{\varepsilon})$ neighborhood of the origin. The solution with $\varepsilon = 10^{-4}$ is plotted in Fig. 2. It is useful to recognize the two reduced solutions $y_1 = 0$ (to all orders in ε) and $y_2(t) = t + \varepsilon/t - 2\varepsilon^2/t^3 - 2\varepsilon^3/t^5 + \cdots$ (valid away from the origin). It is easy to see that y_1 is unstable before the origin and stable after and that y_2 is stable before the origin and unstable afterwards. It is desirable that the approximation have the same dynamical behavior.

In Table 1 we summarize some results obtained with the BDF code LSODE. Since the magnitude of the solution to (3.1) is between 0 and 1, we request absolute local error control, LTOL, in Table 1. If LTOL is larger than ε, then the solution continues

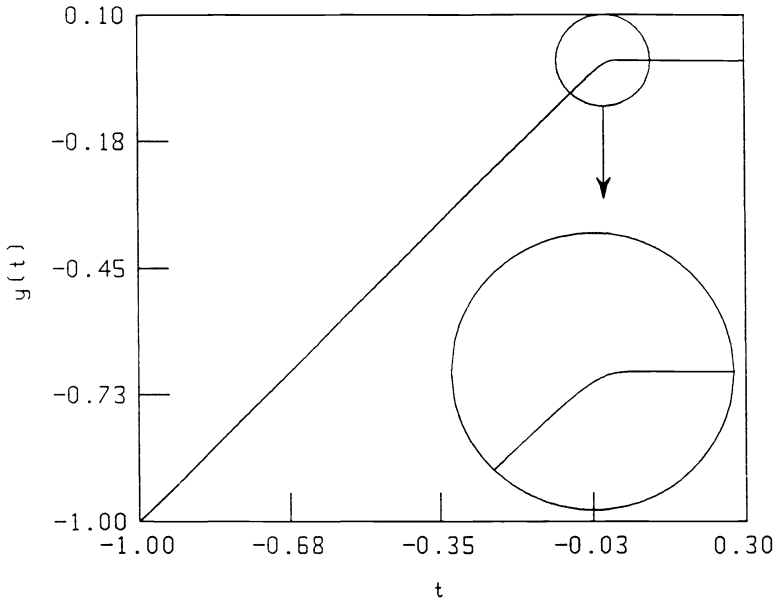


FIG. 2. *The knee problem.* $\varepsilon = 10^{-4}$.

TABLE 1
BDF *and local error.*

$\varepsilon = 10^{-p}$	LTOL $= 10^{-q}$, $q \in \mathbb{N}$	Value reached at $t = 1$
$p = 2, 4, 6$	$q < p$	$y_2(1)$
$p = 2, 4, 6$	$q \geqq p$	$y_1(1)$

along the y_2-branch. The stepsize is increased in the region before the transition and the transition region around the origin is missed; past the origin the formulas are stable and the stepsize remains quite large. Below we explain this behavior.

In Table 2, we give the results obtained with DG. Similar outcomes are obtained for other values of ε. For this problem, the global error control, together with DG, is accurate and recovers the behavior of the exact solution. On the other hand, these schemes yield a noticeably more conservative timestep sequence on this problem.

TABLE 2
DG *and global error.*

$\varepsilon = 10^{-p}$	GTOL	Value reached at $t=1$
$p=4$	10^{-2}	$y_1(1)$
$p=4$	10^{-1}	$y_1(1)$
$p=4$	.5	$y_1(1)$
$p=4$	1	$y_2(1)$

Ideally, if the change of stability is reflected in the timestep control, the steps should increase outside the transition region and remain small in the transition region. This is what happens with the third-order DG scheme and the BDF in the successful runs.

The following discussion is valid as a qualitative picture. There is a complex relationship between the stepsize control and the parameters in the problem (ε, the interval for $t, \cdots$) which makes it difficult to exactly quantify this discussion. However, we believe that the behavior is well depicted.

Analysis of a scheme with local error control. The BEM step for $t_n \to t_{n+1}$ on (3.1) is

$$\varepsilon y_{n+1} - h_n y_{n+1}(y_{n+1} - t_{n+1}) - \varepsilon y_n = 0. \tag{3.2}$$

Because we want to understand why the computed solution might not leave the reduced solution $y_2(t)$, we assume that at t_n, y_n is on $y_2(t)$. Because we want to understand what happens in this situation when the step is relatively large, we assume that $\varepsilon / t_n h_n \approx 0$, in which case the two roots of (3.2) can be expanded:

$$y_{n+1} = \frac{1}{2}\left\{\left(t_n + h_n + \frac{\varepsilon}{h_n}\right) \pm \left(\left|t_n + h_n + \frac{\varepsilon}{h_n}\right| - \frac{2(\varepsilon/h_n)t_n}{|t_n + h_n + \varepsilon/h_n|} + O(\varepsilon^2)\right)\right\}$$

or

$$y_{n+1} \simeq \begin{cases} y_{1,n+1} := O(\varepsilon), \\ y_{2,n+1} := t_{n+1} + O(\varepsilon). \end{cases} \tag{3.3}$$

Subsequently we drop the subscript on h_n for clarity's sake. Thus, the two solutions of (3.2) are close to the reduced solutions $y_1(t)$ and $y_2(t)$.

A standard way to solve the nonlinear equation (3.2) is by Newton's method with initial guess given by (2.2), which for (3.1) reads, $y_{n+1}^{(0)} = y_n + (h/\varepsilon) y_n (y_n - t_n)$. When $y_n = t_n + \varepsilon / t_n + \cdots$, this gives

$$y_{n+1}^{(0)} = t_{n+1} + \varepsilon\left(\frac{1}{t_n} + \frac{h}{t_n^2}\right) + \cdots. \tag{3.4}$$

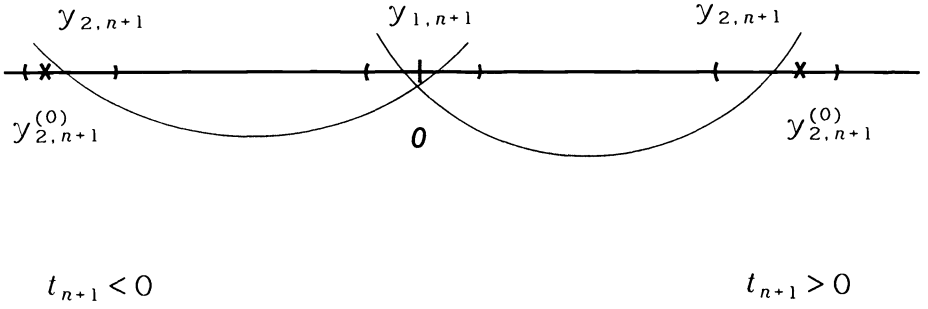
In the case where $t_{n+1} > 0$, Newton's method converges to $y_{2,n+1}$ if $y_{n+1}^{(0)} > \frac{1}{2}(t_{n+1} + \varepsilon/h)$ and to $y_{1,n+1}$ if $y_{n+1}^{(0)} < \frac{1}{2}(t_{n+1} + \varepsilon/h)$. Using (3.4) we conclude that Newton's method converges to $y_{2,n+1}$ if

$$\frac{t_{n+1}}{2} + \varepsilon\left(\frac{t_{n+1}}{t_n^2} - \frac{1}{2h}\right) > 0.$$

Similarly, if $t_{n+1} < 0$, then Newton's method converges to $y_{2,n+1}$ if

$$\frac{t_{n+1}}{2} + \varepsilon\left(\frac{t_{n+1}}{t_n^2} - \frac{1}{2h}\right) < 0.$$

Therefore, the initial guess for Newton's method is always close to the BEM solution $y_{2,n+1}$ in (3.3) and the iteration converges to the branch $y_2(t)$. Graphically, this situation is depicted below.



To summarize, if $h \gg \varepsilon$ and if at t_n the computation is on the $y_2(t)$ branch, then the next stop will remain on the $y_2(t)$ branch. If $h \leqq \varepsilon$, then this reasoning might break down.

We can think of problem (3.1)-(3.2) as generating a family of parabolas which have at least one root located near the origin. As t goes from -1 to 0, the left root of this parabola approaches the origin, then, for $t > 0$, it stays there. The root which was near the origin moves to the right for $t > 0$. Thus, the root located near the origin changes stability as t passes through 0. If $t_n < 0$ and $t_{n+1} > 0$, that is, h_n is large enough so the scheme jumps over the origin, then this picture tells us that the BEM converges to the wrong solution branch.

Next we show that the stepsizes can be increased dramatically in the time period before the transition region. The local error estimate is based on (2.3). An adaptive code uses this estimate both to check the current error and to predict a new (optimal) stepsize. Ignoring for a moment the upper limit that is usually placed on a step increase, if we require that the local error be kept at the local error tolerance LTOL, the new step ideally would be obtained from the implicit equation for $h_{\text{new}} = h_{n+1}$, $\frac{1}{2}|y_{n+2} - y_{n+2}^{(0)}| =$ LTOL. Here, the step from t_n to t_{n+1} and the check on the local error (2.3) have been made, so the quantity $\text{LO} := \frac{1}{2}|y_{n+1} - y_{n+1}^{(0)}|$ is available. Regardless of the success (LO $\leqq$ LTOL) or failure (LO > LTOL) of this last step, the next stepsize h_{new} will be a multiple of h_n, that is, $h_{\text{new}} = Ch_n$. Ideally, we will require that the new estimate, call it LN, of the local error obtained with h_{new}, be equal to LTOL. Since the local error is an $O(h^2)$ quantity, it is typically argued (as in the codes of [H]) that

$$\text{LN} \simeq \left(\frac{h_{\text{new}}}{h_n}\right)^2 \text{LO} \quad \text{or} \quad C = \sqrt{\frac{\text{LTOL}}{\text{LO}}}. \tag{3.5}$$

Under the previous assumptions after (3.3), (3.4), we can now estimate LO. Neglecting

terms of higher order in ε, we get

$$\mathrm{LO} \simeq \frac{\varepsilon}{2}\left|\frac{1}{(t_{n+1}+\varepsilon/h_n)} - \frac{t_{n+1}}{t_n^2}\right|.$$

Since ε/h_n is small, we have

$$\mathrm{LO} \simeq \frac{\varepsilon}{2}\left|\frac{(t_n+t_{n+1})h_n}{t_n^2 t_{n+1}}\right|. \tag{3.6}$$

From (3.6) we see that if we start at $t_0=-1$ with a small h_0, then $C \simeq \sqrt{\mathrm{LTOL}/\varepsilon h_0}$. We conclude that the stepsize will increase sharply for all LTOL unless LTOL is set unrealistically small (i.e., $O(\varepsilon h_0)$). In practice, the size increases are limited by some factor (usually 20 to 10 times) over a single step. Nonetheless, after only a few steps, h_n can become very large since it is clear from (3.6) that the stepsize selector will call repeatedly for a larger h_{new} as long as t_n and t_{n+1} are not too far from -1.

On the other hand, the ratio $\mathrm{LTOL}/\varepsilon$ determines whether the stepsizes will continue to be increased as the transition region is approached. If at some point both t_n and t_{n+1} are a small multiple of the current step from the origin, then using (3.6), we conclude that the steps will be increased or decreased more or less as the ratio $\sqrt{\mathrm{LTOL}/\varepsilon}$. If $\mathrm{LTOL}/\varepsilon \leqq 1$ and a small initial step is used, we would expect to see the initial step increased at the maximum rate at first, and then become constant or even decrease as the transition region is approached. If $\mathrm{LTOL}/\varepsilon > 1$, then the steps will be increased as we bypass the transition region.

If h_n is so large that the solution skips over the transition region in one step from t_n to t_{n+1}, then we have to check whether the large step will be accepted by the error criterion $\mathrm{LO} \leqq \mathrm{LTOL}$. With LO as estimated in (3.6), the step will be accepted if $\mathrm{LTOL} = c\varepsilon$, $c > 1$ a constant that depends on the distance of t_n and t_{n+1} to zero. In this problem, the change in stability of a solution is not reflected in the timestep control when large steps are used.

In our experiments with $\varepsilon = 10^{-4}$, we found that regardless of whether LTOL is less than or greater than ε, h is between .1 and .5 when the critical step from $t_n < 0$ to $t_{n+1} > 0$ is attempted. By (3.5), this step will be accepted if $\mathrm{LTOL} > 2\varepsilon$ and rejected otherwise. These results are reflected in Table 1. When this step is rejected and the stepsize is decreased, then the computation enters the transition region for the solution and our considerations no longer hold. Our experimentation suggests that in this case the automatic step selection process will handle the transition successfully. Also, the empirical evidence suggests that if we force the computation to step inside the transition region (forcing a step at $t=0$, say), then the computation usually follows the correct branch, even if $\mathrm{LTOL} > \varepsilon$.

Analysis of a scheme with global error control. The DG1 step for $t_n \to t_{n+1}$ on (3.1) is

$$\varepsilon y_{n+1} - h_n y_{n+1}\left(y_{n+1} - \frac{(t_{n+1}+t_n)}{2}\right) - \varepsilon y_n = 0. \tag{3.7}$$

If we again make the assumption that h is large compared to ε, then an analysis similar to that above shows that the roots of (3.7) lie close to $y_1(t)$ and $y_2(t)$. Using as initial guess the value of the "lagged" DG1: $y_{n+1}^{(0)} = y_n + (h/\varepsilon)y_n(y_n - (t_n + t_{n-1})/2)$, implies that the Newton iteration always converges to the root on $y_2(t)$. Thus, the DG formulas are susceptible to superstability, if large steps are taken.

However, for this problem, the global error control keeps the steps small. From Table 2, we see that DG1 implemented with (2.8) always performs accurately as a result. To use (2.8), we check whether

$$2h_n\|f_y\|_n\frac{|y_{n+1}-y_n|}{h_n}\leqq \text{GTOL} \tag{3.8}$$

using h_n predicted through

$$h_n=\frac{\text{GTOL}\,h_{n-1}}{2\|f_y\|_{n-1}|y_n-y_{n-1}|}. \tag{3.9}$$

In practice, we only compute values at meshpoints and so $\|f_y\|_{n-1}=\max(|f_y(y_{n-1})|,|f_y(y_n)|)$. Under the assumption that y_{n-1} and y_n are close to the $y_2(t)$ branch, this gives

$$h_n\simeq\frac{\varepsilon\,\text{GTOL}}{2\max(|f_n|,|f_{n-1}|)}. \tag{3.10}$$

Thus, the timesteps are very small and slowly increasing for times moderately far from zero. The computation follows the correct solution branch into the region of the layer where (3.10) is no longer valid. Of course, after zero, (3.9) essentially gives no bound on the stepsize and large steps are taken.

Using a conservative time sequence is one way to guarantee that a method correctly follows these types of internal layers. We consider a modification of the previous test case in order to demonstrate that the global error control does not always lead to a conservative sequence. Consider the *Modified Knee Problem*:

$$\begin{cases}\varepsilon\dot{y}=y^2+ty-\varepsilon^p, & p=\frac{1}{2},1,\frac{3}{2},2,\\ y(-1)=-\varepsilon\quad(\text{also } y(-1)=-\frac{1}{2}) & -1\leqq t\leqq 1,\quad 0<\varepsilon\ll 1.\end{cases} \tag{3.11}$$

A power series representation of the solution can be written down but it is not very insightful. Figures 3 and 4 are plots of the solution for some of the p listed. Again there are two reduced solutions

$$y_1(t)=\frac{\varepsilon^p}{t}-\varepsilon^{p+1}\frac{t+1}{t^3}+\text{h.o.t.},\quad \text{valid for all } p$$

and

$$y_2(t)=\begin{cases}-t-\varepsilon^p/t+\text{h.o.t.}, & p<1,\\ -t,\quad \text{all } \varepsilon, & p=1,\\ -t+\varepsilon/t+\text{h.o.t.}, & p>1.\end{cases}$$

$y_1(t)$ and $y_2(t)$ are stable before and past the origin, respectively. Tables 3 and 4 list the results for the BDF and for DG. Both classes of methods and error strategies can give incorrect results because the integrator steps over the transition region. Below, we analyze the first-order schemes for (3.11) with $p=1$ and 2. As before, the key point is the possible step over the origin.

Analysis of local error control. Assuming the computation is on the $y_1(t)$ branch, the forward Euler prediction step gives

$$y_{n+1}^{(0)}=\begin{cases}\dfrac{\varepsilon}{t_n}-h\dfrac{\varepsilon}{t_n}+\text{h.o.t.}, & p=1,\\ \dfrac{\varepsilon^2}{t_n}-h\dfrac{\varepsilon^2}{t_n}-h\dfrac{\varepsilon^2}{t_n^2}+\text{h.o.t.}, & p=2.\end{cases} \tag{3.12}$$

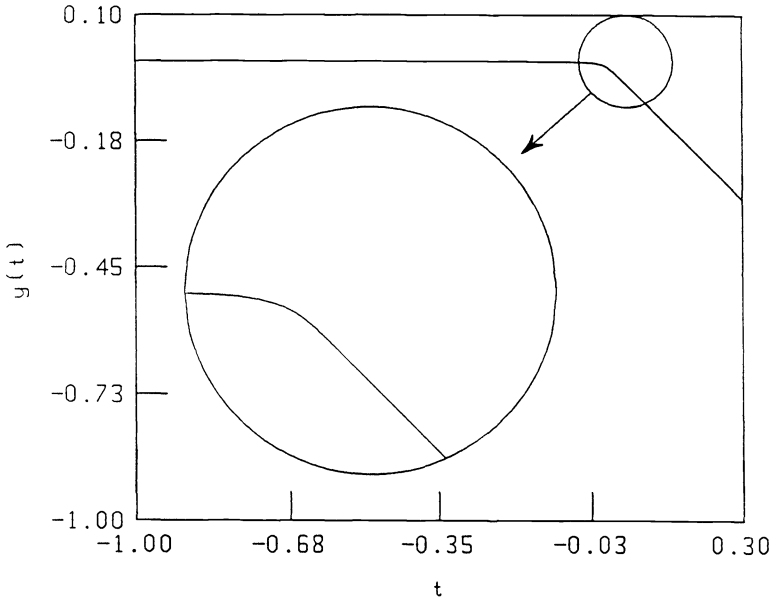


FIG. 3. *Modified knee problem.* $\varepsilon = 10^{-4}$, $p = 1$.

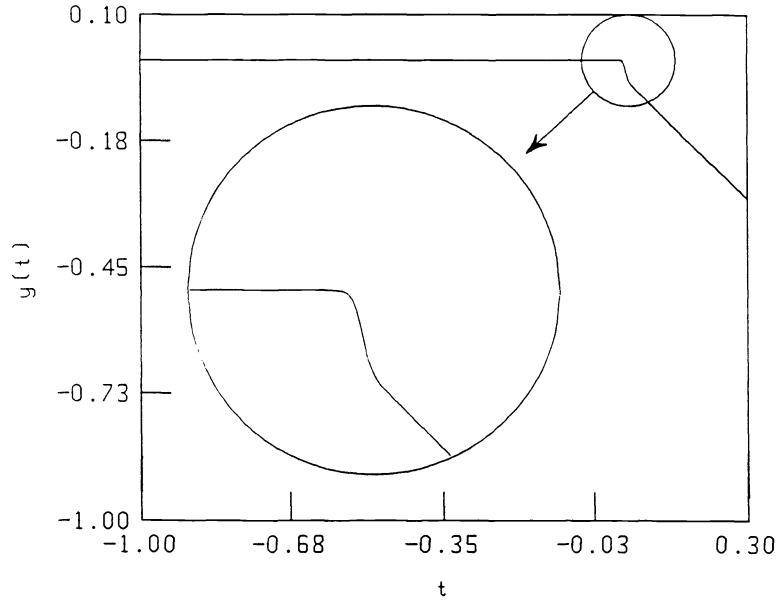


FIG. 4. *Modified knee problem.* $\varepsilon = 10^{-4}$, $p = 2$.

The backward Euler correction gives a quadratic equation with the two solutions

$$y_{n+1} = \begin{cases} \dfrac{\varepsilon}{t_{n+1}} + \text{h.o.t.} =: y_{1,n+1} \\ -t_{n+1} - \dfrac{\varepsilon}{t_{n+1}} + \dfrac{\varepsilon}{h} + \text{h.o.t.} =: y_{2,n+1} \end{cases} \quad \text{if } p = 1, \tag{3.13}$$

TABLE 3
BDF *and local error,* $\varepsilon = 10^{-4}$.

p	LTOL $= 10^{-q}$	Value reached at $t = 1$
$\frac{1}{2}$	$q > 2$	$y_2(1)$
1	$q < 4$	$y_1(1)$
	$q \geqq 4$	$y_2(1)$
$\frac{3}{2}$	$q < 6$	$y_1(1)$
	$q \geqq 6$	$y_2(1)$
2	$q < 8$	$y_1(1)$
	$q \geqq 8$	$y_2(1)$

TABLE 4
DG *and global error,* $\varepsilon = 10^{-4}$.

p	GTOL $= 10^{-q}$	Value reached at $t = 1$
$\frac{1}{2}$	$q \geqq 1$	$y_2(1)$
1	$q \geqq 1$	$y_2(1)$
$\frac{3}{2}$	$q \leqq 2$	$y_1(1)$
	$q \geqq 3$	$y_2(1)$
2	$q \leqq 4$	$y_1(1)$
	$q \geqq 5$	$y_2(1)$

$$y_{n+1} = \begin{cases} \dfrac{\varepsilon^2}{t_{n+1}} + \text{h.o.t.} =: y_{1,n+1} \\ -t_{n+1} + \dfrac{\varepsilon}{h} + \text{h.o.t.} =: y_{2,n+1} \end{cases} \qquad \text{if } p = 2. \tag{3.14}$$

Thus Newton's method for the BEM with initial guess given by (3.12) will converge to $y_{1,n+1}$ (reasoning in a similar way to the previous discussion). The local error control becomes

$$\begin{aligned} \frac{h_n \varepsilon}{2} \left| \frac{t_{n+1} - 1}{t_n t_{n+1}} \right| &\leqq \text{LTOL}, \qquad p = 1, \\ \frac{h_n \varepsilon^2}{2} \left| \frac{t_n^2 + h_n t_n + h_n}{t_n^2 t_{n+1}} \right| &\leqq \text{LTOL}, \qquad p = 2. \end{aligned} \tag{3.15}$$

It is clear that once again the stepsizes increase sharply in the first few steps unless LTOL is unrealistically small and that large steps will be used until the method is close to the layer. When t_n and t_{n+1} are within $O(h_n)$ from zero and h_n is large enough, (3.15) tells us that in this region the steps will increase or decrease approximately as (recall (3.5)) $\sqrt{\text{LTOL}/\varepsilon}$ and $\sqrt{\text{LTOL}/\varepsilon^2}$ for $p = 1$ and 2, respectively. Moreover, the integrator will accept or reject this large step over the origin based on (3.15) thus depending on the ratios LTOL$/\varepsilon$ and LTOL$/\varepsilon^2$ for $p = 1$ and 2, respectively, exactly as verified by the results of Table 3.

Analysis of a global error control. In this case, by using (3.8) to control the error, the stepsize h is chosen according to (3.9). If we are initially on the y_1-branch, this reduces to

$$h_n \simeq \text{GTOL}\, \varepsilon^{1-p} \min(|t_n|, |t_{n-1}|). \tag{3.16}$$

We see immediately that h will be always restricted if $p \leqq 1$, but if $p > 1$, h can grow very fast unless GTOL is small enough. For example, for $p = 2$ we would need GTOL $\leqq \varepsilon$ (see Table 4). Otherwise the integration remains on the y_1-branch.

Remarks. (1) Based on the results for our test problems, it is clear that the superstability phenomenon is not a result of our choice of error control. We stress that the integrators always preserve numerical accuracy. They simply check this accuracy for the wrong solution.

(2) We only discussed the first-order schemes. We have observed that the two classes of methods behave similarly to their first-order representatives. To illustrate, in Fig. 5 we show plots of the stepsizes chosen by the DG3 integrator for the modified knee problem. The solid and dashed lines represent superstable and correct solutions, respectively; the patterns of the stepsizes for these two cases are similar to each other.

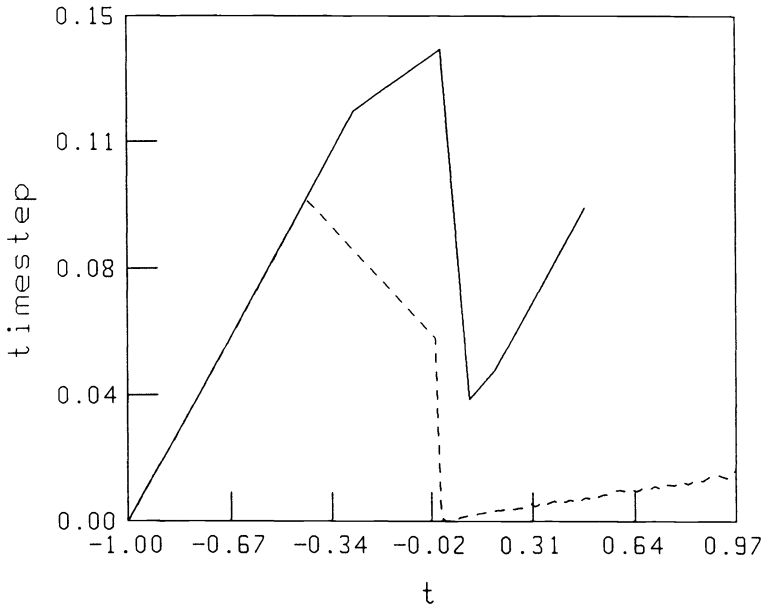


FIG. 5. *Modified knee problem.* $\varepsilon = 10^{-4}$, $p = 2$; GTOL $= 10^{-4}, 10^{-5}$.

(3) Another important aspect to mention in this context is a related one about codes which change order adaptively. We used the implicit Adams–Moulton formulas option provided by LSODE (as is well known, the stability regions of Adams's formulas, orders 2 to 12, do not extend into the right half-plane). By solving the nonlinear system with Newton's method and *not* with functional fixed point iteration, we observed qualitatively identical results to those summarized in Tables 1 and 3. What happens is that around the transition regions, the code always uses the order 1 formula, which is the BEM. The choice of order is logically dictated by the desire of economy and of as large steps as possible for comparable accuracy: a dangerous strategy to use for formulas with different stability characteristics.

At this point, it might appear that superstability is the byproduct of using superstable formulas. To show that this is not the case, we next report on the use of the trapezoidal rule (the Crank–Nicolson scheme), and show that superstability can occur in this case as well.

In Tables 5 and 6 we report on the results obtained for the knee problem (3.1) and modified knee problem (3.11), respectively. It is clear from these results that this scheme is prone to superstability as well. However, along a superstable branch, the scheme behaves differently from the superstable schemes: the stepsizes are never further increased, roundoff errors are not damped out, and the solution betrays an unstable behavior; on several runs, this instability leads to a rapidly growing solution and subsequent failure of the integration. We call this a "delayed effect," since it is sensed after the integration has switched on the wrong branch, often much after $t=1$ (the point we originally wanted to reach). Below we interpret the results of Tables 5 and 6, in a similar way to how we did for the first-order schemes. We only report on the analysis for the modified knee-problem (3.11) and $p=1,2$.

Analysis of the trapezoidal rule with local error control. Assuming that the computation is on the $y_1(t)$ branch, the prediction step gives (after some manipulation)

$$y_{n+1}^{(0)} = \begin{cases} \dfrac{\varepsilon}{t_n} + \varepsilon \dfrac{h_n}{t_{n-1}} \left(\dfrac{h_n + 2h_{n-1}}{2t_n} - 1 \right) + O(\varepsilon^2) & \text{if } p=1, \\ \dfrac{\varepsilon^2}{t_n} + \varepsilon^2 h_n \left[\dfrac{h_n}{2h_{n-1}} \dfrac{t_{n-1}+1}{t_{n-1}^2} - \left(\dfrac{h_n}{2h_{n-1}} + 1 \right) \dfrac{t_n+1}{t_n^2} \right] + O(\varepsilon^3) & \text{if } p=2. \end{cases} \tag{3.17}$$

Using analysis similar to the previous one, the trapezoidal rule step (2.9) gives a quadratic equation with two solutions, which can be written as follows:

$$y_{n+1} = \begin{cases} -\dfrac{\varepsilon}{|t_{n+1}|} + \text{h.o.t.} =: y_{1,n+1} \\ -t_{n+1} + \dfrac{\varepsilon}{h_n} + \dfrac{\varepsilon}{|t_{n+1}|} + \text{h.o.t.} =: y_{2,n+1} \end{cases} \quad \text{if } p=1, \tag{3.18}$$

$$y_{n+1} = \begin{cases} -\dfrac{\varepsilon^2}{|t_{n+1}|} + \text{h.o.t.} =: y_{1,n+1} \\ -t_{n+1} + \dfrac{\varepsilon}{h_n} + \text{h.o.t.} =: y_{2,n+1} \end{cases} \quad \text{if } p=2. \tag{3.19}$$

TABLE 5
(3.1) + *trapezoidal rule.*

$\varepsilon = 10^{-l}$	LTOL $= 10^{-q}$, $q \in \mathbb{N}$	Outcome
$l=3$	$q \geqq 2$	Correct solutions
$l=5,6$	$q \leqq 3$	Superstable and fails
$l=5,6$	$q \geqq 4$	Correct solution

TABLE 6
(3.11) + *trapezoidal rule*: $\varepsilon = 10^{-4}$.

p	LTOL $= 10^{-q}$, $q \in \mathbb{N}$	Outcome
1	$q \leqq 3$	Superstable or fails
	$q \geqq 4$	Correct
1.5	$q \leqq 5$	Superstable or fails
	$q \geqq 6$	Correct
2	$q \leqq 6$	Superstable or fails
	$q \geqq 7$	Correct

Once again, we conclude that Newton's method for the trapezoidal rule with starting value given by (3.17) will converge to $y_{1,n+1}$. Clearly, this can be undesirable when the key step we are analyzing is the possible one over the transition region. To check whether this is accepted or refused, we must look at the local error control. For this problem, the error control based on (2.11) reads

$$\begin{aligned}
&\left|\frac{\varepsilon h_n^2}{3(h_n+h_{n-1})}\left(-\frac{1}{t_n t_{n+1}}+\frac{h_n}{t_{n-1}}\,\frac{t_{n-1}-h_n-2h_{n-1}}{2t_n}\right)\right| \leqq \mathrm{LTOL}, \qquad p=1,\\
&\left|\frac{\varepsilon^2 h_n^2}{3(h_n+h_{n-1})}\left(-\frac{1}{t_n t_{n+1}}+\frac{h_n}{2h_{n-1}}\,\frac{t_{n-1}+1}{t_{n-1}^2}-\left(\frac{h_n}{2h_{n-1}}+1\right)\frac{t_n+1}{t_n^2}\right)\right| \leqq \mathrm{LTOL}, \qquad p=2.
\end{aligned} \tag{3.20}$$

From (3.20), we conclude that the stepsizes are allowed to increase initially, unless LTOL is extremely small. Since, as it is done in our implementation, we have $\frac{1}{2}h_{n-1} \leqq h_n \leqq 2h_{n-1}$, when t_n, t_{n+1}, and t_{n-1} are all $O(h_n)$ quantities and h_n is large enough, (3.20) become

$$\frac{2}{9}\frac{\varepsilon}{h_n} \leqq \mathrm{LTOL}, \qquad p=1,$$

$$\frac{1}{6}\frac{\varepsilon^2}{h_n} \leqq \mathrm{LTOL}, \qquad p=2.$$

Therefore, the step from t_n to t_{n+1} (the one over the origin) will be accepted if $h_n \geqq \frac{2}{9}(\varepsilon/\mathrm{LTOL})$, for $p=1$, or $h_n \geqq \frac{1}{6}(\varepsilon^2/\mathrm{LTOL})$ for $p=2$. For the superstable runs in Tables 5 and 6, we have h between .4 and .8 when this step is attempted, thus confirming this above result.

Final remarks. (1) Remarks (1)-(3) after (3.16), and the above analysis of the trapezoidal rule, suggest the following reinterpretation of the basic issue. There are three problems involved when numerically solving a differential equation by an implicit scheme: (i) the given continuous problem, (ii) the discretized problem, (iii) the iterative method used to solve (ii). We think that (ii) by itself does not suffice to describe the dynamical behavior of the discrete problem, and that we must consider (ii) and (iii) as a unique process. In this case, we realize that the iterative solution process for the nonlinear system might superimpose its own dynamical portrait on that of the given problem. In particular, convergence of Newton's method (assuming enough smoothness) depends upon how close the initial guess is to the root, but *not at all* on whether this root is along a stable trajectory. Our analysis shows that superstability can occur because of this fact. Omitting a consideration of (iii) can give a faulty analysis and incorrect interpretation of the numerical results.

(2) For the simple test problems above, it is easy to give criteria to detect the superstability occurrence (e.g., check the sign of f). But, in general, this is not easy to do. Various possibilities are the following: (i) Check local stability of the computed solution trajectory by computing the eigenvalues of f_y. This is generally very expensive and probably not practical (an exception is for differential Riccati equations [D2]). Alternatively, investigate the stability of the computed solution by attempting to correctly solve the linearized problem all over. This is again very expensive and it is not totally transparent how to proceed in practice. (ii) Determine the initial guess for Newton's method in a different way; we have experimented with the "naive" choice $y_{n+1}^{(0)} = y_n$, rather than extrapolating forward from past values. This choice adds considerably to the expense and, moreover, it is not totally reliable. For example, it helps to avoid some of the superstability occurrences listed in Tables 1 and 3, but none of

those of Table 4. (iii) Use a different iteration than Newton's method to solve the nonlinear algebraic system. In particular, choose an iteration whose convergence to a fixed point depends upon the (dynamical) stability of the fixed point. A possible choice is to use functional fixed point iteration. As implemented in LSODE, this has been very successful. However, this restricts the stepsize everywhere in order to achieve convergence, which is what we were trying to avoid in the first place.

(3) It is possible that changing the implementations of the schemes we have analyzed, and/or the strategies for the error control, could lessen the potential for superstability. However, we think that the difficulties we have discussed are not peculiar to our choice of schemes nor to the adopted strategies. Different schemes require an analysis on their own. But any analysis which is based upon numerical accuracy is potentially faulty.

4. The method of lines for partial differential equations and superstability. For the two case studies of § 3 it was possible to follow in some detail the interaction of error control, stepsize selection, and superstability occurrence. In general, however, this is very difficult. In particular, it is so for large stiff nonlinear systems such as those arising from using the method of lines (NMOL) to solve parabolic problems. For example, solving this well-posed problem:

$$\begin{cases} au_t - u_{xx} = b[u(u-t)+a], \quad t\in[-5,5], \quad x\in[0,1], \\ \text{Dirichlet or Neumann BCs}, \end{cases} \tag{4.1}$$

with the NMOL and adaptive time integration behaves similarly to the knee problem (3.1). We have a reduced solution trajectory that is initially very stable and that loses stability when crossing another reduced solution, which thereafter becomes stable.

A more interesting problem where, in addition to the usual goal of accuracy over a short time, numerical methods should reproduce the physical characteristics of the model quantity, is given by

$$\begin{cases} u_t - u_{xx} = \lambda(u-u^3) =: \lambda f(u), \qquad x\in[0,1], \quad t>0, \quad \lambda \geqq 0 \\ \text{Dirichlet or Neumann boundary conditions} \\ u(x,0) = u_0(x), \qquad x\in[0,1]. \end{cases} \tag{4.2}$$

With Neumann conditions [FH], this is a model for the mixing of two phases of a substance at the transition temperature, and with Dirichlet conditions [C] of the temperature in an electric ballast resistor with constant current. In either case, the solution has rich dynamical behavior, which increases in complexity as λ increases.

Stability information about a steady-state solution of (4.2) is gained by studying the eigenvalues of the (Sturm–Liouville) BVP obtained by linearizing the equation in (4.2) (with $u_t = 0$) around such a solution. For example, around $\bar{U} \equiv 0$ with $\lambda = 0$, the eigenvalues are all real and nonnegative, $\sigma_l = (l\pi)^2$, $l = 0, 1, 2, \cdots$, indicating that 0 is the stable fixed point. When λ is positive, the spectrum is shifted to the left by λ and 0 becomes unstable. (At the same time, $\bar{U} \equiv 1$ and $\bar{U} \equiv -1$ become stable.) As λ crosses the values $(l\pi)^2$, unstable steady-state solutions bifurcate from $\bar{U} \equiv 0$. For this problem, it is critical that unstable steady-state solution branches be unstable for the numerical method as well.

The experiments on which we report below show that an unstable steady-state solution of (4.2), with Neumann or Dirichlet boundary conditions, can be numerically stable due to the superstability phenomenon, when the BDF and DG methods are used in an NMOL discretization of (4.2). This obviously leads to disastrous physical

interpretations. We choose λ and the number of lines, and we study the behavior of solutions with $u_0(x)$ "close" to zero.

The stability of the numerical method depends on the eigenvalues of the Jacobian of the IVP system. By using a piecewise linear, continuous finite element in space with a lumped mass quadrature, then the NMOL becomes

$$B\begin{pmatrix} u_0 \\ \vdots \\ u_m \end{pmatrix}_t - A\begin{pmatrix} u_0 \\ \vdots \\ u_m \end{pmatrix} = \lambda B\begin{pmatrix} f(u_0) \\ \vdots \\ f(u_m) \end{pmatrix}, \qquad t>0,$$

$$B\begin{pmatrix} u_0 \\ \vdots \\ u_m \end{pmatrix}_{t=0} = \begin{pmatrix} u_0(x_0) \\ \vdots \\ u_m(x_m) \end{pmatrix}, \tag{4.3}$$

where $u_0(t), \cdots, u_m(t)$ are the nodal values of the second-order approximant $U(t)$ to u, B is the approximate mass matrix, and A is the stiffness matrix. For a fixed uniform mesh and Neumann boundary conditions (similar reasoning holds for Dirichlet conditions), we have

$$B = h\begin{bmatrix} \frac{1}{2} & & & \\ & 1 & & 0 \\ & & \ddots & \\ & 0 & & 1 \\ & & & & \frac{1}{2} \end{bmatrix}, \quad A = \frac{1}{h}\begin{bmatrix} -1+1 & & & \\ +1-2+1 & & & 0 \\ & & \ddots & \\ & 0 & & +1-2+1 \\ & & & +1-1 \end{bmatrix}, \quad h := \frac{1}{m}.$$

For a steady-state solution $\bar{U} = (\bar{u}_0, \cdots . \bar{u}_m)^T$, the Jacobian is

$$B^{-1}A + \lambda F'(\bar{U}) \tag{4.4}$$

where $F'(\bar{U}) := (f'(\bar{u}_0), \cdots, f'(\bar{u}_m))^T$. All the eigenvalues of this Jacobian, call them σ_l^*, are real and simple and approximate the corresponding eigenvalues of the continuous, linearized BVP. For $\bar{U} \equiv 0$, the error in the numeric eigenvalues is bounded (for h small) by

$$|\sigma_l^* - \sigma_l| \leqq Ch^2 (\sigma_l + \lambda)^2, \tag{4.5}$$

for some constant C. We conclude that for $\lambda > 0$, there are approximately $\lfloor \sqrt{\lambda}/\pi \rfloor$ positive eigenvalues, including one within Ch^2 of λ, and hence that $\bar{U} = 0$ is unstable for $\lambda > 0$. In the same way, we can check that all the eigenvalues associated to $\bar{U} = 1$ and $\bar{U} = -1$ (again, for Neumann conditions) are negative.

In the following results for Neumann conditions, we have set $\lambda = 100$, used 50 lines, and let $u_0(x) = \alpha > 0$, a small constant. For this problem, the solution should monotonically increase to the stable fixed point $u = 1$. From Tables 7 and 8, we see that superstability of the formulas might affect the results for both the BDF and DG

TABLE 7
BDF *and local error control.*

α	LTOL $= 10^{-q}$, $q \in \mathbb{N}$	Converges to 0
10^{-4}	$q < 3$	yes
10^{-4}	$q \geqq 3$	no
10^{-5}	$q < 4$	yes
10^{-5}	$q \geqq 4$	no

TABLE 8
DG *and global error control.*

α	GTOL $= 10^{-q}$, $q \in \mathbb{N}$	Converges to 0
10^{-4}	$q < 5$	yes
10^{-4}	$q \geqq 5$	no
10^{-5}	$q < 6$	yes
10^{-5}	$q \geqq 6$	no

method. The next results also use Neumann conditions, with $\lambda = 100$, and 50 lines. The initial profile is $u_0(x) = \alpha \cos(\nu \pi x)$, for several values of the amplitude α. The chosen values for the frequencies ν ensure that the positive eigenvalues have a nonzero component in the initial profile. In Tables 9 and 10, we report on some typical results obtained with $\alpha = 10^{-5}$. Again superstability is apparent, and we remark that the global error control fails for all realistic values of GTOL.

Remarks. (1) For constant initial data, the analysis of the scalar test problems can be repeated to show that the outcome of superstability for (4.3) is determined by the same factors as previously noticed.

(2) Our experiments show that varying the error tolerance affects the patterns of the flow observed. For example, the exact times of transition from a pattern to another, the sequence of patterns observed, and whether we reach the fixed point 1 or -1 all change for different tolerance ranges. However, there might be other factors which cause this outcome besides superstability, and we plan to study this in the future.

5. Conclusions. The phenomenon of superstability in the adaptive integration of stiff initial value problems is characterized by the numerical integrator not detecting (or detecting too late) that the underlying solution is unstable. All the schemes and error strategies we have considered, to different extents, are affected by the superstability occurrence. Our analysis shows that the iterative solution process for the nonlinear algebraic system is a key factor in provoking this occurrence.

TABLE 9
BDF *and local error control.*

ν	LTOL $= 10^{-q}$, $q \in \mathbb{N}$	converges to 0
52	$q = 3, 4$	yes
52	$q = 8$	no
2	$q = 4$	yes
2	$q = 5$	no

TABLE 10
DG *and global error control.*

ν	GTOL $= 10^{-q}$, $q \in \mathbb{N}$	Converges to 0
52	$q = 5$	yes
52	$q = 6$	no
2	$q < 5$	yes
2	$q \geqq 5$	no

For this reason, unless an explanation of the interaction between the iterative solution of the nonlinear system and the chosen integration scheme is provided, the numerical results of many experiments should be treated with some skepticism. We think that error control strategies for stiff initial value problems ought to be redesigned to take into account stability information of the continuous problem. This is not yet available, and we plan to consider this problem in future extensions of this work.

REFERENCES

[C] N. CHAFEE, *The electric ballast resistor: Homogeneous and nonhomogeneous equilibria*, in Nonlinear Differential Equations: Invariance, Stability and Bifurcation, P. De Motoni and L. Salvadori, eds., Academic Press, New York, 1981, pp. 97-127.

[DESS] G. DAHLQUIST, L. EDSBERG, G. SKOLLERMO, AND G. SODERLIND, *Are the numerical methods and software satisfactory for chemical kinetics?*, in Numerical Integration of DE and Large Linear Systems, J. Hinze, ed., Springer-Verlag, Berlin, New York, 1982, pp. 149-164.

[DHT] M. DELFOUR, W. HAGER, AND F. TROCHU, *Discontinuous Galerkin methods for ordinary differential equations*, Math. Comp., 36 (1981), pp. 455-473.

[D1] L. DIECI, *Some aspects of using* IV *software to solve* BVP *via a Riccati transformation*, Appl. Math. Comput., 34 (1989), pp. 463-472.

[D2] ———, *Numerical integration of the differential Riccati equation and some related issues*, SIAM J. Numer. Anal., 1992, to appear.

[EJ] K. ERIKSSON AND C. JOHNSON, *Adaptive finite element methods for parabolic problems* I: *A linear model problem*, Preprint 1988-31, Department of Mathematics, Chalmers University of Technology, Göteborg, Sweden.

[EJT] K. ERIKSSON, C. JOHNSON, AND V. THOMÉE, *Time discretization of parabolic problems by the discontinuous Galerkin method*, RAIRO Model. Math. Anal. Numér., 19 (1985), pp. 611-643.

[EL1] D. ESTEP AND S. LARSSON, *Time discretization of semilinear parabolic problems by the discontinuous Galerkin method*, CDSNLS Tech. Report, Georgia Institute of Technology, Atlanta, GA, 1990.

[EL2] ———, *Fully discrete discontinuous Galerkin methods for semilinear parabolic problems*, in preparation.

[FH] G. FUSCO AND J. K. HALE, *Slow-motion manifolds, dormant instability, and singular perturbations*, J. Dynamics Differential Equations, 1 (1989), pp. 75-94.

[G] C. W. GEAR, *Numerical Initial Value Problems in* ODEs, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[H] A. C. HINDMARSH, ODEPACK, *A systematized collection of* ODE *solvers*, in Scientific Computing, R. S. Steepleman, ed., North-Holland, Amsterdam, 1983, pp. 55-64.

[J] C. JOHNSON, *Error estimates and adaptive time step control for a class of one step methods for stiff ordinary differential equations*, SIAM J. Numer. Anal., 25 (1988), pp. 908-926.

[L] B. LINDBERG, *On a dangerous property of methods for stiff differential equations*, BIT, 14 (1974), pp. 430-436.

[SG] L. F. SHAMPINE AND C. W. GEAR, *A user's view of solving stiff ordinary differential equations*, SIAM Rev., 21 (1979), pp. 1-17.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1304-1313, November 1991
© 1991 Society for Industrial and Applied Mathematics
004

COMPUTATION OF THIN-PLATE SPLINES*

ROBIN SIBSON† AND G. STONE†‡

Abstract. Thin-plate splines are an attractive method for interpolating and smoothing arbitrarily spaced points in the plane. A major problem in the application of thin-plate splines is that their computation involves the solution of a linear system that is ill-conditioned for large data sets. A technique for preconditioning the system is proposed that greatly extends the usefulness of thin-plate splines.

Key words. bivariate splines, thin-plate splines, spatial regression, nonparametric regression, ill-conditioned matrices, preconditioning, Dirichlet tessellation

AMS(MOS) subject classifications. 65U05, 65D07

1. Introduction.

1.1. Background. Splines have been used in a wide variety of applications for both interpolation and smoothing. Data analysts have found cubic splines to be a flexible and efficient alternative to parametric regression functions. Silverman (1985) gives an account of nonparametric regression based on cubic splines, and full details and many references may be found in his paper.

The success of the cubic spline approach in the one-dimensional case suggests that higher-dimensional extensions of it should be sought. In this paper we concentrate on a particular variational generalisation, one that provides a regression method independent of choice of origin and coordinate axes. The technique is therefore suitable for application to spatial regression problems. There are many papers on this topic in the literature; major references on the general problem are Duchon (1976a), (1976b) and Meinguet (1979a), (1979b). The extension to two dimensions has been carried through in a fairly direct way, to produce the method of *thin-plate splines*, discussed in a data-analytic context, including a treatment of smoothing, by Wahba (1979a); followed up by Wahba (1979b), (1980a), (1980b), (1981), (1983); Dyn and Wahba (1979); Wahba and Wendelberger (1980); Wendelberger (1981), (1982); and Bates and Wahba (1983).

There are three major differences between one-dimensional cubic splines and their two-dimensional counterparts, and one of these makes the computation much more laborious in two dimensions. It is the improvement of the computational procedures involved that concerns us in this paper.

1.2. Notation and mathematical formulation. We consider initially the generalisation to K dimensions. We shall use the term *data sites* for the points in K-space at which observations have been made, to emphasise that it may not be possible to exercise complete control over what these positions actually are. The data sites will be denoted as column coordinate vectors in some particular coordinate system by u_n for $n = 1, \cdots, N$. The observed values at the data sites will be called *data values*, and will be denoted by Z_n. Denote by z_n the fitted values at the data sites; then the usual

* Received by the editors December 4, 1989; accepted for publication (in revised form) September 7, 1990. This research was partially supported by a research grant from the United Kingdom Science and Engineering Research Council.

† School of Mathematical Sciences, University of Bath, Claverton Down, Bath, BA2 7AY, United Kingdom.

‡ The work of this author was done while he was supported by a Science and Engineering Research Council Research Studentship.

residual-sum-of-squares error penalty is

$$E = \sum_{n=1}^{N} (Z_n - z_n)^2, \tag{1.2.1}$$

which arises in the same form and for the same reasons as in many other contexts, and might similarly be generalised.

A typical function on K-space will be denoted by z, and if the fitted values arise from such a function then z_n is simply $z(u_n)$. In one dimension, following the usual discussion of the cubic spline, the roughness of such a function is the squared second derivative, integrated over the whole line. This situation corresponds to a well-known physical model for the deformation of a thin stiff wire. In higher dimensions, a direct generalisation is obtained as follows. Write

$$R(u) = \sum_{i=1}^{K} \sum_{j=1}^{K} \left[\frac{\partial^2 z}{\partial u^i \, \partial u^j} (u) \right]^2 \tag{1.2.2}$$

and call this the *local roughness* at u, and then define the *roughness* to be

$$R = \int_{R^K} R(u) \, du, \tag{1.2.3}$$

where the notation is to be understood as implying a K-dimensional multiple integral.

Note that, although the second derivative (Hessian) matrix is symmetric, off-diagonal terms have to be counted both above and below the diagonal to achieve rotational invariance for the local roughness. The roughness then has the essential property of invariance under the choice of a coordinate system. In two dimensions this penalty again arises as the strain energy in a physical model, in this case for the infinitesimal deformation of a thin stiff lamina. It is from this model that the name "thin-plate splines" derives.

The interpolation problem is the problem of minimising the roughness penalty R subject to the exact interpolation conditions $z(u_n) = Z_n$. For the smoothing problem, a total penalty $E + \omega R$ (where $\omega > 0$ is the *smoothing parameter*) has to be minimised without constraints.

1.3. General form of the solution. It can be shown that the appropriate solution to the above variational problem has the form

$$z(u) = a + b^T u + \sum_{n=1}^{N} c_n \varphi(u - u_n), \tag{1.3.1}$$

where φ is a fundamental solution, given (see Meinguet (1979b)) by

$$\varphi(u) = \begin{cases} (12)^{-1} |u|^3 & \text{in one dimension,} \\ (16\pi)^{-1} \|u\|^2 \log \|u\|^2 & \text{in two dimensions,} \\ (-8\pi)^{-1} \|u\| & \text{in three dimensions.} \end{cases} \tag{1.3.2}$$

In each case, the definition of the fundamental solution is for the whole space excluding the origin, but may be extended by continuity to the whole space. As a basis for nonparametric regression, the variational problem from which the one-dimensional cubic spline arises accordingly generalises satisfactorily to two dimensions, but with (as the first major difference we identify) a drop from twice to once continuous differentiability in the solution. In three dimensions the solution is no better than continuous, and is not differentiable at the data sites. The generalisation to three

dimensions is accordingly unacceptable as the basis for a data-analytic smoothing or interpolation method. In four or more dimensions the generalisation collapses completely, because the fundamental solution cannot be extended to the origin by continuity.

The only suggestion known to us for making the higher-dimensional extensions of the variational problem well-posed is to modify the roughness penalty by replacing the second derivatives by higher-order derivatives. Wahba and Wendelberger (1980) give an example of the practical use of such functions in a meteorological problem. The difficulty with this approach is that the class of penalty-free functions is extended to include polynomials of degree greater than one, and it seems to us to be unsatisfactory not to penalise such functions as possessing some degree of roughness. Possibly a generalisation involving second- and higher-order derivatives together in the roughness penalty might provide what is wanted, but as far as we are aware this approach is unexplored.

The two-dimensional case accordingly has a special status as the only immediately acceptable generalisation to higher dimensions of the variational problem yielding the cubic spline in one dimension, and the remainder of this paper is concerned exclusively with bivariate splines.

2. Thin-plate splines over the whole plane.

2.1. The linear system. In the bivariate case the expression (1.3.1) has the form

$$z(u) = a + b^T u + \sum_{n=1}^{N} c_n (16\pi)^{-1} \|u - u_n\|^2 \log \|u - u_n\|^2 \tag{2.1.1}$$

with $(N+3)$ parameters. Where convenient, we can write $u = [x\, y]^T$, $u_n = [x_n, y_n]^T$. For interpolation, the parameters in (2.1.1) are determined by the interpolation conditions

$$z(u_m) = Z_m \quad \text{for } m = 1, \cdots, N \tag{2.1.2}$$

and by the three additional equations

$$\sum c_n = 0, \quad \sum c_n x_n = 0, \quad \sum c_n y_n = 0, \tag{2.1.3}$$

which arise from boundary conditions. Equations (2.1.2) and (2.1.3) may be written in the form

$$\begin{bmatrix} K & T \\ T^T & 0 \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} Z \\ 0 \end{bmatrix}, \tag{2.1.4}$$

where K is an $N \times N$ matrix whose (m, n)th entry is $(16\pi)^{-1} \|u_m - u_n\|^2 \log \|u_m - u_n\|^2$, and that is accordingly symmetric, where T is an $N \times 3$ matrix whose nth row is $[1\ x_n\ y_n]$, and where $d = [a\, b^T]^T$. The entire system is thus symmetric. The system of equations arising for the smoothing problem is similar:

$$\begin{bmatrix} K + \rho I_N & T \\ T^T & 0 \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} Z \\ 0 \end{bmatrix}, \tag{2.1.5}$$

where I_N is the $N \times N$ unit matrix, and $\rho = N\omega$. The scale on which the smoothing parameter is measured is seldom of interest, so a fairly cavalier attitude towards the choice of scale is harmless.

2.2. The use of B-splines. The analogous system in one dimension is of course of size $(N+2)$, and the boundary conditions are equivalent to a requirement of first-degree tail-out at either end of the range of the data sites. The structure of the system, in the interpolation case in both one and two dimensions, is that the diagonal entries are zero, and all off-diagonal entries in the main $N \times N$ block are in general nonzero

provided that data sites are distinct. In one dimension the solution procedure always takes advantage of some transformation which avoids this problem of nonsparsity, for example a change of basis from the original fundamental solutions to a basis of B-splines. The solution procedure, as described by Silverman (1985), depends only linearly on the number of data sites.

The terminology of "B-splines" has been applied to a number of constructions of functions of compact support over the plane. These appear to be either tensor products of univariate B-splines (de Boor (1978), Hu and Schumaker (1986)), or piecewise polynomial functions (Dahmen (1980), (1981)); they are certainly not special thin-plate splines in the way that univariate B-splines are special cubic splines.

In fact, thin-plate B-splines in the true sense do not exist; that is, a thin-plate spline cannot have compact support unless it is identically zero. This result, which is an immediate consequence of familiar theorems of analysis, applies to all types of "radial basis function" as reviewed generally by Powell (1985) and discussed specifically by Hardy (1971), Duchon (1976a), and Dyn and Levin (1983). This is the second major difference which we identify between cubic splines and thin-plate splines. It is not essential to have basis functions of compact support to achieve a sparse linear system, but in their absence it is necessary to build basis functions which are zero at all but a few data sites. This is possible, but is equivalent to solving the interpolation problem with a selection of specially chosen sets of data values, and thus represents a reformulation of the problem rather than a solution of it.

The nonexistence of B-splines, and the implication that there will be no effective way to reformulate the linear system (2.1.4) or (2.1.5) in sparse form, is the second major difference encountered in the move from one to two dimensions. It fundamentally alters the computational character of the problem. The third major difference arises from the imposition of a general "window" Ω in place of the whole of the line or plane as the domain of integration for the local roughness (Dyn and Levin (1982)). It is well known that in one dimension Ω may be taken as any interval containing all the data sites without altering the problem. For reasons similar to those preventing the construction of B-splines, in two dimensions the choice of window always affects the problem. We shall discuss the computational and practical implications of this in a further paper, but in this paper we limit discussion solely to the case where Ω is the whole plane.

2.3. Solution procedure for the linear system. The interpolation system (2.1.4) is the special case of (2.1.5) with $\rho = 0$. Wahba (1979a) deals with the more general problem which arises in an arbitrary number of dimensions and with a higher order of differentiation; in such a case the matrix corresponding to T has in general more than three columns. Wahba suggests the following procedure. First, construct the projection matrix $(I_N - T(T^TT)^{-1}T^T)$ and find its spectral decomposition. The eigenvectors corresponding to the unit eigenvalues form an orthonormal basis for the orthogonal complement of the space spanned by the columns of T.

In the specific case of interest here, that of the true thin-plate spline in two dimensions where T has three columns, a much more efficient procedure is available, as follows. First find a large triangle of data sites to serve as a reference triangle. It need not actually have maximal area, and a good enough algorithm is to start with any triangle of data sites and successively replace each vertex by that data site which maximises the area of the triangle which it makes with the other two, until no further improvement can be made. Without loss of generality, these data sites may be renumbered as 1, 2, 3. Next, construct vectors corresponding to the data sites $4, \cdots, N$ which

are not vertices of the reference triangle. The vector arising from data site n has entry -1 in position n and its entries in positions 1, 2, and 3 are the homogeneous coordinates obtained when data site n is expressed as an affine combination of the three data sites selected to form the reference triangle. All its remaining entries are zero.

All such vectors automatically lie in the orthogonal complement of the subspace spanned by the columns of T, and are independent, and there are $(N-3)$ of them, so they form a basis for it. The usual Gram-Schmidt procedure can be applied to orthonormalise this basis very efficiently. The relevant inner products are sums of four terms only, except at the normalisation step. The column vectors so determined form an $N\times(N-3)$ matrix which we shall denote by Q. Wendelberger (1981) suggested a rather different modification of Wahba's original procedure which has a similar objective to ours, of avoiding a spectral decomposition step at this stage, but which is designed to cope with the general case rather than to be very efficient for the true thin-plate case.

At this point we rejoin Wahba's procedure. Since $T^Tc=0$, we can write $c=Qe$ for some (unique) $(N-3)$-vector e, and substituting this in the equation $(K+\rho I_N)c+Td=Z$ and premultiplying by Q^T (which annihilates the second term) yields

$$Q^T(K+\rho I_N)Qe=Q^TZ \tag{2.3.1}$$

and on rearranging,

$$(Q^TKQ+\rho I_{(N-3)})e=Q^TZ, \tag{2.3.2}$$

which is a symmetric linear system of size $(N-3)$. Once this system has been solved for e, then c can be obtained directly from it, and d is then found by back substitution and solution of a system which reduces to 3×3. The importance of making the columns of Q orthonormal is now apparent. It ensures that the second term in the brackets is a multiple of the unit matrix, so all that is now needed to solve the system *for any* ρ, is to form the spectral decomposition of the matrix $B=Q^TKQ$. This is the matrix that arises in any case in the interpolation problem. Assuming all the data sites are distinct, and that they are not all collinear, B is a positive-definite symmetric matrix.

2.4. The role of the condition number. The limitations on the scale of the problem that can be solved are imposed both by time and storage considerations, and also by conditioning. In the smoothing problem, a multiple of the unit matrix is added to B, so we need be concerned only with the conditioning of B, which is the worst case. In the present context of a positive-definite symmetric matrix an appropriate measure of conditioning is the ratio of the largest to the smallest eigenvalue. Wahba (1979a), probably writing in the context of some form of single-precision arithmetic, suggests that the largest acceptable condition number in this sense may be about 10^6 or 10^7, and this view has not to our knowledge been seriously challenged. We would suggest that this may be overly optimistic for IEEE single-precision arithmetic, where we believe that a value of 10^5 may be a prudent bound; whereas for IEEE double-precision arithmetic very much larger values, perhaps up to about 10^{12}, should be acceptable.

There are two factors contributing to ill-conditioning. The first is the steady increase in condition number as the number of data sites increases, however benign their positioning. A useful quantitative picture of this situation is given by the condition number for a square grid of data sites, for grids of various sizes. For a 7×7 grid (49 data sites), it is about 250; for a 10×10 grid (100), about 1100; for a 14×14 grid (196), about 4200; for a 20×20 grid (400), about 18,000. This is in line with the theoretical result that the condition number for this particular layout of data sites increases approximately in proportion to the square of the number of data sites (Wahba (1979b),

Cox (1984), Utreras (1988)). On this basis a 30×30 grid would give a condition number around our suggested safe limit of 10^5 for single-precision arithmetic. Double-precision arithmetic should be able to handle any scale of problem on a square grid for which storage and time are available.

The second is the occurrence of close pairs of data sites. In the limit as two data sites move towards coincidence, the corresponding rows and columns of the matrix K tend to equality and the matrix B tends to singularity through cases with a small eigenvalue and consequently a large condition number. When randomly scattered data sites are generated for test purposes, the condition number obtained can fluctuate wildly. In a small number of replications we obtained one configuration of 100 points with a condition number below 5000 and another with a condition number exceeding 500,000. In such circumstances there is no "typical" condition number, and the conditioning of each problem has to be checked individually for acceptability, even if double-precision arithmetic is to be used. No guarantees can be given purely on the basis of the number of data sites. Dyn, Levin, and Rippa (1986) mention a limit of about 150 data sites "due to computational difficulties," which by implication are difficulties arising from ill-conditioning; their examples are all on highly structured evenly spaced systems of data sites, so they are probably not considering the effect of close points.

Strictly speaking, ill-conditioning could also arise from a tendency for the data sites all to approach collinearity; this is not a problem in practice and we shall ignore it.

2.5. Scaling the linear system. Changing scale isotropically in the plane, and separately changing the scale on which the data values are measured, has the same effect as varying the parameters α and β in the more general form

$$\varphi(u) = \alpha \|u\|^2 \log \beta \|u\|^2 \tag{2.5.1}$$

of the fundamental solution. Apart from an overall choice of the scale of B, and associated possible reinterpretation of the meaning of actual numerical values of the smoothing parameter, all of these choices leave B unaffected, except that changing Q for another matrix with the same properties will pre- and postmultiply B by an orthogonal matrix, which does not affect its spectrum, and hence does not alter its condition number. This is immediately obvious except for the effects of varying β, where the property depends on the orthogonality of the columns of Q to the columns of T.

These remarks are not true of the matrix on the left-hand side of (2.1.4) or (2.1.5), which is not even a positive-definite matrix; the actual spectral structure of this matrix is affected by rescaling, and by the way the columns of T are chosen to span the space of first-degree functions evaluated at the data sites. It is accordingly not very meaningful to quote values for the condition number of the matrix occurring in (2.1.4), as Dyn, Levin, and Rippa (1986) appear to have done.

The arithmetic performed on values of the fundamental solution in the calculation of B is most stable when its values do not have very large magnitude within the region occupied by the data sites, neither are they uniformly very small across it. In applications where the natural unit of scale leads to a coordinate range over the axes which is not of the order of magnitude of unity, for example some cartographic applications, then it may well be worth choosing a new scale for the coordinates which does achieve this, with a consequent change of scale for the smoothing parameter, whose interpretation does of course depend on these choices of scale.

2.6. Solution procedures in the ill-conditioned case. We have seen that ill-conditioning is likely to cause difficulties primarily where there are pairs of data sites close together. It can be argued that in such a case, exact interpolation is sensible only if the values at the two close data sites are themselves close, so the eigenvector corresponding to the small eigenvalue will make little contribution to the solution and it may be safer to ignore it completely at the cost of slightly failing to satisfy the interpolation conditions. A similar effect can be achieved by replacing the interpolation problem by a smoothing problem with a smoothing parameter which is small, but substantially larger than the small eigenvalue(s). Because what is being done is more readily interpretable, this may well be a preferable approach. Various other forms of approximation technique, oriented both to reducing the overall burden of computation as well as to the problem of conditioning, have been suggested by Bates and Wahba (1983) and Dyn, Levin, and Rippa (1986).

If the problem is actually an ill-conditioned smoothing problem, then it is only at small values of the smoothing parameter that any difficulty over condition number will become apparent. If there is a positive lower bound ρ_0 to the range of values of the smoothing parameter ρ that are of interest, then $B+\rho I$ can be written as $(B+\rho_0 I)+(\rho-\rho_0)I$ and the numerical linear algebra can be carried through with $B+\rho_0 I$ replacing B and $\rho-\rho_0$ replacing ρ, thereby reducing the condition number.

If it is really necessary to solve an ill-conditioned interpolation problem, then it is useful to recall that the only reason for insisting on orthonormality of the columns of Q is so as to keep a multiple of the unit matrix as the term arising from smoothing. If that term is absent, all that is necessary is to ensure that the columns of Q are independent, and are orthogonal to those of T, and they may be chosen in an attempt to improve the conditioning of B. Some of the numerical difficulties removed from the spectral decomposition problem by such preconditioning may reappear elsewhere in the arithmetic, but will generally be less harmful there.

2.7. A preconditioning method. We have developed a preconditioning method of the above type, where the matrix Q is replaced by a matrix R whose $(N-3)$ columns are merely required to be orthogonal to the columns of T, and independent. R is constructed as follows.

First a working "window" is placed round the configuration of points. This is required simply to provide a bounded convex polygonal region of the plane in which to work; the technique proposed is not sensitive to its choice. It may be conveniently taken to be a rectangular region whose sides are aligned with the coordinate axes and which has a reasonable clearance from the data sites, say, 10 percent of their spread.

The Dirichlet tessellation of the data sites within this window is then calculated. This is the subdivision of the window into convex polygonal *tiles*, one for each data site, such that the tile of a data site comprises that part of the plane nearer to that data site than to any other. An efficient computational procedure for finding Dirichlet tessellations has been proposed by Green and Sibson (1978). The *neighbours* of a data site in the tessellation are those other data sites whose tiles share a common boundary with its own. The strength of this neighbour relationship can be measured by assigning a weight to each pair which is zero for nonneighbours and is the length of the common boundary of their tiles divided by the distance apart for neighbours. These boundary-over-distance weights are symmetric, and are large for close neighbours because of the inverse-distance factor. They also have the property that each data site is the weighted centroid of its neighbours. This *local balance* property is trivial to prove for the Dirichlet tessellation in the plane; it was proved in the K-dimensional case by

Christ, Friedberg, and Lee (1982) from the divergence theorem, and can alternatively be proved from a similar result of Sibson (1980). Boundary-over-distance weights appear to have been first used for statistical purposes by Cliff and Ord (1969), but that application made no mention or use of local balance.

The columns of R are then constructed for the $(N-3)$ data sites which are not vertices of the reference triangle selected as in § 2.3. Each column is first zeroed, and then for each neighbour of the current data site, the boundary-over-distance weight is added to the entry for that neighbour and subtracted from the entry for the current data site itself. Orthogonality to the columns of T is guaranteed by the choice of weight for the current data site, and by local balance.

For data sites which are contiguous to the window edge, a special procedure has to be followed. Each such contiguity is treated as a neighbour relationship with a virtual point which is the reflexion of the current data site in that straight line segment of the window. The boundary-over-distance weight for the virtual point is then split between the data sites which are the vertices of the reference triangle in proportion to its homogeneous coordinates with respect to them. This retains the orthogonality property to the columns of T.

In either case, the column is finally given an appropriate normalisation by division by the tile area. Apart from this normalisation step, the procedure differs from the original construction set out in § 2.3 by expressing each data site as a convex combination of its neighbours rather than as an affine combination of the vertices of the reference triangle, these being used only if they happen to occur as neighbours, or if it is necessary to reexpress a virtual point (when negative weights may arise).

This procedure sounds complicated, but is in fact computationally very efficient. The motivation for it is that it generates linear combinations of the original fundamental solution which, in a rather crude sense, correspond to the application of the Laplacian operator. Large weights arise for close pairs, and this has precisely the appropriate effect to remove the ill-conditioning arising from such pairs, any numerical difficulty being transferred to the process of forming the linear combinations. In fact this "discrete Laplacian" is effectively applied twice over, because the matrix K is premultiplied by R^T as well as postmultiplied by R. If the Laplacian operator itself were applied twice to the fundamental solution, then a delta-function would result; the double application of its discrete analogue has the effect of producing a matrix in more nearly diagonal form, and the results of the preconditioning procedure are accordingly beneficial even in the case of regularly spaced data sites. This idea, of applying a discrete Laplacian to the original basis functions, was proposed for gridded data sites by Dyn and Levin (1983) and generalised for scattered data sites by Dyn, Levin, and Rippa (1986); the latter paper even used the Delaunay triangulation, the dual of the Dirichlet tessellation, but did not use boundary-over-distance weights and their properties.

This procedure has a dramatic effect on the condition number of B. For the regular square grid, it becomes 52 for 7×7, 72 for 10×10, 110 for 14×14, and 170 for 20×20. For 100 scattered points, it ranges from 35 to 95 on cases which include those for which the raw condition number is below 5000 and is above 500,000. Thus the condition number is reduced greatly, and grows more slowly with the number of data sites, even in the regular case; in the scattered case the bad properties arising from close pairs are eliminated; and the regular case now appears as typical within the range of scattered cases.

2.8. Limits on problem size. Even with single-precision arithmetic, it is clear that the limitation on the scale of the problem that can be tackled is imposed by storage

and time limitations rather than conditioning once this preconditioning procedure has been applied, and this is true as much for scattered data sites as for regularly spaced ones.

Two matrices, of size approximately $N \times N$, must be stored, and this is the dominant component of the storage requirement; in single-precision arithmetic this requires $8N^2$ bytes, so for example 8 Mbytes would be required for 1000 data sites; at the time of this writing, this is well within scientific workstation capabilities, but a problem on 10,000 data sites would tax the capabilities of a large mainframe.

The dominant component of the workload is the computation of the spectral decomposition. The time taken by the procedure we use increases as N^3 for large N. We estimate that it would take about four hours on a 1000×1000 matrix on the computer used for our tests (Sun-4/260); whilst no doubt absolute timing will become out of date as quickly as usual, the rapid rate of growth of both the storage and the time workload with N suggests that the practicable limit of scale is going to be below 10,000 data sites for some time to come, even allowing for the possible introduction of alternative algorithms taking advantage of parallel processing technologies.

Methods of solution which avoid the use of the spectral decomposition may well be worth exploring. Solution of (2.3.2) is possible by a variety of methods both direct and iterative, and where only a single vector Z of observed values, and a single value of the smoothing parameter are of interest, then a considerable gain in efficiency should be possible. Dyn, Levin, and Rippa (1986) discuss a variety of such methods, and conclude that the conjugate gradient approach is superior to the alternatives considered in their tests. But such methods still rapidly become laborious for large problems.

Practical problems often arise with many more than 10,000 data sites; for example, in aeromagnetic survey work it is common to have 50,000 to 100,000 observations in a single data set. We believe that such problems will indefinitely remain beyond the scope of thin-plate splines. We plan to report work on the development of splinelike methods for large-scale problems (and for the rapid solution of medium-scale problems) in future papers.

3. Conclusions. Thin-plate splines have been shown to form, in a suitable mathematical sense, the natural multidimensional generalisation of cubic smoothing splines. We have demonstrated why this generalisation breaks down without further modification beyond two dimensions. Focusing our attention on the two-dimensional case, we have identified three major differences between the behaviour of cubic splines and thin-plate splines. The lack of a sparse reformulation of the linear system characterising the solution is fundamental to the nature of the two-dimensional problem. It completely changes its computational character compared with that of the cubic spline problem. We have identified the sources of ill-conditioning in the linear system, and proposed a preconditioning procedure which eliminates ill-conditioning as a source of computational difficulty in the interpolation problem, which is the worst case. The use of this procedure, together with other appropriate numerical methods that we identify, extends the scale on which thin-plate splines can be applied without undue difficulty to data sets with 1000 observations.

REFERENCES

D. M. BATES AND G. WAHBA (1983), *A truncated singular value decomposition and other methods for generalized cross-validation*, Tech. Report 715, Department of Statistics, University of Wisconsin, Madison, WI.

N. H. CHRIST, R. FRIEDBERG, AND T. D. LEE (1982), *Weights of links and plaquettes in a random lattice*, Nuclear Phys. B, Field Theory Statist. Sys., 210, pp. 337–346.

A. D. CLIFF AND J. K. ORD (1969), *The problem of spatial autocorrelation*, in Studies in Regional Science, A. J. Scott, ed., Pron., London, pp. 25-55.

D. D. COX (1984), *Multivariate smoothing spline functions*, SIAM J. Numer. Anal., 21, pp. 789-813.

W. DAHMEN (1980), *On multivariate* B*-splines*, SIAM J. Numer. Anal., 17, pp. 179-191.

——— (1981), *Approximation by linear combinations of multivariate* B*-splines*, J. Approx. Theory, 31, pp. 299-324.

C. DE BOOR (1978), *A Practical Guide to Splines*, Springer-Verlag, Berlin, New York.

J. DUCHON (1976a), *Interpolation des fonctions de deux variables suivant le principe de la flexion des plaques minces*, RAIRO Anal. Numér., 10, pp. 5-12.

——— (1976b), *Fonctions spline du type "Plaque mince" en dimension* 2, RR231, Mathématiques Appliquées, Université Scientifique et Médicale, Grenoble.

N. DYN AND D. LEVIN (1982), *Construction of surface spline interpolants of scattered data over finite domains*, RAIRO Anal. Numér., 16, pp. 201-209.

——— (1983), *Iterative solution of systems originating from integral equations and surface interpolation*, SIAM J. Numer. Anal., 20, pp. 377-390.

N. DYN, D. LEVIN, AND S. RIPPA (1986), *Numerical procedures for surface fitting of scattered data by radial functions*, SIAM J. Sci. Statist. Comput., 7, pp. 639-659.

N. DYN AND G. WAHBA (1979), *On the estimation of functions of several variables from aggregated data*, Tech. Report 1974, Mathematics Research Centre, University of Wisconsin, Madison, WI.

P. J. GREEN AND R. SIBSON (1978), *Computing Dirichlet tessellations in the plane*, The Computer J., 21, pp. 168-173.

R. L. HARDY (1971), *Multiquadric equations of topography and other irregular surfaces*, J. Geophysical Res., 76, pp. 1905-1915.

C. L. HU AND L. L. SCHUMAKER (1986), *Complete spline smoothing*, Numer. Math., 49, pp. 1-10.

J. MEINGUET (1979a), *An intrinsic approach to multivariate spline interpolation at arbitrary points*, in Polynomial and Spline Approximation, B. N. Sahney, ed., D. Reidel, New York.

——— (1979b), *Multivariate interpolation at arbitrary points made simple*, Z. Angew. Math. Phys., 30, pp. 292-304.

M. J. D. POWELL (1985), *Radial basis functions for multivariable interpolation: a review*, presented at the IMA Conference on Algorithms for the Approximation of Functions and Data, RMCS, Shrivenham.

R. SIBSON (1980), *A vector identity for the Dirichlet tessellation*, Math. Proc. Cambridge Philos. Soc., 87, pp. 151-155.

B. W. SILVERMAN (1985), *Some aspects of the spline smoothing approach to non-parametric regression curve fitting* (with discussion), J. Roy. Statist. Soc., Ser. B, 47, pp. 1-52.

F. I. UTRERAS (1988), *Convergence rates for multivariate smoothing spline functions*, J. Approx. Theory, 52, pp. 1-27.

G. WAHBA (1979a), *How to smooth curves and surfaces with splines and cross-validation*, in Proc. 24th Design of Experiments Conference, F. Dressel, ed., Academic Press, New York.

——— (1979b), *Convergence rates of "thin plate" smoothing splines when the data are noisy*, in Smoothing Techniques for Curve Estimation, T. Gasser and M. Rosenblatt, eds., Lecture Notes in Mathematics, Vol. 757, Springer-Verlag, Berlin, New York, pp. 232-245; Tech. Report 557, Department of Statistics, University of Wisconsin, Madison, WI, 1979.

——— (1980a), *Ill posed problems: Numerical and statistical methods for mildly, moderately, and severely ill posed problems with noisy data*, Tech Report 595, Department of Statistics, University of Wisconsin, Madison, WI.

——— (1980b), *Spline bases, regularization, and generalized cross validation for solving approximation problems with large quantities of noisy data*, Tech. Report 595, Department of Statistics, University of Wisconsin, Madison, WI.

——— (1981), *Spline interpolation and smoothing on the sphere*, SIAM. J. Sci. Statist. Comput., 2, pp. 5-16; Tech. Report 584, Department of Statistics, University of Wisconsin, Madison, WI, 1979.

——— (1983), *Bayesian "confidence intervals" for the cross validated smoothing spline*, J. Roy. Statist. Soc., Ser. B, 45, pp. 133-150; Tech. Report 645, Department of Statistics, University of Wisconsin, Madison, WI, 1981.

G. WAHBA AND J. WENDELBERGER (1980), *Some new mathematical methods for variational objective analysis using splines and cross-validation*, Monthly Weather Rev., 108, pp. 1122-1143; Tech. Report 578, Department of Statistics, University of Wisconsin, Madison, WI, 1979.

J. G. WENDELBERGER (1981), *The computation of Laplacian smoothing splines with examples*, Tech. Report 648, Department of Statistics, University of Wisconsin, Madison, WI.

——— (1982), *Smoothing noisy data with multidimensional splines and generalized cross validation*, Ph.D. thesis, Department of Statistics, University of Wisconsin, Madison, WI.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1314-1323, November 1991
© 1991 Society for Industrial and Applied Mathematics
005

DAMPED JACOBI PRECONDITIONING AND COARSE GRID DEFLATION FOR CONJUGATE GRADIENT ITERATION ON PARALLEL COMPUTERS*

LOIS MANSFIELD†

Abstract. A combination of damped Jacobi preconditioning and deflation is given as a way to improve the convergence of conjugate gradient iteration on local memory multiprocessor computers such as the hypercube. Experimental results are given for the linear system associated with the elastic bending of a cantilever beam on the iPSC/2 with both 4 and 16 processors.

Key words. conjugate gradient method, preconditioned conjugate gradient, parallel

AMS(MOS) subject classifications. 65F10, 65N20

1. Introduction. The purpose of this paper is to present and give experimental results using a combination of damped Jacobi preconditioning and deflation to improve the convergence rate of conjugate gradient iteration on local memory multiprocessor computers such as the hypercube. We assume that the linear equations to be solved arise from the finite element discretization of a second order elliptic partial differential equation or of a system of second order elliptic partial differential equations. We consider conjugate gradient iteration used both to solve the entire linear system and to solve the Schur complement system obtained from domain decomposition.

To solve linear finite element systems of equations on a local memory multiprocessor computer it is natural to partition the region Ω of the underlying partial differential equation or system of partial differential equations into as many subregions Ω_i as processors. The system to be solved, $Au=f$, can then be ordered as

$$\begin{pmatrix} B & V \\ V^T & \bar{C} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}, \tag{1}$$

where B contains all rows and columns associated with internal nodes on any processor and $\bar{C}$ contains all rows and columns associated with the separator nodes. One way to solve (1) is to solve the Schur complement system

$$Cu_1 = g_1, \tag{2}$$

where $C=\bar{C}-V^TB^{-1}V$ and $g_1=f_1-V^TB^{-1}f_0$, and then determine u_0 from

$$Bu_0=f_0-Vu_1. \tag{3}$$

Since B is a block diagonal matrix with one block corresponding to each processor, the subsystem in (3) can be solved in parallel with no communication between processors. One might choose to solve (3) by direct methods and (2) by an iterative method such as conjugate gradient.

In fact, in [4], we showed that if the partial differential equation or system of partial differential equations satisfies standard ellipticity assumptions and if the finite element grid is quasi-uniform, the condition number of the Schur complement system C is $O(1/h)$, where h is the mesh parameter. This is in contrast to the condition number of the whole system which is $O(1/h)^2$. The ordering of (1) has also been used to obtain preconditioners for $Au=f$, (see [2]), by using approximations for C, B, and possibly V in (2) and (3).

* Received by the editors July 18, 1990; accepted for publication (in revised form) December 10, 1990. This research was supported by National Science Foundation grant ASC 8922539.

† Department of Applied Mathematics, University of Virginia, Charlottesville, Virginia 22903.

In § 2 we discuss damped Jacobi preconditioning. We show that it has much of the same effect that smoothing has in multigrid iteration. It reduces the large eigenvalues of the matrix to nearly 1. Deflation was originally introduced by Nicolaides in [5]. In [4] we proposed using deflation vectors determined from a coarse grid discretization. We also showed in [4] that for the Schur complement system it has the effect of raising the lower bound on the eigenvalues of the system. The same thing can be shown for coarse grid deflation applied to the entire system. We discuss coarse grid deflation in § 3. In § 4 we give experimental results obtained for the linear system associated with the elastic bending of a cantilever beam on the iPSC/2 with both 4 and 16 processors.

2. Damped Jacobi preconditioning. In order to solve the symmetric positive definite linear system $Au = f$ by preconditioned conjugate gradient methods, the system $Au = f$ is replaced by the system $\tilde{A}\tilde{u} = \tilde{f}$, where $\tilde{A} = G^{-1}AG^{-1}$, $\tilde{u} = Gu$, $\tilde{f} = G^{-1}f$, and $M = G^2$. Conjugate gradient iteration is applied to $\tilde{A}\tilde{u} = \tilde{f}$. The algorithm may be expressed in terms of the original system $Au = f$ as

Choose u^0 and set $r^0 = f - Au^0$.
Solve $M\hat{r}^0 = r^0$.
Set $p^0 = \hat{r}^0$ and calculate $(r^0, \hat{r}^0)$.
For $k = 0, 1, 2, \cdots$
  $\gamma = (\hat{r}^k, r^k)/(p^k, Ap^k)$.
  $u^{k+1} = u^k + \gamma p^k$.
  $r^{k+1} = r^k - \gamma Ap^k$.
  Check for convergence.
  $M\hat{r}^{k+1} = r^{k+1}$.
  $\beta = (\hat{r}^{k+1}, r^{k+1})/(\hat{r}^k, r^k)$.
  $p^{k+1} = \hat{r}^{k+1} + \beta p^k$.

M is the preconditioning matrix. Since the rate of convergence of conjugate gradient iteration is related to the condition number of the matrix, M is chosen so that $M\hat{r}^{k+1} = r^{k+1}$ is easy to solve and so that the condition number of $\tilde{A}$ is smaller than the condition number of A.

Damped or underrelaxed Jacobi preconditioning results from applying one or more steps of damped or underrelaxed Jacobi iteration to $A\hat{r} = r$ with initial approximation $\hat{r}^{(0)} = 0$. Damped Jacobi iteration may be described as

$$\tilde{r}^{(m+1)} = (I - D^{-1}A)\hat{r}^{(m)} + D^{-1}r,$$

$$\hat{r}^{(m+1)} = (1-\alpha)\hat{r}^{(m)} + \alpha\tilde{r}^{(m+1)},$$

or

$$\hat{r}^{(m+1)} = (I - \alpha D^{-1}A)\hat{r}^{(m)} + \alpha D^{-1}r,$$

where D is the diagonal of A. Ordinary Jacobi iteration is obtained by setting $\alpha = 1$. If M^{-1} is chosen to represent m steps of damped Jacobi iteration

$$M^{-1} = (I + I - \alpha D^{-1}A + (I - \alpha D^{-1}A)^2 + \cdots + (I - \alpha D^{-1}A)^{m-1})\alpha D^{-1}.$$

Similarly to Theorem 1 of [1], it can be shown that for m odd, M is always positive definite since D is positive definite, and for m even, M is positive definite if and only if $2D - \alpha A$ is positive definite. Note that the requirement for damped Jacobi iteration to be convergent (see [7], Thm. 3.3) is that $2D - \alpha A$ and A be positive definite. Of course, $2D - \alpha A$ can be made to be positive definite by appropriate choice of α.

The choice $m=1$ essentially amounts to diagonal preconditioning. As is shown in [1] the benefits of taking more steps of an m-step preconditioner decreases as m increases. We have found the most benefit to occur for $m=2$ and for the remainder of the paper will limit discussion to this case. For $m=2$

$$M^{-1}A=\alpha(2I-\alpha D^{-1}A)D^{-1}A.$$

The eigenvalues of $M^{-1}A$ are given by

$$\gamma_i=\alpha(2-\alpha\lambda_i)\lambda_i, \qquad i=1,\cdots,n,$$

where λ_i, $i=1,\cdots,n$, are the eigenvalues of $D^{-1}A$. With λ_n the largest eigenvalue of $D^{-1}A$, we indicate the relationship between eigenvalues of $M^{-1}A$ and $D^{-1}A$ for two choices of α, $\alpha=2/\lambda_n$ and $\alpha=4/3\lambda_n$ in the graph of Fig. 1. Note that $2I-\alpha D^{-1}A$ is positive definite only if $\alpha<2/\lambda_n$ and so $\alpha=2/\lambda_n$ corresponds to ordinary Jacobi iteration.

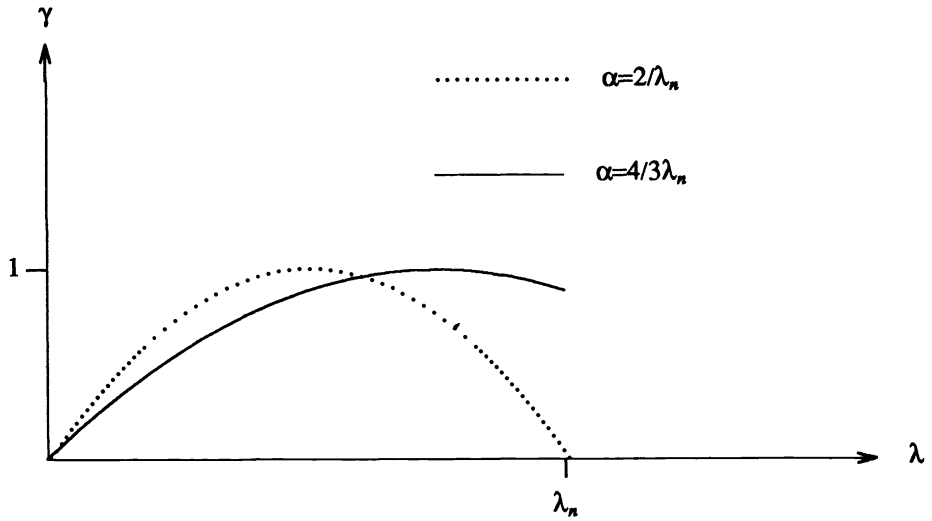


FIG. 1

One measure of the effectiveness of M as a preconditioner is how close to 1 are the eigenvalues of $M^{-1}A$. For $\alpha=2/\lambda_n$ only the eigenvalues of $M^{-1}A$ corresponding to eigenvalues in the middle of the spectrum of $D^{-1}A$ are near 1. The effect of reducing α is to shift the part of the spectrum taken to nearly 1 to the high end and to increase its range. The value of $\alpha=(2/3)(2/\lambda_n)$ is a rough estimate of the best way to do this. Note that $\alpha=4/3\lambda_n$ does less well with the smallest eigenvalues than does $\alpha=2/\lambda_n$.

Thus we propose to use damped Jacobi preconditioning much as smoothing is used in multigrid iteration, to handle the large eigenvalues of $D^{-1}A$. We propose to combine this with coarse grid deflation, which we show in the next section attacks the smaller eigenvalues.

3. Coarse grid deflation. The deflation method of Nicolaides [5] to solve $Au=f$, where A is an $n\times n$ symmetric positive definite matrix may be described as follows. Let E be an $n\times m$ matrix whose columns are a set of linearly independent vectors. First solve $Au=f$ in the subspace $\mathcal{E}$ generated by the columns of E. This amounts to solving

$$E^TAEd=E^Tf. \tag{4}$$

Then solve

$$A(I-P)v = f - AEd \tag{5}$$

in span $\{E\}^{\perp}$, where P is the projection operator

$$P = E(E^TAE)^{-1}E^TA, \tag{6}$$

which is the orthogonal projector in the norm

$$\|x\|_A = (x^TAx)^{1/2}.$$

Then

$$u = Ed + (I-P)v. \tag{7}$$

The effect of (4) is to find the best approximation in the norm $\|\cdot\|_A$ to the solution u to $Au = f$ which can be found among linear combinations of the columns of E. Since m steps of conjugate gradient iteration does the same thing over the Krylov subspace $\mathcal{K}(A, r^0, m)$, the effectiveness of deflation depends on the solution being better approximated by the columns of E than the Krylov subspace. In [3] we proposed defining the columns of E in certain situations to reflect known qualitative features of the solution such as being nearly constant along certain lines in the domain. More generally, we propose to define the columns of E using a nodal basis for a coarse grid over the domain. The matrix E then defines an interpolation operator mapping the coarse grid to the problem grid.

In [4] we showed that if (1) arises from the finite element discretization of a second order elliptic partial differential equation or system of second order elliptic partial differential equations satisfying standard ellipticity assumptions, and if the finite element grid is quasi-uniform, the Schur complement matrix C has its smallest eigenvalue $\lambda_{\min}^C$ and largest eigenvalue $\lambda_{\max}^C$ bounded by

$$\lambda_{\min}^C \geqq c_1 h^{N-1} d, \qquad \lambda_{\max}^C \leqq c_2 h^{N-2}, \tag{8}$$

so that $\operatorname{cond}_2(C) = O(1/hd)$, where N is the dimension of the domain Ω, h is the mesh parameter of the finite element grid, and d is the mesh parameter of the subregions Ω_i of the decomposition of the domain. In [4] we also showed that if the coarse grid is of the same order as the subregions Ω_i, the matrix $C(I-P)$ has its smallest positive eigenvalue $\lambda_{\min}^{C(I-P)}$ and largest eigenvalue $\lambda_{\max}^{C(I-P)}$ bounded by

$$\lambda_{\min}^{C(I-P)} \geqq c_1 \frac{h^{N-1}}{d}, \qquad \lambda_{\max}^{C(I-P)} \leqq c_2 h^{N-2}. \tag{9}$$

Thus deflation reduces the condition number by raising the bound on the smallest eigenvalue.

Under the assumptions above, the matrix A has its smallest eigenvalue $\lambda_{\min}^A$ and largest eigenvalue $\lambda_{\max}^A$ bounded by

$$\lambda_{\min}^A \geqq c_1 h^N, \qquad \lambda_{\max}^C \leqq c_2 h^{N-2}, \tag{10}$$

so that $\operatorname{cond}_2(A) = O(1/h^2)$. Using a nearly parallel proof to that of [4], it can be shown that the matrix $A(I-P)$ has its smallest positive eigenvalue $\lambda_{\min}^{A(I-P)}$ and largest eigenvalue $\lambda_{\max}^{A(I-P)}$ bounded by

$$\lambda_{\min}^{A(I-P)} \geqq \frac{c_1 h^N}{d^2}, \qquad \lambda_{\max}^{A(I-P)} \leqq c_2 h^{N-2}. \tag{11}$$

To obtain (11) it is not necessary that the computation grid be a refinement of the coarse grid.

4. Experimental results. In this section we give experimental results obtained in solving the linear system of equations resulting from the finite element approximation of the linear elastic bending of a cantilever beam subjected to a constant tip bending moment, which can be modeled as a linearly varying surface traction $\hat{\sigma}$. This problem is described in more detail in [3]. This problem is also considered in [6]. We took the beam to be 4 inches high and either 10 or 20 inches long. By applying appropriate boundary conditions along the neutral axis it is only necessary to model the upper half of the beam. The quantities being solved for are the horizontal displacements u_1 and the vertical displacements u_2. The essential boundary conditions are $u_1 = 0$ and $u_2 = 0$ along the left (built-in side) of the beam, and $u_1 = 0$ on the bottom (neutral axis) of the beam.

Let $\mathbf{H}^1(\Omega) = H^1(\Omega) \times H^1(\Omega)$ and $\mathbf{H}^1_e(\Omega)$ be the subspace of $\mathbf{H}^1(\Omega)$ consisting of all elements of $\mathbf{H}^1(\Omega)$ which satisfy the essential boundary conditions. The variational problem to be solved is: find $\mathbf{u} \varepsilon \mathbf{H}^1_e(\Omega)$ such that

$$\int_\Omega (\mathbf{Dv})^T \mathbf{EDu}\, dx\, dy = \int_0^1 \hat{\sigma} y v_2\, dy, \qquad \text{all } \mathbf{v} \varepsilon \mathbf{H}^1_e(\Omega),$$

where

$$\mathbf{D} = \begin{pmatrix} \dfrac{\partial}{\partial x} & 0 \\ 0 & \dfrac{\partial}{\partial y} \\ \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial x} \end{pmatrix}$$

and

$$\mathbf{E} = \frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1}{2}(1-\nu) \end{pmatrix}$$

where E is Young's modulus and $0 < \nu < \frac{1}{2}$ is Poisson's ratio.

The finite elements used were piecewise quadratic polynomials over the triangulation obtained by drawing in the diagonal from the upper left-hand corner to the lower right-hand corner of each rectangle of a rectangular grid containing N_x rectangles in the x-direction and N_y rectangles in the y-direction. For four processors the region Ω was divided into four rectangles each half as long and wide as Ω. For 16 processors the region Ω was divided into 16 rectangles each one fourth as long and wide as Ω.

Tables 1–4 contain results obtained solving the entire system by conjugate gradient with the preconditionings indicated. Tables 5–8 contain the results obtained when the system was reordered by domain decomposition as in (1) with the Schur complement system (2) solved by conjugate gradient with the preconditionings indicated and (3) solved by Cholesky decomposition. The convergence criterion was that the l^2 norm of the original residual be reduced by a factor of 10^6. We include results for both a beam of length 10 and a beam of length 20 because although more physically realistic, the system for the beam of length 20 is quite ill-conditioned.

TABLE 1
Iterations for complete system for beam of length 10, 4 *processors.*

	8×4 306 equations (setup = .06)	16×8 1,122 equations (setup = .069)	32×16 4,290 equations (setup = .095)
No preconditioning	198 (9.24)	406 (67.21)	806 (512.48)
Diagonal preconditioning	185 (8.92)	388 (67.10)	787 (522.90)
Damped Jacobi preconditioning	100 (8.82)	201 (65.46)	399 (501.87)
Deflation	78 (9.76)	151 (59.88)	296 (416.06)
Deflation with damped Jacobi preconditioning	41 (10.25)	75 (59.49)	149 (418.16)
Deflation with nested ordering	78 (8.05)	151 (50.56)	296 (358.10)
Deflation with damped Jacobi preconditioning and nested ordering	41 (8.38)	75 (50.12)	149 (359.28)

TABLE 2
Iterations for complete system for beam of length 10, 16 *processors.*

	16×8 1,122 equations (setup = .06)	32×16 4,290 equations (setup = .067)
No preconditioning	405 (20.51)	806 (138.66)
Diagonal preconditioning	388 (20.54)	787 (141.22)
Damped Jacobi preconditioning	201 (19.11)	399 (133.41)
Deflation	151 (18.57)	296 (113.99)
Deflation with damped Jacobi preconditioning	75 (18.31)	149 (114.23)
Deflation with nested ordering	151 (15.79)	296 (97.84)
Deflation with damped Jacobi preconditioning and nested ordering	75 (15.39)	149 (97.60)

The numbers in parentheses are the timings in seconds of the solution of the system obtained on the iPSC/2. The times required to set up the linear systems are indicated near the top of each table. For the Schur complement system the setup time is by far the biggest computation cost. For the complete system the setup time includes the time to set up the element matrices and compute the diagonal if required. For the Schur complement system the setup time includes the time to set up the element matrices, the time to set up the system A from the element matrices, the time to reorder it as in (1), the time to factor B, and the time to set up C. The solution time includes the time to solve the Schur complement system (2) along with the time to do the forward and backward solves to solve (3). We set up C explicitly. Each processor sets up the contributions from its region to those rows and columns corresponding to

TABLE 3
Iterations for complete system for beam of length 20, 4 *processors.*

	8×4 306 equations (setup = .06)	16×8 1,122 equations (setup = .069)	32×16 4,290 equations (setup = .095)
No preconditioning	317 (14.52)	659 (109.03)	1,324 (844.74)
Diagonal preconditioning	284 (13.69)	610 (105.97)	1,236 (823.44)
Damped Jacobi preconditioning	156 (13.73)	318 (103.39)	636 (801.70)
Deflation	140 (16.18)	277 (104.99)	551 (754.94)
Deflation with damped Jacobi preconditioning	71 (16.31)	152 (113.77)	286 (781.13)
Deflation with nested ordering	140 (13.57)	277 (89.44)	551 (652.64)
Deflation with damped Jacobi preconditioning and nested ordering	71 (13.60)	152 (97.10)	286 (674.79)

TABLE 4
Iterations for complete system for beam of length 20, 16 *processors.*

	16×8 1,122 equations (setup = .06)	32×16 4,290 equations (setup = .067)
No preconditioning	659 (33.39)	1,324 (227.71)
Diagonal preconditioning	610 (32.28)	1,236 (221.70)
Damped Jacobi preconditioning	318 (30.20)	636 (212.41)
Deflation	277 (32.63)	551 (207.10)
Deflation with damped Jacobi preconditioning	152 (34.97)	286 (213.01)
Deflation with nested ordering	277 (28.02)	551 (178.67)
Deflation with damped Jacobi preconditioning and nested ordering	152 (29.81)	286 (183.09)

separator nodes adjacent to its subregion. If there is sufficient memory it is more efficient to set up C explicitly if the maximum number of separators adjacent to any region is less than the total number of conjugate gradient iterations required. This was true in most of our runs.

To do the deflation we first orthogonalized the columns of E with respect to the inner product y^TAx or y^TCx to obtain V so that V^TAV or V^TCV is the identity matrix. The vectors in AV or in CV were retained in memory to facilitate the evaluation of $A(I-P)p$ or $C(I-P)p$. This eliminated the need to set up E^TAE or E^TCE. For both the complete system and for the Schur complement system, the time to set up E and determine V are contained in the solution time even though if the same system were to be solved again with a different right-hand side these steps need not be repeated.

TABLE 5
Iterations for Schur complement system for beam of length 10, 4 *processors.*

	8×4 47 separators (setup = 2.74)	16×8 95 separators (setup = 37.02)	32×16 191 separators (setup = 632.14)
No preconditioning	52 (1.06)	93 (5.72)	145 (33.84)
Diagonal preconditioning	47 (1.04)	76 (4.97)	114 (28.81)
Damped Jacobi preconditioning	29 (1.08)	44 (5.27)	63 (29.88)
Deflation	21 (0.99)	36 (4.22)	58 (23.16)
Deflation with damped Jacobi preconditioning	13 (1.12)	17 (4.05)	26 (21.57)
Deflation with nested ordering	21 (0.89)	37 (4.02)	58 (22.34)
Deflation with damped Jacobi preconditioning and nested ordering	13 (1.01)	17 (3.77)	26 (20.76)

TABLE 6
Iterations for Schur complement system for beam of length 10, 16 *processors.*

	16×8 273 separators (setup = 6.14)	32×16 561 separators (setup = 86.67)
No preconditioning	160 (8.39)	246 (43.80)
Diagonal preconditioning	140 (7.56)	201 (36.58)
Damped Jacobi preconditioning	76 (7.41)	108 (39.98)
Deflation	80 (7.98)	122 (33.00)
Deflation with damped Jacobi preconditioning	32 (6.47)	46 (25.28)
Deflation with nested ordering	80 (6.70)	122 (29.59)
Deflation with damped Jacobi preconditioning and nested ordering	32 (5.29)	46 (22.29)

The times to set up E and determine V were found to be 19.7 seconds for the complete system and 2 seconds for the Schur complement system for the largest problem on four processors. In both cases the coarse grid was 2×1 in order to keep the number of columns in E small. This meant that the coarse grid was coarser than the subdivision of Ω to the processors. The number of nodes in the coarse grid was 20 for the complete system, 11 for the Schur complement system with four processors and 17 for the Schur complement system with 16 processors.

To combine deflation with damped Jacobi preconditioning, we first applied deflation to A or C and then used damped Jacobi preconditioning to precondition $A(I-P)$ or $C(I-P)$. The number of operations required to orthogonalize the columns of E can be reduced if the nodes in the coarse grid are ordered by nested dissection. Our

TABLE 7
Iterations for Schur complement system for beam of length 20, 4 *processors.*

	8×4 47 separators (setup = 2.74)	16×8 95 separators (setup = 37.02)	32×16 191 separators (setup = 632.14)
No preconditioning	65 (1.28)	122 (7.18)	220 (46.78)
Diagonal preconditioning	50 (1.06)	93 (5.84)	149 (34.95)
Damped Jacobi preconditioning	34 (1.23)	54 (6.25)	85 (37.22)
Deflation	34 (1.35)	58 (5.76)	99 (31.72)
Deflation with damped Jacobi preconditioning	16 (1.28)	24 (4.95)	37 (25.90)
Deflation with nested ordering	33 (1.21)	58 (5.54)	99 (31.04)
Deflation with damped Jacobi preconditioning and nested ordering	16 (1.16)	24 (4.65)	37 (24.95)

TABLE 8
Iterations for Schur complement system for beam of length 20, 16 *processors.*

	16×8 273 separators (setup = 6.14)	32×16 561 separators (setup = 86.67)
No preconditioning	251 (13.03)	391 (68.96)
Diagonal preconditioning	191 (10.24)	273 (49.32)
Damped Jacobi preconditioning	107 (10.31)	151 (51.24)
Deflation	149 (13.65)	232 (58.51)
Deflation with damped Jacobi preconditioning	48 (8.92)	70 (35.71)
Deflation with nested ordering	151 (11.83)	232 (52.82)
Deflection with damped Jacobi preconditioning and nested ordering	48 (7.39)	70 (31.87)

last two preconditionings use a one level nested dissection to order the coarse grid nodes for both deflation and deflation combined with damped Jacobi preconditioning. This reduced the maximum number of columns of E any processor had to deal with to 15 for the complete system, 9 for the Schur complement system with four processors and 12 for the Schur complement system with 16 processors.

Each conjugate gradient iteration requires two inner product calculations and one matrix vector multiply. Each inner product calculation requires a global sum or fan-in fan-out communication among all the processors while each matrix vector multiply requires a nearest neighbor exchange among the processors. For a polynomial preconditioner to be competitive it must reduce the number of iterations required by a factor nearly equal to the increase in the number of matrix vector multiplies. The damped

Jacobi preconditioner requires one additional matrix vector multiply per iteration and in most cases more than halved the number of iterations required. We determined the parameter α experimentally. We used .66 for the complete system for the beam of length 10 and .72 for the complete system for the beam of length 20. We used $\alpha = .71$ for all our runs on the Schur complement system. The timings were not significantly better than with no preconditioning because the inner product calculations did not require a high enough percentage of the total time. Because with our partitioning each processor has two neighbors when four processors are used and at most four neighbors when 16 processors are used, the number of sends and receives for a nearest neighbor exchange and a global sum are the same. If additional processors were used on the same problem, the cost of a nearest neighbor exchange should decrease relative to a global sum. This means that if more processors were used, we would expect the relative performance of damped Jacobi preconditioning to improve. This should be even more true with the next generation of the hypercube, the iPSC/860, which has much increased computation speed relative to communication speed.

For each column of E, the orthogonalization of the columns of E requires one matrix vector multiplication, the inner product of that column with all the previous columns, and an inner product v^TAv or v^TCv to normalize the column. For any column, these inner products with previous columns can be done simultaneously and so require only one fan-in fan-out communication. Thus the orthogonalization requires two fan-in fan-out communications and one nearest neighbor exchange for each column in E, which is the same communication required in each iteration of conjugate gradient. To do the projection $(I - P)p$, inner products of p with all the columns of V must be computed. However, again these can be done simultaneously so that after V is computed the only additional communication is an additional fan-in fan-out in each conjugate gradient iteration. We have found, however, that the coding which sets up E is not trivial to write.

The timings indicate a speedup for the complete system of about 3.2 to about 3.76 in going from four processors to 16 processors. For the Schur complement system the speedup in the setup phase was 7.29, while the solution phase actually became slower. This is due to the fact that we are using different algorithms since the number of subregions was taken to be equal to the number of processors. For the beam of length 10, solving the complete system by conjugate gradient was faster for the largest problem with both four and 16 processors. For the smaller problems for the beam of length 10 and all problems for the beam of length 20, domain decomposition was faster.

REFERENCES

[1] L. ADAMS, *m-step preconditioned conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 452-463.

[2] D. E. KEYES AND W. D. GROPP, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM J. Sci. Statist. Comput., 10 (1987), pp. s166-s202.

[3] L. MANSFIELD, *On the use of deflation to improve the convergence of conjugate gradient iteration*, Comm. Appl. Numer. Methods, 4 (1988), pp. 151-156.

[4] ———, *On the conjugate gradient solution of the Schur complement system obtained from domain decomposition*, SIAM J. Numer. Anal., 27 (1990), pp. 1612-1620.

[5] R. A. NICOLAIDES, *Deflation of conjugate gradients with applications to boundary value problems*, SIAM J. Numer. Anal., 24 (1987), pp. 355-365.

[6] M. D. SNYDER AND K. J. BATHE, *A solution procedure for thermoelastic-plastic and creep problems*, Nucl. Engrg. Des., 64 (1981), pp. 49-80.

[7] D. M. YOUNG, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1324–1331, November 1991
©1991 Society for Industrial and Applied Mathematics
006

WHAT IS THE COVARIANCE ANALOG OF THE PAIGE AND SAUNDERS INFORMATION FILTER?*

M. R. OSBORNE† AND TANIA PRVAN‡

Abstract. In this paper a square root covariance form of the Kalman filter is presented which is in many ways the covariance analog of the Paige and Saunders information filter [*SIAM J. Numer. Anal.*, 14 (1977), pp. 180–193] and possesses many of the advantages of their algorithm. For example, it is a true square root implementation; but the distinctive feature which it shares with the Paige and Saunders algorithm is a compact, convenient, and effective square root implementation of the interpolation smoother. The algorithm is based on the Duncan and Horne generalised least squares form of the filter equations [*J. Amer. Statist. Assoc.*, 67 (1972), pp. 816–821], but uses these in a recursive manner.

Key words. Kalman filter, covariance filter, information filter, square root filter, generalised least squares, smoothing step

AMS(MOS) subject classifications. 60G35, 65F25, 65U05

1. Introduction.

The Kalman filter gives the best estimate (best linear minimum variance estimate) $\mathbf{x}_{k|k}$ of the state vector $\mathbf{x}_k$ (or $\mathbf{x}(t_k)$) together with the covariance matrix $S_{k|k} = \mathcal{E}\{(\mathbf{x}_k - \mathbf{x}_{k|k})(\mathbf{x}_k - \mathbf{x}_{k|k})^T\}$ given observations on the system at $t = t_j$, $j = 1, 2, \cdots, k$ when the evolution of the state vector is determined by the stochastic process having the particular structure that the state vector evolves according to a dynamics equation

$$\mathbf{x}_{k+1} = T(t_{k+1}, t_k)\mathbf{x}_k + \mathbf{u}(t_{k+1}, t_k), \tag{1.1}$$

and information is available on the state vector through an observation equation

$$\mathbf{y}_{k+1} = H_{k+1}\mathbf{x}_{k+1} + \epsilon_{k+1}. \tag{1.2}$$

Here $T(t, s) : R^p \to R^p$ and $H_k : R^p \to R^m$ are assumed to have full rank, and $m \leq p$. Further $\mathbf{u}(t_{k+1}, t_k)$ and ϵ_k are vectors of random variables, uncorrelated with each other and with the past, and possessing known (positive definite) covariance matrices

$$\mathcal{E}\{\mathbf{u}(t_{k+1}, t_k)\mathbf{u}(t_{k+1}, t_k)^T\} = \Omega(t_{k+1}, t_k), \tag{1.3}$$

$$\mathcal{E}\{\epsilon_k \epsilon_k^T\} = R_k. \tag{1.4}$$

Initial conditions must be imposed on the system when $t = t_1$ in order to specify it

*Received by the editors June 6, 1990; accepted for publication (in revised form) January 15, 1991.

† Statistics Research Section, Mathematical Sciences School, Australian National University, G.P.O. Box 4, Canberra, A.C.T. 2601 Australia (mike@thrain.anu.edu.au).

‡ Department of Statistics, University of Glasgow, Glasgow G12 8QW, Scotland (gata36@vms1.glasgow.ac.uk).

completely. Here these are assumed to have the form of an initial estimate $\mathbf{x}_{1|0}$ of $\mathbf{x}_1$ together with the corresponding covariance matrix

$$\mathcal{E}\{(\mathbf{x}_1 - \mathbf{x}_{1|0})(\mathbf{x}_1 - \mathbf{x}_{1|0})^T\} = S_{1|0}.$$

This recursive estimation problem has the solution (for example, Anderson and Moore [1])

$$\mathbf{x}_{k|k-1} = T(t_k, t_{k-1})\mathbf{x}_{k-1|k-1}, \tag{1.5a}$$

$$S_{k|k-1} = T(t_k, t_{k-1})S_{k-1|k-1}T(t_k, T_{k-1})^T + \Omega(t_k, t_{k-1}), \tag{1.5b}$$

$$\varsigma_k = \mathbf{y}_k - H_k\mathbf{x}_{k|k-1}, \tag{1.5c}$$

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + S_{k|k-1}H_k^T(H_kS_{k|k-1}H_k^T + R_k)^{-1}\varsigma_k, \tag{1.5d}$$

$$S_{k|k} = S_{k|k-1} - S_{k|k-1}H_k^T(H_kS_{k|k-1}H_k^T + R_k)^{-1}H_kS_{k|k-1}. \tag{1.5e}$$

To obtain an estimate of $\mathbf{x}_k$ given data for $t = t_1, t_2, \cdots, t_n$ it is necessary to perform the so-called smoothing step, which has the form of a backwards recursion,

$$\mathbf{x}_{i|n} = \mathbf{x}_{i|i} + A(t_{i+1}, t_i)(\mathbf{x}_{i+1|n} - \mathbf{x}_{i+1|i}), \tag{1.6a}$$

$$A(t_{i+1}, t_i) = S_{i|i}T(t_{i+1}, t_i)^T S_{i+1|i}^{-1}, \tag{1.6b}$$

where $A(t_{i+1}, t_i)$ is called the interpolation gain. The recursion for the corresponding covariance matrix $S_{i|n}$ is

$$S_{i|n} = A(t_{i+1}, t_i)S_{i+1|n}A(t_{i+1}, t_i)^T. \tag{1.7}$$

The importance of the smoothing step which gives the dependence on all the data has been underplayed in comparison with the filter equations which give the innovations ς_k and hence provide a basis both for prediction and for maximum likelihood calculations. However, recently the smoothing formulae (1.6), (1.7) have been made more accessible as a result of several neat derivations. See, for example, [9].

Stable evaluation of the filter equations has received some attention, and it is known that so-called square root filters based on the Choleski decomposition of the covariance matrices have preferable numerical properties [2]. An attempt to produce a corresponding square root implementation of the smoothing algorithm has also been made [3]. Again improved numerical properties have been reported [11]. An important feature of the filtering and smoothing algorithms is that they exploit the sparsity of the system in the sense that they have $O(n)$ complexity estimates for large n and fixed p.

An alternative approach which yields the smoothed values directly is due to Duncan and Horne [4]. They show the equivalence of the estimation problem with the generalised least squares problem

$$\min \mathbf{r}^T V^{-1}\mathbf{r}; \qquad \mathbf{r} = X\boldsymbol{\beta} - \mathbf{y} \tag{1.8}$$

where

$$\boldsymbol{\beta} = \mathbf{x}_1 \times \mathbf{x}_2 \times \cdots \times \mathbf{x}_n,$$

$$\mathbf{y} = \begin{bmatrix} \mathbf{x}_{1|0} \\ \mathbf{y}_1 \end{bmatrix} \times \begin{bmatrix} 0 \\ \mathbf{y}_2 \end{bmatrix} \times \cdots \times \begin{bmatrix} 0 \\ \mathbf{y}_n \end{bmatrix},$$

$$X = \begin{bmatrix} I & 0 & 0 & \dots & 0 & 0 \\ H_1 & 0 & 0 & \dots & 0 & 0 \\ -T(t_2,t_1) & I & 0 & \dots & 0 & 0 \\ 0 & H_2 & 0 & \dots & 0 & 0 \\ 0 & -T(t_3,t_2) & I & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -T(t_n,t_{n-1}) & I \\ 0 & 0 & 0 & \dots & 0 & H_n \end{bmatrix}, \tag{1.9}$$

and

$$V = diag\{S_{1|0}, R_1, \Omega(t_2,t_1), R_2, \cdots, \Omega(t_n,t_{n-1}), R_n\}. \tag{1.10}$$

This formulation is the basis of an important square root algorithm for the Kalman filter [10]. Here the idea is to factorize

$$V = LL^T \tag{1.11}$$

and to transform the minimization problem to

$$\min \mathbf{s}^T\mathbf{s}; \qquad \mathbf{s} = L^{-1}X\boldsymbol{\beta} - L^{-1}\mathbf{y}. \tag{1.12}$$

This problem can now be solved by standard orthogonalization techniques [5]. By taking account of the structure of $L^{-1}X$ the steps of the filter are mirrored accurately, and the result is a recursive system for $\mathbf{x}_{k|k}$ and $S_{k|k}^{-T/2}$. The appearance of the inverse covariance matrix identifies this process as an *information filter*. Also the smoothing step appears naturally as the backsubstitution to complete the solution of the least squares problem. The result is a very satisfactory algorithm, and it has an additional attractive feature that if the initial condition is specified by means of a vague prior (for example, by setting

$$\mathbf{x}_{1|0} = 0, \qquad S_{1|0} = \gamma^2 I \tag{1.13}$$

and then letting $\gamma^2 \uparrow \infty$) then the information filter often copes without difficulty while special starting procedures are necessary for methods which recur the covariance matrix rather than its inverse.

However, there is another side to the coin because it is possible for the covariance matrix to become small and/or potentially ill-conditioned through the accumulation of increasingly precise information about some or all components of the state vector. In these circumstances it seems that a filter which recurs the covariance matrix rather than its inverse could be preferable. One application where both theoretical and practical studies show superior stability for the covariance form of the filter has been identified in Osborne and Prvan [8]. They consider the use of the Kalman filter to compute smoothing splines, and show that $\Omega(t_{i+1}, t_i)$ is necessarily ill-conditioned (the ill-conditioning increasing with the order of smoothing required in the spline) so that explicit formation of L^{-1} can introduce problems.

Here a recursive algorithm, based on orthogonal factorization techniques, which recurs the square root of the state covariance matrix, and which automatically provides the information necessary for the smoothing step is presented. The new feature is the manner in which the information necessary for the smoothing step is generated. Our

approach is to make use of a two-stage recursion based on an application of the Duncan and Horn formulation. It is a generalisation and extension of an algorithm developed for the calculation of smoothing splines in [8]. In many ways it appears to be the covariance analog of the Paige and Saunders algorithm which could equally well have been developed in a similar recursive form, and the work required in the two methods is very similar.

2. Development of a covariance filter. To develop a covariance filter based on (1.8) it is necessary to take a different approach to the solution of the generalised least squares problem that provided the basis for the Paige and Saunders information filter algorithm. The idea is to set

$$\mathbf{r} = L\mathbf{s} \tag{2.1}$$

in (1.8) to obtain the problem in the form

$$\min \mathbf{s}^T\mathbf{s}; \qquad L\mathbf{s} = X\boldsymbol{\beta} - \mathbf{y}. \tag{2.2}$$

Let $X : R^q \to R^l$ have rank q; and define $Z : R^{l-q} \to R^l$ by the conditions

(1) $Z^TX = 0$,

(2) Z has its maximum rank $l - q$.

Then (2.2) has the alternative formulation [7]

$$\min \mathbf{s}^T\mathbf{s}; \qquad Z^TL\mathbf{s} = -Z^T\mathbf{y}. \tag{2.3}$$

Typically this will lead to a more compact problem specification when $l - q$ is small. This will prove to be the usual situation in the Kalman filter application .

To solve (2.3) let

$$L^TZ = Q\begin{bmatrix} U \\ 0 \end{bmatrix} = [Q_1 \quad Q_2]\begin{bmatrix} U \\ 0 \end{bmatrix}. \tag{2.4}$$

Substituting in (2.3) gives

$$U^TQ_1^T\hat{\mathbf{s}} = -Z^T\mathbf{y} \tag{2.5}$$

while the least squares condition gives

$$Q_2^T\hat{\mathbf{s}} = 0. \tag{2.6}$$

It follows that

$$\hat{\mathbf{s}} = -Q_1U^{-T}Z^T\mathbf{y}. \tag{2.7}$$

The constraint system in (2.2) is now consistent and any convenient and stable method can be used to recover $\hat{\boldsymbol{\beta}}$. In the Kalman filter the obvious procedure will prove to be a forward recurrence. This has a form familiar in the solution of differential equations where it is known that stability cannot always be guaranteed. But this is a problem already for the innovation sequence which satisfies the same basic recursion and is not peculiar to our method.

The particular generalised least squares problem considered here is defined recursively and is specified by

$$\mathbf{r} = \begin{bmatrix} \mathbf{x}_{i-1} - \mathbf{x}_{i-1|i-1} \\ \mathbf{x}_i - T(t_i, t_{i-1})\mathbf{x}_{i-1} \\ H_i\mathbf{x}_i - y_i \end{bmatrix}, \tag{2.8a}$$

$$V = \begin{bmatrix} S_{i-1|i-1} & 0 & 0 \\ 0 & \Omega(t_i, t_{i-1}) & 0 \\ 0 & 0 & R_i \end{bmatrix}. \tag{2.8b}$$

Here $l \leftarrow 2p+m$, $q \leftarrow 2p$,

$$\beta = \begin{bmatrix} \mathbf{x}_{i-1} \\ \mathbf{x}_i \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} \mathbf{x}_{i-1|i-1} \\ 0 \\ \mathbf{y}_i \end{bmatrix}$$

in adapting the above notation. In this case it is straightforward to write down an appropriate Z:

$$Z = \begin{bmatrix} T(t_i, t_{i-1})^T H_i^T \\ H_i^T \\ -I_m \end{bmatrix} \tag{2.9}$$

and

$$Z^T\mathbf{y} = -\mathbf{y}_i + H_i^T T(t_i, t_{i-1})\mathbf{x}_{i-1|i-1} = -\zeta_i \tag{2.10}$$

where ζ_i is the innovation at the current stage. It would appear to be necessary to work with at least a two-stage recursion in $\mathbf{x}$ as in (2.8)—that is, one which updates both $\mathbf{x}_{i-1}$ and $\mathbf{x}_i$—in order to obtain the information necessary for the derivation of the smoother step (1.6).

The procedure adopted in this section is the following. An algorithm is developed for solving the generalised least squares problem defined in (2.8). It is then verified inductively that the solution of this problem is identical with that produced by the Kalman filter. It is demonstrated

(1) that the resulting method is naturally in square root covariance form, and
(2) that the interpolation smoother in convenient square root form is an immediate by-product.

Appropriate tools for solving (2.8) can be developed by adapting the usual stepwise approach. In this, first the problem based on the first two blocks of (2.8) is solved to give $\mathbf{x}_{i|i-1}$. Then the observation equation is added to give $\mathbf{x}_{i|i}$. Carrying out this program suggests the construction in sequence of a pair of orthogonal transformations $Q^{(p)}$ and $Q^{(u)}$, where p and u stand for prediction and update, respectively.

$$Q^{(p)}: \begin{bmatrix} S_{i-1|i-1}^{T/2} T(t_i, t_{i-1})^T & S_{i-1|i-1}^{T/2} \\ \Omega(t_i, t_{i-1})^{T/2} & 0 \end{bmatrix} = \begin{bmatrix} Q_1^{(p)} & Q_2^{(p)} \end{bmatrix} \begin{bmatrix} S_{i|i-1}^{T/2} & K_{i1} \\ 0 & K_{i2} \end{bmatrix}, \tag{2.11}$$

$$Q^{(u)}: \begin{bmatrix} S_{i|i-1}^{T/2} & S_{i|i-1}^{T/2} H_i^T \\ 0 & R_i^{T/2} \end{bmatrix} = \begin{bmatrix} Q_1^{(u)} & Q_2^{(u)} \end{bmatrix} \begin{bmatrix} S_{i|i}^{T/2} & 0 \\ W_i^T & [R_i + H_i S_{i|i-1} H_i^T]^{T/2} \end{bmatrix} \tag{2.12}$$

where the superscript $T/2$ indicates that the upper Choleski factor is taken. Equations (2.11), (2.12) are related to systems suggested by Morf and Kailath [6] amongst others. However, the appearance of the quantities $K_{ij}, j = 1, 2$ which are used in the smoother step would appear to be an important point of difference. A priori this sequence of computations appears to have some attractive properties. The second equation involves a downdating step (compare (1.5e)), but the matrix which is actually inverted at this stage of the algorithm proves to be

$$[R_i + H_i S_{i|i-1} H_i^T]^{T/2},$$

and this is stabilized here. Also $S_{i|i}^{T/2}$ is augmented in the calculation of $S_{i+1|i}^{T/2}$ in (2.11) when $i \leftarrow i+1$. These orthogonal matrices can be built up by standard

procedures (for example, [6]). The more interesting case is $Q^{(u)}$, and the sequence of operations is indicated in Fig. 2.1 for the case $p = 3$, $m = 2$. Subscripted zeros indicate zeros introduced by the corresponding elementary reflectors, while the other numbers indicate the pattern of fill-in.

$$\begin{array}{ccccc} \times & \times & \times & 0_5 & 0_6 \\ . & \times & \times & 0_3 & 0_4 \\ . & . & \times & 0_1 & 0_2 \\ 5 & 3 & 1 & \times & \times \\ 6 & 4 & 2 & . & \times \end{array}$$

FIG. 2.1. *Application of elementary reflectors to generate* $Q^{(u)}$.

The identifications made in (2.11), (2.12) are verified readily (thus verifying this component of the induction). For example,

$$\begin{bmatrix} S_{i|i-1}^{1/2} & 0 \\ H_i S_{i|i-1}^{1/2} & R_i^{1/2} \end{bmatrix} \begin{bmatrix} S_{i|i-1}^{T/2} & S_{i|i-1}^{T/2} H_i^T \\ 0 & R_i^{T/2} \end{bmatrix} = \begin{bmatrix} S_{i|i-1} & S_{i|i-1} H_i^T \\ H_i S_{i|i-1} & R_i + H_i S_{i|i-1} H_i^T \end{bmatrix}$$

and

$$\begin{bmatrix} S_{i|i}^{1/2} & W_i \\ 0 & [R_i + H_i S_{i|i-1} H_i^T]^{1/2} \end{bmatrix} \begin{bmatrix} S_{i|i}^{T/2} & 0 \\ W_i^T & [R_i + H_i S_{i|i-1} H_i^{T/2}]^{T/2} \end{bmatrix}$$
$$= \begin{bmatrix} S_{i|i} + W_i W_i^T & W_i [R_i + H_i S_{i|i-1} H_i^T]^{T/2} \\ [R_i + H_i S_{i|i-1} H_i^T]^{1/2} W_i^T & R_i + H_i S_{i|i-1} H_i^T \end{bmatrix}.$$

Equating the right-hand sides gives

$$W_i = S_{i|i-1} H_i^T [R_i + H_i S_{i|i-1} H_i^T]^{-T/2} \tag{2.13}$$

and also permits the immediate verification that $S_{i|i}$ is updated correctly.

To construct $\hat{\mathbf{s}}$ note that, using (2.11),

$$L^T Z = \begin{bmatrix} S_{i-1|i-1}^{T/2} T(t_i, t_{i-1})^T H_i^T \\ \Omega(t_i, t_{i-1})^{T/2} H_i^T \\ -R_i^{T/2} \end{bmatrix}$$
$$= \begin{bmatrix} Q_1^{(p)} & Q_2^{(p)} & 0 \\ 0 & 0 & -I_m \end{bmatrix} \begin{bmatrix} S_{i|i-1}^{T/2} H_i^T \\ 0 \\ R_i^{T/2} \end{bmatrix}$$
$$= \begin{bmatrix} Q_1^{(p)} & Q_2^{(p)} & 0 \\ 0 & 0 & -I_m \end{bmatrix} \begin{bmatrix} I_p & 0 & 0 \\ 0 & 0 & I_p \\ 0 & I_m & 0 \end{bmatrix} \begin{bmatrix} S_{i|i-1}^{T/2} H_i^T \\ R_i^{T/2} \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} Q_1^{(p)} & Q_2^{(p)} & 0 \\ 0 & 0 & -I_m \end{bmatrix} \begin{bmatrix} I_p & 0 \\ 0 & 0 \\ 0 & I_m \end{bmatrix} Q_2^{(u)} [R_i + H_i S_{i|i-1} H_i^T]^{T/2}$$
$$= \begin{bmatrix} Q_1^{(p)} & 0 \\ 0 & -I_m \end{bmatrix} Q_2^{(u)} [R_i + H_i S_{i|i-1} H_i^T]^{T/2}. \tag{2.14}$$

Using (2.10) gives

$$\hat{\mathbf{s}} = \begin{bmatrix} Q_1^{(p)} & 0 \\ 0 & -I_m \end{bmatrix} Q_2^{(u)} [R_i + H_i S_{i|i-1} H_i^T]^{-1/2} \boldsymbol{\zeta}_i. \tag{2.15}$$

Substituting back into (2.2) and deleting the last m equations gives

$$\begin{bmatrix} I_p & 0 \\ -T(t_i,t_{i-1}) & I_p \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}}_{i-1} \\ \hat{\mathbf{x}}_i \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{i-1|i-1} \\ 0 \end{bmatrix} + \begin{bmatrix} S^{1/2}_{i-1|i-1} & 0 \\ 0 & \Omega(t_i,t_{i-1})^{1/2} \end{bmatrix} \hat{\mathbf{s}}_- \tag{2.16}$$

where

$$\hat{\mathbf{s}}_- = \begin{bmatrix} Q_1^{(p)} & 0 \end{bmatrix} Q_2^{(u)} [R_i + H_i S_{i|i-1} H_i^T]^{-1/2} \boldsymbol{\zeta}_i. \tag{2.17}$$

The updated state vector corresponds to $\hat{\mathbf{x}}_i$ and is given by

$$\begin{aligned} \hat{\mathbf{x}}_i &= T(t_i,t_{i-1})\mathbf{x}_{i-1|i-1} + \begin{bmatrix} T(t_i,t_{i-1})S^{1/2}_{i-1|i-1} & \Omega(t_i,t_{i-1})^{1/2} \end{bmatrix} \hat{\mathbf{s}}_- \\ &= \mathbf{x}_{i|i-1} + \begin{bmatrix} S^{1/2}_{i|i-1} & 0 \end{bmatrix} Q_2^{(u)} [R_i + H_i S_{i|i-1} H_i^T]^{-1/2} \boldsymbol{\zeta}_i \\ &= \mathbf{x}_{i|i-1} + W_i [R_i + H_i S_{i|i-1} H_i^T]^{-1/2} \boldsymbol{\zeta}_i \\ &= \mathbf{x}_{i|i-1} + S_{i|i-1} H_i^T [R_i + H_i S_{i|i-1} H_i^T]^{-1} \boldsymbol{\zeta}_i \\ &= \mathbf{x}_{i|i} \end{aligned} \tag{2.18}$$

where use has been made of the relations defining $Q^{(p)}$, $Q^{(u)}$, and W_i. This verifies that the solution of the recursively defined least squares problem (2.8) does indeed update the Kalman filter, and hence provides a basis for its implementation. Further, (2.11), (2.12), (2.17), (2.18) show that the implementation is a true square root filter (in particular, the second line of (2.18) shows that much more is done than just evaluating the filter equations (1.5) using the triangular factors), that it recurs the square root of the covariance matrix, and that it is based on orthogonal factorizations which are used explicitly in the filter calculations. In particular, the third line of (2.18), which involves quantities computed in the downdating step (2.12), would appear to provide the most appropriate form to use in updating the state vector. Assuming that the calculation of $Q^{(p)}$ and $Q^{(u)}$ is based on surgery using elementary reflectors, then a quick operations count shows that it requires the same number of transformations as the Paige and Saunders information filter.

To evaluate the interpolation gain $A(t_i,t_{i-1})$ consider the equation for $\hat{\mathbf{x}}_{i-1} = \mathbf{x}_{i-1|i}$ from (2.16). This gives

$$\begin{aligned} \hat{\mathbf{x}}_{i-1} &= \mathbf{x}_{i-1|i-1} + \begin{bmatrix} S^{1/2}_{i-1|i-1} & 0 \end{bmatrix} \begin{bmatrix} Q_1^{(p)} & 0 \end{bmatrix} Q_2^{(u)} [R_i + H_i S_{i|i-1} H_i^T]^{-1/2} \boldsymbol{\zeta}_i \\ &= \mathbf{x}_{i-1|i-1} + \begin{bmatrix} K_{i1}^T & 0 \end{bmatrix} Q_2^{(u)} [R_i + H_i S_{i|i-1} H_i^T]^{-1/2} \boldsymbol{\zeta}_i \\ &= \mathbf{x}_{i-1|i-1} + \begin{bmatrix} K_{i1}^T & 0 \end{bmatrix} \begin{bmatrix} S^{T/2}_{i|i-1} H_i^T \\ R_i^T \end{bmatrix} [R_i + H_i S_{i|i-1} H_i^T]^{-1} \boldsymbol{\zeta}_i \\ &= \mathbf{x}_{i-1|i-1} + K_{i1}^T S^{-1}_{i|i-1} (\mathbf{x}_{i|i} - \mathbf{x}_{i|i-1}), \end{aligned}$$

where (2.11), (2.12), and the filter equations have been used. Comparing this with (1.6a) and noting that $n = i$ for our two-stage recursion gives

$$A(t_i,t_{i-1}) = K_{i1}^T S^{-1/2}_{i|i-1}. \tag{2.19}$$

In particular, quantities needed for the evaluation of $A(t_i,t_{i-1})$ have been constructed in (2.11). This step should also be compared with the Paige and Saunders method. It is clear that the two algorithms stand as a complementary pair.

To compute the associated update for the covariance matrix $S_{i|n}$ in square root form note that by (1.7), (2.19)

$$S_{i|n} = L_i L_i^T \tag{2.20}$$

where

$$L_i = K^T_{(i+1)1} S^{-1/2}_{i+1|i} S^{1/2}_{i+1|n}. \tag{2.21}$$

It follows that

$$L_i^T = Q^{(s)} S^{T/2}_{i|n} \tag{2.22}$$

where $Q^{(s)}$ is the orthogonal matrix in the reduction of L_i^T to upper triangular form.

Acknowledgment. A significant amount of the work reported here was carried out while the first author was visiting the Statistics Department at Texas A&M University: support, encouragement, and hospitality are gratefully acknowledged.

REFERENCES

[1] B. D. O. ANDERSON AND J. B. MOORE, *Optimal Filtering*, Prentice-Hall, Englewood Cliffs, NJ, 1979.

[2] G. J. BIERMAN, *Factorization Methods for Discrete Sequential Estimation*, Academic Press, New York, 1977.

[3] ———, *A new computationally efficient fixed-interval, discrete-time smoother*, Automatica, 19 (1983), pp. 503–511.

[4] D. B. DUNCAN AND S. D. HORN, *Linear dynamic recursive estimation from the point of view of regression analysis*, J. Amer. Stat. Assoc., 67 (1972), pp. 816–821.

[5] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Second Edition, Johns Hopkins University Press, Baltimore, MD.

[6] M. MORF AND T. KAILATH, *Square-root algorithms for linear least-squares estimation*, IEEE Trans. Automatic Control, 20 (1975), pp. 487–497.

[7] M. R. OSBORNE, *Finite Algorithms in Optimization and Data Analysis*, John Wiley & Sons, New York, 1985.

[8] M. R. OSBORNE AND T. PRVAN, *Smoothness and conditioning in generalised smoothing spline calculations*, J. Austral. Math. Soc., Ser. B, 30 (1988), pp. 43–56.

[9] ———, *A direct derivation of the interpolation smoother*, Appl. Math. Letters, (1990), to appear.

[10] C. C. PAIGE AND M. A. SAUNDERS, *Least squares estimation of discrete linear dynamic systems using orthogonal transformations*, SIAM J. Numer. Anal., 14 (1977), pp. 180–193.

[11] T. PRVAN AND M. R. OSBORNE, *A square-root, fixed-interval, discrete-time smoother*, J. Austral. Math. Soc., Ser. B, 30 (1988), pp. 57–68.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1332–1350, November 1991
© 1991 Society for Industrial and Applied Mathematics
007

STRUCTURE-PRESERVING AND RANK-REVEALING QR-FACTORIZATIONS*

CHRISTIAN H. BISCHOF† AND PER CHRISTIAN HANSEN‡

Abstract. The rank-revealing QR-factorization (RRQR-factorization) is a special QR-factorization that is guaranteed to reveal the numerical rank of the matrix under consideration. This makes the RRQR-factorization a useful tool in the numerical treatment of many rank-deficient problems in numerical linear algebra. In this paper, a framework is presented for the efficient implementation of RRQR algorithms, in particular, for sparse matrices. A sparse RRQR-algorithm should seek to preserve the structure and sparsity of the matrix as much as possible while retaining the ability to capture safely the numerical rank. To this end, the paper proposes to compute an initial QR-factorization using a restricted pivoting strategy guarded by incremental condition estimation (ICE), and then applies the algorithm suggested by Chan and Foster to this QR-factorization. The column exchange strategy used in the initial QR factorization will exploit the fact that certain column exchanges do not change the sparsity structure, and compute a sparse QR-factorization that is a good approximation of the sought-after RRQR-factorization. Due to quantities produced by ICE, the Chan/Foster RRQR algorithm can be implemented very cheaply, thus verifying that the sought-after RRQR-factorization has indeed been computed. Experimental results on a model problem show that the initial QR-factorization is indeed very likely to produce RRQR-factorization. Fill-in is comparable with other methods, and little additional effort is required to implement the Chan/Foster postprocessing step. Compared to alternative strategies, the new algorithm allows to a greater extent the use of Householder transformations (instead of Givens rotations), and requires fewer touches of the data, while requiring not more (and sometimes substantially fewer) floating-point operations. These characteristics make the algorithm attractive for sparse problems, and a good candidate for parallel computers as well.

Key words. rank-revealing QR-factorization, rank-deficient problems, numerical rank, incremental condition estimation, sparse matrices

AMS(MOS) subject classifications. 65F25, 65F50

1. Introduction.

In this paper we are concerned with least-squares problems where the matrix in consideration is rank-deficient in the sense that its condition number is very large and that its numerical rank is well determined (see, e.g., [9] and [24]). These problems arise for example in geodesy [25], computer-aided design [29], trust region methods for the solution of nonlinear least-squares problems [35], the solution of integral equations [17] and in the calculation of splines [14], [28].

The common way to approach this problem is via a *QR-factorization* $AP = QR$ of A. For an $m \times n$ ($m \geq n$) matrix A, P is an $n \times n$ permutation matrix, Q is an $m \times n$ matrix with orthonormal columns ($Q^TQ = I$), and R is an upper triangular $n \times n$ matrix. The hope is that in the resulting QR-factorization the numerical rank of A will reveal itself by a trailing subblock of R with numerically small elements. If $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ are the singular values of A and we partition R into

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \tag{1}$$

* Received by the editors September 26, 1989; accepted for publication (in revised form) November 20, 1990. This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy under contract W-31-109-Eng-38.

† Mathematics and Computer Science Division, Argonne National Laboratories, 9700 S. Cass Ave., Argonne, Illinois 60439.

‡ UNI•C (Danish Computing Center for Research and Education), Building 305, Technical University of Denmark, DK-2800 Lyngby. The work of this author was also supported by the Danish Natural Science Foundation.

with an $(n-r) \times (n-r)$ lower right-hand block R_{22}, then it is easy to show [27, p. 19] that $\sigma_{r+1}(A) \leq \|R_{22}\|_2$ and that $\sigma_r(R_{11}) \leq \sigma_r(A)$. Hence, if $\|R_{22}\|_2$ is small and R_{11} is well conditioned, then A can be considered to have at least $n-r$ numerically dependent columns. Such a factorization is called a *rank-revealing QR-factorization* (RRQR-factorization) if $n-r$ is in fact the dimension of the numerical null space ($n-r$ is always a lower bound) and as a consequence r is the numerical rank of A. For a survey of the applications of RRQR-factorizations in numerical linear algebra, see [12].

Let us from now on assume that A has $n-r$ small singular values and that there is a well-defined gap between σ_r and σ_{r+1}. It is shown in [24] that a well-defined gap is necessary to make a sensible decision on the numerical rank of A.

The standard technique is the *QR-factorization with column pivoting* [26], which constructs Q by a sequence of Householder transformations. Here the permutation matrix P is chosen such that, in each step of the algorithm, the pivot column is the one that is farthest away (in the two-norm sense) from the subspace spanned by the columns that have already been selected [27, p. 168, P.6.4-5]. That is, if res_j denotes the 2-norm of the vector consisting of the last $m-k$ entries of column j in the partial QR-factorization of A, then one chooses as pivot column the one which satisfies

(1A) $$\boxed{pvt \leftarrow \arg\max_{i \leq j \leq n}\{res_j\}.}$$

As a consequence of this pivoting strategy, the diagonal entries of R decrease monotonically in magnitude. The modulus of the ith diagonal entry is used as an estimate for the singular values σ_i of A, and the rank estimate is determined by the number of trailing diagonal entries numerically less than τ in modulus, where τ is a suitably chosen threshold.

This technique, however, is not entirely reliable and Foster [19] has recently shown that the probability that this scheme fails to detect rank deficiecy is not insignificant.

An alternative pivoting strategy has been suggested by Foster [18] and Chan [10]. Their idea is first to compute any QR-factorization of A and then "peel off" the small singular values of R one after the other. In each step, one must compute an approximate null-vector and permute the columns according to a permutation determined by the location of the largest element (in modulus) in this null-vector. Application of this permutation leads to a Hessenberg form, and Givens rotations are used to restore triangular form. This method can be considered as a postprocessor to any QR-factorization, and Chan proves that if the rank r is not too small compared to n, this algorithm is guaranteed to compute an RRQR-factorization in the sense that the norm of R_{22} in (1) will be small.

The situation is more complicated when A is large and has a certain structure one would like to preserve. When A has full rank, the sparse QR-factorization schemes described, for example, in [21], [22], and [31] work well with respect to preserving sparsity (they all depend on identifying a column permutation P of A such that the upper triangular factor of AP can be computed with little fill). When A is rank-deficient, Heath [30] and Ng [36] suggest schemes that allow one to compute the minimum-norm solution while preserving an initially chosen column ordering. Another alternative is to use *threshold pivoting* where at step k one chooses as pivot column j the one which is closest to column k and satisfies

(1B) $$\boxed{res_j \geq \max_{k<i\leq n}\{res_i\}/\rho_{colpiv}}$$

for some threshold $\rho_{colpiv} \geq 1$. This can be viewed as trying to preserve the initial

ordering as much as possible while selecting columns that are reasonably independent of the ones already selected. Both threshold pivoting and the methods of Heath and Ng rely on the occurrence of small diagonal elements in the triangular factor, but one can construct triangular matrices which are rank-deficient even though no diagonal element is particularly small (see [19] and [27, p. 167]). Hence those methods are not guaranteed to capture the numerical rank.

The Chan/Foster algorithm seems well suited for sparse matrices, because an initial QR-factorization without column exchanges can take advantage of the sparsity of the matrix, and because the Givens rotations in the postprocessing step introduce fill-in only in columns that are recognized as "dependent" and permuted to the end of A (this result is mentioned without proof in [18] and later proved by Björck in [8, Thm. 4.1]). Nevertheless, the Chan/Foster algorithm postprocessing step may require nonnegligible computational overhead, due to the computation of the null-vectors, and the Givens rotations that have to be applied and stored.

In this paper we describe a new approach to implementing an RRQR algorithm. Instead of not doing any column exchanges in the initial QR-factorization, we employ column exchanges that are restricted in such a way that they do not produce any extra fill (except in columns that are recognized as dependent). For example, permuting columns in cliques of the elimination graph (see [20]), or in a fully assembled front in a multifrontal scheme [16], [23] does not produce any extra fill. Incremental condition estimation [4], [6] is used to identify columns suggested by the restricted pivoting strategy as either dependent or independent. In this fashion we simultaneously preserve the structure of the matrix, compute an approximation to an RRQR-factorization, and produce approximations to the small singular values and left singular vectors. Using this information, we can then implement the Chan/Foster backward pass quite inexpensively since most of the RRQR-factorization is already in place.

The outline of the paper is as follows. In the next section we discuss the concept of numerical rank in more detail and briefly review the ideas behind the Chan/Foster algorithm for computing an RRQR-factorization. Section 3 describes how we compute a QR factorization that is a good approximation of the RRQR structure being sought, using incremental condition estimation and restricted column pivoting. In §4 we show why some column pivoting is a good idea if one wants the initial QR-factorization to be close to an RRQR-factorization. The next section then shows how we can reduce the computational complexity of the Chan/Foster postprocessing scheme due to the information computed in the forward pass while maintaining reliability in revealing the rank. We then present some numerical experiments illustrating that our initial QR-factorization indeed produces a good approximate RRQR-factorization, and that the modified backward pass is indeed inexpensive. Lastly, we summarize our results.

2. Numerical rank and RRQR-factorizations. A matrix A is called *ill-conditioned* if there exists a vector x with $\|x\|_2 = 1$ such that $\|Ax\|_2 \ll \|A\|_2$—hence the columns of A are almost linearly dependent. In this case, we say that x is a numerical null-vector of A. For such an ill-conditioned matrix, it is convenient to introduce the concept of numerical rank which, roughly speaking, is the number of significantly linearly independent columns of A.

To give a more rigorous definition of numerical rank, consider A's singular value decomposition (SVD):

$$A = U\Sigma V^T,$$

where $U = [u_1, \cdots, u_m]$ and $V = [v_1, \cdots, v_n]$ are orthogonal matrices, and

$\Sigma = \mathrm{diag}(\sigma_1, \cdots, \sigma_n)$ is an m-by-n diagonal matrix whose entries σ_i are the singular values of A. Then the *numerical rank* r of A, with respect to the threshold τ, is defined as the number of singular values of A strictly larger than τ: $\sigma_r > \tau \geq \sigma_{r+1}$. Columns $v_{r+1}, \cdots, v_n$ of V are right numerical null-vectors of A since they satisfy $\|Av_i\|_2 \leq \tau$.

As mentioned before, an RRQR-factorization of A is a special QR-factorization which is guaranteed to reveal the numerical rank of A. An RRQR-factorization satisfies

$$AP = QR = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{2}$$

where R_{11} is a well-conditioned upper triangular $r \times r$ matrix, r is the numerical rank of A, and $\|R_{22}\|_2 \ll \|A\|_2 = \|R\|_2$. An RRQR-factorization always exists [33], but it is not unique: any P, Q and R satisfying these conditions constitute an RRQR-factorization of A.

The basic problem in computing an RRQR-factorization of A is to find a column permutation P and a matrix Q such that the requirements mentioned above are satisfied. The following theorem is useful to determine such a pair.

THEOREM 2.1 ([11]). *Let Y be an $n \times p$ matrix ($n \geq p$), partitioned as $Y = \binom{Y_1}{Y_2}$ with Y_2 being $p \times p$. If Y and P satisfy $\|APY\|_2 = \|RY\|_2 = \epsilon$, then R_{22} satisfies*

$$\|R_{22}\|_2 \leq \epsilon \|Y_2^{-1}\|_2. \tag{3}$$

For the special case $p = 1$, one can take $Y = v_n$ and P the permutation that brings the largest element in absolute value of v_n to the last position [24]. This idea was extended by Chan [10] and Foster [18] to higher dimensions, i.e., the case $p > 1$. The idea is first to compute any QR-factorization of A and then "peel off" the small singular values of R one by one: for $i = n, n-1, \cdots, r$ one computes the right null-vectors $y^{(i)}$ of increasingly smaller $i \times i$ triangular submatrices. One then applies a cyclic shift that permutes column j to position i if

$$\boxed{|y_j^{(i)}| = \|y^{(i)}\|_\infty.} \tag{2A}$$

This changes the triangular structure into a Hessenberg form, which is then restored by means of Givens rotations. The permuted $y^{(i)}$'s are padded with zeros and collected in the $n \times p$ matrix Y (so Y_2 in (3) is upper triangular). During this process, the numerical rank is identified by monitoring lower and upper bounds for the singular values σ_i of A. This is done by means of the following theorem:

THEOREM 2.2 ([10]). *Let $Y_2^{(i)}$ denote the bottom $(n-i+1) \times (n-i+1)$ block of Y. Also, let $\delta_i \equiv \sigma_{\min}(R(1:i, 1:i))$ and $\theta_i \equiv \|R(i:n, i:n)\|_2$. Then the ith singular value σ_i of A satisfies*

$$\frac{\sigma_i}{\sqrt{n-i+1}\,\|(Y_2^{(i)})^{-1}\|_2} \leq \delta_i \leq \sigma_i \leq \theta_i \leq \sigma_i \sqrt{n-i+1}\,\|(Y_2^{(i)})^{-1}\|_2\,. \tag{4}$$

It is shown in [11] that for Y constructed in the above-mentioned fashion, we have

$$\|(Y_2^{(i)})^{-1}\|_2 < \sqrt{n}\,2^{n-i},$$

and hence the bounds in (4) are guaranteed to be tight as long as $p = n - r$ is not too large. This allows the implementation of the algorithm in an adaptive fashion: The

algorithm can be terminated if the lower bounds indicate that all the small singular values have been revealed. The extra work in the Chan/Foster algorithm, beyond the initial QR-factorization, is $O((n-r)n^2)$. This overhead is partly due to the null-vector computations and partly due to the annihilation of lower bidiagonal elements that appear in each postprocessing step when the columns of R are put in their final position.

Alternatively, one can derive a *threshold version* of this RRQR-strategy by choosing the pivot column j to be that column, closest to column i, that satisfies

$$\boxed{|y_j^{(i)}| \geq \|y^{(i)}\|_\infty / \rho_{rrqr}} \tag{2B}$$

for some threshold $\rho_{rrqr} \geq 1$. For a moderate threshold ρ_{rrqr} this strategy trades a slightly worse-conditioned Y_2 (and as a result slightly worse bounds in (4)) for the savings resulting from exchanging column i with a "near-by" column. We return to these aspects in §5.

3. Computing an approximate RRQR-factorization. Our key idea is to exploit the fact that even for sparse matrices, some column exchanges can be performed without generating fill-in, and to try to approximate an RRQR-factorization as much as possible using suitably restricted column exchanges. So we must reconcile two objectives:

- Restrict column exchanges in order not to destroy the structure of A, and
- Identify "dependent" columns and permute them to the back of A.

To set the stage for the description of our algorithm, assume that, at the kth stage, we have used k Householder transformations $H_1, \cdots, H_k$ to compute the partial QR-factorization $AP^{(k)} = Q^{(k)}A^{(k)}$, where

$$Q^{(k)} = H_1 \cdots H_k, \qquad A^{(k)} = \begin{pmatrix} R_{11}^{(k)} & R_{12}^{(k)} \\ 0 & A_{22}^{(k)} \end{pmatrix}. \tag{5}$$

Here, $A_{22}^{(k)}$ is $(m-k) \times (n-k)$, and $R_{11}^{(k)}$ is a $k \times k$ matrix such that $\sigma_{\min}(R_{11}^{(k)}) > \tau$, where τ is the threshold below which we can consider a singular value to be numerically zero. We then determine a new candidate pivot column by using the traditional pivoting strategy (1A) or (1B), but *restricted to a subset of columns of* $A_{22}^{(k)}$. Which subset of columns we consider as pivot candidates depends on the application. For a distributed-memory machine, pivoting within the columns residing in the local memory is a natural choice [5]. For general sparse matrices column interchanges within the clique of the elimination graph [20], or within a fully assembled front in a multifrontal scheme [16], [23], do not introduce more fill-in. Even for a dense problem performance improvements can be obtained by restricting pivoting to arrive at a block algorithm [2]. So let us assume that our "pivot window" is limited to a small number $s_{\text{window}}^{(k)}$ of adjacent columns with $s_{\text{window}}^{(k)} \ll n$. In most applications, the size of the pivot window will change throughout the course of the algorithm.

In order to compute an RRQR-factorization, however, we have to guard against our restricted column exchange strategy selecting pivot columns that are essentially dependent on the ones previously chosen. That is, after we would pivot the pivot column into position $k+1$ and apply the $(k+1)$st Householder transformation, we would obtain a $(k+1) \times (k+1)$ upper triangular matrix

$$R_{11}^{(k+1)} = \begin{pmatrix} R_{11}^{(k)} & v \\ 0 & \gamma \end{pmatrix}. \tag{6}$$

Here v consists of the first k elements of the pivot column, and since γ is determined by the information used to select the pivot column, we know $R_{11}^{(k+1)}$ without actually having to compute the $(k+1)$st Householder transformation. Since the smallest singular value of a matrix measures the distance to the set of rank-deficient ones, we can use the smallest singular value of $R_{11}^{(k+1)}$ to decide whether we will actually accept the suggested $(k+1)$st pivot column or reject it.

To do this efficiently, we rely on *incremental condition estimation* (ICE) [4], [6]. Given an approximate left null-vector x_k of R_{11}^k in the sense that

$$\hat{\delta}_k \equiv \|x_k^T R_{11}^k\|_2 \approx \sigma_{\min}(R_{11}^k),$$

ICE computes with only $3k$ flops $s = \sin(\phi)$ and $c = \cos(\phi)$ such that

$$\hat{\delta}_{k+1} \equiv \|x_{k+1}^T R_{11}^{(k+1)}\|_2 \approx \sigma_{\min}(R_{11}^{(k+1)}),$$

where

$$x_{k+1}^T = (s \cdot x_k^T, c).$$

Incremental condition estimation is crucial in restricting column exchanges since it allows us to validate the columns suggested by the restricted pivoting strategy. Based on the estimate $\hat{\delta}_{k+1}$ from ICE, we will either accept the pivot column (if $\hat{\delta}_{k+1} > \tau$), pivot it into the $(k+1)$st position and generate a Householder transformation based on it, or reject it if $\hat{\delta}_{k+1} < \tau$. In the latter case, it is likely that all remaining columns in the pivot window are linearly dependent on the columns previously chosen, and should be permuted to the back of A. To this end, a cyclic shift to the left is most appropriate here since it preserves as much as possible the structure of the leading triangular submatrix. As one of the referees pointed out, this scheme is in certain respects similar to the "delayed elimination" scheme in [16] whose structural implications have been discussed in [34].

So let us then assume that after $\tilde{r}_{ice}$ steps we have reached a stage where the last $n - \tilde{r}_{ice}$ columns of $A^{(\tilde{r}_{ice})}$ in (5) are all considered "dependent," that is, $\hat{\delta}_{\tilde{r}_{ice}} > \tau$ and any of the last $n - \tilde{r}_{ice}$ columns in $A^{(\tilde{r}_{ice})}$ has either been rejected at some stage or was in a pivot window where the candidate pivot column was rejected. Due to the restricted pivoting strategy no additional fill-in (compared to a QR-factorization without any column exchanges) has occurred in $R_{11}^{(\tilde{r}_{ice})}$, and $\tilde{r}_{ice}$ should be close to the numerical rank of A.

We now complete the QR-factorization of A by computing a QR-factorization with traditional column pivoting (strategy (1A)) of $A_{22}^{(\tilde{r}_{ice})}$. Since fill-in tends to concentrate in the lower right-hand side, $A_{22}^{(\tilde{r}_{ice})}$ is likely to be dense if $\tilde{r}_{ice}$ is not too much smaller than n. So there is not too much point in worrying about structure anymore.

Note that we may find some more "independent" columns in the set of columns that had been rejected in the restricted pivoting phase so that at the end of this initial QR-factorization the numerical rank estimate r_{ice} may be larger than $\tilde{r}_{ice}$. This is due to the fact that we reject *all* columns in a pivot window if the candidate pivot column is considered dependent. But if the traditional column pivoting strategy failed and suggested the wrong column, some of the other columns in the pivot window might still have been acceptable.

While we compute the QR-factorization of the last $(n - r_{ice})$ columns, we continue to use incremental condition estimation to compute approximate left null-vectors $x^{(k)}$

and approximate smallest singular values $\hat{\delta}_k$ of $R_{11}^{(k)}$, $k = r_{ice}+1, \cdots, n$. We gather the vectors $x^{(k)}$ in the matrix W where the ith column w_i of W is identical to $x^{(r_{ice}+i)}$ augmented with $n-r_{ice}-i$ zeros. Thus, $W = \binom{W_1}{W_2}$ has upper trapezoidal form and W_2 is upper triangular. The information stored in W will save us work in the backward pass. We mention that W can be stored in a "factored" form by exploiting the special structure of the vectors produced by ICE: by storing only $x_{r_{ice}}$ and $(s_i, c_i), i = r_{ice}+1 \cdots, n$, we can cheaply reconstruct $x^i, i = r_{ice}+1, \cdots, n$. Since a fully assembled W would be dense, this is an important consideration in the sparse setting.

4. The importance of column pivoting. One would think that it would be most advantageous not to pivot at all since this would disturb the structure of A as little as possible. It turns out that this is not a good idea if one strives to compute an approximate RRQR-factorization. The following general result is essential here.

THEOREM 4.1. *Assume we have an approximate right null-vector y of B satisfying*

$$\|By\|_2 = \delta < \|B_1 y_1\|_2, \qquad \|y\|_2 = 1, \tag{7}$$

where we have partitioned $B = (B_1, B_2)$ and $y = \binom{y_1}{y_2}$. Also, assume that the smallest singular value of B_1 satisfies

$$\sigma_{\min}(B_1) = \bar{\sigma} > \delta. \tag{8}$$

Then

$$\|y_2\|_2 \geq \frac{\bar{\sigma}}{2\|B\|_2^2} \sqrt{\|B_2\|_2^2 + (\bar{\sigma}^2 - \delta^2)}\,. \tag{9}$$

Proof. Due to the partitioning of B and y, we have $B_2 y_2 = By - B_1 y_1$, such that

$$\|B_2\|_2 \|y_2\|_2 \geq \|B_2 y_2\|_2 = \|By - B_1 y_1\|_2 \geq | \, \|By\|_2 - \|A_1 y_1\|_2 \, | \,.$$

Due to our assumptions, we thus obtain $\|B_2\|_2 \|y_2\|_2 \geq \|B_1 y_1\|_2 - \|By\|_2 \geq \bar{\sigma}\|w_1\|_2 - \delta$. To proceed, we use $\|y\|_2{}^2 = \|y_1\|_2{}^2 + \|y_2\|_2{}^2 = 1$ to obtain $\|B_2\|_2 \|y_2\|_2 \geq \bar{\sigma}\sqrt{1 - \|y_2\|_2^2} - \delta$, which is equivalent to

$$\|y_2\|_2 \geq \frac{\sqrt{\|B_2\|_2^2 \delta^2 + (\|B_2\|_2^2 + \bar{\sigma}^2)(\bar{\sigma}^2 - \delta^2)} - \|B_2\|_2 \delta}{\|B_2\|_2^2 + \bar{\sigma}^2}.$$

Since both $\|B_2\|_2$ and $\bar{\sigma}$ are less than or equal to $\|B\|_2$, we obtain the bound in (9) by replacing the denominator in the above expression by $2\|B\|_2$. □

Remember that the matrix Y_2 in Theorem 2.2 is built up from the vectors y_2 (for increasingly larger submatrices), and the certainty of having found an RRQR-factorization depends on $\|(Y_2^{(i)})^{-1}\|_2$. In particular, if $\|Y_2\|_2$ is small, then $\|(Y_2^{(i)})^{-1}\|_2$ will be large, and the lower and upper bounds $\hat{\delta}_i$ and θ_i are therefore not guaranteed to be tight. So we would like the right-hand side in (9) to be large. Applying Theorem 4.1 to $A^{(k)}$ in (5), (8) simply means that $R_{11}^{(k)}$ must be well conditioned. In the case where both $\|R_{12}^{(k)}\|_2$ and $\|A_{22}^{(n-k)}\|_2$ are small, we have $\bar{\sigma} \gg \delta$, so the second term under the root in (9) will not be too small; otherwise the first term will not be negligible. Taken together then, $\|y_2\|_2$ will not be small if $\bar{\sigma}$ is not too small. So well-conditioned leading submatrices $R_{11}^{(k)}, k = 1, \cdots, r_{ice}$ are a necessary condition for the vectors building up Y_2 not being too small.

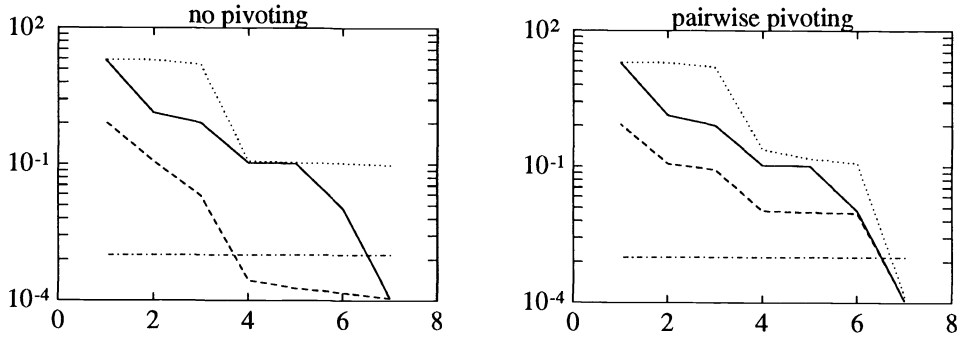


FIG. 1. *Singular value approximations after forward pass for matrix.*

To illustrate this point, consider the 7×7 matrix

$$(10)\, R = \begin{pmatrix} -0.8480 & -0.3521 & 0.3862 & 0.5893 & -1.3705 & 1.2615 & -1.2396 \\ 0 & -0.1241 & 0.1095 & 6.6984 & -5.6058 & 5.7495 & -5.7782 \\ 0 & 0 & 0.0269 & 7.4016 & -8.1668 & 8.0840 & -8.0670 \\ 0 & 0 & 0 & 0.1000 & 0.0296 & -0.0140 & 0.0097 \\ 0 & 0 & 0 & 0 & 0.1000 & 0.0140 & -0.0094 \\ 0 & 0 & 0 & 0 & 0 & 0.1000 & 0.0084 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1000 \end{pmatrix}$$

The singular value distribution of this matrix is shown as the solid line in the plots of Fig. 1. So with respect to the tolerance $\tau = 10^{-3}$ (indicated by the horizontal dash-dotted line), the rank of this matrix is six. Disallowing column exchanges and processing columns in their natural order, the forward pass "accepts" the first three columns and "rejects" the remaining four, arriving at the estimate $r_{ice} = 3$. In detail we obtain

$$R_{ice}(4:7, 4:7) = \begin{pmatrix} -0.1043 & -0.0094 & 0.0062 & -0.0284 \\ 0 & -0.1015 & -0.0062 & 0.0164 \\ 0 & 0 & -0.1009 & -0.0124 \\ 0 & 0 & 0 & 0.0936 \end{pmatrix}$$

and no trailing submatrix is small enough to reveal the rank.

The reason is that the leading 3×3 submatrix is not well conditioned (compared to the large singular values of R), and

$$\|(Y_2^{(i)})^{-1}\|_2 = O(10^3), i = 4, \cdots, 7.$$

Because of this relatively large norm of $Y_2^{(i)}$, the gap between the lower and upper bounds δ_i and θ_i of σ_i is large as is shown in the first plot in Fig. 1, where the lower bounds are indicated by dashed lines and the upper bounds are indicated by dotted lines.

If, on the other hand, we perform pairwise pivoting (i.e., we can exchange adjacent columns), then the third column of R is percolated to the back and we obtain $R_{ice}(7,7) = -0.001$, $\|(Y_2^{(7)})^{-1}\|_2 = 1.24$ and the upper and lower bounds shown in

the second plot in Fig. 1, which are tight for the small singular values. This is why we employ a restricted column pivoting strategy. It allows us, figuratively speaking, to push almost dependent columns ahead of us and keep $R_{11}^{(i)}$ well conditioned.

This argument also suggests a possible weakness in the algorithm suggested in [1], which uses a one-norm condition estimator and processes the columns of A in their natural order. As shown above, this approach may significantly underestimate the rank of A if a leading submatrix of A is not well conditioned.

5. Completing the RRQR-factorization. After completing the forward pass we have computed a QR-factorization

$$AP_{ice} = Q_{ice}R_{ice}. \tag{11}$$

If we partition R_{ice} as in (2), we know that for $i = r_{ice}, \cdots, n$,

$$\hat{\sigma}_{\min}(R_{11}^{(r_{ice})}) > \tau$$

and for any column y in $\begin{pmatrix} R_{12}^{(r_{ice})} \\ R_{22}^{(r_{ice})} \end{pmatrix}$ we have

$$\hat{\sigma}_{\min}\left(\left(\begin{pmatrix} R_{11}^{(r_{ice})} \\ 0 \end{pmatrix}, y\right)\right) \leq \tau.$$

This suggests that (11) is a good approximation to an RRQR-factorization.

However, the forward pass is *not guaranteed* to capture the numerical rank of A, partly because the restricted pivoting scheme may be too restricted to produce a well-conditioned leading triangular submatrix of R, and partly because ICE may overestimate a singular value. So to validate our initial factorization and to "fix it up" if needed, we apply the Chan/Foster RRQR algorithm to R_{ice}. This can be done cheaply, by taking advantage of the information accumulated in the forward pass. Since we have already computed (and saved in W) the left null-vectors $x^{(k)}$ of $R_{ice}(1 : k, 1 : k)$, $k = r_{ice} + 1, \cdots, n$ during the initial QR factorization, we can obtain the corresponding right null-vectors $y^{(k)}$ cheaply via a back substitution with $R_{ice}(1 : r_{ice} + i, 1 : r_{ice} + i)$. The experiments in [6] show that even if ICE should overestimate the smallest singular value, one backsolve is sufficient to obtain estimates that are correct to one digit at least.

Counting down from $i = n$, we can then accept column i of R_{ice} as being in the right place for an RRQR-factorization (without doing *any* additional work beyond the backsolve) as long as the element in the last position of $y^{(i)}$ is not too small.

As soon as $|\ (y^{(i)})_i\ |$ is not large enough, more work has to be done, since otherwise $\|(Y_2^{(i)})^{-1}\|_2$ may be large. So in accordance with the Chan/Foster scheme, we permute the columns of $R_{ice}(1 : i, 1 : i)$ according to the location of the largest entry (in modulus) in $y^{(i)}$ and annihilate the subdiagonal using Givens rotations. After the first such step, the information in the remaining leading columns of W is lost, since R changed due to the column exchanges and Givens rotations. Therefore, from now on we must recompute the right null-vector of every current $R_{ice}(1 : r_{ice} + i, 1 : r_{ice} + i)$. To do this, we use the two-norm condition estimator suggested by Cline, Conn, and Van Loan [13] to compute a left null-vector of $R_{ice}(1 : r_{ice} + i, 1 : r_{ice} + i)$ and then transform it into a right null-vector with a back substitution as before. This strategy involves fewer memory accesses and less computational effort than using the classical

Linpack condition estimator. We would also like to mention that one-norm condition estimators (see, for example, [32]), although computationally efficient, should not be used in this setting because they produce a one-norm null-vector which is likely to have zeros in its last positions.

In deciding whether the last entry of $y^{(i)}$ is large enough, we use the threshold strategy (2B). It has the advantage that we delay the point when we have to do more work in addition to the backsolve with $x^{(i)}$, and even thereafter, it is still advantageous for the pivot column to be as far back as possible in the matrix $R_{ice}(1 : r_{ice} + i, 1 : r_{ice}+i)$, preferably within the last i columns. This reduces both the fill and the number of computations involved in annihilating the bidiagonal. In addition, a pivot column with index greater than r_{ice} allows us to restart ICE to produce a new approximate left null-vector more cheaply than with the Cline, Conn, and Van Loan scheme (we did not implement this, though).

The whole process continues until we compute a lower bound δ_i which is greater than the threshold τ, at which point we stop with the dimension of $R_{ice}(1 : r_{ice}+i, 1 : r_{ice} + i)$ equal to the estimate r_{rrqr_ice} of the numerical rank.

It is worth mentioning that the matrix $Y_2 = R_{22}^{-1}W_2$ has to be well conditioned if we want to have a good chance of approximating the RRQR-factorization. For example, when $\mid r_{r_{ice}+1,r_{ice}+1} \mid$ is almost zero, the first column of Y will be almost the canonical unit vector $e_{r_{ice}+1}$, as will be all other columns of Y, since the smallest singular value of $R_{ice}(1 : i, 1 : i)$ will not drop appreciably more for $i = r_{ice} + 1, \cdots, n$. This is the reason why we employ the column pivoting strategy (1A) when completing the QR-factorization of the "rejected" columns. In this fashion the columns are likely to be reordered such that the smallest singular value of $R_{ice}(1 : r_{ice} + i, 1 : r_{ice} + i)$, $i = 1, \cdots, n-r_{ice}$ drops gradually towards $\sigma_{\min}(A)$. As a result, the diagonal entries of Y_2 should not be too small since they will capture the direction of the new null-vector.

An important question in relation to the threshold criterion (2B) used to determine pivot columns in this backward pass is whether we are still guaranteed that $\|(Y_2^{(i)})^{-1}\|_2$ is not large, where $Y_2^{(i)}$ denotes the lower right $(n - i + 1) \times (n - i + 1)$ submatrix of Y. The following theorem is useful in this context.

THEOREM 5.1. *If Y is upper trapezoidal with columns of unit norm, and if the diagonal elements $(Y_2)_{ii}$ of Y_2 satisfy*

$$\mid (Y_2)_{ii} \mid \geq \xi, \tag{12}$$

where $0 < \xi \leq 1$, then

$$\|(Y_2^{(i)})^{-1}\|_2 < (2/\xi)^{(n-i)}. \tag{13}$$

Proof. For convenience, define $Z = Y_2^{(i)}$. Then $\mid z_{jj} \mid \geq \xi$ and $\mid z_{jk}/z_{jj} \mid \leq \xi^{-1}$. Now, if $y = Zx$, then the elements of x are given by

$$x_j = z_{jj}^{-1}\left(y_j - \sum_{k=j+1}^{n-i} z_{jk}x_k\right).$$

If we assume that $\|y\|_2 = 1$, then $\mid y_j \mid \leq 1$ and we get

$$\mid x_j \mid \leq \xi^{-1} + \sum_{k=j+1}^{n-i} \xi^{-1} \mid x_k \mid .$$

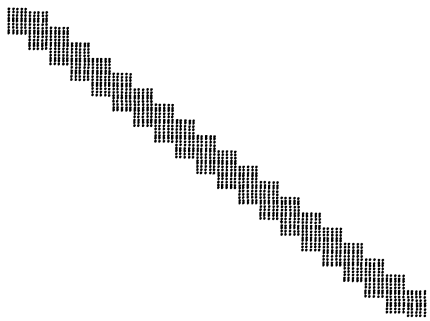


FIG. 2. 100×100 *matrix with staircase structure.*

It is easy to show by induction that $| x_j | \leq (2/\xi)^{(n-i-j)}$ and that $\|x\|_2 < (2/\xi)^{(n-i)}$. From the definition of the matrix two-norm it then follows that $\|(Y_2^{(i)})^{-1}\|_2 = \|Z^{-1}\|_2 = \max_x \|x\|_2 < (2/\xi)^{(n-i)}$. □

Due to the rescaling of the columns of Y and the threshold ρ_{rrqr}, both requirements in Theorem 5.1 are satisfied for $\xi = 1/\sqrt{n}\rho_{rrqr}$, and we can therefore conclude that $\|(Y_2^{(i)})^{-1}\|_2$ is not large. Hence, the lower and upper bounds in Theorem 2.1 are tight, and moreover the columns of Y are linearly independent and form a good numerical basis for the right null-space of R.

We would also like to stress that we have to compute approximate right null-vectors $y^{(i)}$ even for those columns whose associated upper bound $\theta_i = \|R(r_{ice} + i : n, r_{ice} + i : n)\|_2$ is less than τ. Even though in this case the ith column of R_{ice} is guaranteed to be linearly dependent, it is not guaranteed that the corresponding $\|(Y_2^{(i)})^{-1}\|_2$ is not large, which is a necessary condition for the QR-factorization to reveal *all* dependent columns. This suggests that there is not much sense in actually computing the upper bounds θ_i.

In summary, we use the backward pass to make sure that we compute an RRQR-factorization as defined in §2. We apply the Chan/Foster algorithm, using as much information as possible from the forward pass. Since the forward pass already computed an approximation to an RRQR-factorization, we can in all likelihood skip orthogonal transformations for many of the RRQR steps, and only do a backsolve to confirm that a column of A has been permuted to the right place. Even if we have to change the column ordering, it will in all likelihood involve nearby columns towards the end of the matrix, and so few Givens rotations should be required. As a result, we expect the preordering done in the initial QR-factorization to lead to a very inexpensive backward pass. We would like to stress, however, that we are not compromising numerical reliability, since even if the forward pass should fail completely to approximate an RRQR-factorization, the backward pass will recover from it.

6. Numerical experiments. In this section we illustrate the performance of our algorithm on a model problem. Our test matrices were square matrices of order 100 and with a staircase structure as shown in Fig. 2. The size s_{window} of the pivot window was fixed at five. Using PRO-MATLAB, we generated matrices of rank $r = 80$

and rank $r = 95$ with a geometric distribution of singular values between $\sigma_1 = 1$ and $\sigma_r = 10^{-2}$, and also between $\sigma_{r+1} = 10^{-5}$ and $\sigma_n = 10^{-7}$. There is a well-defined gap between "large" and "small" singular values and our threshold $\tau = 5 \cdot 10^{-4}$ is chosen to lie well within this gap. We generated five random matrices of rank 80 and 95, respectively. Since the orthogonal transformations we apply in obtaining a banded structure usually grade a matrix in that numerically small elements tend to appear in the lower right-hand corner, we generated five more test matrices by "flipping" those matrices, i.e., we applied the permutation mapping $[1, 2, \cdots, n]$ into $[n, n-1, \cdots, 1]$ to rows and columns. As a result, if A has a small trailing submatrix, the flipped version of A has a small *leading* submatrix. The latter case is much harder, since the small submatrix has to be transferred to the lower right-hand side during the factorization process in order to reveal the rank.

We compared the following three methods: RRQR_ICE, which is our model algorithm as described above, RRQR_CF, which is an implementation of the Chan/Foster algorithm using an initial QR-factorization with no column exchanges, and QR_PIV, which is an implementation of threshold pivoting. We used a threshold $\rho_{rrqr} = 5$ in RRQR_ICE as well as RRQR_CF, and a threshold $\rho_{colpiv} = 5$ in QR_PIV. To obtain right null-vectors in RRQR_CF, we used the condition estimator suggested by Cline, Conn, and Van Loan [13] followed by a backsolve. This way the generation of an approximate null-vector requires only two "touches" of R, whereas the use of the LINPACK condition estimator required four touches on average [10]. These modifications also allow a fair comparison of the merits of our proposed algorithm compared to the Chan/Foster RRQR scheme applied to a QR-factorization where no column exchanges were performed.

As expected, RRQR_ICE and RRQR_CF identify the rank correctly in all cases, while QR_PIV overestimated the rank by 1 for one matrix of rank 95. Despite the very restricted pivot window, the forward pass of RRQR_ICE identified the rank correctly in most cases; only for one matrix of rank 80 and one of rank 95 it underestimated the rank by one. Figure 3 offers a closer look at the behavior of the different algorithms on the one matrix of rank 80 where the forward pass of RRQR_ICE underestimated the rank by one. The solid line indicates the singular values σ_i of A, the dotted line indicates $\delta_i = \sigma_{\min}(R(1:i, 1:i))$, and the stars are our computed ICE-estimates $\hat{\delta}_i$ of δ_i. The threshold τ is indicated by the horizontal dashed line. The first plot, labelled "QR_ICE," shows the situation after the initial QR-factorization with restricted pivoting. We see that σ_i and δ_i are already close together except for σ_{81} and δ_{81} and that $\hat{\delta}_i$ returned by incremental condition estimation is a very good estimate of δ_i.

This is borne out by the plot for RRQR_ICE, which shows the situation after the backward pass. In this case we immediately accepted the last 17 columns of R_{ice}, i.e., no work was required beyond a back substitution per column. Then we reverted to the Chan/Foster RRQR scheme, thereby adjusting δ_{80} closer to σ_{80} and correcting our rank estimate. The resulting picture is almost identical to the one obtained by RRQR_CF, the traditional and more expensive Chan/Foster-scheme. The picture for QR_PIV gives us an idea why this strategy is prone to overestimating the rank: Its underlying estimates $\hat{\delta}_i \equiv | r_{ii} |$ can overestimate δ_i by a large margin.

In Fig. 4 we have a closer look at the structure of the triangular factor for one of the matrices with rank 80. The matrix A underlying this figure was the "flipped" version as explained above and therefore had a small *leading* submatrix. The task here is hard, since the small leading entries have to be transferred to the end of the matrix to obtain an RRQR-factorization. As can be seen, threshold pivoting in QR_PIV

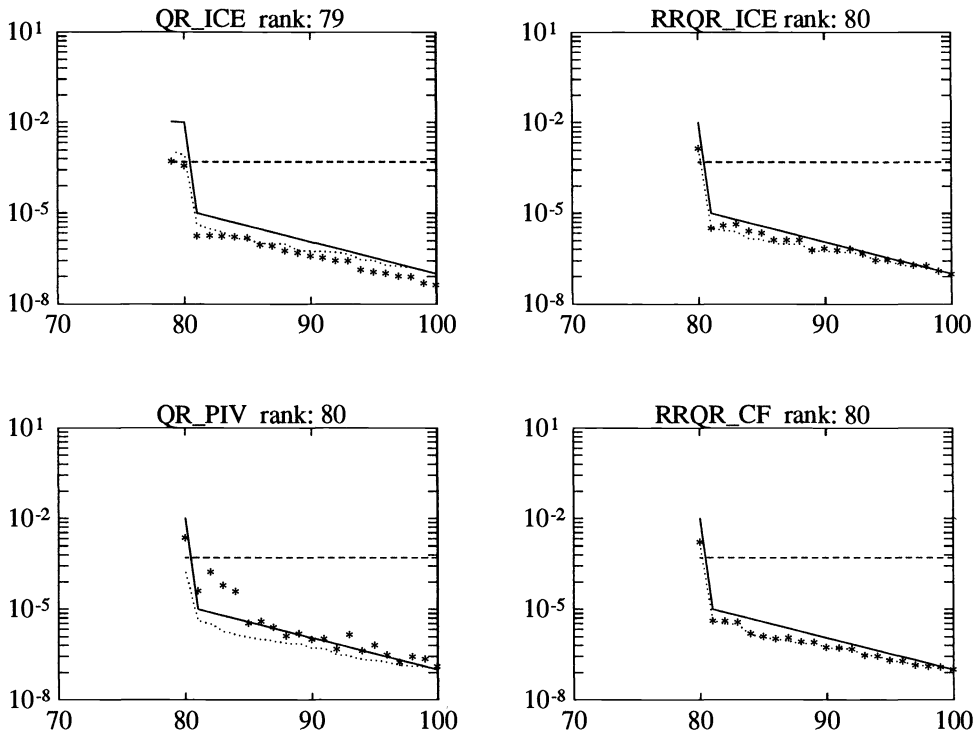


FIG. 3. *Approximations of the small singular values of A.*

pretty much destroys the structure of the matrix as dependent columns percolate to the back of the matrix. RRQR_ICE preserves the structure in the leading part of the triangular matrix and so does RRQR_CF. The situation is very much the same for a matrix of rank 95. Note that the number of "spikes" in the back of the R generated by RRQR_ICE and RRQR_CF is more or less the number of small singular values, whereas no such structure is apparent for QR_PIV.

The number of nonzeros in R for all the tests is shown in Fig. 5 with the dashed line indicating the 1225 nonzeros that would have resulted from a QR-factorization without any column exchanges. The 10 bars for each method represent, from left to right, the first test matrix, the first test matrix "flipped," the second test matrix, the second test matrix "flipped," etc. As expected from the previous example, all strategies generate little fill when the matrix is already close to rank-revealing structure in that it has a small trailing submatrix. On the other hand, when a small leading submatrix has to be transferred to the right-hand side of the matrix, RRQR_ICE and RRQR_CF generate fill that is more or less proportional to the number of small singular values, whereas QR_PIV generates large amounts of fill regardless. We also see that RRQR_ICE generates slightly more fill than RRQR_CF.

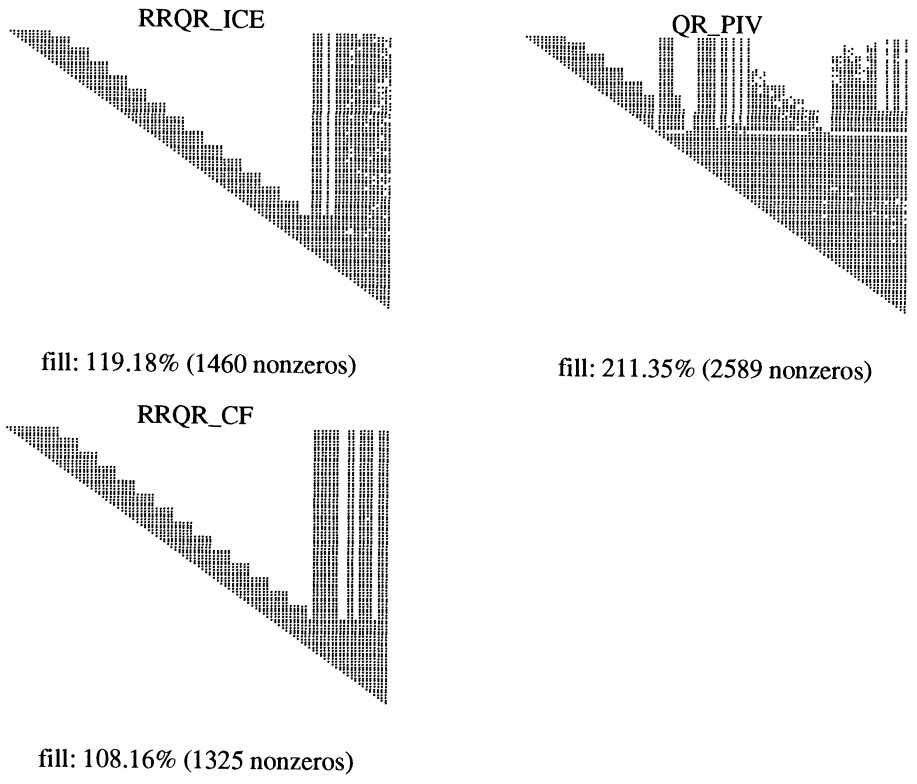


FIG. 4. *Structure of triangular factor for a matrix of rank 80 with a small* leading *submatrix.*

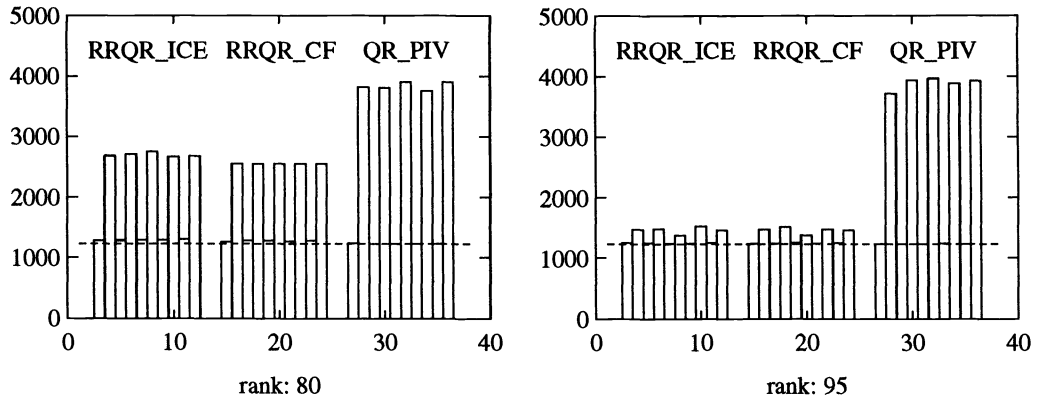


FIG. 5. *Number of nonzeros in R.*

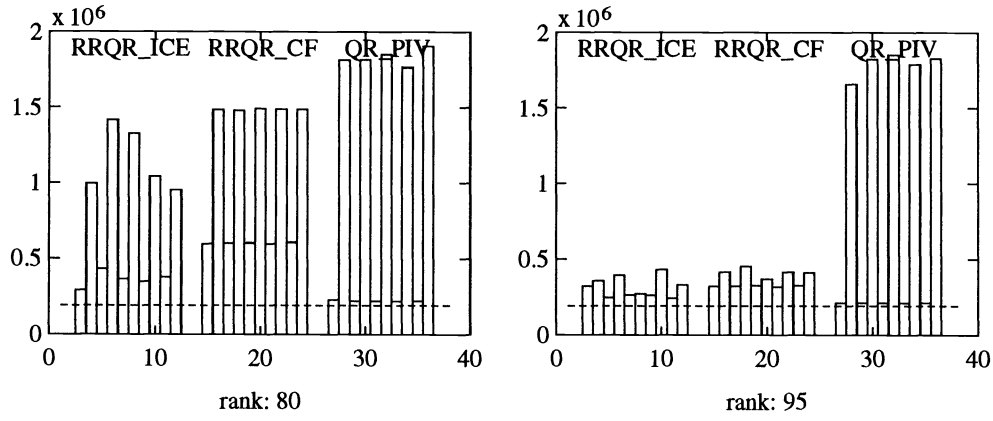


FIG. 6. *Number of floating point operations required.*

Flop counts for the different strategies are shown in Fig. 6. These flop counts comprise the cost of forming R and Q and of computing an approximate null-space matrix Y. Only operations on nonzero elements were counted. These numbers show that RRQR_ICE is a good compromise between column pivoting and the Chan/Foster RRQR algorithm. In particular for the matrices of rank 80, RRQR_ICE requires fewer flops than RRQR_CF and for the easy "random" matrices it requires not too much more work than computing a QR-factorization without pivoting. For the rank 95 matrices, the differences are not so drastic, but RRQR_ICE is in general still cheaper than RRQR_CF. These differences would have been even larger, in favor of RRQR_ICE, if we had used the RRQR algorithm suggested in [10], which uses no threshold and employs the Linpack condition estimator. Again, the QR-factorization with threshold pivoting can be very expensive. The savings in RRQR_ICE are due mainly to the fact that for the rank 80 matrices, RRQR_ICE accepted on average 14.1 of the 20 rejected columns. For the rank 95 matrices, RRQR_ICE accepted on average 3.4 of the 5 rejected columns as being in the right place. As a result, RRQR_ICE performs fewer triangular solves, a significant advantage on parallel machines, where triangular solves are rather expensive.

At this point it seems as if there would be no dramatic difference between RRQR_CF and RRQR_ICE; both compute an RRQR-factorization reliably, RRQR_CF experiences a little bit less fill, but RRQR_ICE can require significantly fewer floating point operations. What is substantially different between those two methods is *how* these results were obtained. Table 1 shows how many nonzeros are required for Householder transformations (which are used in the initial QR-factorization) and Givens rotations (which are used in the Chan/Foster backward pass), respectively, assuming that a Givens rotation is stored in one floating-point word [27, p. 46]. The column "random" refers to the random matrices we generated and the column "flipped" refers to their flipped counterparts. While the total number of orthogonal transformations required by RRQR_ICE and RRQR_CF is more or less equal, RRQR_ICE computes the orthogonal transformation almost entirely via Householder transformations in the forward pass, whereas RRQR_CF relies mainly on Givens rotations during the post-processing. In many situations, Householder rotations are preferable over Givens rotations: Givens rotations require twice as many floating point operations and since

TABLE 1
Average storage requirements for Q.

		Rank 80		Rank 95	
		Random	Flipped	Random	Flipped
RRQR_ICE	Householder	717	2168	684	771
	Givens	11	23	2	69
RRQR_CF	Householder	680	680	680	680
	Givens	25	1401	12	164

TABLE 2
Number of average touches of R.

	Rank 80		Rank 95	
	Random	Flipped	Random	Flipped
RRQR_ICE	27.4	30.6	8.6	10.5
RRQR_CF	42.9	57.6	12.0	13.7

they are vector–vector operations versus the matrix–vector nature of Householder transformations, their memory access cost is usually higher. In our setting we have the additional shortcoming that the application of Givens rotations to eliminate a lower band is essentially sequential, allowing at most for fine-grained pipelining to exploit parallelism. The Householder transformations generally can exploit parallelism more easily due to their coarser granularity.

Table 2 shows how many times we had to touch R in the backward pass. The Cline/Conn/Van Loan condition estimator and a back substitution count as one touch each. The touch cost for the elimination of a lower diagonal due to column exchanges was defined as the fraction of the nonzeros of R that were read during this update. It can be seen that the memory-access cost of the backward pass of RRQR_ICE is substantially lower than that of RRQR_CF, particularly for matrices of rank 80. This is important since good implementations will store sparse matrices in some compressed form [15], [20], thereby making memory accesses expensive. In addition, many computing environments employ hierarchical memory schemes, and so operations where data in fast memories are not reused (as is the case in a series of backsolves, for example), will not exploit the computational potential of the machine. These issues are discussed in more detail in [3].

We also mention that all methods gave good approximations to the numerical null-space. After the forward pass, the subspace angle (see for example [27, p. 429]) between the approximate left null-vectors W and the space spanned by the vectors of U corresponding to small singular values was $O(10^{-3})$. The angle between the right numerical null-space and the approximate null-vectors Y returned by RRQR_ICE and RRQR_CF was $O(10^{-6})$, and after two steps of inverse subspace iterations the angle was reduced to the order of the machine precision. Both methods computed well-conditioned bases for the left and right null-spaces, respectively, in that the condition number of W and Y was less than 500 in all cases.

In summary, these experiments suggest that RRQR_ICE and RRQR_CF are both superior to QR_PIV, mainly because QR_PIV can miss the right rank and can produce unpredictable amounts of fill. For RRQR_ICE and RRQR_CF, the amount of fill generated is roughly proportional to the number of small singular values of A. RRQR_ICE generates slightly more fill than RRQR_CF but requires fewer floating

point operations and memory references. On the other hand, RRQR_ICE does most of its work during the initial QR-factorization, when, for example, Householder or block Householder transformations [7], [37] could be employed, and requires substantially less work in the backward pass. In particular, it avoids for the most part the essentially sequential application of Givens rotations.

This seems computationally more advantageous for common high-performance architectures, where hierarchical memories are likely to "reward" RRQR_ICE for requiring fewer memory accesses. In addition, RRQR_ICE avoids for the most part the sequential application of Givens rotations in the backward pass, which could lead to a serious loss of efficiency on parallel architectures.

7. Conclusions. We presented a new RRQR-factorization algorithm, which we believe has the right characteristics to allow for an efficient implementation for sparse matrices in high-performance computing environments. By using a restricted column-pivoting strategy guarded by incremental condition estimation during the initial QR-factorization, we try to approximate an RRQR structure as much as possible, while at the same time preserving the structure of the matrix. We also showed why some column exchanges are desirable for the forward pass to give good approximations to the desired RRQR-factorization. This effort is rewarded in the subsequent Chan/Foster backward pass which validates, and, if necessary, improves, the RRQR structure of the initial QR-factorization. Due to the preordering achieved in the forward pass, generally very little remains to be done in the backward pass. This is computationally advantageous, since in the forward pass it is possible to use Householder matrices, and we have the possibility of applying block orthogonal schemes to dense subblocks. Compared to Givens rotations, of which we have to do quite a few if no preordering is computed in the forward pass, Householder transformations require fewer memory accesses and due to their coarser-grained nature, are advantageous in most parallel computing environments. The cost of memory access for the new algorithm is further reduced by the fact that many fewer passes over the matrix are required in the backward pass. Again, this is advantageous in most high-performance computing environments, where memory accesses, in particular to a sparse matrix stored in some compressed scheme, are rather expensive compared to floating-point operations. Numerical experiments using PRO-MATLAB on a model problem illustrated these points. They also show that the new algorithm is as reliable as the original Chan/Foster scheme, and that the QR-factorization with threshold pivoting does not seem to be well suited for sparse matrices since it can generate unpredictable amounts of fill and lead to an overestimate of the numerical rank.

Acknowledgments. The authors would like to thank the anonymous referees, whose insightful comments helped us to improve the focus and presentation of this paper.

REFERENCES

[1] J. L. BARLOW, *The accurate solution of sparse weighted and equality constrained least-squares problems using a static data structure*, Tech. Report CS–89–03, Department of Computer Science, Pennsylvania State University, University Park, PA, January 1989.

[2] ———, *A block QR factorization algorithm using restricted pivoting*, in Proceedings SUPERCOMPUTING '89, ACM Press, Baltimore, MD, 1989, pp. 248–256.

[3] ———, *Fundamental linear algebra computations on high-performance computers*, Vol. 250, Informatik Fachberichte, Springer-Verlag, Berlin, 1990, pp. 167–182.

[4] ———, *Incremental condition estimation*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 312–322.

[5] C. H. BISCHOF, *A parallel* QR *factorization algorithm with controlled local pivoting*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 36–57.

[6] C. H. BISCHOF AND P. T. P. TANG, *A robust incremental condition estimation scheme*, Tech. Report MCS-P225, 0391, Mathematics and Computer Sciences Division, Argonne National Laboratory, Argonne, IL, 1991.

[7] C. H. BISCHOF AND C. F. VAN LOAN, *The WY representation for products of Householder matrices*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s2–s13.

[8] Å. BJÖRCK, *A direct method for sparse least squares problems with lower and upper bounds*, Numer. Math., 54 (1988) pp. 19–32.

[9] ———, *Difference Methods—Solutions of Equations in* R^n, Vol. I, *Handbook of Numerical Analysis*, Elsevier Publishers, Amsterdam, 1990.

[10] T. F. CHAN, *Rank revealing* QR *factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.

[11] T. F. CHAN AND P. C. HANSEN, *Computing truncated* SVD *least squares solutions by rank revealing* QR *factorizations*, SIAM J. Sci. Statist. Comput., 11 (1990) pp. 519–530.

[12] ———, *Some applications of the rank revealing* QR *factorization*, Tech. Report 90-09, Department of Mathematics, University of California at Los Angeles, Los Angeles, CA, 1990.

[13] A. K. CLINE, A. R. CONN, AND C. F. VAN LOAN, *Generalizing the* LINPACK *Condition Estimator*, Vol. 909, Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1982, pp. 73–83.

[14] F. R. DE HOOG AND M. F. HUTCHINSON, *An efficient method for calculating smoothing splines using orthogonal transformations*, Numer. Math., 50 (1987), pp. 311–320.

[15] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Oxford Press, London, 1987.

[16] I. DUFF AND J. REID, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[17] L. ELDÉN AND R. SCHREIBER, *An application of systolic arrays to linear discrete ill-posed problems*, SIAM J. Sci. Statist. Comput., 7 (1986) pp. 892–903.

[18] L. V. FOSTER, *Rank and null space calculations using matrix decomposition without column interchanges*, Linear Algebra Appl., 74 (1986), pp. 47–71.

[19] ———, *The probability of large diagonal elements in the* QR *factorization*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 531–544.

[20] A. GEORGE AND J. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[21] A. GEORGE, J. LIU, AND E. NG, *A data structure for sparse* QR *and* LU *factorization*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 100–121.

[22] A. GEORGE AND E. NG, *Orthogonal reduction of sparse matrices to upper triangular form using Householder transformations*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 460–472.

[23] J. G. LEWIS, D. J. PIERCE, AND D. C. WAH, *A multifrontal Householder* QR *factorization*, Tech. Report ECA–TR–127, Boeing Computer Services, Engineering and Scientific Services Division, Seattle, WA, 1989.

[24] G. H. GOLUB, V. KLEMA, AND G. W. STEWART, *Rank degeneracy and least squares problems*, Tech. Report TR–456, Dept. of Computer Science, University of Maryland, College Park, MD, 1976.

[25] G. H. GOLUB, P. MANNEBACK, AND P. L. TOINT, *A comparison between some direct and iterative methods for certain large scale geodetic least-squares problem*, SIAM J. Sci. Statist. Comput., 7 (1986) pp. 799–816.

[26] G. H. GOLUB, *Numerical methods for solving linear least squares problems*, Numer. Math., 7 (1965), pp. 206–216.

[27] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[28] T. A. GRANDINE, *An iterative method for computing multivariate* C^1 *piecewise polynomial interpolants*, Comput. Aided Geom. Design, 4 (1987), pp. 307–319.

[29] ———, *Rank deficient interpolation and optimal design: An example*, Tech. Report SCA–TR–113, Boeing Computer Services, Engineering and Scientific Services Division, Seattle, WA, February 1989.

[30] M. T. HEATH, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 223–237.

[31] ———, *Numerical methods for large sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 497–513.

[32] N. J. HIGHAM, FORTRAN *codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation*, ACM Trans. Math. Software, 14 (1988), pp. 381–396.

[33] Y. P. HONG AND C.-T. PAN, *The rank revealing* QR *decomposition and* SVD, Tech. Report

MCS-P188-1090, Argonne National Laboratory, Mathematics and Computer Sciences Division, 1990.

[34] J. W. H. LIU, *A tree model for sparse symmetric indefinite matrix factorization*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 26–39.

[35] J. MORÉ, *The Levenberg–Marquardt algorithm: Implementation and theory*, in Proceedings of the Dundee Conference on Numerical Analysis, G. A. Watson, ed., Springer-Verlag, Berlin, 1978.

[36] E. NG, *A scheme for handling rank-deficiency in the solution of sparse linear least squares problems*, Tech. Report ORNL/TM-10980, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, Oak Ridge, TN, November 1988; SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1173–1183.

[37] R. SCHREIBER AND C. VAN LOAN, *A storage efficient* WY *representation for products of Householder transformations*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 53–57.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1351–1372, November 1991
© 1991 Society for Industrial and Applied Mathematics
008

THE DYNAMICS OF THE THETA METHOD*

A. M. STUART† AND A. T. PEPLOW†‡

Abstract. The dynamics of the theta method for arbitrary systems of nonlinear ordinary differential equations are analysed. Two scalar examples are presented to demonstrate the importance of spurious solutions in determining the dynamics of discretisations. A general system of differential equations is then considered. It is shown that the choice $\theta = \frac{1}{2}$ does not generate spurious solutions of period 2 in the timestep n. Using bifurcation theory, it is shown that for $\theta \neq \frac{1}{2}$ the theta method does generate spurious solutions of period 2. The existence and form of spurious solutions are examined in the limit $\Delta t \to 0$. The existence of spurious steady solutions in a predictor-corrector method is proved to be equivalent to the existence of spurious period 2 solutions in the Euler method. The theory is applied to several examples from nonlinear parabolic equations. Numerical continuation is used to trace out the spurious solutions as Δt is varied. Timestepping experiments are presented to demonstrate the effect of the spurious solutions on the dynamics and some complementary theoretical results are proved. In particular, the linear stability restriction $\Delta t/\Delta x^2 \leq \frac{1}{2}$ for the Euler method applied to the heat equation is generalised to cope with a nonlinear problem. This naturally introduces a restriction on Δt in terms of the initial data; this restriction is necessary to avoid the effect of spurious periodic solutions.

Key words. dynamics of numerical methods, asymptotic instabilities, dissipative problems, timestep as bifurcation parameter

AMS(MOS) subject classifications. 65M10, 35A40, 35K57

1. Introduction.

In this paper we study the relationship between the asymptotic behaviour of a numerical simulation and that of the true solution itself, *for fixed values of the mesh-spacing.* Our viewpoint here is to consider the numerical method as a dynamical system and to analyse its possible asymptotic states. Our aim is to determine methods for which spurious asymptotic states are not introduced by the discretisation, either for all values of the timestep, or for all values of the timestep sufficiently small.

We study one-step methods for systems of nonlinear ordinary differential equations (ODEs). Such systems may be of interest in their own right or may arise from spatial discretisations of partial differential equations (PDEs). Many of the features we describe are qualitatively robust to spatial mesh refinement and this justifies studying the problem in isolation from the effect of spatial derivatives—essentially a method of lines approach. In fact, many of the results in this paper can be generalised from an ODE in $\Re^n$ to an ODE in a Banach space.

We study the following ODE: find $u(t) \in \Re^m$ satisfying

$$u_t = G(u), \qquad u(0) = u_0. \tag{1.1}$$

The regularity of $G(u)$ will be specified when required. The theta method consists of finding a sequence $U_n \in \Re^m$ satisfying

$$U_{n+1} - U_n = \Delta t[(1-\theta)G(U_n) + \theta G(U_{n+1})], \tag{1.2}$$

for $\theta \in [0,1]$.

* Received by the editors December 6, 1989; accepted for publication (in revised form) October 3, 1990.

† School of Mathematical Sciences, University of Bath, Bath BA2 7AY, United Kingdom.

‡ Present address, Department of Civil Engineering, University of Bradford, Bradford BD7 1DP, United Kingdom. The work of this author was supported by the Science and Engineering Research Council, United Kingdom.

We also consider the following splitting of G :

$$G(u) = Au + H(u), \tag{1.3}$$

where both A and H map $\Re^m$ into itself. A is assumed to be a matrix, whilst H is a nonlinear function. This splitting enables us to consider the semi-implicit method which is popular in the solution of many dissipative PDEs: find $U_n \in \Re^m$ satisfying

$$U_{n+1} - U_n = \Delta t[AU_{n+1} + H(U_n)]. \tag{1.4}$$

Note that U_n approximates $u(n\Delta t)$. Under fairly mild assumptions, both the schemes (1.2) and (1.4) yield convergent solutions on compact time intervals. The main result of this paper is proved in Lemmas 3.1, 3.2, and 3.3. It is given in the following theorem.

MAIN THEOREM. *If $\theta = \frac{1}{2}$, then the numerical method* (1.2) *cannot have period* 2 *solutions in n. The method* (1.4) *and the method* (1.2) *with $\theta \neq \frac{1}{2}$ can have period* 2 *solutions in n.*

Such period 2 solutions are spurious and generated by discretisation. They are important because their presence can cause the dynamics of the discretisation to differ substantially from the dynamics of the underlying differential equation. This happens in the following two different ways:

(i) If the period 2 solutions are unstable (that is, repel data in their vicinity), then such solutions and their stable manifolds often delineate classes of initial data for which the correct asymptotic behaviour is reproduced by the numerical method and classes of initial data which lead to blow-up of the scheme or nonexistence of numerical solutions. (See Example 2.1 and Theorem 4.2.)

(ii) If the period 2 solutions are stable (that is, attract data in their vicinity), then U_n may converge to a period 2 solution as $n \to \infty$. All initial data lying within the domain of attraction of this period 2 solution will then yield spurious results. (See Example 2.2.)

In §2 we describe two simple examples which illustrate the importance of spurious periodic solutions in determining the dynamics of discretisations. These examples indicate why the nonexistence of period 2 solutions for the trapezoidal rule gives it an advantage over other methods. Section 3 contains the proof of the main theorem in the general case. In addition, we state and prove a theorem about the behaviour of spurious solutions as $\Delta t \to 0$. Specifically we prove that for C^1 nonlinearities $G(u)$ spurious solutions approach infinity in norm as $\Delta t \to 0$ (if they exist for arbitrarily small Δt). This is essentially a consequence of the convergence of the schemes. We also show that the existence of period 2 solutions for the Euler method is equivalent to the existence of spurious steady solutions for a predictor-corrector method and hence the theory of spurious periodic solutions has far-reaching consequences. The results from §3 are illustrated further in §4 by means of several examples arising from discretisations of parabolic PDEs; numerical continuation is used to trace out the spurious solutions as Δt is varied, and timestepping experiments are presented which illustrate the effect of the spurious solutions. Complementary analysis of the specific problems is also presented. In particular, Theorem 4.2 provides a generalisation of the classical stability restriction for the numerical solution of the linear heat equation to a nonlinear problem. As would be expected, this nonlinear restriction involves dependence upon initial data, reflecting the existence of spurious solutions. Section 5 contains the conclusions.

The first reference to the relevance of period 2 solutions in generating spurious dynamics is by Newell [13]. This important work has not received the attention it deserves in the numerical analysis community, mainly because it is couched in the language of fluid mechanics. Many of the ideas in this and other papers are present in embryonic form in [13]; in particular, Newell recognized the importance of the linear stability limit in predicting the existence of spurious period 2 solutions. Newell's work on dissipative problems dealt mainly with a cubic viscous Burgers' equation and touched on the issue of implicit methods. Subsequently, further studies of period 2 solutions were undertaken by Mitchell and his co-workers ([6], [10], [11], [17], [19]), and the role of the linear stability limit in predicting their existence was firmly established. Mitchell's work has concentrated on explicit methods with quadratic nonlinearities (in particular, Fisher's equation) and the existence of stable period 2 solutions for Δt above the linear stability limit. Following the work of Newell and of Mitchell, a general result establishing the existence of spurious periodic solutions in the neighbourhood of the linear stability limit was proved for explicit discretisations of arbitrary reaction-diffusion-convection equations [22]. That work showed that spurious solutions can exist below the linear stability limit and in some circumstances for Δt arbitrarily small. This paper extends the work in [22] to more general numerical methods and to more general differential equations; we also include new results about the qualitative behaviour of the spurious solutions and several numerical examples which illustrate the application of the theory to PDEs. A unified treatment of spurious steady and periodic solutions in Runge–Kutta methods, linear multistep methods, and predictor-corrector methods is contained in [14]. The main result in this paper disproves a conjecture made by one of the present authors in [21], namely, that the Crank–Nicolson method for nonlinear parabolic equations can have period 2 solutions which bifurcate from $\Delta t = \infty$.

In general, dynamically varying timesteps should be used in time-dependent simulations of nonlinear problems, whenever feasible. Here we are concerned entirely with fixed timestepping strategies. There are two main reasons why we believe that the study of fixed timestepping strategies is important. First, time-like iterations are often used to solve steady problems (or to find other asymptotic states) and in such a context it is the asymptotic properties of the scheme that are of most importance; transient behaviour is irrelevant. Hence fixed timesteps are frequently used, the aim being to choose a timestepping algorithm which maximises the domain of attraction of the target solution; an important step in this direction is to avoid the existence of spurious solutions. Second, it is our belief that a thorough understanding of the problems associated with the fixed timestep strategy is necessary in order to develop simple variable-step strategies for large-scale PDEs; it seems likely that the emphasis in designing timestepping strategies will shift from schemes designed purely to control local error to schemes designed to reproduce the long-time dynamics of differential equations as closely as possible. Clearly these two approaches are intimately related, but further work is required on the subject. We claim that, for both fixed and variable timestepping strategies, bifurcation theory is an excellent tool for studying the dynamics of discretisations since it enables the study of changes in the topology of a dynamical system as a parameter (some measure of the timestep) is varied.

An alternative, and very powerful, algebraic approach to the study of spurious solutions is introduced in [8], where spurious steady solutions are considered. This approach can be generalised to cope with period 2 solutions; see [14]. The dynamics of variable timestepping algorithms are analysed in [5]

2. Motivation for the methods and results. In this section we present two examples which illustrate the effect of spurious period 2 solutions on the dynamics of discretisations. Another example illustrating the effect of unstable spurious solutions may be found in the paper of Brezzi, Ushiki, and Fujii [1]; and the effect of stable spurious solutions is discussed in the paper of Sleeman et al. [19].

Example 2.1. *Unstable spurious solutions.* This example illustrates the effect of unstable spurious solutions. Consider the ODE

$$u_t = -u^3. \tag{2.1}$$

It is clear that $u(t) \to 0$ as $t \to 0$, for any initial data. Consider now the forward Euler discretisation of (2.1)

$$U_{n+1} = U_n - \Delta t U_n^3. \tag{2.2}$$

It is straightforward to verify that (2.2) has a period 2 solution of the form $U_n = (-1)^n U_c$, where $U_c = \sqrt{2/\Delta t}$. The following result is proved in [23].

Result. The following three possibilities hold for (2.2):

(i) If $|U_0| < |U_c|$, then $|U^n| \to 0$ as $n \to \infty$.

(ii) If $|U_0| = U_c$, then $U_n = U_c$.

(iii) If $|U_n| > U_c$, then $|U^n| \to \infty$ as $n \to \infty$.

Notice the important role of the unstable spurious period 2 solution: data lying below it has the correct asymptotic behaviour and data lying above it diverges to infinity. *Thus the spurious solution divides the phase space into regions where correct and incorrect asymptotic behaviour are generated by the numerical method.* Of course, this result is particularly simple since we are in one dimension, but similar results can be proved in higher dimensions (see Theorem 4.2.) Consequently we claim that unstable spurious solutions play a very important role in the dynamics of discretisations.

Example 2.2. *Stable spurious solutions.* This example illustrates the effect of stable spurious solutions. Consider the scalar ODE

$$u_t = u^3. \tag{2.3}$$

The dynamics of this equation are very simple. The solution is

$$u(t) = \frac{u(0)}{((1 - 2tu(0))^2)^{\frac{1}{2}}}.$$

All initial data other than $u(0) = 0$ diverge to infinity monotonically in finite time $1/2u(0)^2$. The situation is summarised in Fig. 1; the variation with Δt is trivial for the differential equation and is included for the purpose of comparison with the discretisation.

Consider now the backward Euler discretisation of (2.3):

$$U_{n+1} = U_n + \Delta t U_{n+1}^3. \tag{2.4}$$

As for the differential equation, there is an unstable steady solution $U_n = 0$. It is a conseqeunce of [24, Thm. 2.3] that there is no strictly positive or strictly negative sequence satisfying (2.4) for all $n \geq 0$. Thus the numerical solution cannot diverge to infinity monotonically as the true solution does. What happens is that inital data other than $U_0 = 0$ evolves towards a stable period 2 solution as n increases. It is straightforward to show that (2.4) has a period 2 solution of the form $U_n = (-1)^n U_c$,

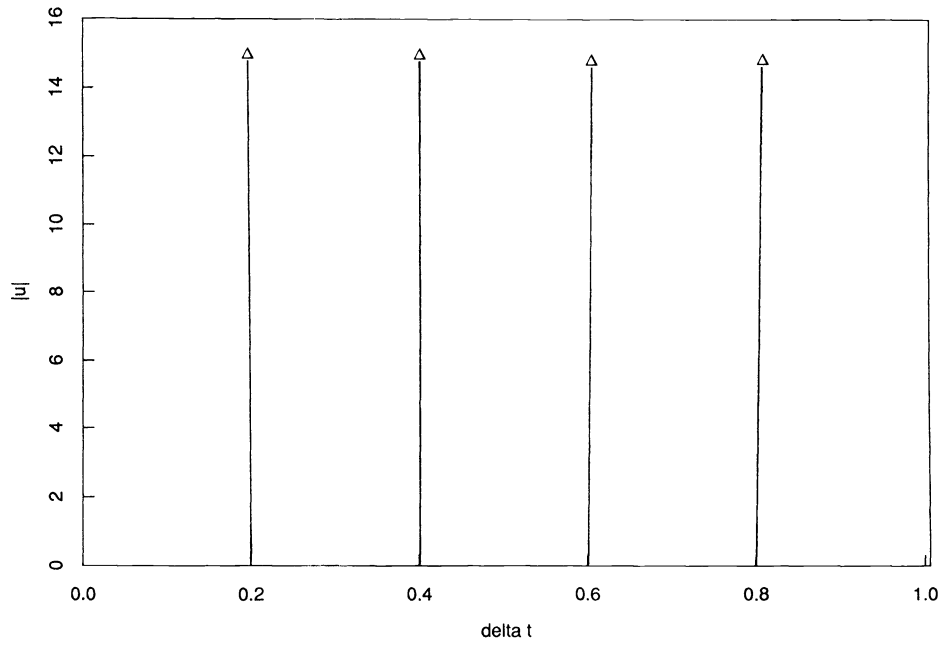


FIG. 1. *The dynamics of* (2.3). *Arrows indicate the evolution of* $|u(t)|$ *with time.*

with U_c as defined in the previous example. This solution is stable with a local contraction rate of $\frac{1}{25}$ over two timesteps. As in Example 2.1, the spurious solution exists for arbitrarily small Δt and moves off to infinity as $\Delta t \to 0$; see Theorem 3.4.

We solve (2.4) numerically using Newton iteration at each step, with an initial guess provided by the forward Euler method. Iteration is performed until convergence, for each value of n. All nonzero initial data evolves to the stable period 2 solution as n increases. Thus the dynamics of the numerical solution can be summarised as in Fig. 2, which should be compared with Fig. 1.

Figure 3 shows the solution of (2.4) with $U_0 = 1$ and $\Delta t = 0.0001$. The evolution towards the stable period 2 solution is clear.

Thus the spurious solution attracts all nonzero initial data. The correct asymptotics are not reproduced for any nonzero initial data. Consequently we claim that stable spurious solutions play a very important role in the dynamics of discretisations.

3. The Main Theorem and related results. In this section we prove the Main Theorem stated in §1. The proof is broken into three lemmas (Lemmas 3.1–3.3) concerned with the trapezoidal rule, the theta method for $\theta \neq \frac{1}{2}$, and the semi-implicit method. We also prove Theorem 3.4 concerning the behaviour of spurious solutions as $\Delta t \to 0$; we show that spurious solutions must tend to infinity in norm as $\Delta t \to 0$ (if they exist) and derive a condition under which spurious solutions do not exist for Δt arbitrarily small. In addition, we prove Theorem 3.5, which yields the same conclusion as Lemma 3.2, namely, the existence of spurious solutions, but with different assumptions. Finally, in Theorem 3.6, we prove that the existence of spurious period 2 solutions for $\theta = 0$ is equivalent to the existence of spurious steady solutions

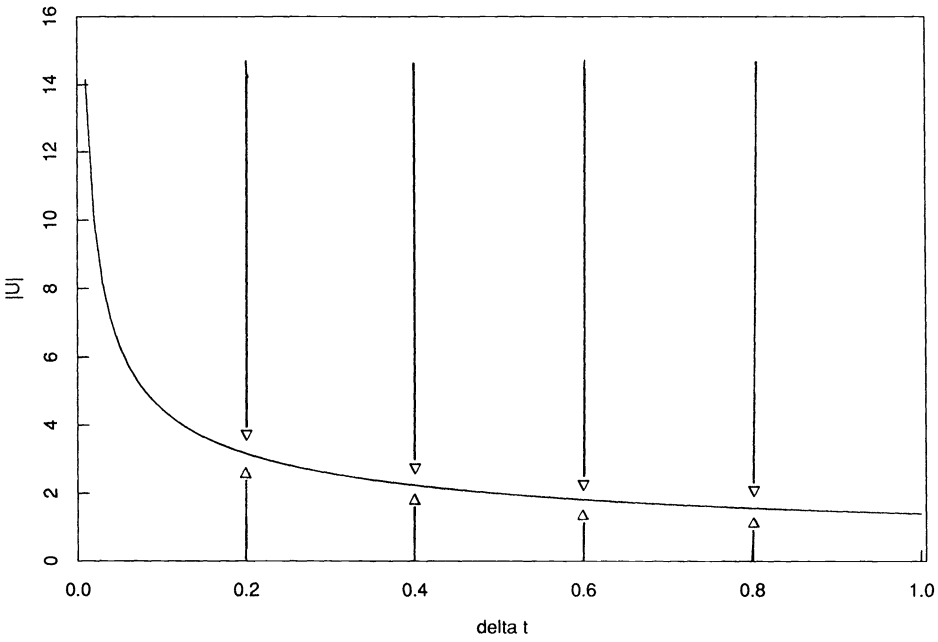


FIG. 2. *The dynamics of* (2.4). *Arrows indicate the evolution of* $|U_n|$ *with* n.

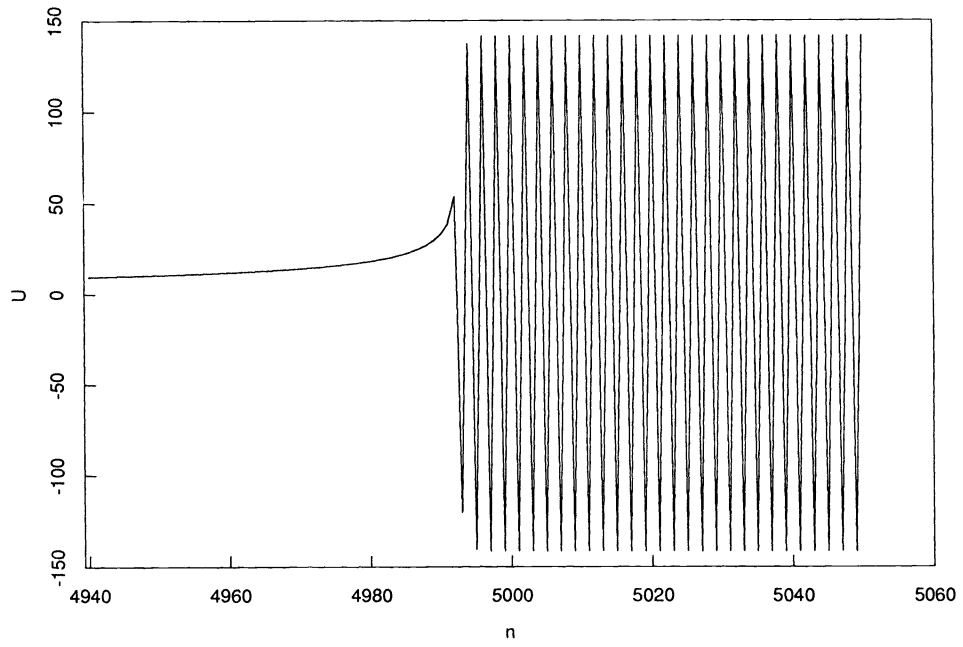


FIG. 3. *Solution of* (2.4). $U_0 = 1, \Delta t = 0.01$.

for a predictor-corrector method. Extensions and alternative proofs of Theorems 3.4 and 3.6 can be found in [9].

Period 2 solutions of (1.2) are pairs $(U, V) \in \Re^m \times \Re^m$ with $U \neq V$, satisfying

$$V - U = \Delta t[(1-\theta)G(U) + \theta G(V)], \tag{3.1}$$

$$U - V = \Delta t[(1-\theta)G(V) + \theta G(U)]. \tag{3.2}$$

Period 2 solutions of (1.4) are pairs $(U, V) \in \Re^m \times \Re^m$ with $U \neq V$, satisfying

$$V - U = \Delta t[AV + H(U)], \tag{3.3}$$

$$U - V = \Delta t[AU + H(V)]. \tag{3.4}$$

LEMMA 3.1. *For $\theta = \frac{1}{2}$, (1.2) cannot have period 2 solutions.*

Proof. When $\theta = \frac{1}{2}$, (3.1) and (3.2) give

$$2(V - U) = \Delta t[G(U) + G(V)],$$

$$2(U - V) = \Delta t[G(V) + G(U)].$$

Subtracting, we obtain $U = V$ and so period 2 solutions cannot exist. □

Assumptions on $G(u)$ for Lemmas 3.2 and 3.3.

(i) $G(0) = 0$.

(ii) $G(u) \in C^2(\Re^m, \Re^m)$ for u near 0.

(iii) Let $dG(0)$ denote the Jacobian of G at $u = 0$. Then we assume that $dG(0)$ has a real, nonzero, simple eigenvalue η. The corresponding eigenvector is y. (This is required only for Lemma 3.2.)

(iv) Let $dH(0)$ denote the Jacobian of $H(u)$ at $u = 0$. Then we assume that $A - dH(0)$ has a real, nonzero, simple eigenvalue γ. The corresponding eigenvector is w. (This is required only for Lemma 3.3.)

(v) The matrix $dG(0)$ is nonsingular.

We now discuss the necessity of the assumptions for the proof of the lemmas. Our method of proof is to consider the bifurcation of spurious period 2 solutions from steady solutions. Steady solutions u satisfy $G(u) = 0$ and without loss of generality we may assume that $u = 0$. Thus Assumption (i) is necessary for our method of proof. (However, it is possible for spurious period 2 solutions to exist which do not bifurcate from a steady solution at any finite value of Δt, as Examples 2.1 and 2.2 show.) Assumption (ii) is required so that we may employ standard bifurcation theory as in [3]. The existence of a real nonzero eigenvalue in Assumption (iii) (respectively, Assumption (iv)) for Lemma 3.2 (respectively, Lemma 3.3) is necessary. That the eigenvalue be simple is not strictly necessary, but simplifies the analysis. (Note that by a different method of proof we are able to relax the condition on the existence of a real eigenvalue, provided that $G(u)$ has two zeros; see Theorem 3.5.) Assumption (v) precludes the bifurcation of steady solutions in the differential equation itself. This could be allowed for in the analysis, but would complicate it unnecessarily; the simultaneous bifurcation of steady and periodic solutions is analysed in [21].

Lemmas 3.2 and 3.3 are proved by application of standard "bifurcation from a simple eigenvalue" arguments; see [3, Chap. 5] or [7]. Here Δt plays the role of

bifurcation parameter. Note that the lemmas provide information about the structure of the period 2 solutions near to the bifurcation point. This is required to initiate numerical continuation procedures which trace out the spurious solutions; see §5.

LEMMA 3.2. *Period 2 solutions of* (1.2) *bifurcate from the trivial solution* $(U,V) = (0,0)$ *at* $\Delta t = 2/(2\theta - 1)\eta$. *Furthermore, the branches of nontrivial solutions satisfy, for* $\mu \ll 1$,

$$U = -V = \mu y + O(|\mu|^2),$$

$$\Delta t = \frac{2}{(2\theta - 1)\eta} + O(|\mu|).$$

Proof. Consider (3.1) and (3.2). Clearly these are satisfied by $(U,V) = (0,0)$ for all Δt. Straightforward application of [3, Thm. 5.3, Chap. 5] shows that bifurcation occurs at simple eigenvalues Δt of the following problem:

$$\beta - \alpha = \Delta t[(1-\theta)dG\alpha + \theta dG\beta],$$

$$\alpha - \beta = \Delta t[(1-\theta)dG\beta + \theta dG\alpha].$$

Throughout this proof dG is evaluated at $u = 0$. Adding the two equations shows that $dG(\alpha + \beta) = 0$. Since dG is invertible, we obtain $\alpha = -\beta$. Thus the equations reduce to

$$[2 - \Delta t(2\theta - 1)dG]\beta = 0.$$

By assumption, dG has a real, nonzero, simple eigenvalue η and so we deduce that the linearised system is singular at

$$\Delta t = \frac{2}{(2\theta - 1)\eta}.$$

A little calculation shows that this eigenvalue is simple (in the sense defined in [3, Chap. 5]) and the conclusions of the lemma follow in a straightforward way. □

LEMMA 3.3. *Period 2 solutions of* (1.4) *bifurcate from the trivial solution* $(U,V) = (0,0)$ *at* $\Delta t = \frac{2}{\gamma}$. *Furthermore, the branches of nontrivial solutions satsify, for* $\mu \ll 1$,

$$U = -V = \gamma w + O(|\mu|^2),$$
$$\Delta t = \frac{2}{\gamma} + O(|\mu|).$$

Proof. The proof is very similar to Lemma 3.2 and is omitted. □

An important question for the numerical analyst is to determine to where the branches of solutions $(U(\mu), V(\mu), \Delta t(\mu))$ go to in function and parameter space. In particular, it is important to know whether the branches extend to arbitrarily small values of Δt. This depends on the global structure of the nonlinear terms and no general answer is possible. It is, however, an important area for future research.

We now prove a result about the behaviour of spurious periodic solutions as $\Delta t \to 0$. Roughly, Theorem 3.4 states that if spurious solutions exist for arbitrarily small Δt then they must move off to infinity in norm—essentially a consequence of the convergence of the scheme.

THEOREM 3.4 *Consider genuine period 2 solutions* (U,V) *of* (1.2) *and of* (1.4) *with* $U \neq V$. *Assume that* $G(U) \in C^1(\Re^m, \Re^m)$. *The following results hold:*

(i) *If a pair* (U, V) *can be found for arbitrarily small* Δt, *then* $\|U\|_\infty$ *and* $\|V\|_\infty \to \infty$ *as* $\Delta t \to 0$.

(ii) *Furthermore, if* $G(U) = BU + J(U)$, *where* B *is an invertible matrix and* $J(U)$ *is uniformly bounded for all* U, *then period 2 solutions cannot exist for arbitrarily small* Δt.

Note on Theorem 3.4. The assumption that $G(u)$ is a C^1 function is strictly necessary. See [9, Ex. 4.8] for an illustration of this fact. It is shown that, for $G(u) = -\sqrt{(u)}, u > 0$ and $G(u) = \sqrt{(|u|)}, u < 0$ the Euler method has period 2 solutions of the form $U_n = (-1)^n \Delta t^2/4$.

Proof. We restrict our proofs to the theta method (3.1), (3.2) since the proofs for the semi-implicit method (3.3), (3.4) are slight modifications. Consider (1.2). Period 2 solutions satisfy (3.1), (3.2). Adding these equations shows that for $\Delta t \neq 0$,

$$G(U) + G(V) = 0.$$

Using this result in (3.1), (3.2), we obtain

$$V - U = \Delta t(1 - 2\theta)G(U) \tag{3.5}$$

and

$$U - V = \Delta t(1 - 2\theta)G(V). \tag{3.6}$$

Subtracting (3.6) from (3.5) and using the Mean Value Theorem gives

$$\left[2I + \Delta t(1 - 2\theta)\int_0^1 dG(tU + (1-t)V)dt\right](V - U) = 0. \tag{3.7}$$

We first prove part (i) of Theorem 3.4. We assume that U remains bounded in norm as $\Delta t \to 0$ and we obtain a contadiction. From (3.5) we deduce that V remains bounded. Thus the linear operator in (3.7) is invertible for all Δt sufficiently small and $U = V$. Hence period 2 solutions do not exist for all Δt sufficiently small. This is the required contradiction which establishes that U becomes unbounded. Similarly, it may be shown that V becomes unbounded.

We now prove Theorem 3.4(ii). We assume that a pair (U, V) with $U \neq V$ can be found for Δt arbitrarily small and we obtain a contradiction. Period 2 solutions satisfy (3.5). Using (3.5) to eliminate V, and using $G(U) = BU + J(U)$, we obtain

$$BU + J(U) + B[U + \hat{\Delta}tBU + \hat{\Delta}tJ(U)] + J(V) = 0.$$

Here $\hat{\Delta}t = (1 - 2\theta)\Delta t$. Rearranging gives

$$(2B + \hat{\Delta}tB^2)U = -J(U) - J(V) - \hat{\Delta}tBJ(U).$$

From the boundedness of J and the invertibility of B we deduce that U remains bounded as $\Delta t \to 0$. By (3.5) so too does V. Hence, by (3.7), $U = V$ for all Δt sufficiently small so that period 2 solutions cannot exist. □

We now prove another theorem on the existence of spurious periodic solutions in the θ method. Lemma 3.2 proves the existence of spurious periodic solutions which bifurcate from a steady solution by means of a simple symmetry-breaking bifurcation. However, to prove Lemma 3.2 it is necessary to assume that the Jacobian $dG(0)$ is invertible and that $dG(0)$ has a real, nonzero eigenvalue. We now prove the existence

of spurious period 2 solutions assuming the existence of two steady solutions but under the weaker assumption that the Jacobian is invertible at both steady solutions; we do not need to assume the existence of a real eigenvalue. The result holds for $\Delta t \gg 1$ but may be useful as a starting point for numerical continuation to small values of Δt.

THEOREM 3.5. *Let $\theta \neq \frac{1}{2}$. Assume that there exist U_0, V_0 with $U_0 \neq V_0$ satisfying $G(U_0) = G(V_0) = 0$ and that $dG(U_0)$ and $dG(V_0)$ are invertible. Then for $|\Delta t|$ sufficiently large, there exist spurious period 2 solutions of the discretisation* (1.2).

Proof. Period 2 solutions of (1.2) are pairs (U, V) satisfying (3.1) and (3.2). For $\Delta t = \infty$ these equations have the solution $U = U_0$ and $V = V_0$. We now prove that these solutions can be uniquely extended to solutions for $|\Delta t|$ sufficiently large. By the Implicit Function Theorem, this can be done provided that the following problem for (α, β) has the unique solution $(0, 0)$:

$$(1-\theta)dG(U_0)\alpha + \theta dG(V_0)\beta = 0,$$

$$(1-\theta)dG(V_0)\beta + \theta dG(U_0)\alpha = 0.$$

If $\theta = 0$ or 1 we deduce that $\alpha = \beta = 0$ automatically. If $\theta \neq 0, 1$ then simple manipulations show that $\alpha = \beta = 0$ provided that $\theta \neq \frac{1}{2}$. □

The final theorem in this section relates the existence of spurious periodic solutions in the Euler method to the existence of spurious steady solutions in the following predictor-corrector method for (1.1):

$$U_{n+1}^p = U_n + \Delta t G(U_n), \tag{3.8}$$

$$U_{n+1} = U_n + \frac{\Delta t}{2}[G(U_{n+1}^p) + G(U_n)]. \tag{3.9}$$

THEOREM 3.6. *Consider the predictor-corrector method* (3.8), (3.9). *This method has a spurious steady solution satisfying $U_{n+1} = U_n$ whenever the theta method with $\theta = 0$ has a spurious period 2 solution.*

Proof. Steady solutions of (3.8), (3.9) satisfy

$$V = U + \Delta t G(U), \tag{3.10}$$

$$0 = G(U) + G(V). \tag{3.11}$$

Using (3.11) in (3.10) we obtain

$$U = V + \Delta t G(V). \tag{3.12}$$

Equations (3.10) and (3.12) are identical to (3.1) and (3.2) with $\theta = 0$. Hence the result follows. □

4. Application to nonlinear parabolic equations. Here we apply the results in §4 to nonlinear parabolic equations of the form

$$u_t = u_{xx} + \lambda u + h(u, u_x) \tag{4.1}$$

with boundary conditions

$$u(0, t) = u(1, t) = 0 \tag{4.2}$$

and some initial conditions. For simplicity we shall assume that $h(0,0) = 0$ and that $h(a,b)$ is superlinear in its arguments a and b. These assumptions are not necessary (see [22]) but simplify the analysis considerably; we seek period 2 solutions bifurcating from zero and our assumptions ensure that the linear eigenvalue problem governing bifurcation is symmetric and explicitly solvable.

We discretise (4.1), (4.2) in space using a centred approximation to the second derivative and a consistent approximation to the nonlinear term (which may or may not involve upwinding). We obtain the system of $J-1$ nonlinear ODEs

$$\frac{du_j}{dt} = \frac{\delta_x^2 u_j}{\Delta x^2} + \lambda u_j + g(u_{j-1}, u_j, u_{j+1}), \tag{4.3}$$

for $j = 1, \cdots, J-1$. Here $J\Delta x = 1, \delta_x^2 u_j = u_{j+1} - 2u_j + u_{j-1}$, and

$$u_0 = u_J = 0. \tag{4.4}$$

From our assumptions on $h(a,b)$ we deduce that $g(0,0,0) = 0$ and that g is superlinear in its three arguments. (Note that g may depend upon Δx but that we supress this for notational convenience.)

We can apply Lemma 3.2 directly to (4.3), (4.4). We let $U = [U_1, U_2, \cdots, U_{J-1}]^T$ and $V = [V_1, V_2, \cdots, V_{J-1}]^T$. At zero, the Jacobian of the right-hand side is a symmetric, tridiagonal matrix with $J-1$ distinct eigenvalues $\eta^{(k)}$ and corresponding eigenvectors $y^{(k)}$ [4]. These are

$$\eta^{(k)} = \lambda - \frac{4\sin^2(k\pi/2J)}{\Delta x^2}, \tag{4.5}$$

with corresponding eigenvectors given componentwise by

$$y_j^{(k)} = \sin(k\pi j/J). \tag{4.6}$$

The index $k = 1, \cdots, J-1$. For $y_j^{(k)}$ the subscript j denotes the jth component of the vector $y^{(k)}$. By Lemma 3.2 we can find $J-1$ branches of period 2 solutions when applying the θ method to (4.2) and locally these satisfy

$$U_j = -V_j = \mu y_j^{(k)} + O(|\mu^2|), \tag{4.7}$$

$$\Delta t_k = \frac{2}{(2\theta - 1)\eta^{(k)}} + O(|\mu|). \tag{4.8}$$

Note that the smallest of the positive values of Δt is the asymptotic stability limit for the numerical method linearised about zero.

The local results (4.7) and (4.8) provide the starting points for a numerical continuation procedure which traces out spurious period 2 solutions in the theta method as Δt is varied. We now describe several examples corresponding to different choices of nonlinearity $h(a,b)$. Throughout the following, we use the nonlinear solver PITCON [16] to follow the solution branches of the nonlinear equations governing existence of period 2 solutions. Note that the bifurcation of period 2 solutions is necessarily of pitchfork type [7], [22] and so there are two branches emanating from each bifurcation point. It is possible to move from one branch to the other by interchanging the roles of U and V. For this reason the two branches are indistinguishable in the figures.

Example 4.1. Consider the equation

$$u_t = u_{xx} - u^3 \tag{4.9}$$

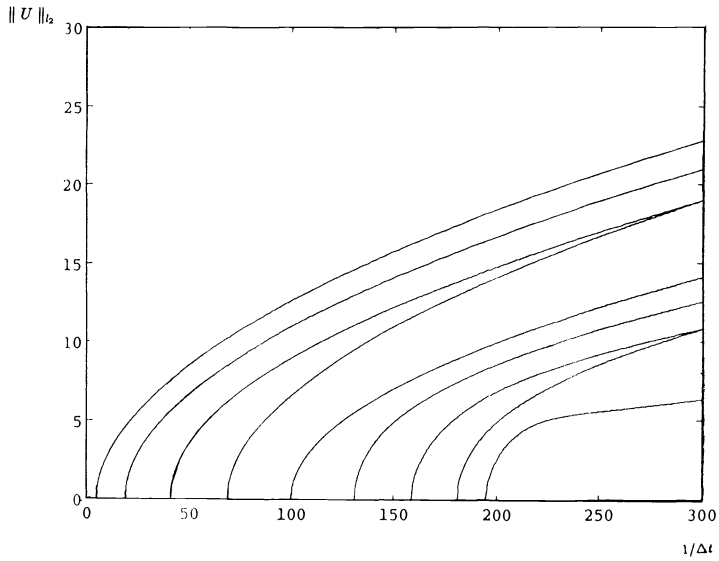


FIG. 4. *l_2 norm of spurious periodic solutions versus $1/\Delta t$ for equations* (4.10) *and* (4.11).

together with boundary conditions (4.2). A Liapunov functional approach shows that all solutions converge to zero as $t \to \infty$. We discretise the equation using explicit Euler timestepping (θ=0) and we take $g(a, b, c) = -b^3$. We obtain, for $j = 1, \cdots, J-1$,

$$u_j^{n+1} = u_j^n + r\delta_x^2 u_j^n - \Delta t (u_j^n)^3, \tag{4.10}$$

boundary conditions

$$u_0^n = u_J^n = 0, \tag{4.11}$$

and some initial condition on u_j^0. Here $r = \Delta t/\Delta x^2$.

We seek period 2 solutions satisfying the Z_2 symmetry $U = -V$ and find that they satisfy

$$\frac{\delta_x^2 U_j}{\Delta x^2} - U_j^3 + \frac{2U_j}{\Delta t} = 0, \qquad j = 1, \cdots, J-1 \tag{4.12}$$

together with

$$U_0 = U_J = 0. \tag{4.13}$$

Using the estimate (4.7), (4.8) we can follow the branches of spurious periodic solutions satisfying (4.12), (4.13). Figure 4 shows a plot of these branches in the case $\Delta t = 0.1$; thus $J = 10$ and there are nine branches. Note that the branches exist for Δt arbitrarily small. This is proved in [22], where it is shown that the solution branches approach infinity at a rate proportional to $\sqrt{1/\Delta t}$. This scaling follows from a balance between U_j^3 and $2U_j/\Delta t$. The fact that they approach infinity is a consequence of Theorem 3.4(i).

Recall that the solution of the differential equation tends to zero as t tends to infinity. The following theorem gives conditions for the Euler approximation to behave similarly. The theorem reflects the effect of the spurious periodic solutions in that Δt must be restricted in terms of the reciprocal of the square of the magnitude of the initial data; this is precisely the scaling of the spurious solutions with Δt. The theorem shows that the phase space can be divided into regions where correct and incorrect asymptotic behaviour is observed.

THEOREM 4.2. *Consider the solution of* (4.10), (4.11). *Let*

$$u_{\max} = \max_{0 \le j \le J} |u_j^0|.$$

Then we find the following:

(i) *If*

$$\Delta t \le \frac{\Delta x^2}{2 + 3(\Delta x u_{\max})^2}.$$

Then $u_j^n \to 0$ *as* $n \to \infty$ *for all initial data (correct asymptotics).*

(ii) *If* $J = 0 \pmod 3$, *then there exists initial data such that if*

$$\Delta t > \frac{2\Delta x^2}{3 + (\Delta x u_{\max})^2}$$

then $|u_j^n| \to \infty$ *as* $n \to \infty$ (*spurious asymptotics*).

Proof. (i) Define

$$u_{\max}^n = \max_{0 \le j \le J} u_j^n, \qquad u_{\min}^n = \min_{0 \le j \le J} u_j^n.$$

Note that $u_{\max}^n \ge 0$ and $u_{\min}^n \le 0$ by virtue of (4.11). The proof proceeds by induction. Assume that

$$\Delta t \le \frac{\Delta x^2}{2 + 3(\Delta x u_{\max}^n)^2}, \tag{4.14}$$

$$\Delta t \le \frac{\Delta x^2}{2 + 3(\Delta x u_{\min}^n)^2}. \tag{4.15}$$

This holds for $n = 0$ by assumption. Rearranging (4.10), we obtain, for $j = 1, \cdots, J-1$

$$u_j^{n+1} = r u_{j-1}^n + [1 - 2r - \Delta t (u_j^n)^2] u_j^n + r u_{j+1}^n.$$

Using the monotonicity of $(1-2r)u - \Delta t u^3$ implied by (4.14), (4.15), we deduce that

$$u_j^{n+1} \le [1 - \Delta t (u_{\max}^n)^2] u_{\max}^n. \tag{4.16}$$

Similarly,

$$u_j^{n+1} \ge [1 - \Delta t (u_{\min}^n)^2] u_{\min}^n. \tag{4.17}$$

This shows that, if $u_{\max}^{n+1}$ (respectively, $u_{\min}^{n+1}$) is nonzero and attained in the interior ($j \ne 0, J$), then it strictly decreases (respectively, increases) over one step. If $u_{\max}^{n+1}$ (respectively, $u_{\min}^{n+1}$) is attained at the boundary ($j = 0$ or $j = J$), then $u_{\max}^{n+1} = 0$

(respectively, $u_{\min}^{n+1} = 0$.) Thus equations (4.14) and (4.15) hold for $n+1$. By induction we deduce that $u_j^n \to 0$ as $n \to \infty$.

(ii) For this part we assume that $J = 0 \pmod 3$. With this assumption it may be shown by substitution that

$$u_j^n = A^n \sin(2\pi j/3) \tag{4.18}$$

is an exact solution of (4.10), (4.11) provided that A^n satisfies the recursion

$$A^{n+1} = [1 - 3r - \tfrac{3}{4}\Delta t(A^n)^2]A^n. \tag{4.19}$$

This exact nonlinear solution satisfies the recursion because of the property of aliasing: to establish (4.19) we use $\sin^3 x = \frac{3}{4}\sin x - \frac{1}{4}\sin 3x$. For $x = 2\pi j/3$, $\sin 3x = 0$ and hence (4.19) follows. If

$$1 - 3r - \frac{3}{4}\Delta t(A^n)^2 < -1, \tag{4.20}$$

it is clear that $|A^{n+1}| > |A^n|$ and hence, by induction, that $|A^n| \to \infty$ as $n \to \infty$. Hence $u_j^n \to \infty$ as $n \to \infty$ provided that

$$\Delta t > \frac{8\Delta x^2}{12 + 3(\Delta x A^0)^2}. \tag{4.21}$$

However, $u_{\max} = A^0 \sin(2\pi/3)$. Thus (4.21) gives

$$\Delta t > \frac{2\Delta x^2}{3 + (\Delta x u_{\max})^2}$$

as required. □

We now describe some timestepping experiments which illustrate Theorem 4.2. We solve the recursion (4.10), (4.11) with $1/\Delta t = 300.557$ and with $J = 10$. We use initial data of the form

$$u_j^0 = aU_j,$$

where U_j is the spurious periodic solution lying on the uppermost branch in Figure 4 at $\frac{1}{\Delta t} = 300.557$. Figures 5 and 6 show plots of the l_2 norm of u_j^n,

$$\sqrt{\sum_{j=0}^{J-1}(u_j^n)^2},$$

versus n for two different values of a. In Fig. 5 we take $a = 1$. The true solution is pure period 2 in n. However, rounding errors excite an instability and the solution diverges to infinity (spurious asymptotics.) This occurs in a very small number of steps; the l_2 norm is approximately constant for eight steps (since we start on the period 2 solution which has the symmetry $U_j = -V_j$) and in the last two steps the instability rapidly takes over and the solution diverges to infinity. In Fig. 6 we take $a = 0.99$. Here the solution lies just beneath the spurious periodic solution and it is attracted to zero (correct asymptotics).

Example 4.3. In this example we study the Ginzburg–Landau equation

$$u_t = u_{xx} + \lambda(u - u^3),$$

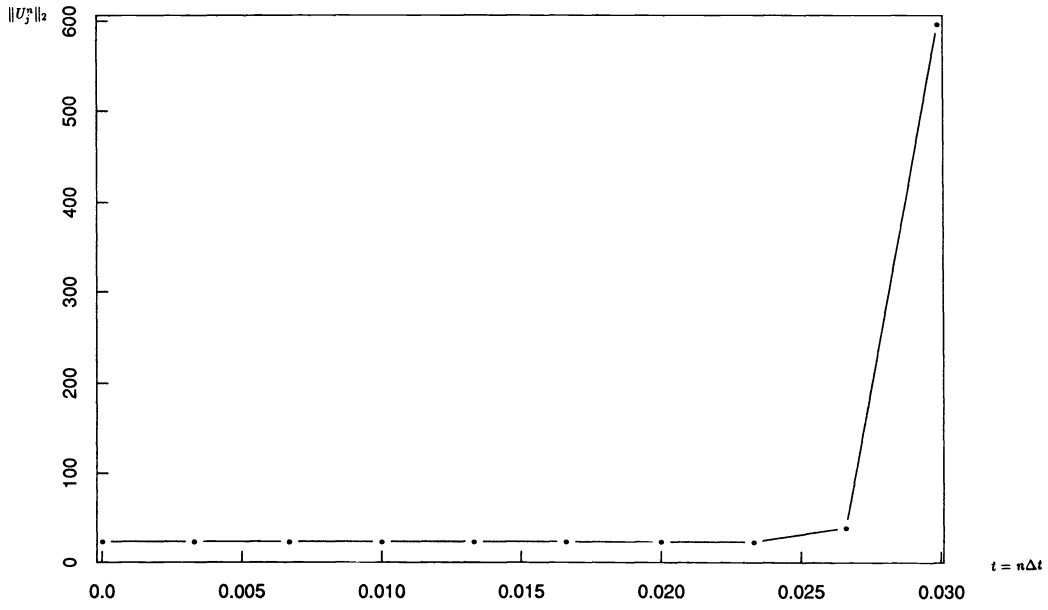


FIG. 5. $a = 1.0$. *Plots of the* l_2 *norm of* u_j^n *versus* $t = n\Delta t$.

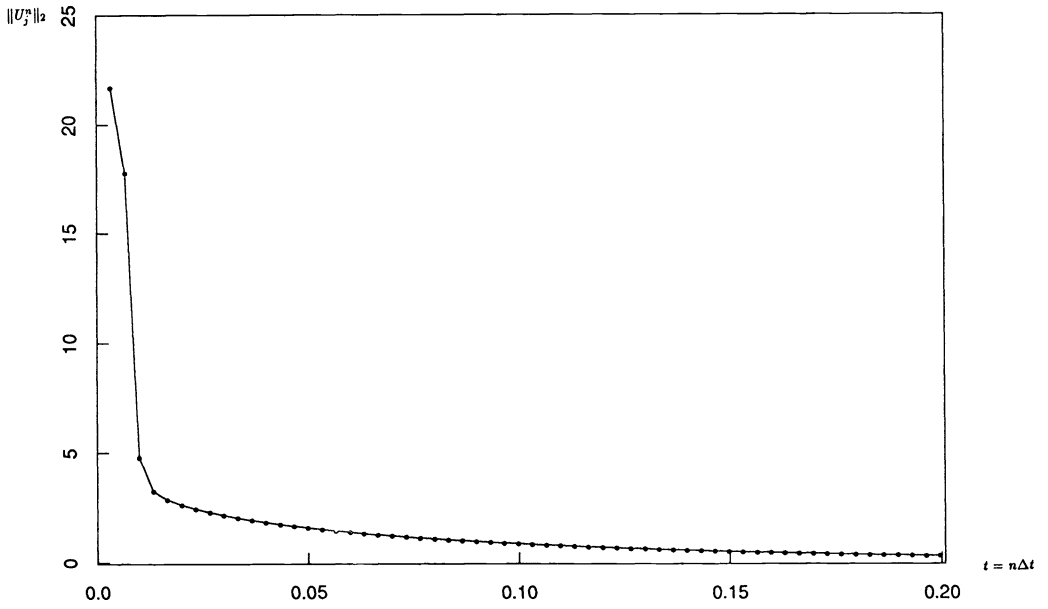


FIG. 6. $a = 0.99$. *Plots of the* l_2 *norm of* u_j^n *versus* $t = n\Delta t$.

together with boundary conditions (4.2). The dynamics of this problem are very well understood: the solution converges to zero for λ sufficiently small and to the unique positive or negative solution for larger values of λ for almost all initial data. See [2].

We discretise the equation using explicit Euler timestepping ($\theta = 0$) and seek period 2 solutions satisfying the Z_2 symmetry $U = -V$. We obtain the defining problem

$$\frac{\delta_x^2 U_j}{\Delta x^2} + \lambda(U_j - {U_j}^3) + \frac{2U_j}{\Delta t} = 0, \qquad j = 0, \cdots, J-1, \tag{4.22}$$

together with boundary conditions (4.13). This problem is related to (4.12), (4.13) by a simple group of continuous transformations: we let $\gamma = 2/\Delta t$; then (4.22), (4.13) can be transformed into (4.12), (4.13) by

$$U_j \to \lambda^{-\frac{1}{2}} U_j \tag{4.23}$$

and

$$\gamma \to \gamma - \lambda. \tag{4.24}$$

Thus we expect the spurious periodic solutions of the Ginzburg–Landau equation to be of the same form as the solutions of Example 4.1 but scaled and shifted. The bifurcation points are shifted to the left (in $1/\Delta t$) and the branches are scaled down. This can be seen in Fig. 7, which should be compared with Fig. 4. The spurious periodic solutions affect the dynamics of the Euler method in a similar way to Example 4.1; it is necessary to restrict Δt in terms of the magnitude of the initial data to obtain satisfactory results.

Example 4.4. Consider the equation

$$u_t = u_{xx} - \frac{u^3}{1 + \delta u^4}, \tag{4.25}$$

together with boundary conditions (4.2). When discretised in space alone, the system of nonlinear ODEs satisfies the criteria of Theorem 3.4(ii). Consequently we expect that branches of spurious periodic solutions cannot extend back to arbitrarily small Δt.

We discretise the equation using explicit Euler timestepping ($\theta = 0$) and seek period 2 solutions satisfying the Z_2 symmetry $U = -V$. We obtain the problem

$$\frac{\delta_x^2 U_j}{\Delta x^2} - \frac{U_j^3}{1 + \delta U_j^4} + \frac{2U_j}{\Delta t} = 0, \qquad j = 1, \cdots, J-1, \tag{4.26}$$

together with boundary conditions (4.13). The numerical solution of this problem is shown in Fig. 8. Notice that the branches of spurious periodic solutions are bounded uniformly from $\Delta t = 0$. This is in accordance with Theorem 3.4(ii).

Notice also that the branches of solutions in Fig. 8 emanating from $\|U\| = 0$, $\Delta t = \Delta t_k$ are asymptotic to the same value Δt_k at $\|U\| = \infty$. This results from the fact that the nonlinear source term is asymptotically negligible for $\|U\| \ll 1$ and for $\|U\| \gg 1$. This form of bifurcation from infinity can be established rigorously by applying the Asymptotic Bifurcation Theorem of Krasnosel'skii quoted in [25].

Example 4.5. Consider the viscous Burgers' equation

$$u_t = u_{xx} + uu_x, \tag{4.27}$$

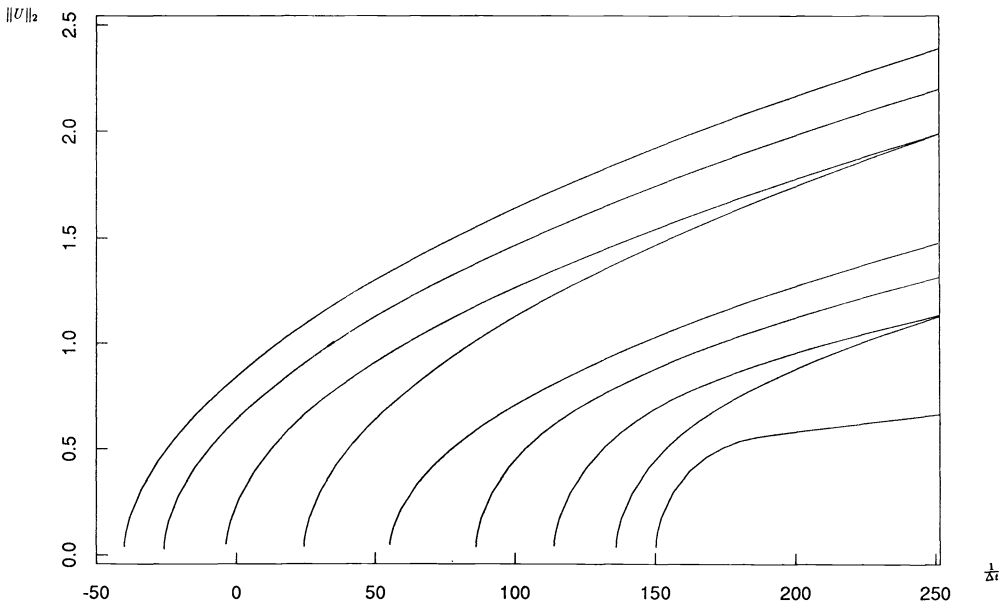


FIG. 7. *l_2 norm of spurious periodic solutions versus Δt for the Ginzburg–Landau equation.* $\lambda = 90$.

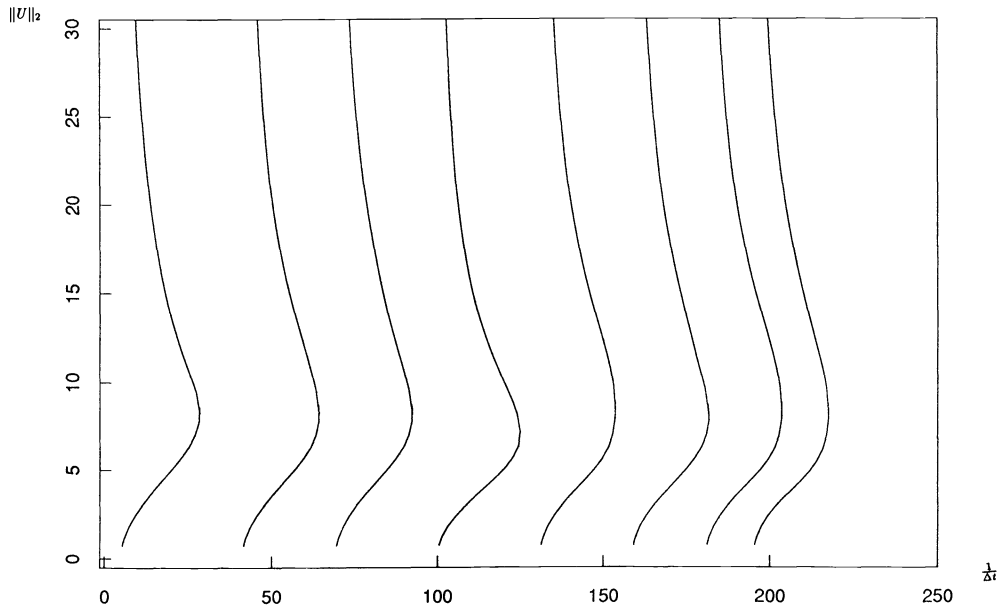


FIG. 8. *l_2 norm of spurious periodic solutions versus Δt for Euler discretisation of* (4.25).

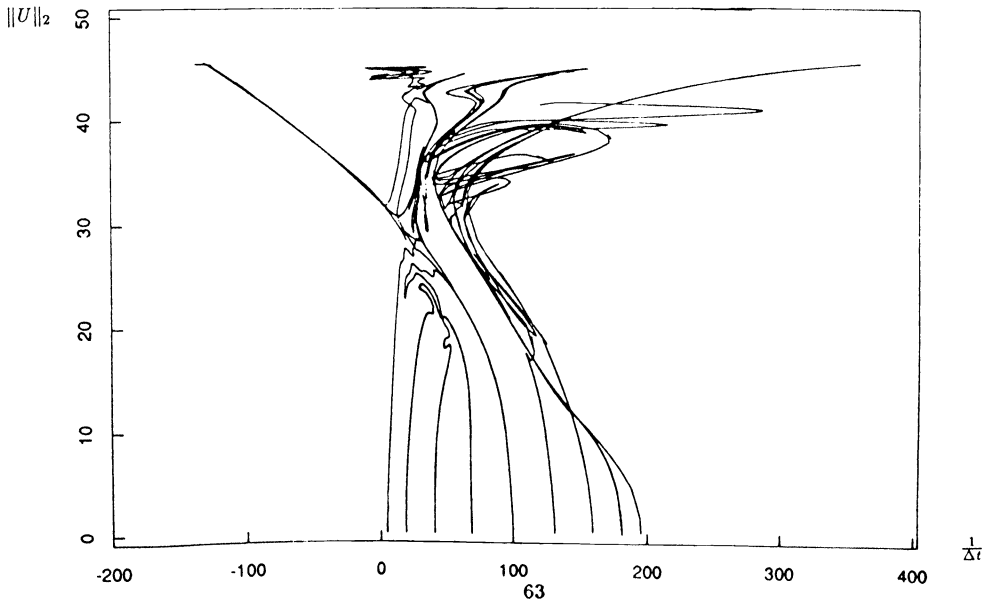


FIG. 9. *l_2 norm of spurious periodic solutions versus $1/\Delta t$ for the upwind scheme* (4.27).

together with boundary conditions (4.2). A Liapunov functional approach shows that all solutions converge to zero as $t \to \infty$. We consider two different schemes for the solution of (4.27), (4.2). Both involve Euler timestepping but in one we use an upwind scheme and in the other a centred scheme.

Consider first the *upwind scheme* for which we take

$$g(a,b,c) = \frac{c^2 - b^2}{2\Delta x} \quad \text{for} \quad j = 1, \cdots, J-1.$$

Numerical continuation was implemented to follow the nontrivial branches away from the bifurcation points given by (4.7), (4.8). Figure 9 shows a plot of these branches in the case $\Delta x = 0.1$. Most of the branches are irrelevant since they exist only above the linear stability limit ($1/\Delta t \approx 200$).

The branch emanating from the eighth bifurcation point (from left to right in $1/\Delta t$) enters into the region of linearised stability for the scheme. This branch is shown in Fig. 10. Note that the branch seems to terminate at $1/\Delta t \approx 360$, contradicting the Rabinowitz global bifuration theory [15]. What actually happens is that the solutions U and V satisfy $U = V$ at this point and beyond this the roles of U and V interchange and the solution follows a symmetric transformation of the original branch back to the bifurcation point; in $(U, V, \Delta t)$ space it is actually a loop and there is no contradiction. This branch is of particular interest. At the point where $U = V$, we have obtained a steady solution of the discretisation of Burgers' equation. This steady solution is spurious and caused by *spatial discretisation* (in contrast to the spurious periodic solutions which are caused by temporal discretisation). The existence of spurious steady solutions for Burgers' equation (with nonhomogeneous boundary conditions)

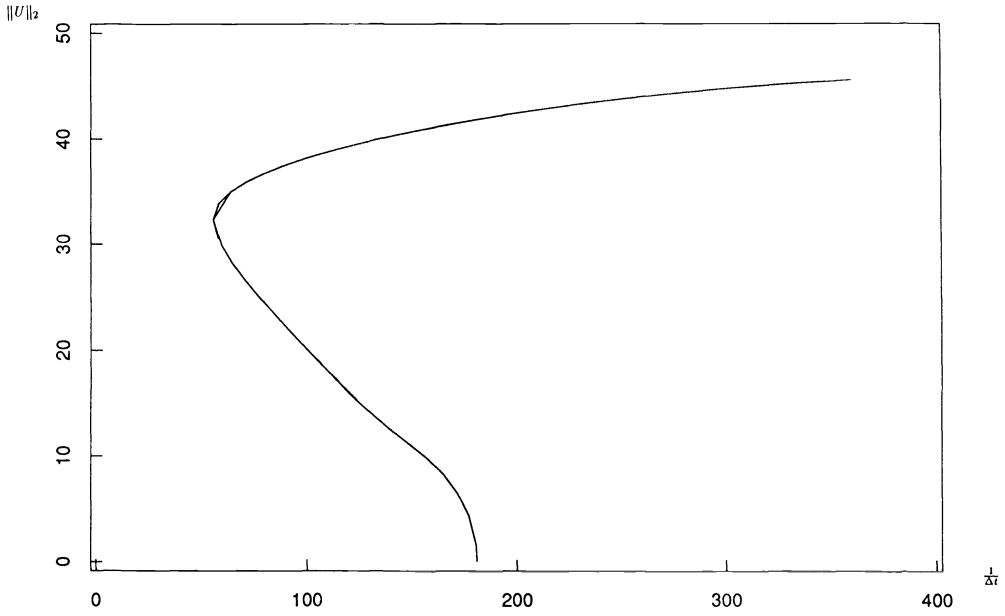


FIG. 10. *l_2 of a spurious periodic solution versus $1/\Delta t$ for the upwind scheme* (4.27).

is discussed in [20], for the driven cavity problem in [18], and for nonlinear elliptic problems in [12].

We now discretise Burgers' equation by *centred differencing*. Thus we take

$$g(a,b,c) = \frac{c^2 - a^2}{4\Delta x} \quad \text{for } j = 1, \cdots, J-1. \tag{4.28}$$

The spurious periodic solutions are shown in Fig. 11. The linear theory determining the bifurcation points is identical to that for the upwind scheme. Note, however, that the behaviour of the solution branches is entirely different far from the bifurcation points. This illustrates an important point: the interaction between spatial and temporal discretisation is crucial in the design of schemes which minimise the effect of spurious solutions.

Example 4.6. Consider the equation

$$u_t = u_{xx} + \lambda(u + u^3), \tag{4.29}$$

together with boundary conditions (4.2). We discretise the equation using backward Euler timestepping ($\theta = 1$) and seek period 2 solutions. The points at which spurious period 2 solutions bifurcate from the trivial solution are given by (4.8), which reduces to

$$\Delta t_k = -\frac{2\Delta x^2}{4\sin^2(k\pi/2J) - \lambda\Delta x^2} \quad \text{for} \quad i = 1, \cdots, J-1. \tag{4.30}$$

We choose λ sufficiently large that $\Delta t_1 > 0$. Figure 12 shows the spurious periodic solution emanating from Δt_1. Note that it can be found for all Δt sufficiently small and moves off to infinity in norm as predicted by Theorem 3.4(i).

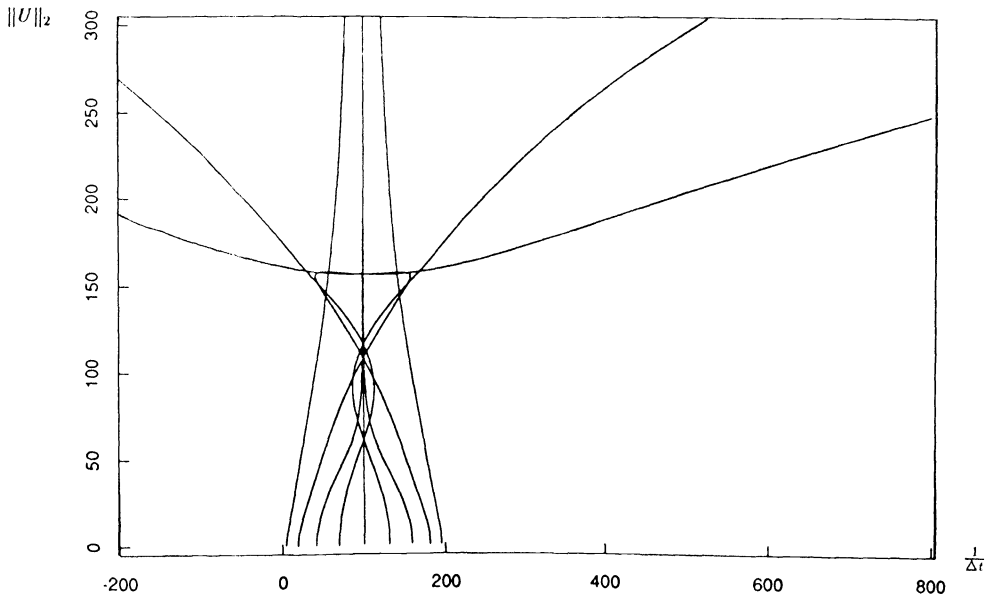


FIG. 11. *l_2 norm of spurious periodic solutions versus $1/\Delta t$ for the centred scheme.*

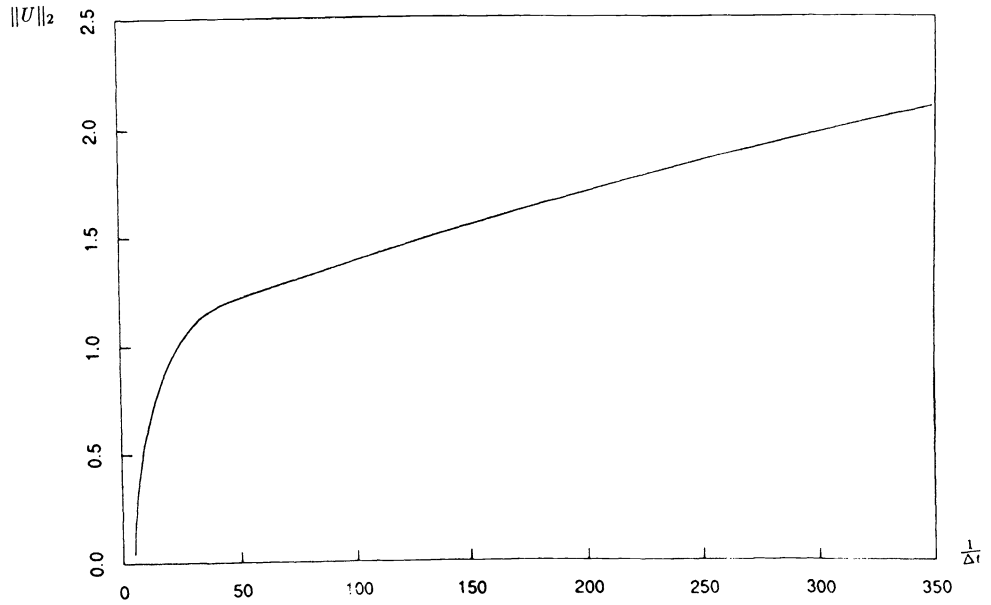


FIG. 12. *l_2 norm of spurious periodic solutions versus $\frac{1}{\Delta t}$ for backward Euler discretisation of* (4.29).

5. Conclusions. In this paper we have investigated the dynamics of numerical methods for nonlinear differential equations. We have shown, by means of illustrative examples, that spurious periodic solutions can seriously degrade the performance of the numerical methods (§2). Consequently, it is important to design schemes which minimise the effect of spurious solutions. We have shown that the trapezoidal rule never possesses spurious solutions of period 2 in n; on the other hand, the theta method with $\theta \neq \frac{1}{2}$ does possess spurious solutions of period 2 in n. (This and other related results are in §3.) We have applied our theory to a number of specific examples arising from discretisation of semilinear parabolic equations (§4). In a specific case, we have generalised the asymptotic stability criterion $\Delta t/\Delta x^2 \leq \frac{1}{2}$ for the Euler discretisation of the heat equation to a nonlinear problem (Theorem 4.2). This result reflects the dependence upon initial conditions required in the nonlinear case; the precise form of the restriction for the nonlinear problem is intimately related to the existence of spurious periodic solutions in the discretisation.

Acknowledgements. The authors thank both referees, whose comments have improved the presentation. A.M. Stuart is grateful to A.R. Mitchell and C.M. Elliott for several helpful discussions.

REFERENCES

[1] F. Brezzi, S. Ushiki, and H. Fujii, *Real and ghost bifurcation dynamics in difference schemes for ordinary differential equations*, in Numerical Methods for Bifurcation Problems, T. Kupper, H. D. Mittlemann, and H. Weber, eds., Birkhäuser-Verlag, Boston, 1984.

[2] N. Chafee and E. Infante, *A bifurcation problem for a nonlinear parabolic problem*, J. Applic. Anal., 4 (1974), pp. 17–37.

[3] S.-N. Chow and J. Hale, *Methods of Bifurcation Theory*, Springer-Verlag, Berlin, New York, 1982.

[4] S. D. Conte and C. de Boor, *Elementary Numerical Analysis*, Third Edition, McGraw-Hill, New York, 1980.

[5] D. F. Griffiths, *The dynamics of some linear multistep methods with step-size control*, in Numerical Analysis, D. F. Griffiths and G. A. Watson, eds., Pitman, Boston, 1987.

[6] D. F. Griffiths and A. R. Mitchell, *Stable periodic bifurcations of an explicit discretisation of a nonlinear partial differential equation in reaction diffusion*, IMA J. Numer. Anal., 8 (1988), pp. 435–454.

[7] J. Guckenheimer and P. Holmes, *Nonlinear Oscillations, Dynamical Systems and Bifurcations of Vector Fields*, in Applied Mathematical Sciences, 42, Springer-Verlag, Berlin, New York, 1983.

[8] E. Hairer, A. Iserles, and J. M. Sanz-Serna, *Equilibria of Runge–Kutta methods*, Numer. Math., to appear.

[9] A. R. Humphries, *Numerical methods for ordinary differential equations and dynamical systems*, Mathematical Tripos, Part III, University of Cambridge, Cambridge, 1990.

[10] A. R. Mitchell and D. F. Griffiths, *Beyond the linearised stability limit in nonlinear problems*, in Numerical Analysis, D. F. Griffiths and G. A. Watson, eds., Research Notes in Mathematics, 140, Pitman, Boston, 1986.

[11] A. R. Mitchell and S. W. Schoombie, *Nonlinear diffusion and stable period 2 solutions of a discrete reaction-diffusion model*, J. Comput. Appl. Math., 25 (1989), pp. 363–372.

[12] T. Murdoch and C. J. Budd, *Convergent and spurious solutions of nonlinear elliptic equations*, in preparation.

[13] A. C. Newell *Finite amplitude instabilities of partial difference equations*, SIAM J. Appl. Math., 33 (1977), pp. 133–160.

[14] A. Iserles, A. T. Peplow, and A. M. Stuart, *A unified approach to spurious solutions introduced by time-discretisation*, Part I, SIAM J. Numer. Anal., 28 (1991), to appear.

[15] P. H. Rabinowitz, *Some global results for nonlinear eigenvalue problems*, J. Funct. Anal., 7 (1971), pp. 487–513.

[16] W. C. Rheinboldt, *Numerical Analysis of Parameterised Nonlinear Equations*, Wiley-Interscience, New York, 1986.

[17] S. W. SCHOOMBIE AND A. R. MITCHELL, *Numerical studies of a discretised version of Fisher's equation beyond the linearised stability limit*, SIAM J. Sci. Statist. Comput., submitted.

[18] R. SHREIBER AND H. B. KELLER, *Spurious solutions of the driven cavity calculations*, J. Comput. Phys., 49 (1983), pp. 165–172.

[19] B. D. SLEEMAN, D. F. GRIFFITHS, A. R. MITCHELL, AND P. D. SMITH, *Stable periodic solutions in nonlinear difference equations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 543–557.

[20] A. B. STEPHENS AND G. R. SHUBIN, *Multiple solutions and bifurcations of finite difference approximations to some steady problems of fluid mechanics*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 404–415.

[21] A. M. STUART *Nonlinear instability in dissipative finite difference schemes*, SIAM Rev., 31 (1989), pp. 191–220.

[22] ———, *Linear instability implies spurious periodic solutions*, IMA J. Numer. Anal., 9 (1989), pp. 465–486.

[23] ———, *The global attractor under discretisation*, in Continuation and Bifurcations: Numerical Techniques and Applications, D. Roose, B. De Dier, and A. Spence, eds., NATO ASI Series, Kluwer, the Netherlands, 1990.

[24] A. M. STUART AND M. S. FLOATER, *On the computation of blow-up*, European J. Appl. Math., 1 (1990), pp. 47–71.

[25] J. F. TOLAND, *Asymptotic linearity and nonlinear eigenvalue problems*, Quart. J. Math., 24 (1973), pp. 241–250.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1373–1400, November 1991
© 1991 Society for Industrial and Applied Mathematics
009

THE NESTED RECURSIVE TWO-LEVEL FACTORIZATION METHOD FOR NINE-POINT DIFFERENCE MATRICES*

OWE AXELSSON† AND VICTOR EIJKHOUT‡

Abstract. Nested recursive two-level factorization methods for nine-point difference matrices are analyzed. Somewhat similar in construction to multilevel methods for finite element matrices, these methods use recursive red-black orderings of the meshes, approximating the nine-point stencils by five-point ones in the red points and then forming the reduced system explicitly. Because this Schur complement is again a nine-point matrix (on a skew grid this time), the process of approximating and factorizing can be applied anew. Progressing until a sufficiently coarse grid has been reached, this procedure gives a multilevel preconditioner for the original matrix. Solving the levels in V-cycle order will not give an optimal order method (that is, with a total work proportional to the number of unknowns), but we show that using certain combinations of V-cycles and W-cycles will give methods of both optimal order of numbers of iterations and computational complexity.

Since all systems to be solved during a preconditioner solve are of diagonal form, the method is suitable for execution on massively parallel architectures.

Key words. preconditioning, multilevel method, recursive two-by-two level, approximate factorization

AMS(MOS) subject classifications. 65F10, 65N30

1. Introduction.

Recently the first author and Vassilevski [6], [7] have derived and analyzed algebraic multilevel iteration methods for finite element matrices, in particular for piecewise linear approximations. In the present paper we propose somewhat similar methods for nine-point difference matrices. Because nine-point difference approximations can have a higher degree of approximation, while being just as easy to construct as five-point approximations, it is of interest to have preconditioners based directly on the nine-point matrix. In fact, the method proposed in this paper is also directly applicable to five-point difference matrices.

The methods presented here differ from the methods proposed earlier in that, apart from the top level, they do not use coefficient matrices derived from the differential operator. Instead, the Schur complement system of the previous level is used. This is possible because the factorization proposed ensures recursively a nine-point structure on each level. Namely, if a red-black ordering is imposed on the grid, and the nine-point stencil in the red points is suitably modified to a five-point one, elimination of these red points will give a skew nine-point stencil on the black points. A fundamental property of this procedure is that the red points become mutually uncoupled, thus facilitating the solution of systems, and, indeed, enabling the explicit formation of the Schur complement.

Continuing this recursive factorization process until a sufficiently coarse grid is obtained, we obtain a multilevel preconditioner which can be used in a conjugate gradient or other iterative method. For systems on the coarse grid one can for instance use a direct solution method. However, using the preconditioner as a simple V-cycle (descending to the coarsest grid during the forward solve, and ascending again to the finest grid in the back substitution) will not give spectral equivalence to the origi-

* Received by the editors January 16, 1990; accepted for publication (in revised form) December 28, 1990.

† Faculty of Mathematics and Informatics, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, the Netherlands.

‡ Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, 104 S. Wright St., Urbana, Illinois 61801.

nal matrix. We show that the relative condition number grows with the meshsize h as $O(h^{-q})$ for some positive q. Therefore, we use the nested polynomial approximations proposed earlier. Each visit to a level then entails multiple visits to the next coarser level, thereby making the preconditioner into a general W-cycle.

More precisely we show that for polynomial degree $\nu = 2$ the relative condition number is $O(1)$, but as the number of arithmetic operations grows linearly with the number of levels, the result is a method with computational complexity of order $O(h^{-2}\log h^{-1})$. Using alternately first and second (or even third) degree polynomials, we show that a method of optimal order of computational complexity results.

The optimality or close-to-optimality is also valid for problems with anisotropy, if the coefficients differ by a factor sufficiently close to 1. The bilinear approximation then permits the ratio of the coefficients to be between $\frac{1}{2}$ and 2, while the standard nine-point discretization allows values between $\frac{1}{5}$ and 5. For other ranges the method will converge, but not with optimal order in general.

Earlier use of similar red-black orderings for five-point difference matrices can be found in [5]. The use of "intermediate" (i.e., skewed) grids has been described in [17] and in [9] and [15]. A method similar to the present one, but for finite element meshes using bisections of triangles, has been analyzed in [2]. A first description of five-point/nine-point methods, but without analysis of the influence of polynomials on the condition number, and not based on explicit use of the Schur complement, can be found in [1].

Besides the multilevel methods mentioned above in which standard finite element basis functions are used, there is the multilevel method using hierarchical basis functions [18]. However, in this method there is no use of nested polynomials, and the condition number grows slowly ($O(\log h)$) for two-dimensional, or more rapidly ($O(h^{-1})$) for three-dimensional elliptic problems. A method based on Schur complements and several mesh levels has been used in [12]. Here also the nested recursive definition of the preconditioners, which is essential for an optimal order method, was not used.

This paper proceeds as follows. In §2 we present the preconditioner in matrix form; in §3 we derive the actual coefficients involved. Section 4 gives an element matrix analysis of the preconditioner to arrive at the value of a certain series of constants μ_k. A global analysis of the condition number is then given in §5. Here we will show that a condition number of $O(1)$ is attainable. Section 6 concerns itself with the computational complexity of the method. The result here is that for methods with a condition number of $O(1)$ an optimum number of operations can also be attained. Finally, §7 gives numerical tests and discussion.

2. Construction of the level k coefficient and preconditioning matrices. The nested recursive factorization method resembles in some respects recursive two-level methods proposed in [6], [7]. It differs in the fact that, whereas the two-level methods use a finite element stiffness matrix on every level, the nested recursive factorization uses on each level the Schur-complement of the previous (finer) level to generate the coefficient matrix on that level.

The nested recursive factorization can be described formally (that is, in terms of matrices, and without reference to stencils) as follows. Let some symmetric positive definite coefficient matrix $A^{(p)}$ be given. Let for some value $k \leq p-1$ the matrix

$$A^{(k+1)} = \begin{pmatrix} D & C^t \\ C & E \end{pmatrix}$$

(where D and E are symmetric and positive definite) be the coefficient matrix on level $k+1$. The coefficient matrix $A^{(k)}$ on the next level is formed as an approximation of the Schur complement

$$S^{(k)} = E - CD^{-1}C^t$$

in such a way that for all vectors u there exist a $\mu_k \in (0,1)$ such that

$$0 < \mu_k u^t S^{(k)} u \leq u^t A^{(k)} u \leq u^t S^{(k)} u. \tag{1}$$

The exact mechanics of this approximation are discussed in the next section. This recursive factorization is pursued until we arrive at a sufficiently coarse level, say, level 0, where direct solution with $A(k)$ or $S(k)$ becomes feasible.

We precondition $A^{(k+1)}$ by

$$M^{(k+1)} = \begin{pmatrix} D & 0 \\ C & Z^{(k)} \end{pmatrix} \begin{pmatrix} I & D^{-1}C^t \\ 0 & I \end{pmatrix}, \tag{2}$$

where I stands generically for an identity matrix of the proper order, and $Z^{(k)}$ is an approximation to the Schur complement $S^{(k)}$, defined by

$$\begin{aligned} {Z^{(k)}}^{-1} &= (I - P_\nu({M^{(k)}}^{-1} S^{(k)})){S^{(k)}}^{-1} && \text{version (i)} \\ {Z^{(k)}}^{-1} &= (I - P_\nu({M^{(k)}}^{-1} A^{(k)})){A^{(k)}}^{-1} && \text{version (ii).} \end{aligned} \tag{3}$$

Here $P_\nu = P_{\nu_k}^{(k)}$ is a polynomial of degree $\nu = \nu_k$ where the order zero term has coefficient 1. We will specify the choice of the polynomials in §5; subscripts and superscripts k to indicate level-dependencies will be omitted when they are not needed.

We will see later on that, if we aim at preconditioned solution of a nine-point system, the matrix on the finest mesh is an $S^{(k+1)}$-matrix. Alternatively, one could start out with a mixed five-/nine-point system. The matrix on the finest mesh is then an $A^{(k+1)}$-matrix.

3. Recursive derivation of the level k coefficients. In the previous section we did not mention the stencils involved in the $S^{(k)}$ and $A^{(k)}$ matrices; in particular, we left unspecified how the approximation of the Schur complement system $S^{(k)}$ by $A^{(k)}$ is performed. This will be discussed now.

We may consider on each level a nine-point matrix $S^{(k)}$ to be given. A standard red-black structure is then imposed on the grid, the nine-point stencils are modified to five-point stencils in the red points, and are left untouched in the black points. The resulting matrix is called $A^{(k)}$. In the transition from this level to the next, the red points are eliminated; as we will show below, in the resulting Schur complement system $S^{(k-1)}$ the black points are then connected by a nine-point stencil but on a skew grid. Note that this Schur complement system is derived from *exact* factorization of $A^{(k)}$; factorization of $S^{(k)}$ would give rise to fill-in outside the nine-point stencil on the skew grid.

The coefficient matrix $A^{(k-1)}$ on the grid of the black points is again derived by imposing a red-black structure on these points, and modifying the stencils in the new red points in order to obtain five-point connections. Eliminating the red points will then give nine-point stencils on a horizontal/vertical grid, which is the same structure that we had at the outset, but with different coefficients, and on a double distance grid.

Suppose that we start off in the black points with a nine-point molecule composed of s times $(-\Delta_5^{(h,+)})$ plus t times $(-\Delta_5^{(h,\times)})$, where s and t are positive numbers and

$$-\Delta_5^{(h,+)} = \frac{1}{h^2}\begin{Bmatrix} \cdot & -1 & \cdot \\ -1 & 4 & -1 \\ \cdot & -1 & \cdot \end{Bmatrix}; \qquad -\Delta_5^{(h,\times)} = \frac{1}{(\sqrt{2}h)^2}\begin{Bmatrix} -1 & \cdot & -1 \\ \cdot & 4 & \cdot \\ -1 & \cdot & -1 \end{Bmatrix}$$

with grid spacing h in both stencils are the axi-parallel and skew five-point difference molecules, i.e., our initial molecule in the black points is

$$\frac{1}{h^2}\begin{Bmatrix} -t/2 & -s & -t/2 \\ -s & 4s+2t & -s \\ -t/2 & -s & -t/2 \end{Bmatrix}.$$

The stencil in the red points is then

$$\begin{matrix} \cdot & -s & \cdot \\ -s & 4s & -s \\ \cdot & -s & \cdot \end{matrix}.$$

Now, in order to eliminate the red points (i.e., the horizontal and vertical neighbours of the black points), we multiply on the horizontal and vertical cross connections by

$$\begin{matrix} \cdot & 1/4 & \cdot \\ 1/4 & 1 & 1/4 \\ \cdot & 1/4 & \cdot \end{matrix}.$$

Such algebra of stencils can already be found in [11]. This gives a resulting (skewed) nine-point stencil

$$\frac{1}{2h^2}\begin{Bmatrix} \cdot & \cdot & -s/2 & \cdot & \cdot \\ \cdot & -(s+t) & \cdot & -(s+t) & \cdot \\ -s/2 & \cdot & 6s+4t & \cdot & -s/2 \\ \cdot & -(s+t) & \cdot & -(s+t) & \cdot \\ \cdot & \cdot & -s/2 & \cdot & \cdot \end{Bmatrix}$$

(or $(s+t)(-\Delta_5^{(h,\times)}) + s(-\Delta_5^{(2h,+)})$) on the black points.

Next we impose a red-black structure on the black points with the black points occupying the positions of an axi-parallel grid of meshwidth $2h$. We then modify the stencil in the red points by eliminating the horizontal/vertical connections and moving them to the central node; that is, we eliminate the $s(-\Delta_5^{(2h,+)})$ term at the red points. Thus we obtain the stencil

$$\frac{1}{2h^2}\begin{Bmatrix} -(s+t) & \cdot & -(s+t) \\ \cdot & 4(s+t) & \cdot \\ -(s+t) & \cdot & -(s+t) \end{Bmatrix}$$

in the red points. Eliminating the red points of the skew grid gives the double distance nine-point molecule

$$\left(\frac{1}{2h}\right)^2\begin{Bmatrix} -(t+s)/2 & \cdot & -(2s+t) & \cdot & -(s+t)/2 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ -(2s+t) & \cdot & 10s+6t & \cdot & -(2s+t) \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ -(t+s)/2 & \cdot & -(2s+t) & \cdot & -(s+t)/2 \end{Bmatrix}$$

or $(2s+t)(-\Delta_5^{(2h,+)}) + (s+t)(-\Delta_5^{(2h,\times)})$. Thus the factorization progresses with stencils

$$\begin{array}{lrcl} \text{on level } k & a_k(-\Delta_5^{(h,+)}) & + & b_k(-\Delta_5^{(h,\times)}) \\ k-1 & a_{k-1}(-\Delta_5^{(h,\times)}) & + & b_{k-1}(-\Delta_5^{(2h,+)}) \\ k-2 & a_{k-2}(-\Delta_5^{(2h,+)}) & + & b_{k-2}(-\Delta_5^{(2h,\times)}) \end{array}$$

and coefficients satisfying

$$a_{k-1} = a_k + b_k, \quad b_{k-1} = a_k \quad \text{for } k = p, p-1, \cdots, 1.$$

We see that (after a normalization) the coefficients satisfy a Fibbonacci series, with the coefficients for the finest mesh (level p) given. In explicit form

$$a_k = c_1\left(\tfrac{1+\sqrt{5}}{2}\right)^{p-k} + c_2\left(\tfrac{1-\sqrt{5}}{2}\right)^{p-k}$$

with

$$c_1 = \frac{1}{2}\left(1+\frac{1}{\sqrt{5}}\right)a_p + \frac{1}{\sqrt{5}}b_p, \qquad c_2 = \frac{1}{2}\left(1-\frac{1}{\sqrt{5}}\right)a_p - \frac{1}{\sqrt{5}}b_p.$$

In the course of the factorization (that is, for $k \downarrow 0$) the quotient b_k/a_k converges as

$$\frac{b_k}{a_k} = \frac{a_{k+1}}{a_k} \to \sigma = \frac{\sqrt{5}-1}{2} \approx 0.618;$$

its maximal value is $a_p/(a_p+b_p)$ if $c_2 > 0$, it is b_p/a_p if $c_2 < 0$. Furthermore, if $b_p = \sigma a_p$, then $b_k/a_k = \sigma$ for all k.

4. Local analysis. In this section we analyze the error that we make by modifying a Schur complement system $S^{(k)}$ to obtain the five-point/nine-point coefficient matrix $A^{(k)}$ on that level. As was shown in the previous section, the Schur complement matrix itself was derived by eliminating the red points in the coefficient matrix $A^{(k+1)}$ one level higher.

To this purpose we construct element matrices that can be associated with the global matrices and analyze their relative condition numbers. For simplicity of presentation we start by treating isotropic problems.

4.1. Isotropic problems. Suppose that on a certain (horizontal/vertical) level we have a coefficient matrix $A^{(k+1)}$ of the type with a nine-point stencil in the black points and a five-point stencil in the red point. If this matrix was derived by discarding the skew connections in the red points, we can write the stencils as follows:

$$\text{black: } \begin{bmatrix} -b/2 & -a & -b/2 \\ -a & 4a+2b & -a \\ -b/2 & -a & -b/2 \end{bmatrix} \qquad \text{red: } \begin{bmatrix} . & -a & . \\ -a & 4a & -a \\ . & -a & . \end{bmatrix}.$$

The Schur complement $S^{(k)}$ obtained after eliminating the red points will then have the nine-point stencil

$$\begin{bmatrix} . & . & -a/4 & . & . \\ . & -(b+a)/2 & . & -(b+a)/2 & . \\ -a/4 & . & 3a+2b & . & -a/4 \\ . & -(b+a)/2 & . & -(b+a)/2 & . \\ . & . & -a/4 & . & . \end{bmatrix}.$$

We can associate local element matrices with the above Schur complement, without any need to actually construct any underlying basisfunctions. Consider then the numbering of the nodes of a local element

$$\begin{array}{ccc} & (1) & \\ (3) & & (4) \\ & (2) & \end{array}$$

on the skew grid of the black points. Each center node has support of four rotated squares, and we find the element matrix

$$S_e = \begin{pmatrix} (3a+2b)/4 & -a/4 & -(b+a)/4 & -(b+a)/4 \\ -a/4 & (3a+2b)/4 & -(b+a)/4 & -(b+a)/4 \\ -(b+a)/4 & -(b+a)/4 & (3a+2b)/4 & -a/4 \\ -(b+a)/4 & -(b+a)/4 & -a/4 & (3a+2b)/4 \end{pmatrix}$$

by the reverse process of that by which we usually construct global matrices from element matrices.

Letting the points 3 and 4 be the red points of the next level, we equip them with a five-point stencil

$$\begin{array}{ccc} -(b+a)/2 & . & -(b+a)/2 \\ . & 2a+2b & . \\ -(b+a)/2 & . & -(b+a)/2 \end{array}$$

and we eliminate the couplings between the red points. For the resulting coefficient matrix on this level we find an element matrix

$$A_e = \begin{pmatrix} (3a+2b)/4 & -a/4 & -(b+a)/4 & -(b+a)/4 \\ -a/4 & (3a+2b)/4 & -(b+a)/4 & -(b+a)/4 \\ -(b+a)/4 & -(b+a)/4 & (2a+2b)/4 & 0 \\ -(b+a)/4 & -(b+a)/4 & 0 & (2a+2b)/4 \end{pmatrix}.$$

The zeros in positions $(3,4)$ and $(4,3)$ are due to the uncoupling of the red points 3 and 4.

Now we want to solve the generalized eigenvalue problem

$$S_e \begin{pmatrix} x \\ y \end{pmatrix} = \lambda A_e \begin{pmatrix} x \\ y \end{pmatrix}.$$

Since S_e and A_e have the same nullspace, the eigenvalue problem is understood as finding the solutions in the complementary space to the nullspace. For this we partition the matrices into 2×2 blocks and make a block factorization

$$S_e = \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} = \begin{pmatrix} S_{11} & 0 \\ S_{21} & I \end{pmatrix} \begin{pmatrix} I & S_{11}^{-1} S_{12} \\ 0 & S_{22} - S_{21} S_{11}^{-1} S_{12} \end{pmatrix},$$

$$A_e = \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} S_{11} & 0 \\ S_{21} & I \end{pmatrix} \begin{pmatrix} I & S_{11}^{-1} S_{12} \\ 0 & A_{22} - S_{21} S_{11}^{-1} S_{12} \end{pmatrix}$$

and we find that the solution to the generalized eigenvalue problem satisfies:

1. $\lambda = 1$ and $S_{22} y = A_{22} y$, or
2. $x = -S_{11}^{-1} S_{12} y$ and $(S_{22} - S_{21} S_{11}^{-1} S_{12}) y = \lambda (A_{22} - S_{21} S_{11}^{-1} S_{12}) y$, where y is not in the nullspace of the two matrices.

An elementary computation reveals that the vector $y = \binom{1}{1}$ satisfies $S_{22}y = A_{22}y$, so the generalized eigenvalue problem in the second point above simplifies to

$$\frac{2a+b}{a+b}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \lambda\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix},$$

where $y_1 \neq y_2$. Hence, the eigenvalues to the generalized eigenproblem are

$$\lambda = 1 \quad \text{(twice)} \qquad \text{and} \qquad \lambda = \frac{2a+b}{a+b}, \tag{4}$$

the latter for a vector with $y_1 \neq y_2$.

Changing a nine-point matrix into a five-point matrix by the process of moving elements to the central node corresponds to subtracting a certain positive semidefinite matrix, so from the positiveness of a and b at every level we conclude that the coefficient matrix $A^{(k)}$ and the Schur complement matrix $S^{(k)}$ derived from the previous level satisfy

$$\text{for all } x: \quad \mu_k x^t S^{(k)} x \leq x^t A^{(k)} x \leq x^t S^{(k)} x \tag{5}$$

where 4 shows that $\mu_k = (a+b)/(2a+b)$; that is, $\mu_k \in (0.5, 1)$ for all levels $k < p$.

The estimates based on element matrices can be used for the global matrix since

$$x^t A x = x^t \left(\sum_{\text{elements}} A_e^{(k)} \right) x$$

if the global matrix is a Toeplitz form, or if we can identify the element that will give an upper bound for all elements.

Note that if the coefficient matrix on the finest level was a Toeplitz matrix, the matrices on the coarser levels need not be so. A simple calculation, however, shows that after an elimination step, for the elements along the boundary of the domain an even sharper bound than (4) holds. Therefore, inequalities of the form (5) hold on all levels of the factorization of a Toeplitz form, and the quantity μ_k can be computed locally.

In fact, similar to the corresponding analysis in [6], [7], we find that if we construct the nine-point stencil recursively from an initial coarse grid to finer levels, we can even permit the coefficients of the underlying elliptic differential equation to be discontinuous on the elements of the coarsest grid. The quantity μ_k can still be computed locally. The relation between μ_k and the parameter γ_k used in [6], [7], is $\mu_k = 1 - \gamma_k^2$.

Hence, as an example, they are valid for the nine-point discretization of $-\underline{\nabla}\rho\underline{\nabla}u = f$ in Ω, where Ω consists of four rectangular boxes with $\rho = \rho_i$, $i = 1, 2, 3, 4$.

ρ_1	ρ_2
ρ_3	ρ_4

In particular, they are valid for $\rho_4 = \infty$, which corresponds to a problem on an L-shaped domain, with Neumann-type boundary conditions in the interior boundary part of the "L."

Equation (5) could also have been derived in the following way. Consider the original nine-point matrix and its approximation by using a five-point stencil in the red points. From the generalized eigenvalue problem for the corresponding element matrices we find in this case

$$\mu_k = a_p/(a_k + b_k) = a/(a+b).$$

The relation between the entries in the element matrices for the original mesh and the next, skew, mesh is

$$b \leftarrow a/2, \qquad a \leftarrow (b+a)/2.$$

Hence

$$\mu_{k-1} = \frac{(b+a)/2}{(b+a)/2 + a/2} = \frac{b+a}{b+2a}$$

which gives (4). Therefore, using the recurrence relations

$$a_{k-1} = a_k + b_k, \qquad b_{k-1} = a_k$$

we find

$$\mu_k = \frac{1}{1+\mu_{k-1}}, \qquad k = p, p-1, \cdots, 1$$

which shows that the μ_k converge to the positive solution of the equation

$$\mu^2 + \mu - 1 = 0 \Rightarrow \mu = \sigma \approx 0.618.$$

We will now calculate the value of μ_k for the finest mesh (that is, at level $k = p$) for some nine-point stencils.

For the standard nine-point difference stencil (which is of fourth order of approximation after suitable modification of the right-hand side)

$$\begin{array}{ccc} -1 & -4 & -1 \\ -4 & 20 & -4 \\ -1 & -4 & -1 \end{array}$$

we have

$$\left.\begin{array}{l} a = 4 \\ b = 2 \end{array}\right\} \Rightarrow \mu_p = \frac{4}{4+2} = \frac{2}{3},$$

which is the largest μ_k in this case. For the nine-point molecule arising from bilinear finite elements

$$\begin{array}{ccc} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{array}$$

we find

$$\left.\begin{array}{l} a = 1 \\ b = 2 \end{array}\right\} \Rightarrow \mu_p = \frac{1}{3},$$

which is the smallest μ_k in this case: on all coarser levels $\mu_k \geq .5$.

4.2. Anisotropic problems. In the analysis of the previous section, in particular in the derivation of the inequalities in (5), the fact that the operator stencil was of positive type was of crucial importance. Unfortunately, nine-point stencils for the anisotropic elliptic selfadjoint differential equation

$$\mathcal{L}u = -\epsilon_1 u_{xx} - \epsilon_2 u_{yy}$$

do not always give difference approximations of positive type. For instance, bilinear finite elements will give a stencil

$$\begin{matrix} -(\epsilon_1+\epsilon_2)/2 & -(2\epsilon_2-\epsilon_1) & -(\epsilon_1+\epsilon_2)/2 \\ -(2\epsilon_1-\epsilon_2) & 4(\epsilon_1+\epsilon_2) & -(2\epsilon_1-\epsilon_2) \\ -(\epsilon_1+\epsilon_2)/2 & -(2\epsilon_2-\epsilon_1) & -(\epsilon_1+\epsilon_2)/2 \end{matrix},$$

which is not of positive type if the coefficients differ by more than a factor of 2. The general form for the fourth order (again, assuming a suitable modification of the right-hand side) nine-point box can be derived by noting that

$$\epsilon_1 u_{xx} + \epsilon_2 u_{yy} = \frac{\epsilon_1+\epsilon_2}{2}(u_{xx}+u_{yy}) + \frac{\epsilon_1-\epsilon_2}{2}u_{xx} + \frac{\epsilon_2-\epsilon_1}{2}u_{yy}$$

and using the nine-point difference scheme for the first term, and central differences for the second and third; the resulting stencil (multiplied by 6) is then

$$\begin{matrix} -(\epsilon_1+\epsilon_2)/2 & -(5\epsilon_2-\epsilon_1) & -(\epsilon_1+\epsilon_2)/2 \\ -(5\epsilon_1-\epsilon_2) & 10(\epsilon_1+\epsilon_2) & -(5\epsilon_1-\epsilon_2) \\ -(\epsilon_1+\epsilon_2)/2 & -(5\epsilon_2-\epsilon_1) & -(\epsilon_1+\epsilon_2)/2 \end{matrix}, \tag{6}$$

which is only of positive type if the coefficients differ by at most a factor of 5.

To get a difference approximation of positive type we therefore modify the stencil when $5\epsilon_2 - \epsilon_1 < 0$ (or when $5\epsilon_1 - \epsilon_2 < 0$) by moving the off-diagonal positive entries to the diagonal:

$$\begin{matrix} -(\epsilon_1+\epsilon_2)/2 & 0 & -(\epsilon_1+\epsilon_2)/2 \\ -(5\epsilon_1-\epsilon_2) & 12\epsilon_1 & -(5\epsilon_1-\epsilon_2) \\ -(\epsilon_1+\epsilon_2)/2 & 0 & -(\epsilon_1+\epsilon_2)/2 \end{matrix} \tag{7}$$

and proceed with this as described before.

For the analysis of the spectral relation between the Schur complements and the approximations thereof for a general scheme of anisotropic type, we start with a stencil

$$\begin{matrix} -b/2 & -c & -b/2 \\ -a & 2(a+b+c) & -a \\ -b/2 & -c & -b/2 \end{matrix}. \tag{8}$$

The Schur complement obtained after eliminating the red points (where we use

$$\begin{matrix} & -c & \\ -a & 2(a+c) & -a \\ & -c & \end{matrix}$$

for the five-point stencil) has the skew nine-point stencil

$$\begin{matrix} & & -\frac{c^2}{2(a+c)} & & \\ & -\frac{b}{2}-\frac{ac}{a+c} & \cdot & -\frac{b}{2}-\frac{ac}{a+c} & \\ -\frac{a^2}{2(a+c)} & \cdot & 4(a+c+b)(a+c) & \cdot & -\frac{a^2}{2(a+c)} \\ & -\frac{b}{2}-\frac{ac}{a+c} & -2a^2-2c^2 & -\frac{b}{2}-\frac{ac}{a+c} & \\ & & \cdot & & \\ & & -\frac{c^2}{2(a+c)} & & \end{matrix}.$$

Corresponding to this stencil we get two kinds of element matrices for square elements: those lying to the left or right of the central node, and those lying over or under it. For an element to the right of the central node, the element matrix and modified element matrix (where a five-point stencil is used in the red points) are

$$S_e = \begin{pmatrix} \frac{b}{2} + \frac{a^2}{2(a+c)} & -\frac{a^2}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} \\ -\frac{a^2}{2(a+c)} & \frac{b}{2} + \frac{a^2}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} \\ -\frac{b}{4} - \frac{ac}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} & \frac{b}{2} + \frac{c^2}{2(a+c)} & -\frac{c^2}{2(a+c)} \\ -\frac{b}{4} - \frac{ac}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} & -\frac{c^2}{2(a+c)} & \frac{b}{2} + \frac{c^2}{2(a+c)} \end{pmatrix},$$

$$A_e = \begin{pmatrix} \frac{b}{2} + \frac{a^2}{2(a+c)} & -\frac{a^2}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} \\ -\frac{a^2}{2(a+c)} & \frac{b}{2} + \frac{a^2}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} \\ -\frac{b}{4} - \frac{ac}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} & \frac{b}{2} + \frac{ac}{a+c} & 0 \\ -\frac{b}{4} - \frac{ac}{2(a+c)} & -\frac{b}{4} - \frac{ac}{2(a+c)} & 0 & \frac{b}{2} + \frac{ac}{a+c} \end{pmatrix};$$

the other matrices are the same, but with a and c interchanged. Solving the generalized eigenvalue problem

$$S_e x = \lambda A_e x$$

orthogonal to the null space of the matrices, results in eigenvalues $\lambda = 1$ (twice) and

$$\lambda = \max \left\{ \frac{2a+b}{\frac{2ac}{a+c}+b}, \frac{2c+b}{\frac{2ac}{a+c}+b} \right\}.$$

Hence, assuming that a, b, and c are nonnegative, we find that the new coefficient matrix $A^{(k)}$ and the Schur complement matrix $S^{(k)}$ are related by

$$\text{for all } x\text{: } \mu_k x^t S^{(k)} x \leq x^t A^{(k)} x \leq x^t S^{(k)} x, \tag{9}$$

where

$$\mu_k = \min \left\{ \frac{\frac{2ac}{a+c}+b}{2a+b}, \frac{\frac{2ac}{a+c}+b}{2c+b} \right\}. \tag{10}$$

For the nine-point scheme (6), substituting $b = \epsilon_1 + \epsilon_2$, $a = 5\epsilon_1 - \epsilon_2$, $c = 5\epsilon_2 - \epsilon_1$, (10) shows that if $1/5 \leq \epsilon_1/\epsilon_2 \leq 5$ holds, then $\mu_k \geq 1/9$.

For the modified nine-point scheme (7) we obtain with $b = \epsilon_1 + \epsilon_2$, $a = 5\epsilon_1 - \epsilon_2$, $c = 0$ that $\mu_k \geq 1/11$ for all values of ϵ_1 and ϵ_2. Unfortunately, the results in §5 show that an optimal, or nearly optimal, method requires $\mu_k \geq 1/4$.

A similarly modified bilinear scheme gives, with $a = 2\epsilon_1 - \epsilon_2$, $b = \epsilon_1 + \epsilon_2$, $c = 0$, a value of

$$\mu_k = \frac{1}{5} \frac{1 + \epsilon_2/\epsilon_1}{1 - \epsilon_2/5\epsilon_1},$$

so $\mu_k > 1/5$ for all values of ϵ_1 and ϵ_2. In case $\epsilon_2/\epsilon_1 > 1/5$ we have $\mu_k > 1/4$.

Next we have to investigate how the value of μ changes during the factorization. Consider then a skew nine-point stencil

$$\begin{array}{ccc} & -c & \\ -b & \cdot & -b \\ -a \quad \cdot & 2(a+2b+c) & \cdot \quad -a\,; \\ -b & \cdot & -b \\ & -c & \end{array}$$

after equipping the red points (that is, the cross connections) with a five-point stencil and eliminating them, we obtain

$$\begin{array}{ccccc} -b/4 & . & -c-b/2 & . & -b/4 \\ \cdot & & \cdot & & \cdot \\ -a-b/2 & . & 2a+2c+3b & . & -a-b/2\,. \\ \cdot & & \cdot & & \cdot \\ -b/4 & . & -c-b/2 & . & -b/4 \end{array}$$

For the analysis, we assume, in terms of the quantities a, b, and c of 8, that $c \geq a$. Now note that

$$\mu^{-1} = \max\{1+\delta_a, 1+\delta_c\},$$

where

$$\begin{aligned} \delta_a &= \frac{2a^2}{2ac+ab+bc} = \frac{2}{2\frac{c}{a}+\frac{b}{a}+\frac{b}{a}\frac{c}{a}} \\ \delta_c &= \frac{2c^2}{2ac+ab+bc} = \frac{2}{2\frac{a}{c}+\frac{b}{c}+\frac{b}{c}\frac{a}{c}}. \end{aligned}$$

Two steps of the factorization, that is, going from one level to one with double mesh-size, transform the quantity c/a as follows:

$$\frac{c}{a} \mapsto \frac{c(a+c)+(ab+bc)}{a(a+c)+(ab+bc)} \in [1, c/a],$$

so both δ_a and δ_c may increase during the factorization, but they remain bounded uniformly in the initial coefficients. In fact, in practice we often see them converging rather rapidly to 1.

The analysis for the case $a > c$ follows by interchanging a and c, and we can perform a similar analysis to the above for the spectral condition bounds on the skew levels.

Another approach to obtaining nine-point schemes of positive type would be imaginable, namely, to start out with the standard five-point difference scheme and let the method be based on the nine-point scheme of the Schur complement after elimination of the red points. For anisotropic problems, however, one then finds (with $a = \epsilon_1$, $c = \epsilon_2$, and $b = 0$) that on the finest mesh

$$\mu = \min\left\{\frac{c}{a+c}, \frac{a}{a+c}\right\} = \frac{\min\{\epsilon_1, \epsilon_2\}}{\epsilon_1+\epsilon_2}.$$

Hence in this case the lower eigenvalue bound is not uniform in ϵ_1 and ϵ_2.

5. Estimation of the condition number. In this section we analyze the relative condition numbers of the coefficient matrix $A^{(k)}$ and the Schur complement matrix $S^{(k)}$ to the preconditioner $M^{(k)}$ on the kth level. This is done by comparing the condition numbers on two consecutive levels and performing a limit analysis. Ultimately, we are interested in the condition number of $M^{(p)}$ relative to either $A^{(p)}$ or $S^{(p)}$. As the latter two are related by the simple inequalities (1), we can look at either.

We show that if the polynomial P_ν in the preconditioner is taken as a linear polynomial, the relative condition number increases with the meshsize h as h^{-q}, where $q = \log_2 \sigma^{-1}, \sigma \approx 0.618$. If a higher degree polynomial is taken, the condition number can be reduced to $O(1)$.

However, as only half of the points per level are eliminated, such higher degree polynomials do not give a preconditioner of an optimal order in the number of arithmetic operations. Therefore we also consider preconditioners alternately taking polynomials of first and higher degrees; this will again lead to an $O(1)$ condition number. In §6 it is shown that such preconditioners have an optimal order of the number of arithmetic operations for polynomial degrees 2 and 3.

5.1. Analysis of optimal order. We begin by recalling that the preconditioner $M^{(k+1)}$ satisfies

$$M^{(k+1)} = \begin{pmatrix} D & C^t \\ C & Z^{(k)} + CD^{-1}C^t \end{pmatrix} = A^{(k+1)} + \begin{pmatrix} 0 & 0 \\ 0 & Z^{(k)} - S^{(k)} \end{pmatrix} \tag{11}$$

where

$$S^{(k)} = E - CD^{-1}C^t$$

and where $Z^{(k)}$ is defined as

$$Z^{(k)^{-1}} = (I - P_\nu(M^{(k)^{-1}} S^{(k)}))S^{(k)^{-1}}$$

for version (i), and

$$Z^{(k)^{-1}} = (I - P_\nu(M^{(k)^{-1}} A^{(k)}))A^{(k)^{-1}}$$

for version (ii). At the lowest level $k = 0$ (i.e., the coarsest mesh) we let

$$A^{(0)} = M^{(0)}.$$

The polynomials P_ν can have both their degree and their coefficients defined per level; they are chosen to have

$$\text{for all } x \in I_k: \qquad 0 \le P(x) < 1,$$

where the interval I_k is defined as

$$\text{for version (i):} \qquad I_k = \left[\inf_u \frac{u^t S^{(k)} u}{u^t M^{(k)} u}, \sup_u \frac{u^t S^{(k)} u}{u^t M^{(k)} u} \right],$$

$$\text{for version (ii):} \qquad I_k = \left[\inf_u \frac{u^t A^{(k)} u}{u^t M^{(k)} u}, \sup_u \frac{u^t A^{(k)} u}{u^t M^{(k)} u} \right].$$

Also we assume that P_ν is chosen to have at least one zero in I_k.

For the further analysis we need the following basic lemma.

LEMMA 5.1. *The following relations between coefficient matrices and Schur complement matrices hold.*

1. *For all* v

$$0 < \mu_k v^t S^{(k)} v \le v^t A^{(k)} v \le v^t S^{(k)} v.$$

2. *For all* $v = (v_1, v_2)$ *partitioned consistently with* $A^{(k+1)}$

$$v^t A^{(k+1)} v \ge v_2^t S^{(k)} v_2$$

and equality is attainable.

Proof. Part 1 is (1) carried over from the element matrices to the global matrix; the value of μ_k was derived in (5). Part 2 follows by noting that

$$\begin{aligned} v^t A^{(k+1)} v &= (v_1 - D^{-1}C^t v_2)^t D (v_1 - D^{-1}C^t v_2) + v_2^t (E - CD^{-1}C^t) v_2 \\ &\ge v_2^t S^{(k)} v_2 \end{aligned}$$

with equality if $v_1 = D^{-1}C^t v_2$. □

The definition of the polynomial implies the following a priori bounds on the relative condition of $S^{(k)}$ and $Z^{(k)}$:

LEMMA 5.2. *For version* (i) *of the method*

$$\text{for all } u: \quad \frac{u^t S^{(k)} u}{u^t Z^{(k)} u} \in (0, 1]; \tag{12}$$

for version (ii) *of the method*

$$\text{for all vectors } u: \quad \frac{u^t S^{(k)} u}{u^t Z^{(k)} u} \in (0, \mu_k^{-1}], \tag{13}$$

with the lower and upper bounds satisfying

$$\inf_u \frac{u^t S^{(k)} u}{u^t Z^{(k)} u} \le 1 \le \sup_u \frac{u^t S^{(k)} u}{u^t Z^{(k)} u}.$$

Proof. The assertions for version (i) follow trivially from the definition of the polynomial. In order to obtain the bounds for version (ii), use in addition Lemma 5.1, and note that for block vectors with a zero first block component

$$\frac{u^t S^{(k)} u}{u^t Z^{(k)} u} = \frac{u^t A^{(k)} u}{u^t Z^{(k)} u} \in (0, 1],$$

which implies by the definition of the polynomial that the lower bound does not exceed 1. Also, for any $u \neq 0$

$$\frac{u^t S^{(k)} u}{u^t Z^{(k)} u} = \frac{u^t S^{(k)} u}{u^t A^{(k)} u} \frac{u^t A^{(k)} u}{u^t Z^{(k)} u} \le \mu_k^{-1}.$$

If u is such that $P(u^t A^{(k)} u / u^t M^{(k)} u) = 0$ (and note that we have required P to have a zero in I_k), then we find

$$\frac{u^t S^{(k)} u}{u^t Z^{(k)} u} \ge \frac{u^t A^{(k)} u}{u^t Z^{(k)} u} = 1,$$

which establishes that the upper bound is ≥ 1. □

Investigation of $u^t S^{(k)} u / u^t Z^{(k)} u$ gives us bounds for $u^t A^{(k+1)} u / u^t M^{(k+1)} u$:

LEMMA 5.3. *For both versions of the method*

$$\inf_u \frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} = \inf_{u_2} \frac{u_2^t S^{(k)} u_2}{u_2^t Z^{(k)} u_2},$$

and

$$\textit{for version (i):} \qquad \sup_u \frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} = 1;$$

$$\textit{for version (ii):} \qquad \sup_u \frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} = \sup_{u_2} \frac{u_2^t S^{(k)} u_2}{u_2^t Z^{(k)} u_2}.$$

Proof. For the lower bound let

$$\tau = \inf_u \frac{u^t S^{(k)} u}{u^t Z^{(k)} u},$$

and note that Lemma 5.2 shows that $\tau \leq 1$; using (11), the bound then follows from

$$\begin{aligned} \frac{u^t M^{(k+1)} u}{u^t A^{(k+1)} u} &= 1 + \frac{u^t (M^{(k+1)} - A^{(k+1)}) u}{u^t A^{(k+1)} u} \\ &= 1 + \frac{u_2^t (Z^{(k)} - S^{(k)}) u_2}{u^t A^{(k+1)} u} \leq 1 + (\tau^{-1} - 1) \frac{u_2^t S^{(k)} u_2}{u^t A^{(k+1)} u} \leq \tau^{-1}, \end{aligned} \tag{14}$$

where the last inequality follows from $\tau \leq 1$ and Lemma 5.1.

For the upper bound we have to consider the two versions of the method separately. For version (i), (11) and (12) imply that

$$u^t M^{(k+1)} u = u^t A^{(k+1)} u + u_2^t (Z^{(k)} - S^{(k)}) u_2 \geq u^t A^{(k+1)} u;$$

for version (ii) let

$$\tau = \inf_{u_2} \frac{u_2 Z^{(k)} u_2}{u_2 S^{(k)} u_2},$$

and note that $\tau \leq 1$ by Lemma 5.2. Now we have for all vectors $u = (u_1, u_2)$,

$$(1 - \tau) u^t A^{(k+1)} u \geq (1 - \tau) u_2^t S^{(k)} u_2 \geq u_2^t (S^{(k)} - Z^{(k)}) u_2$$

so

$$u^t M^{(k+1)} u = u^t A^{(k+1)} u + u_2^t (Z^{(k)} - S^{(k)}) u_2 \geq \tau u^t A^{(k+1)} u.$$

Also in this case we have equality for the vectors $u = (u_1, u_2)$ for which $S^{(k)} u_2 = A^{(k+1)} u$. □

LEMMA 5.4. *The following upper bounds hold for the quadratic forms of $A^{(k+1)}$ and $S^{(k+1)}$ relative to $M^{(k+1)}$: for version* (i)

$$\frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} \leq 1; \qquad \frac{u^t S^{(k+1)} u}{u^t M^{(k+1)} u} \leq \mu_{k+1}^{-1},$$

and for version (ii)

$$\frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} \leq \mu_k^{-1}; \qquad \frac{u^t S^{(k+1)} u}{u^t M^{(k+1)} u} \leq \mu_k^{-1} \mu_{k+1}^{-1}.$$

Proof. We proved the bounds for $A^{(k+1)}$ in the previous lemma. The bounds for $S^{(k+1)}$ then follow from Lemma 5.1. □

COROLLARY 5.5. *The interval I_k can be given by*

$$I_k = \left[\inf_u \frac{u^t S^{(k-1)} u}{u^t Z^{(k-1)} u}, \mu_{k-\sigma}^{-1} \right],$$

where $\sigma = 0$ for version (i) *and $\sigma = 1$ for version* (ii).

Proof. Combine Lemmas 5.3 and 5.4, and for the lower bound for version (i) use in addition that

$$\frac{u^t S^{(k)} u}{u^t M^{(k)} u} \geq \inf \frac{u^t S^{(k)} u}{u^t A^{(k)} u} \frac{u^t A^{(k)} u}{u^t M^{(k)} u}$$

for all $u \neq 0$, and Lemma 5.1 (1). □

We will now give the definition of the polynomials $P_{\nu_k}^{(k)}$. The best approximation to zero on the interval I_k among polynomials of degree ν satisfying $0 \leq P_\nu(x) < 1$ and $P_\nu(0) = 1$, is the shifted and scaled Chebyshev polynomial

$$P_{\nu_k}^{(k)}(t) = \frac{1 + T_{\nu_k}\left(\frac{\beta_k + \alpha_k - 2t}{\beta_k - \alpha_k}\right)}{1 + T_{\nu_k}\left(\frac{\beta_k + \alpha_k}{\beta_k - \alpha_k}\right)} \tag{15}$$

where $T_\nu(x) = \frac{1}{2}[(x + \sqrt{x^2 - 1})^\nu + (x - \sqrt{x^2 - 1})^\nu]$ and

$$\alpha_k = \inf_u \frac{u^t S^{(k-1)} u}{u^t Z^{(k-1)} u}$$
$$\beta_k = \begin{cases} \mu_k^{-1} & \text{version (i)} \\ \mu_{k-1}^{-1} & \text{version(ii)} \end{cases}.$$

Remark. For $\nu_k = 1$ the polynomial reduces to

$$P_{\nu_k}^{(k)}(t) = 1 - t/\beta_k = \begin{cases} 1 - \mu_k t \\ 1 - \mu_{k-1} t \end{cases}$$

for versions (i) and (ii), respectively.

LEMMA 5.6. *If we define ϕ_k by*

$$\textit{for version (i):} \qquad \phi_k = \inf_u \frac{u^t S^{(k)} u}{u^t M^{(k)} u},$$
$$\textit{for version (ii):} \qquad \phi_k = \inf_u \frac{u^t A^{(k)} u}{u^t M^{(k)} u},$$

then a lower bound for $u^t S^{(k)} u / u^t Z^{(k)} u$ is given by

$$\frac{u^t S^{(k)} u}{u^t Z^{(k)} u} \geq 1 - P_{\nu_k}^{(k)}(\phi_k)$$

for both versions.

Proof. The assertion follows as

$$\frac{u^t S^{(k)} u}{u^t Z^{(k)} u} = 1 - P_{\nu_k}^{(k)} \left(\frac{\tilde{u}^t S^{(k)} \tilde{u}}{\tilde{u}^t M^{(k)} \tilde{u}} \right) \quad \text{for some } \tilde{u}$$

for version (i), and

$$\frac{u^t S^{(k)} u}{u^t Z^{(k)} u} = \frac{u^t S^{(k)} u}{u^t A^{(k)} u} \left(1 - P_{\nu_k}^{(k)} \left(\frac{\tilde{u}^t A^{(k)} \tilde{u}}{\tilde{u}^t M^{(k)} \tilde{u}} \right) \right) \quad \text{for some } \tilde{u}$$

for version (ii), where Lemma 5.1 has also been used. □

Remark. Note that also for version (i) Lemma 5.3 shows that

$$\text{for all } u: \qquad \frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} \geq \phi_k.$$

We collect the above results into a theorem.

THEOREM 5.7. *The following bounds hold for the quadratic forms of* $A^{(k+1)}$ *and* $S^{(k+1)}$ *relative to* $M^{(k+1)}$*: for version* (i)

$$\frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} \in [1 - P_{\nu_k}^{(k)}(\phi_k), 1]; \qquad \frac{u^t S^{(k+1)} u}{u^t M^{(k+1)} u} \in [1 - P_{\nu_k}^{(k)}(\phi_k), \mu_{k+1}^{-1}],$$

and for version (ii)

$$\frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} \in [1 - P_{\nu_k}^{(k)}(\phi_k), \mu_k^{-1}]; \qquad \frac{u^t S^{(k+1)} u}{u^t M^{(k+1)} u} \in [1 - P_{\nu_k}^{(k)}(\phi_k), \mu_k^{-1} \mu_{k+1}^{-1}],$$

where the quantities ϕ_k *satisfy the recurrence* $\phi_k = 1 - P_{\nu_{k-1}}^{(k-1)}(\phi_{k-1})$.

Note that the above theorem is somewhat independent of the actual choice of the polynomials; the results hold for any polynomial such that its maximum in the interval I_k is taken at the left boundary.

Also it is worthwhile to remark that there is an essential difference in behaviour between polynomials of odd and even degree in that polynomials of even degree are very sensitive to estimates of the upper bound of I_k. If we underestimate the upper bound, in both cases the condition $P_\nu(x) \in [0, 1]$ is violated, but for polynomials of odd degree it gives $P_\nu(x) < 0$ which merely result in eigenvalues of $M^{-1}A$ greater than 1. For even degree polynomials, on the other hand, it gives $P_\nu(x) > 1$, leading to negative eigenvalues, so it may result in divergent methods.

To analyze the spectral condition number λ_k of $M^{(k)^{-1}} A^{(k)}$ for version (i), we note that for $\nu = 1$

$$\lambda_{k+1} = 1/(1 - P_\nu(\phi_k)) = 1/(\phi_k \mu_k) = \mu_k^{-1} \lambda_k,$$

so the condition number on the finest mesh

$$\lambda_p = \prod_{k=0}^{p-1} \mu_k^{-1}.$$

As $\mu_k^{-1} \geq 1 + \tau$ for some positive τ, the condition number grows geometrically with the number of levels. An upper bound for the number of iterations to achieve a certain

relative error ϵ in the norm $\{r^t M^{(p)^{-1}} r\}^{1/2}$, where r is the residual, is found to be (see [3])

$$\#iterations = \frac{1}{2}\sqrt{\lambda_p}\log 2/\epsilon.$$

For the standard nine-point stencil we have

$$\lambda_p \to \frac{1}{3}c_1(\sigma^{-1})^p \qquad (c_1 = (5+2\sqrt{5})/15, \sigma^{-1} \approx 1.618)$$

so

$$\#iterations = O((1/\sigma)^{p/2})\log 2/\epsilon = O(2^{(\log_2 1/\sigma)p/2})\log 2/\epsilon.$$

With $h^{-1} = 2^{p/2}$ we find

$$\#iterations = O(h^{-\log_2 1/\sigma})\log 2/\epsilon, \qquad \text{for } h \to 0,$$

which incidentally shows that the number of iterations grows asymptotically slower than for the incomplete factorization method where the number of iterations grows as $O(h^{-1})$, but faster than for the modified incomplete factorization method (see [3], [8], or [13], for instance) where it is $O(h^{-.5})$.

For version (ii) and for the relative condition of $S^{(k)}$ and $M^{(k)}$ we get a similar analysis, but involving ratios of μ_k's.

Consider now the case $\nu \geq 2$. Actually, from the results in §6, we deduce that only $\nu = 2$ is of interest because of the high computational complexity for $\nu \geq 3$. For $\nu = 2$, (15) shows that for version (i)

$$P_{\nu_k}^{(k)}(t) = \left(\frac{\mu_k^{-1} + \phi_k - 2t}{\mu_k^{-1} + \phi_k}\right)^2, \qquad P_{\nu_k}^{(k)}(\phi_k) = \left(\frac{\mu_k^{-1} - \phi_k}{\mu_k^{-1} + \phi_k}\right)^2$$

so

$$\phi_{k+1} = 4\phi_k\mu_k^{-1}/(\mu_k^{-1} + \phi_k)^2, \qquad k = 0, \cdots, p-1. \tag{16}$$

It is readily seen that if $\mu_k \equiv \hat{\mu}$, the recursion converges to a positive fixed-point

$$\hat{\phi} = 2\hat{\mu}^{-1/2} - \hat{\mu}^{-1}$$

if $\hat{\mu} > 1/4$. Even though the sequence μ_k is in fact not constant, the quantities ϕ_k are bounded below by a number independent of p. The condition numbers are

$$\lambda_k = \phi_k^{-1},$$

and

$$\lambda_{k+1} = (1 + \mu_k\lambda_k^{-1})^2/(4\mu_k\lambda_k^{-1}), \qquad k = 0, \ldots, p-1.$$

We have seen earlier that $b_0 = \sigma a_0$ implies $\mu_k = \sigma$, in which case

$$\lambda_k \to \hat{\lambda} = \hat{\mu}/(2\sqrt{\hat{\mu}} - 1) \approx 1.09,$$

that is, a very small condition number. The condition numbers for the standard nine-point difference stencils converge to the same limit and should be close to this.

Consider next preconditioners where on alternating levels polynomials of first and higher degree are used; that is

$$\nu_k = 1, \quad \text{and} \quad \nu_{k+1} = 2 \quad \text{or} \quad \nu_{k+1} = 3.$$

For $\nu_{k+1} = 2$ we find

$$\lambda_{k+2} = \mu_{k+1}^{-1}\lambda_{k+1} = \mu_{k+1}^{-1}(1 + \mu_k\lambda_k^{-1})^2/(4\mu_k\lambda_{k+1}^{-1}). \tag{17}$$

Similar to the above derivation for the case where $\nu = 2$ was used throughout, we find that (17) has a fixed-point

$$\hat{\lambda} = \hat{\mu}/(2\hat{\mu} - 1)$$

if $\mu_k \equiv \hat{\mu}$ and $\hat{\mu} > 1/2$. For $\hat{\mu} = \sigma$ we find

$$\hat{\lambda} = \sigma/(2\sigma - 1) = (3 + \sqrt{5})/2 \approx 2.62.$$

For $\nu_{k+1} = 3$ we have

$$\begin{aligned} 1 - P_3^{(k)}(\phi_k) &= \left[T_3\left(\frac{\beta_k + \alpha_k}{\beta_k - \alpha_k}\right) - 1\right] \Big/ \left[T_3\left(\frac{\beta_k + \alpha_k}{\beta_k - \alpha_k}\right) + 1\right] \\ &= \left[1 + \frac{1}{8}\frac{(\beta_k - \alpha_k)^2}{(\beta_k + \alpha_k)\beta_k}\right] \Big/ \left[1 + \frac{1}{8}\frac{(\beta_k - \alpha_k)^2}{(\beta_k + \alpha_k)\alpha_k}\right] \end{aligned}$$

so with $\beta_{k+1} = \mu_{k+1}^{-1}$, $\alpha_{k+1} = \phi_{k+1}$,

$$\begin{aligned} \lambda_{k+2} &= \mu_{k+1}^{-1}\lambda_{k+1} \\ &= \mu_{k+1}^{-1}\left[1 + \frac{1}{8}\frac{(\lambda_k - \mu_k)^2}{(\lambda_k + \mu_k)\mu_k}\right] \Big/ \left[1 + \frac{1}{8}\frac{(\lambda_k - \mu_k)^2}{(\lambda_k + \mu_k)\lambda_k}\right]. \end{aligned} \tag{18}$$

If $\mu_k \equiv \hat{\mu}$, this recurrence has a fixed-point $\hat{\lambda}$ satisfying

$$\hat{\lambda}\left[1 + \frac{1}{8}\frac{(\hat{\lambda} - \hat{\mu})^2}{(\hat{\lambda} + \hat{\mu})\hat{\lambda}}\right] = \frac{1}{\hat{\mu}}\left[1 + \frac{1}{8}\frac{(\hat{\lambda} - \hat{\mu})^2}{(\hat{\lambda} + \hat{\mu})\hat{\mu}}\right]$$

and we find if $\hat{\mu} > 1/3$

$$\hat{\lambda} = \hat{\mu}\frac{3 - \hat{\mu}}{3\hat{\mu} - 1},$$

which is

$$\hat{\lambda} = \frac{3}{2} + \frac{\sqrt{5}}{10} \approx 1.72$$

if $\hat{\mu} = \sigma$.

5.2. Analysis of parameter-free polynomials . The previous section gave an analysis for the case where the polynomials $P_{\nu_k}^{(k)}$ were optimized with respect to the quantities α_k and β_k. This of course presumes that these quantities are computable, which in general, and in particular for problems with strongly variable coefficients, may not be the case. In this section we will therefore suggest polynomials that do

not require explicit knowledge of parameters. In particular, we will give an analysis of the quantities

$$\inf_u \frac{u^t A^{(k)} u}{u^t M^{(k)} u}, \qquad \sup_u \frac{u^t A^{(k)} u}{u^t M^{(k)} u}$$

for several specific classes of polynomials. Quantities $u^t S^{(k)} u / u^t M^{(k)} u$ differ from these by at most a simple factor, as was shown in the previous section.

We start by considering separately polynomials of even and odd degree, where the latter are required to have degree $\nu \geq 3$. We will then consider the case $\nu = 1$, and the use of mixed first and higher degree polynomials.

CASE I. $P(x) = \mu^\nu (x - \mu^{-1})^\nu$ for ν even.

In this case we show that the polynomial P assumes only values in $(0, 1)$ for some value of $\mu > 1$. Then Lemma 5.5 applies, and we find that if I_k is such that for some $\phi_k \in (0, 1]$

$$I_k = [\phi_k, \mu_k^{-1}]$$

then

$$I_{k+1} = [\phi_{k+1}, \mu_{k+1}^{-1}], \qquad \phi_{k+1} = \min\{1 - P(\phi_k), 1 - P(\mu_k^{-1})\}.$$

It is seen that $\mu^{-1} = \min_k \mu_k$ satisfies the above assumption.

CASE II. $P(x) = (1 - \epsilon x)^\nu$ for ν odd and $\nu > 1$; $\epsilon \in (0, 1]$.

First let us assume that $\epsilon = 1$. In this case we show that there exist quantities β_k such that

$$\text{for all } u: \qquad \frac{u^t A^{(k)} u}{u^t M^{(k)} u} \in [1, \beta_k]$$

and such that the sequence $k \mapsto \beta_k$ is bounded. Suppose inductively that for all u

$$\frac{u^t A^{(k)} u}{u^t M^{(k)} u} \in [1, \beta_k]$$

for some $\beta_k > 1$. Then

$$\frac{u^t S^{(k)} u}{u^t M^{(k)} u} \in [1, \beta_k \mu_k^{-1}],$$

and, as $1 - P$ is increasing and $1 - P(1) = 1$, we find

$$\frac{u^t S^{(k)} u}{u^t Z^{(k)} u} \in [1, \beta_{k+1}], \qquad \beta_{k+1} = 1 - P(\beta_k \mu_k^{-1}).$$

Now use Corollary 5.5 and the second half of the proof of Lemma 5.4 to find that also

$$\frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} \in [1, \beta_{k+1}].$$

In order to show that the β_k's converge if $\nu = 2\nu' + 1$ (for $\nu' > 0$), we show that the recursion

$$x_0 = 1; \qquad x_{k+1} = 1 - (1 - \mu^{-1} x_k)^3$$

has a fixed point for μ sufficiently close to 1. It is easily seen that, if this is the case, also for higher values of ν' a fixed point exists. Elementary computation shows that this recurrence has a fixed point if

$$\mu \geq \frac{3}{4}.$$

As this condition is not met in practice, we choose ϵ such that a fixed point is ensured. For constant values of μ_k, $\epsilon = \mu$ would be a convenient choice; in practice it suffices to take $\epsilon = 1/2$. Note, however, that in the numerical tests on the Poisson problem even for $\epsilon = 1$ the method converges in a number of iterations that is of optimal order, or close to it.

CASE III. $P(x) = 1 - x$.

The important observation in the case of $P(x) = 1 - x$ is that

$$Z^{(k)} = M^{(k)},$$

so if β_k is such that

$$\text{for all } u: \qquad \frac{u^t A^{(k)} u}{u^t M^{(k)} u} \leq \beta_k,$$

then

$$\begin{aligned} \sup_u \frac{u^t A^{(k+1)} u}{u^t Z^{(k+1)} u} &= \sup_{u_2} \frac{u_2^t S^{(k)} u_2}{u_2^t M^{(k)} u_2} \leq \sup_{u_2} \frac{u_2^t S^{(k)} u_2}{u_2^t A^{(k)} u_2} \sup_{u_2} \frac{u_2^t A^{(k)} u_2}{u_2^t M^{(k)} u_2}, \\ &\leq \mu_k^{-1} \beta_k \end{aligned}$$

that is,

$$\beta_{k+1} = \mu_k^{-1} \beta_k.$$

The lower bound of 1 carries over from the above analysis for polynomials of odd degree. Thus we find in this case an (approximately) geometric increase of the condition number, and consequently an increase of the number of iterations inversely proportional to the meshwidth.

CASE IV. $P(x)$ is of first and higher degree on alternating levels.

It is shown in §6 that if second- or third-degree polynomials are only employed on every other level, the number of operations per preconditioner solve will be of the order of the number of unknowns. In order to show that the condition number $O(1)$ can still be reached, we have to extend the above analysis somewhat. Again we consider the cases of odd and even polynomials separately.

Assume that k is such that on level $k+1$ we have $P(x) = 1 - x$, while on level k the polynomial is of even degree. Let α_k and β_k be such that

$$\text{for all } u: \qquad \frac{u^t A^{(k)} u}{u^t M^{(k)} u} \in [\alpha_k, \beta_k],$$

and let the polynomial satisfy

$$x \in [\alpha_k, \beta_k \mu_k^{-1}] \Rightarrow P(x) \in [0, 1]. \tag{19}$$

For a lower bound on the condition number we find, referring to Lemma 5.4

$$\begin{aligned}\inf_u \frac{u^t A^{(k+2)} u}{u^t M^{(k+2)} u} &= \inf_{u_2} \frac{u_2^t S^{(k+1)} u_2}{u_2^t Z^{(k+1)} u_2} = \inf_{u_2} \frac{u_2^t S^{(k+1)} u_2}{u_2^t M^{(k+1)} u_2} \\ &\geq \inf_{u_2} \frac{u_2^t A^{(k+1)} u_2}{u_2^t M^{(k+1)} u_2} = \inf_{u_{22}} \frac{u_{22}^t S^{(k)} u_{22}}{u_{22}^t Z^{(k)} u_{22}} \\ &= \inf_{u_{22}} \left\{ 1 - P\left(\frac{u_{22}^t S^{(k)} u_{22}}{u_{22}^t M^{(k)} u_{22}} \right) \right\}.\end{aligned}$$

Again using the proof of Lemma 5.4 and taking the requirements on the polynomial into account we find that

$$\frac{u^t S^{(k)} u}{u^t Z^{(k)} u} \in [0,1], \qquad \text{so} \qquad \frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} \leq 1$$

and

$$\begin{aligned}\sup_u \frac{u^t S^{(k+1)} u}{u^t Z^{(k+1)} u} = \sup_u \frac{u^t S^{(k+1)} u}{u^t M^{(k+1)} u} &\leq \mu_{k+1}^{-1} \sup_u \frac{u^t A^{(k+1)} u}{u^t M^{(k+1)} u} \\ &\leq \mu_{k+1}^{-1},\end{aligned}$$

which shows that

$$\text{for all } u: \qquad \frac{u^t A^{(k+2)} u}{u^t M^{(k+2)} u} \leq \mu_{k+1}^{-1}.$$

Thus we have proved inductively that

$$0 < \alpha_{k+2} = \inf_{x \in [\alpha_k, \beta_k \mu_k^{-1}]} (1 - P(x)); \qquad \beta_{k+2} = \mu_{k+1}^{-1}.$$

An easy way to have condition (19) satisfied is to choose

$$P(x) = \mu^\nu (x - \mu^{-1})^\nu,$$

where μ is a lower bound to all of the μ_ks.

Next let the polynomial be of odd degree on level k; specifically, let

$$P(x) = (1 - \epsilon x)^\nu, \tag{20}$$

where $\epsilon \in (0,1]$. We begin by assuming that $\epsilon = 1$ is chosen. Our inductive assumption is

$$\text{for all } u: \qquad \frac{u^t A^{(k)} u}{u^t M^{(k)} u} \in [1, \beta_k].$$

As

$$\frac{u^t S^{(k+2)} u}{u^t M^{(k+2)} u} \in [1, \beta_k \mu_k^{-1}] \Rightarrow \frac{u^t S^{(k+2)} u}{u^t Z^{(k+2)} u} \in [1, 1 - P(\beta_k \mu_k^{-1})],$$

so following the above analysis for odd degree polynomials, we get the lower bound

$$\text{for all } u: \qquad \frac{u^t A^{(k+2)} u}{u^t M^{(k+2)} u} \geq 1$$

and the upper bound

$$\sup_u \frac{u^t A^{(k+2)} u}{u^t M^{(k+2)} u} \leq \sup_{u_2} \frac{u_2^t S^{(k+1)} u_2}{u_2^t Z^{(k+1)} u_2} = \sup_{u_2} \frac{u_2^t S^{(k+1)} u_2}{u_2^t M^{(k+1)} u_2}$$
$$\leq \mu_{k+1}^{-1} \sup_{u_2} \frac{u_2^t S^{(k+1)} u_2}{u_2^t Z^{(k+1)} u_2} \leq \mu_{k+1}^{-1} \sup_{u_{22}} \frac{u_{22}^t S^{(k)} u_{22}}{u_{22}^t Z^{(k)} u_{22}}.$$

As the recurrence

$$\beta_{k+1} = \mu^{-1}(1 - (1 - \mu^{-1}\beta_k)^\nu)$$

does not have a fixed-point for any value of μ, we will have to choose again an $\epsilon < 1$. In practice a value of $\epsilon = .5$ is sufficient.

6. Computational complexity. The solution of a system $M^{(k+1)}a = b$ involves the solution of a system of lower dimension with $Z^{(k)}$. From the definition of $Z^{(k)}$, 3, we see that solving

$$Z^{(k)} y = x$$

is equivalent to computing

$$y = (I - P_\nu(M^{(k)^{-1}} S^{(k)}))S^{(k)^{-1}} x \quad \text{for version (i)} \tag{21}$$

or

$$y = (I - P_\nu(M^{(k)^{-1}} A^{(k)}))A^{(k)^{-1}} x \quad \text{for version (ii).} \tag{22}$$

As was already remarked before, we choose for P_ν a polynomial which can be written

$$P_\nu(x) = 1 - c_1 x - \cdots - c_\nu x^\nu,$$

so for instance the solution process (21) boils down to

$$y = \left[c_\nu (M^{(k)^{-1}} S^{(k)})^{\nu-1} M^{(k)^{-1}} + \cdots + c_1 M^{(k)^{-1}}\right] x$$
$$= M^{(k)^{-1}} \left[c_1 + S^{(k)} M^{(k)^{-1}} (\cdots (c_{\nu-1} + S^{(k)} M^{(k)^{-1}} c_\nu) \cdots)\right] x,$$

which can be written algorithmically as

$y \leftarrow 0$
for $k = 0, \cdots, \nu - 1$ do
    $y \leftarrow M^{(k)^{-1}}(c_{\nu-k} x + S^{(k)} y)$

for version (i); for version (ii) we must replace $S^{(k)}$ by $A^{(k)}$.

We will now proceed to give operation counts for the solution of a system with preconditioners using combined five-point/nine-point stencils and the elimination procedure described above. If we take N to denote the number of points on the finest mesh, at level p, we define $N_p = N$, and the number of points N_k on the mesh at level k equals (disregarding boundary effects)

$$N_k = N/2^{p-k}.$$

From the definition (2), (3) of the preconditioner and the algorithm to solve systems with $Z^{(k)}$ above, we find that the number of operations ω_{k+1} needed to solve a system with $M^{(k+1)}$ can (for version (i)) be given by

$$\begin{aligned}\omega_{k+1} &= 2 \cdot N_k && \text{solve with } D^{(k)} \text{ twice}\\ &\quad + 2 \cdot 4 \cdot N_k && \text{multiply by } C \text{ and } C^t\\ &\quad + \nu\omega_k + (\nu + 9(\nu-1)) \cdot N_k && \text{solve system with } Z^{(k)}\\ &= (5\nu + 1/2)N_{k+1} + \nu\omega_k && \text{(Total)}.\end{aligned}$$

For version (ii) we find similarly

$$\omega_{k+1} = (4\nu + 3/2)N_{k+1} + \nu\omega_k.$$

These operation counts concern the number of multiplications. The number of additions is somewhat smaller. When the coefficients are locally constant, we can actually reduce the number of multiplications substantially, using a method of summations of differences, or taking averages for the computation of matrix vector multiplications. For further details see [3]. However, in the interest of not presenting too many variations of our method, we do not give the corresponding work estimate here.

It is now easy to estimate the amount of work for various values of ν. Let the amount of work ω_0 at the coarsest level be given. As $N_k = N_{k+1}/2$ we solve the recurrence for the amount of work ω_k explicitly:

$$\omega_k = (5\nu + 1/2)(1 + \cdots + (\tfrac{\nu}{2})^{k-1})N_k + \nu^k\omega_0.$$

For some specific values of ν we find for the total amount of work at the finest level

$$\begin{aligned}\nu = 1\text{: } \omega_p &\le \quad 11N_p + \quad \omega_0\\ \nu = 2\text{: } \omega_p &\le \ 10.5pN_p + 2^p\omega_0\\ \nu = 3\text{: } \omega_p &\le 31(\tfrac{3}{2})^pN_p + 3^p\omega_0\end{aligned}$$

for version (i); for version (ii) we find somewhat smaller constants, except for $\nu = 1$ when the methods coincide. Thus, for $\nu = 2$ or larger the preconditioner can no longer be of optimal order of computational complexity.

As it is desirable to have polynomial degrees higher than 1 (see the analysis of the condition number in §5), we shall propose a preconditioner that uses different polynomials on different levels.

In particular, let us consider a preconditioner which uses first and either second or third degree polynomials on alternate levels. Let k be such that a first degree polynomial is used on level $k+1$. For the amount of work we then find for version (i)

$$\begin{aligned}\omega_{k+2} &= 5.5N_{k+2} + (5\nu + 1/2)N_{k+1} + \nu\omega_k\\ &= (23/4 + 5\nu/2)N_{k+2} + \nu\omega_k.\end{aligned}$$

The explicit solution of this recurrence is

$$\omega_{2k} = (23/4 + 5\nu/2)(1 + \cdots + (\nu/4)^{k-1})N_{2k} + \nu^k\omega_0$$

which gives (neglecting the last term)

$$\nu = 2 : \omega_p \le 21.5N_p; \qquad \nu = 3 : \omega_p \le 53N_p$$

for version (i), and

$$\nu = 2 : \omega_p \leq 20.5 N_p; \qquad \nu = 3 : \omega_p \leq 49 N_p$$

for version (ii). Hence both versions of the method give, when using on alternating levels polynomials of degree 1 and 2 or 3, a preconditioner with computational work per preconditioning step of optimal order, $O(N)$.

7. Numerical tests. We have performed a number of problems,

- to test the optimality of the methods,
- to compare the Chebyshev polynomials and the parameter-free polynomials, and
- to investigate the sensitivity of the methods to anisotropy.

The method used throughout is the conjugate gradient method with a multilevel preconditioner M with various degrees of polynomials, subject to a stopping criterion

$$\sqrt{g^t M^{-1} g} < 10^{-10},$$

where g is the residual. The starting vector was the zero vector and the solution of the differential equation was $u \equiv 1$ throughout.

The coefficient matrix was derived from the nine-point difference box. We have tested the Poisson problem, and the anisotropic problem

$$-u_{xx} - 100 u_{yy} = f \quad \text{on } \Omega = (0,1)^2.$$

The coefficient matrix was a nine-point matrix (that is, an $S^{(p)}$ matrix) derived from the fourth order box finite difference stencil, and the preconditioner was a version (i). The incomplete factorization was pursued all the way down to where the size of the system was 1, so the number of levels was $p = 2\lceil \log_2 h^{-1} \rceil$.

Some observations:

- For the methods with alternating first and higher degree polynomials, it makes a difference if the first degree polynomial is taken on the axi-parallel or the skew grid. It turns out that for the Chebyshev polynomials one has to take the axi-parallel grid, but for the parameter-free polynomials it is better to take the skew grid for the first degree polynomials. In both cases the numbers of iterations may differ, especially for the anisotropic problem, by a factor of 2 depending on which grid is taken. We have reported only the optimal number of iterations.
- For first degree polynomials the method with the Chebyshev polynomials converges surprisingly slowly, to such an extent that for the anisotropic problem no convergence within the first 100 iterations was reached for $h^{-1} > 20$. The convergence speed for the parameter-free first degree polynomials is probably due to an effect of improved condition by underestimation of the upper bound of the spectrum, as was described above.
- When parameter-free polynomials of alternating degree are used on the anisotropic problem, the value of ϵ in the definition of the polynomials is crucial. Although for the Poisson problem $\epsilon = 1$ seems optimal with other values giving slightly more iterations, for anisotropic problems this gives a very slowly converging method. Taking $\epsilon = 1/2$, on the other hand, gives, for third degree polynomials on every other level, satisfactory convergence speeds that are within a factor of 2 of those for the (optimal) Chebyshev polynomials.

Numerical tests indicate that this value of $\epsilon = 1/2$ suffices for most values of the anisotropy; only for the Poisson problem, performance slightly improves when $\epsilon = 1$ is chosen.

- For parameter-free polynomials of the second degree, the choice of μ is crucial. It turned out not to be necessary to choose $\mu = \max_k \mu_k^{-1}$: about half to one third of that value sufficed. The resulting numbers of iterations were within a factor of two from those of the methods with optimal polynomials. Choosing μ smaller led to divergent methods, and choosing it larger rapidly led to slower convergence. However, since no practical method for finding this optimal μ seems to exist, we have only given results for this variant for the Poisson problem where $\mu = 1$ suffices.

We start by giving in Tables 1 and 2 the iteration counts on the Poisson problem. On this problem the "V-cycle" methods (polynomial degree 1) give numbers of iterations that seem to indicate a condition number of less than $O(h^{-1})$; the "W-cycle" methods (polynomial degree ≥ 2) give constant numbers of iterations, but using alternating first and higher degree polynomials gives a slowly increasing number of iterations for the parameter-free polynomials.

TABLE 1

Numbers of iterations for the Δ_5/Δ_9 difference stencil on the Poisson problem using optimal polynomials.

$1/h$	10	20	40	80
$\nu = 1$	10	13	17	22
2	6	6	6	6
3	6	6	6	6
1&2	7	7	7	7
1&3	6	6	6	6

TABLE 2

Numbers of iterations for the Δ_5/Δ_9 difference stencil on the Poisson problem using parameter-free polynomials.

$1/h =$	10	20	40	80
$\nu = 1$	7	10	12	16
2	7	8	8	8
3	5	6	6	6
1&2	7	8	8	9
1&3	6	7	9	12

As a prototype of a more difficult and thus more realistic problem, we consider the anisotropic problem

$$-u_{xx} - 100u_{yy} = f.$$

Although we give in Table 3 the numbers of iterations for the Chebyshev polynomials of second and third degree on each level, these data, indicating that the preconditioner is spectrally equivalent in this case too, should be regarded as reference material only because of the high operation count per iteration.

Our interest then lies with the first order methods, and those with first and higher degree polynomials alternating. As already remarked above, however, using first degree polynomials in a V-cycle method with Chebyshev polynomials gave a very slowly converging method, so these results were not reported.

For the methods with alternating degrees of polynomials, the tests show a reasonably slow increase in the numbers of iterations for the second and in particular third degree polynomials. The preconditioners with parameter-free polynomials converge in a number of iterations that stays within a factor 2 of the optimal number on the Chebyshev polynomials.

TABLE 3
Numbers of iterations for the Δ_5/Δ_9 difference stencil on the anisotropic problem.

$1/h =$	10	20	40	80
Chebyshev: $\nu = 2$	13	16	17	17
3	10	11	12	12
1&2	24	34	38	44
1&3	13	15	16	17
Parameter-Free: $\nu = 1$	15	20	23	26
1&3	16	20	25	28

Tests not reported here on problems with even stronger anisotropy, or with large discontinuities in the coefficients, show essentially the same behaviour, with numbers of iterations differing not much from those given here.

Next we consider the efficiency of these methods compared to some incomplete block factorization preconditioners, namely, the standard recursive incomplete line block preconditioner (denoted L_r in [4] and $INVC1$ in [10]), and a vectorizable variant of this where series expansion of pivot blocks is used in the preconditioner solve (denoted $INVV_j$ in [16] and, more general, $L_x(p)$ in [4]). For the methods with alternating degree polynomials we use for the Poisson problem second degree polynomials (Table 4), as for third degree the number of iterations is the same, but the number of operations is about twice as high; for the anisotropic problem, however, second degree polynomials give an increasing number of iterations, so we present then the figures for third degree polynomials (Table 5).

TABLE 4
Numbers of iterations, condition number estimates, and operations/n^2 per PCG *for multilevel and incomplete factorization preconditioners on the Poisson problem using Chebyshev polynomials.*

$h^{-1} =$	10			20			40			80			160		
L_r	5	1.3	106	8	1.6	160	13	1.9	250	24	2.2	448	42	.	772
$L_x(3)$	5	1.8	162	8	2.3	240	13	2.9	370	24	3.7	656	42	.	1124
$\nu = 1$	7	1.8	167	10	2.0	231	12	2.2	274	16	2.4	358	.	.	
$\nu = 1\&2$	7	1.38	263	7	1.4	263	7	1.42	263	7	1.42	263	.	.	

TABLE 5
Numbers of iterations, condition number estimates, and operations/n^2 per PCG *for multilevel and incomplete factorization preconditioners on the anisotropic problem; Chebyshev polynomials for the case 1&3.*

$h^{-1} =$	10			20			40			80			160		
L_r	3	1.003	40	4	1.03	88	6	1.2	144	12	1.6	242	23	.	430
$L_x(3)$	6	3.5	188	10	5.8	292	20	9.2	552	41	12.7	1098	75	.	1982
$\nu = 1$	15	6.2	337	20	8.7	442	23	10.9	505	26	16.3	568	.	.	
$\nu = 1\&3$	13	2.1	925	15	2.2	1051	16	2.3	1114	17	2.4	1177	.	.	

First of all we notice with respect to scalar efficiency that it is hard to outperform the recursive preconditioner L_r, especially as its performance improves with increasing

anisotropy. Only for the Poisson problem is the multilevel preconditioner with $\nu =$ 1&2 more efficient for sufficiently fine grids.

On vector computers and parallel architectures, however, where a recursive preconditioner is less desirable, we have to compare the multilevel preconditioners to the vectorizable preconditioner L_x. It is then apparent that, even for the anisotropic problem, for relatively coarse meshes multilevel preconditioners can be competitive with respect to scalar efficiency. The simple V-cycle method ($\nu = 1$) with parameter-free polynomials seems then preferable to W-cycle methods, as it has at most a slightly higher complexity on the Poisson problem, but a noticeably lower operation count on the anisotropic problem. Extrapolating the results for the anisotropic problem would lead us to expect that the W-cycle methods become more efficient only for grids with $h^{-1} >\sim 600$.

Note that solving a system with the multilevel preconditioner is vectorizable, as only diagonal systems need to be solved. However, the sequence of grids presupposes that the architecture is capable of efficient gather/scatter.

It is only on massively parallel architectures that W-cycle methods are the uncontested winners with a number of iterations that is both lower than that of the V-cycle method, and constant or at most very slowly increasing.

In the last two tables estimates of the condition number of the preconditioned system are also given, based on parameters computed during the iterative process; see for instance [14]. Comparing for instance the incomplete factorization preconditioner $L_x(3)$ and the first degree multilevel preconditioner on the anisotropic problem, we see that the latter has a higher condition number, but it converges faster. Thus these figures are not the best measure of the relative value of the methods. However, we do see that for the multilevel preconditioners the condition numbers are slowly growing for the first degree methods, and almost constant for the higher degree ones.

8. Conclusion. We have investigated multilevel preconditioners for selfadjoint nine-point difference operators where the incompleteness of the preconditioner derives from replacing the nine-point stencil by a five-point one on the red points of a red-black structure imposed on each grid. Each coarser grid consists then of the black points of the previous grid.

We have shown that if the grids are traversed using certain polynomials to obtain a general W-cycle–like method, it is possible to have a preconditioner that is both spectrally equivalent to the original matrix and has a number of operations that is proportional to the number of unknowns. As an added bonus, only systems with diagonal matrices need to be solved.

Such multilevel preconditioners can function competitively on vector machines if there is an efficient gather/scatter mechanism to overcome the increasing grid sizes. On massively parallel architectures they can be extremely fast, involving no global synchronization and only local communications.

REFERENCES

[1] O. AXELSSON, *On multigrid methods of the two-level type*, in Multigrid Methods, Proceedings, Köln-Porz, W. Hackbusch and U. Trottenberg, eds., 1982, pp. 352–367. Lecture Notes in Mathematics, Vol. 960.

[2] ———, *A multilevel solution method for nine-point difference approximations*, in Parallel Supercomputing: Methods, Algorithms and Applications, G. F. Carey, ed., John Wiley & Sons, New York, 1989, pp. 199–205.

[3] O. AXELSSON AND A. BARKER, *Finite Element Solution of Boundary Value Problems. Theory and Computation*, Academic Press, Orlando, FL, 1984.
[4] O. AXELSSON AND V. EIJKHOUT, *Vectorizable preconditioners for elliptic difference equations in three space dimensions*, J. Comp. Appl. Math., 27 (1989), pp. 299–321.
[5] O. AXELSSON AND I. GUSTAFSSON, *On the use of preconditioned conjugate gradient methods for red-black ordered five-point difference schemes*, J. Comp. Physics, 35 (1980), pp. 284–299.
[6] O. AXELSSON AND P. VASSILEVSKI, *Algebraic multilevel preconditioning methods*, I, Numer. Math., 56 (1989), pp. 157–177.
[7] ———, *Algebraic multilevel preconditioning methods*, II, Tech. Report 1988-15, Inst. for Sci. Comput., University of Wyoming, Laramie, WY, 1988; SIAM J. Numer. Anal., to appear.
[8] R. BEAUWENS, *On Axelsson's perturbations*, Lin. Alg. Appl., 68 (1985), pp. 221–242.
[9] D. BRAESS, *The contraction number of a multigrid method for solving the Poisson equation*, Numer. Math., 37 (1981), pp. 387–404.
[10] P. CONCUS, G. GOLUB, AND G. MEURANT, *Block preconditioning for the conjugate gradient method*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 220–252.
[11] T. DUPONT, R. KENDALL, AND H. RACHFORD, *An approximate factorization procedure for solving self-adjoint elliptic difference equations*, SIAM J. Numer. Anal., 5 (1968), pp. 559–573.
[12] H. ELMAN, *Approximate Schur complement preconditioners on serial and parallel computers*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 581–605.
[13] I. GUSTAFSSON, *A class of first-order factorization methods*, BIT, 18 (1978), pp. 142–156.
[14] L. HAGEMAN AND D. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.
[15] T. MEISS, *Schnelle Lösung von Randwertaufgaben*, Z. Angew. Math. Mech., 62 (1982), pp. 263–270.
[16] G. MEURANT, *The block preconditioned conjugate gradient method on vector computers*, BIT, 24 (1984), pp. 623–633.
[17] M. RIES, U. TROTTENBERG, AND G. WINTER, *A note on MGR methods*, Lin. Alg. Appl., 49 (1983), pp. 1–26.
[18] H. YSERENTANT, *On the multilevel splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379–412.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1401–1419, November 1991
© 1991 Society for Industrial and Applied Mathematics
010

FRONT TRACKING APPLIED TO A NONSTRICTLY HYPERBOLIC SYSTEM OF CONSERVATION LAWS*

NILS HENRIK RISEBRO† AND ASLAK TVEITO‡

Abstract. The application of a front tracking method to a nonstrictly hyperbolic system of conservation laws is described in one space dimension. The front tracking method is based on approximate solutions of Riemann problems. The method is compared with the random choice scheme and the upwind scheme.

Key words. front tracking, Riemann problem, hyperbolic conservation laws

AMS(MOS) subject classifications. 35L65, 65M99, 76S05

1. Introduction.

In this paper we discuss the application of a front tracking method to the following system of nonstrictly hyperbolic conservation laws,

$$
\begin{aligned}
&s_t + f(s, c_1, \cdots, c_n)_x = 0, \\
&(sc_i + a_i(c_i))_t + (c_i f(s, c_1, \cdots, c_n))_x = 0, \qquad i = 1, \cdots, n.
\end{aligned} \tag{1}
$$

Here $(s, c_1, \cdots, c_n)$ denotes the unknown state vector and f and $a_i = a_i(c_i)$ for $i = 1, \cdots, n$ are given smooth functions. This system models the displacement of oil by water, containing n dissolved polymer components, in a porous medium. The saturation of the aqueous phase is given by the variable s, and $c = (c_1, \cdots, c_n)$ denotes the concentration of the polymer components in the aqueous phase. The function f is usually called the fractional flow function and $a_i = a_i(c_i)$ for $i = 1, \cdots, n$ are referred to as the adsorption functions.

The system (1) has been studied through a series of papers, starting with the work of Isaacson [14], who solved the Riemann problem of (1) in the case $n = 1$ and $a_1(c_1) \equiv 0$. Johansen and Winther [17] generalized this analysis by taking the effects of adsorption into account. In [18] they proved the existence of a unique solution of the Riemann problem for (1) for $n \geq 1$. Their constructive argument for the existence of a solution of the Riemann problem was presented in an algorithmic form in [16].

The Cauchy problem of (1) without the adsorption term has been studied by Temple [25] and Isaacson and Temple [15]. They proved, using the random choice scheme, the existence of a weak solution of the Cauchy problem with initial data having bounded total variation. In [27] the existence of a unique and stable solution of the Cauchy problem of (1) for $n = 1$ without the adsorption term and with "smooth" initial data, was proved.

Some numerical illustrations of solutions of Riemann problems for the system were presented in [16]. These examples clearly indicated that the solutions of (1) may develop a very complicated behavior, even for Riemann problems. Although standard numerical methods seem to converge to the correct solution of the problem, the convergence is slow and it is very time consuming to compute an accurate solution. This motivates the development of faster methods.

* Received by the editors June 13, 1990; accepted for publication December 4, 1990.

† Department of Mathematics, University of Oslo, P.O. Box 1053 Blindern, N-0316 Oslo 3, Norway. The research of this author was supported by the Royal Norwegian Council for Technical and Industrial Research.

‡ Center for Industrial Research, P. O. Box 124, Blindern, O314 Oslo 3, Norway. The research of this author has been supported by VISTA, a research cooperation between the Norwegian Academy of Science and Letters and Den norske stats oljeselskap a.s. (Statoil).

The purpose of this paper is to describe the application of a front tracking strategy to (1). Our numerical method is a generalization of Dafermos' method [6] for scalar conservation laws. The main idea of his method is to approximate the flux function by a piecewise linear function, and the initial function by a piecewise constant function. Then the exact solution of the problem can be computed. Hedstrom [12] studied a generalization of Dafermos' method to the "p-system" applying piecewise linear approximation to the p-function and thereby utilizing the fact that the only nonlinear function involved in the system is a function of one single variable. It seems difficult to pursue this idea in the context of general systems of hyperbolic conservation laws.

Front tracking has been a successful computational tool in many areas of applied mathematics, cf. [1], [2], [3], [7], [10], [13], and [24]. The main advantage of this family of methods is their ability to represent the complicated behavior of solutions of nonlinear partial differential equations, especially when the solutions develop discontinuities.

The front tracking method studied in the present paper is a generalization of Dafermos' method to systems where the solution of the general Riemann problem can be computed. The method is based on approximate solutions of Riemann problems. The approximate solution consists of constant states separated by discontinuities in the $x - t$ space. Rarefaction waves are approximated by several small inadmissible shocks. The discontinuities are tracked until they interact, i.e., collide, then they define a Riemann problem. The approximate solution of this Riemann problem defines a new family of discontinuities which are tracked, and so on. A more precise algorithm will be presented in §4.

The main advantage of the present method compared with the random choice scheme is that our method utilizes all the information from the approximate solution of a Riemann problem, whereas the random choice scheme uses only the information from one single point of the solution path from the left to the right state in order to define the approximate solution on a grid block. The numerical experiments illustrate that the front tracking method is computationally more efficient than the random choice scheme.

In [21] this approach was studied from a theoretical point of view in the case of strictly hyperbolic systems of conservation laws with "small" data. In fact, an alternative proof of Glimm's [9] famous existence proof was established. The method was applied to the classical system of gas dynamics in [22]. Our scheme turned out to be much faster than the random choice scheme and the Lax–Friedrichs scheme.

In the next section we present the mathematical model under consideration. In §3 we will briefly review the elementary waves for (1). We will also describe the computation of approximative solutions to these problems. In §4, the front tracking method will be presented. To compare the method with standard numerical methods for hyperbolic conservation laws, we have implemented the upwind scheme and the random choice scheme. The results of comparisons on two test problems are reported in §5. Our second test problem is of great practical interest since it models the displacement of oil in a porous medium by water containing a polymer "slug." We summarize our results in §6.

2. The mathematical model. In this section we will present the mathematical model under consideration. We refer to Pope [20] and Johansen and Winther [17] for a derivation of the model.

Define $\mathbf{I} = [0, 1]$, and let $\mathcal{S} = \mathbf{I}^{n+1}$ denote the state space of the model. Recall that $s \in \mathbf{I}$ denotes the water saturation, and $c_i \in \mathbf{I}$ denotes the concentration of the

ith polymer component in the aqueous phase. We assume that the fractional flow function f is a smooth function which for each fixed $c = (c_1, \cdots, c_n)$ has the following properties (cf. Fig. 1),

(i) $f(0, c) \equiv 0$, $f(1, c) \equiv 1$,
(ii) $f_s(s, c) = \frac{\partial f(s,c)}{\partial s} > 0$ for $0 < s < 1$,
(iii) $\exists s^I = s^I(c) \in \mathbf{I}$ such that

$$f_{ss}(s, c) > 0 \quad \text{for } s \in (0, s^I(c)) \text{ and } \quad f_{ss}(s, c) < 0 \quad \text{for } s \in (s^I(c), 1),$$

(iv) $\frac{\partial f(s,c)}{\partial c_i} < 0$ for $0 < s < 1$, $i = 1, \cdots, n$.

Let

$$h_i(c_i) = \frac{da_i(c_i)}{dc_i}.$$

We assume that the adsorption functions $a_i(c_i)$ are smooth functions satisfying the following requirements (cf. Fig. 2),

(i) $a_i(0) = 0$,
(ii) $h_i(c_i) > 0$,
(iii) $\frac{dh_i(c_i)}{dc_i} < 0$

for all $i = 1, \cdots, n$. Under these assumptions we will discuss the implementation of a front tracking method for the following system of nonstrictly hyperbolic conservation laws,

$$\begin{aligned} &s_t + f(s, c_1, \cdots, c_n)_x = 0 \\ &(sc_i + a_i(c_i))_t + (c_i f(s, c_1, \cdots, c_n))_x = 0, \qquad i = 1, \cdots, n, \end{aligned} \tag{2}$$

with initial data

$$(s(x, 0), c(x, 0)) = (s^0(x), c^0(x)). \tag{3}$$

The initial data are supposed to satisfy

1. $(s^0(x), c^0(x)) \in \mathcal{S} \qquad \forall x \in \mathbb{R}$,
2. $(s^0(x), c^0(x)) = (s^L, c^L)$ for $x < x^L$, and
3. $(s^0(x), c^0(x)) = (s^R, c^R)$ for $x > x^R$.

Here (s^L, c^L) and (s^R, c^R) denote constant states in the state space $\mathcal{S}$, and $x^L < x^R$ are finite real numbers. Thus, the initial functions are supposed to take on constant values for $|x|$ sufficiently large. As a part of the initialization of the front tracking method, the initial functions will be approximated by piecewise constant functions with a finite number of jumps.

3. The Riemann problem. The Riemann problem for (2) is the initial value problem with data of the form

$$(s(x, 0), c(x, 0)) = \begin{cases} (s^L, c^L), & x < 0 \\ (s^R, c^R), & x > 0. \end{cases} \tag{4}$$

Johansen and Winther [18] proved the existence of a unique solution of this Riemann problem for arbitrary left state (s^L, c^L) and right state (s^R, c^R) in the state space $\mathcal{S}$. In [16] the implementation of a Riemann problem solver for the system is discussed. In the present paper we apply this Riemann solver as a part of our front tracking method. Since (2) is nonstrictly hyperbolic, the solution of the Riemann problem is,

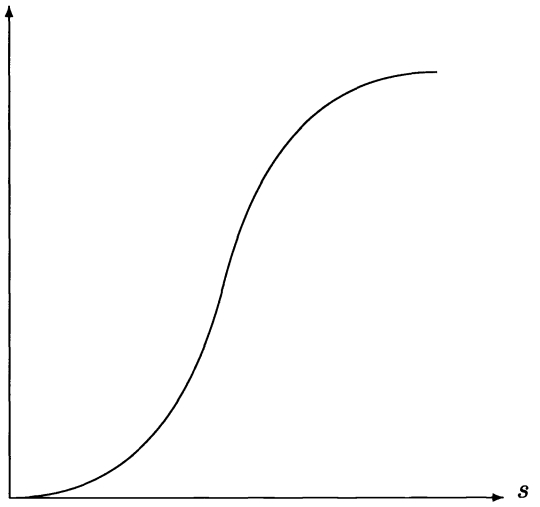


FIG. 1. *The fractional flow function f as a function of s for a fixed c.*

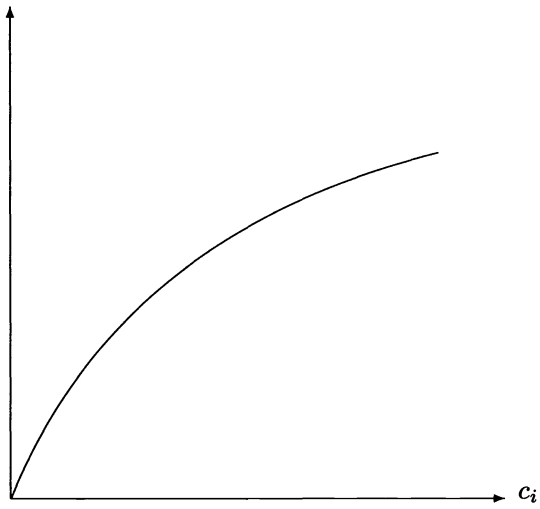


FIG. 2. *A typical adsorption function $a_i(c_i)$.*

in general, rather complicated. Thus, we consider this system to be well suited as a test of the applicability of our front tracking method.

The solution of the Riemann problem is a composition of elementary waves for the system. The elementary waves of (2) can be divided into two types; s-waves and c-waves. These waves can either be rarefaction or shock waves. The Riemann solver is a description of how to put the elementary waves together in order to get a solution path from the left state (s^L, c^L) to right state (s^R, c^R).

The computation of approximate solutions of Riemann problems is a very important part of the front tracking method. In fact, about 95 percent of the CPU time of a simulation is spent on computing such approximate solutions. In our implementation

we use the algorithm outlined in [16]. The only difference is the computation of the elementary waves. The approximation of these waves will be described below, but first we briefly review the mathematical structure of these waves.

3.1. The s-waves. If $c^L = c^R$ in (4), the system (2) degenerates to the scalar Buckley–Leverett equation,

$$s_t + f(s)_x = 0,$$

where $f(s) = f(s, c^L) = f(s, c^R)$. The elementary waves and the solution of the Riemann problem for a scalar conservation law is well known, cf. Gelfand [8].

3.2. The c-waves. Assume that $c^L \neq c^R$, and that (s^L, c^L) can be connected to (s^R, c^R) by a single elementary wave. Such a wave is called a c-wave. The c-waves are not scalar waves; they involve the s-variable and at least one of the c-components. We divide the c-waves into c-shock and c-rarefaction waves.

The c-rarefaction waves are smooth solutions of (2). For smooth solutions, the system can be written in the form

$$\begin{pmatrix} s \\ c \end{pmatrix}_t + A(s,c) \begin{pmatrix} s \\ c \end{pmatrix}_x = 0,$$

where

$$A(s,c) = \begin{pmatrix} \frac{\partial f}{\partial s} & \frac{\partial f}{\partial c_1} & \frac{\partial f}{\partial c_2} & \cdots & \frac{\partial f}{\partial c_n} \\ 0 & f/(s+h_1) & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & f/(s+h_n) \end{pmatrix}$$

cf. [18]. The eigenvalues of the matrix A are given by

$$\lambda^s = \lambda^s(s,c) = \frac{\partial f}{\partial s}(s,c)$$
$$\lambda^i = \lambda^i(s,c) = \frac{f(s,c)}{s + h_i(c_i)}, \qquad i = 1, \cdots, n.$$

Here, the first eigenvalue, λ^s, corresponds to the s-waves; the speed of these waves are given by λ^s.

The speed of a c-wave is given by λ^i. Since no assumption is made on the relation between the adsorption functions, the family $\{\lambda^i\}$ may have coinciding eigenvalues. Suppose $c_i^L \neq c_i^R$ for some fixed index i. Let J denote the set of indices corresponding to the rest of the c-components involved in the simple c-rarefaction wave, i.e.,

$$J = \{j : 1 \leq j \leq n, j \neq i, \lambda^i = \lambda^j\}.$$

Along the rarefaction path, the relation $\lambda^i = \lambda^j$ defines c_j as a function of c_i;

$$c_j = g_{i,j}(c_i) \quad \text{for} \quad j \in J,$$

where we have defined

$$g_{i,j}(c_i) = h_j^{-1} h_i(c_i).$$

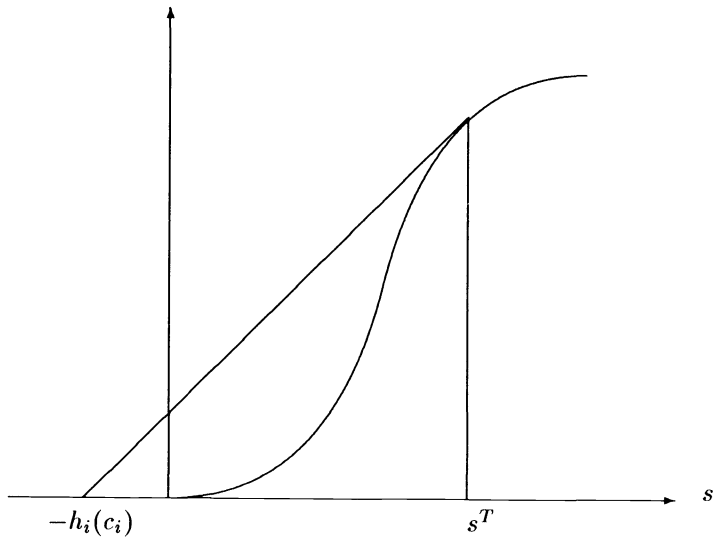


FIG. 3. *There exists exactly one* $s = s^T > 0$ *such that* $f_s(s^T, c) = f(s^T, c)/(s^T + h_i(c_i))$ *for fixed* c *and* i.

The path this elementary wave traverses in the state space can be computed by solving the following ordinary differential equation (cf. [18]),

$$\left(\frac{\partial f}{\partial c_i} + \sum_{j \in J} \frac{\partial f}{\partial c_j}\frac{dg_{i,j}}{dc_i}\right)\frac{dc_i}{ds} = \lambda^i - \lambda^s$$

$$c_j = g_{i,j}(c_i), \qquad j \in J \tag{5}$$

$$c_j = c_j^L \ (= c_j^R) \qquad \text{for } j \neq i \text{ and } j \notin J.$$

The rarefaction waves are directed, in the state space, towards increasing values of the associated eigenvalue. For the present system, this implies that the rarefaction curves are directed towards increasing values of c_i. In fact, the c-rarefaction waves are nondecreasing in all the c-components, (cf. [18]).

The eigenvalue λ^i may also coincide with λ^s. We assume that, for a fixed c and i, there is exactly one value of $s \in (0,1)$ such that $\lambda^i = \lambda^s$, cf. Fig. 3.

The c-shocks are weak solutions of (2) of the form

$$(s,c)(x,t) = \begin{cases} (s^L, c^L), & x/t < \sigma \\ (s^R, c^R), & x/t > \sigma, \end{cases} \tag{6}$$

where σ denotes the shock speed. In order to obtain a unique solution of the Riemann problem, the c-shocks must satisfy both the Rankine–Hugoniot condition and an entropy condition, cf. [18]. The computation of theses shocks is described in [16].

3.3. Approximate solutions of the Riemann problem. In the front tracking method we need an approximation of the solution of the Riemann problem represented as a piecewise constant function of $\xi = x/t$. The accuracy of the approximation is specified by a discretization parameter δ. The approximation to the solution of the Riemann problem is generated by a modification of the procedure described in [16]. The only difference is that the rarefaction waves are computed with an accuracy of

order δ. The shocks are computed in exactly the same manner as described in [16]. Also the structure of the Riemann solver is the same.

Since both the structure of the Riemann solver and the computation of the shock waves are as described in [16], we only need to present the approximation of the simple rarefaction waves. We remark that each elementary wave is computed separately.

The piecewise constant approximation of a rarefaction wave, considered as a function of ξ, consists of small inadmissible shocks. The approximation is computed in two steps; first we compute an approximation to the solution in the state space, then this approximation is regarded as a function in the $x-t$ space, and "small" waves are removed. More precisely, a discontinuity is removed if the speed of the discontinuity is closer than δ to the speed of the previous discontinuity. We remark that this method of data reduction for the approximation of the solution of the Riemann problem does not reduce the complexity of the solution. This is due to the monotonicity of the rarefaction waves (cf. [18]).

We shall describe the computation of the state space approximation of the c and the s-rarefaction waves, and we start with the latter. We recall that the s-rarefaction waves traverse paths in the state space where the c-vector is constant. Hence these waves are scalar and they can be approximated in a conservative manner.

Suppose the state (s^L, c) can be connected to the state (s^R, c) with a single s-rarefaction. Let $M = 1 + \text{trunc}(|s^R - s^L|/\delta)$ and let $\xi^0 = -\infty$, $\xi^{M+1} = \infty$. Furthermore, we define

$$s^k = s^L + \frac{k-1}{M}(s^R - s^L), \qquad k = 1, \cdots, M+1,$$

and

$$\xi^k = \frac{f(s^{k+1}, c) - f(s^k, c)}{s^{k+1} - s^k}, \qquad k = 1, \cdots, M.$$

Then the state space approximation, (s_δ, c_δ), to the single s-rarefaction wave can be defined by

$$(s_\delta(\xi), c_\delta(\xi)) = (s^k, c) \quad \text{for} \quad \xi \in [\xi^{k-1}, \xi^k), \qquad k = 1, \cdots, M+1.$$

The c-rarefaction waves are approximated in state space by integrating the ordinary differential equation (5) by Euler's explicit method with a steplength of order δ. Define $s^1 = s^L$ and $c^1 = c^L$. Writing the ordinary differential equation as

$$\frac{dc_i}{ds} = \phi(s, c),$$

where

$$\phi(s, c) = \frac{\lambda^i - \lambda^s}{\frac{\partial f}{\partial c_i} + \sum_{j \in J} \frac{\partial f}{\partial c_j} \frac{dg_{i,j}}{dc_i}},$$

we compute

$$\begin{aligned}
s^{k+1} &= s^k + \Delta s, \\
c_i^{k+1} &= c_i^k + \phi(s^{k+1}, c^k)\Delta s, \\
c_j^{k+1} &= g_{i,j}(c_i^{k+1}) \qquad j \in J, \\
c_j^{k+1} &= c_j^L \qquad \text{for } j \neq i \text{ and } j \notin J,
\end{aligned}$$

when $|\phi(s^k, c^k)| < 1$. Here Δs denotes the steplength, and we recall that J denotes the set of indices corresponding to the c-components involved in the c-rarefaction wave in addition to the c_i-component. In our implementation, we have used $\Delta s = \delta(0.005 + |\phi(s^k, c^k)|)$. This choice of steplength is motivated by the fact that the rarefaction curves have a very large curvature in the neighborhood of the transition curve where $\phi \equiv 0$. If $|\phi(s^k, c^k)| \geq 1$, we compute

$$\begin{aligned} c_i^{k+1} &= c_i^k + \Delta c, \\ c_j^{k+1} &= g_{i,j}(c_i^{k+1}) \qquad j \in J, \\ c_j^{k+1} &= c_j^L \quad \text{for } j \neq i \quad \text{and} \quad j \notin J, \\ s^{k+1} &= s^k + \Delta c / \phi(s^k, c^{k+1}), \end{aligned}$$

where we have used the steplength $\Delta c = \delta$. The speed of each discontinuity is defined by

$$\xi^k = \lambda^i(s^k, c^k).$$

When computing this approximation to a c-rarefaction wave from (s^L, c^L) to (s^R, c^R), where s^L, c^L, c^R are given, the adaptive Euler iteration is performed until we have passed c^R. Then linear interpolation is used to compute s^R.

The procedure described above generates a fairly accurate piecewise constant approximation, (s_δ, c_δ), to a rarefaction wave. In order to increase the efficiency of the front tracking method, a data reduction strategy is applied to this approximation. We apply data reduction separately to each of the elementary waves. In this way we only reduce the representation of monotone parts of the solution of the Riemann problem.

We have implemented the data reduction in the following way: the first and the last discontinuity of the approximation of a single rarefaction wave are kept as they are. These discontinuities are essential in order to maintain the complexity of the solution. Then, recursively, starting with the second discontinuity, each discontinuity which has a speed that is closer than δ to the speed of the previous discontinuity, is removed.

Having performed the data reduction, we are left with a sequence $(\bar{s}^k, \bar{c}^k)$ approximating the rarefaction curve in the state space. In order to obtain a piecewise constant approximation of the rarefaction wave as a function of ξ, the speed of the discontinuities must be computed again. For the approximation to the s-rarefaction waves, we can compute the speed such that these waves are approximated in a conservative manner. This is done by defining

$$\bar{\xi}^k = \frac{f(\bar{s}^{k+1}, c) - f(\bar{s}^k, c)}{\bar{s}^{k+1} - \bar{s}^k}.$$

The speed of the c-rarefaction waves are defined by

$$\bar{\xi}^k = (\lambda^i(\bar{s}^k, \bar{c}^k) + \lambda^i(\bar{s}^{k+1}, \bar{c}^{k+1}))/2.$$

The approximation of the c-rarefaction waves is not conservative. Small errors in the mass-balance are introduced in this approximation. The numerical experiments clearly indicate that these errors remain small throughout the simulation.

We have also implemented a fourth order Runge–Kutta iteration to integrate the ordinary differential equation. This method has, however, two drawbacks; firstly,

the computation of the rarefaction curve close to the transition curve is unstable in the sense that the approximation leaves the state space when using a coarse grid. Secondly, even if these instabilities do not occur, the overall computational cost is increased compared with the simpler Euler method.

3.4. Small waves. We shall conclude this section by describing how to compute an approximation to a Riemann problem consisting only of one single discontinuity. This approximation is used in the front tracking method when the "distance" in the state space from the left state to the right state is smaller than δ. To simplify the notation, we describe this approximation applied to a general system of conservation laws,

$$u_t + F(u)_x = 0.$$

Here $u \in \mathbb{R}^n$ and $F : \mathbb{R}^n \to \mathbb{R}^n$ is a smooth function with components $F_i, i = 1, \cdots, n$. Let

$$u(x,0) = \begin{cases} u^L, & x < 0 \\ u^R, & x > 0. \end{cases} \tag{7}$$

Suppose that $u^L \approx u^R$, then an approximation to the solution of the Riemann problem consisting of only one discontinuity can be defined by computing the best shock speed in a least square sense. Define

$$p(\sigma) = \sum_{i=1}^{n} \left\{ \sigma(u_i^R - u_i^L) - (F_i(u^R) - F_i(u^L)) \right\}^2.$$

The speed $\bar{\sigma}$ of the single discontinuity is uniquely defined by the equation $p'(\bar{\sigma}) = 0$, and then the approximation can be written in the form

$$u_\delta(x,t) = \begin{cases} u^L, & x/t < \bar{\sigma} \\ u^R, & x/t > \bar{\sigma}. \end{cases} \tag{8}$$

We remark that this approximation is used in the front tracking method only when the left state of a Riemann problem is very close to the right state. This approximation is introduced because the structure of the Riemann problem solver is rather complicated, even for "small" data. Thus, it is desirable to be able to compute a rough approximation to such problems efficiently. The fastest way of solving the small problems is, of course, to ignore them. However, numerical experiments strongly indicate that by ignoring these problems, relatively large errors in the mass-balance may occur. These errors are reduced considerably by using the approximation described above.

4. The front tracking scheme. We will describe the application of a front tracking scheme to (2). The scheme was first used as a computational method in [22], where it was applied to the classical equations of gas dynamics. For this system the scheme compared favorably with the random choice scheme and the Lax–Friedrichs method. The equations of gas dynamics are, however, especially well suited for this front tracking scheme since the solution of the Riemann problem for this system is fairly simple and has been carefully studied; very fast and reliable Riemann problem solvers are available.

Assume that the initial function is piecewise constant with a finite number of jumps. This initial function defines a sequence of Riemann problems. For each Riemann problem, we compute an approximate solution represented as a piecewise constant function of x/t as described in the previous section. The approximate solution

of the Riemann problems defines a family of discontinuities emanating from the origin of each problem, and between these discontinuities the solution is constant. The discontinuities define an approximate solution of the global Cauchy problem until they interact. When two discontinuities interact (i.e., collide), they define a Riemann problem. An approximate solution to this Riemann problem is computed as explained above. This defines the approximate solution in the vicinity of the collision point after the collision. Then the next collision is handled similarly, and so on.

In order to present a more precise algorithm, we will need some data structure. We first define a front:

$$\textbf{Front} : (s^L, c^L),\ (s^R, c^R),\ \sigma,\ (x_0,\ t_0),\ (x_1, t_1),\ Nx,\ Px,\ Nt.$$

Thus a front is an object containing the left state (s^L, c^L) and the right state (s^R, c^R) of a discontinuity. The discontinuity starts in (x_0, t_0) and its speed is given by σ. The time of interaction of the discontinuity with its right-hand side neighbor is given by (x_1, t_1). The fronts are organized in two lists; one X-list and one T-list. In the X-list the fronts are placed from left to right (left $\sim -\infty$, right $\sim \infty$), by using the Nx (Next x) and the Px (Previous x) pointers. The T-list organizes the fronts with respect to collision time. The front with the smallest collision time, t_1, is always the first element of the T-list. This front has a Nt (Next t) pointer to the front with the second smallest collision time, and so on. Suppose that P is a pointer to one of the objects in the X-list. Then $P.s^L$ denotes the value of s^L in the object at which P is pointing; similarly for the rest of the real variables in the front object. Furthermore, $P.Nx$ points at the next front in the X-list; similarly for $P.Px$ and $P.Nt$. All the front objects are stored only once, hence the X-list and the T-list use the same objects.

We remark that the data structure applied here is slightly different from the structure used in [22]. In particular, here we store both the left and the right state of a discontinuity. This is of course not necessary, but it slightly simplifies the implementation.

4.1. The algorithm. The algorithm consists of two parts; an initialization part and an iteration part. In the initialization part we assume that the initial function is represented as a piecewise constant function. There are several ways to construct a piecewise constant approximation to a function, one of which is described in [22]. In the present implementation, we have simply divided the interval $[x^L, x^R]$ (cf. §2) into N equally sized intervals and defined the constant value on each interval to be the value of the initial function evaluated at the center of the interval. Furthermore, we have defined $\delta = 1/N$, thus the accuracy of the method is related to the accuracy of the initial approximation. In the iteration part, we compute an approximate solution to the Cauchy problem from $t = 0$ to $t = t_{\max}$.

Algorithm

INITIALIZATION

- Solve the Riemann problems defined by the piecewise constant initial function.
- Store the discontinuities as front objects in the X-list. If a discontinuity will collide with its right-hand neighbor, then the collision time t_1 and place x_1 must be computed. If not, we let $t_1 = \infty$.
- Organize the fronts of the X-list in a T-list according to increasing values of collision time t_1, using the Nt pointers.

ITERATION

WHILE (T-list not empty) and ($T.t_1 < t_{\max}$) DO

- $L := T$; $R := rightmost(L)$; $q := |R.s^R - L.s^L| + \sum_{j=1}^{n} |R.c_j^R - L.c_j^L|$;

- IF $q < \delta/1000$ THEN (cf. Fig. 4)
  - IF $L.Px.\sigma > R.Nx.\sigma$ THEN
    Compute $L.Px.x_1$, $L.Px.t_1$ and update the T-list.
  - ENDIF ELSE $L.Px.t_1 := \infty$;
  - Remove L and R from the X and T-list.
- ELSEIF $q < \delta$ THEN (cf. Fig. 5)
  - $P.s^L := L.s^L$; $P.s^R := R.s^R$; $P.c^L := L.c^L$;
    $P.c^R := R.c^R$; $P.x_0 := L.x_1$; $P.t_0 := L.t_1$;
  - Compute $P.\sigma$ as described in (8)
  - Replace L and R by P in the X-list.
  - IF $P.Px.\sigma > P.\sigma$ THEN
    Compute $P.Px.x_1$, $P.Px.t_1$ and update the T-list.
  - ENDIF ELSE $P.Px.t_1 := \infty$;
  - IF $P.\sigma > P.Nx.\sigma$ THEN
    Compute $P.x_1$, $P.t_1$ and update the T-list.
  - ENDIF ELSE $P.t_1 := \infty$;
- ELSE (cf. Fig. 6)
  - Compute an approximate solution of the Riemann problem with left state $(L.s^L, L.c^L)$ and right state $(R.s^R, R.c^R)$.
  - Remove the fronts L and R from the X and T-list and include the fronts generated by the solution of the Riemann problem in the X-list. Let $\bar{L}$ and $\bar{R}$ be pointers to the first and the last wave of the solution of the Riemann problem.
  - IF $\bar{L}.Px.\sigma > \bar{L}.\sigma$ THEN
    Compute $\bar{L}.Px.x_1$, $\bar{L}.Px.t_1$ and update the T-list.
  - ENDIF ELSE $\bar{L}.Px.t_1 := \infty$;
  - IF $\bar{R}.\sigma > \bar{R}.Nx.\sigma$ THEN
    Compute $\bar{R}.x_1$, $\bar{R}.t_1$ and update the T-list.
  - ENDIF ELSE $\bar{R}.t_1 := \infty$:
- ENDIF
- $T := T.Nt$

ENDWHILE

The function *rightmost*(L) used in the algorithm returns a pointer to a front on the right-hand side of L. Usually it will return $L.Nx$, but if three ore more fronts collide, the function *rightmost*(L) will return the rightmost front involved in the collision.

As mentioned above, this scheme is not conservative. Small errors in the mass balance are introduced by the approximation of c-rarefaction waves as small shocks. We remark that since in general rarefaction curves and shock curves do not coincide, the approximation of a rarefaction wave by a number of small shocks must be nonconservative. However, for the class of systems with coinciding rarefaction and shock curves, cf. Temple [26], the approximation will be conservative. Then, for any $\delta > 0$, the entire approximation generated by the front tracking scheme will be a weak solution of the initial value problem having a piecewise constant initial function. We also remark that in (2) with all adsorption functions set identically equal to zero, the rarefaction curves and the shock curves coincide. In fact, the c-rarefaction waves degenerate to contact discontinuities when the adsorption functions are neglected, cf. Isaacson [14]. In this case the front tracking scheme is conservative.

Also the approximation of the solution of the Riemann problem with $u^L \approx u^R$, is nonconservative. This approximation is, however, much better than simply ignoring these waves. It is also much faster than computing an approximation by the Riemann problem solver. A further discussion of the issue of mass balance will be given in the next section.

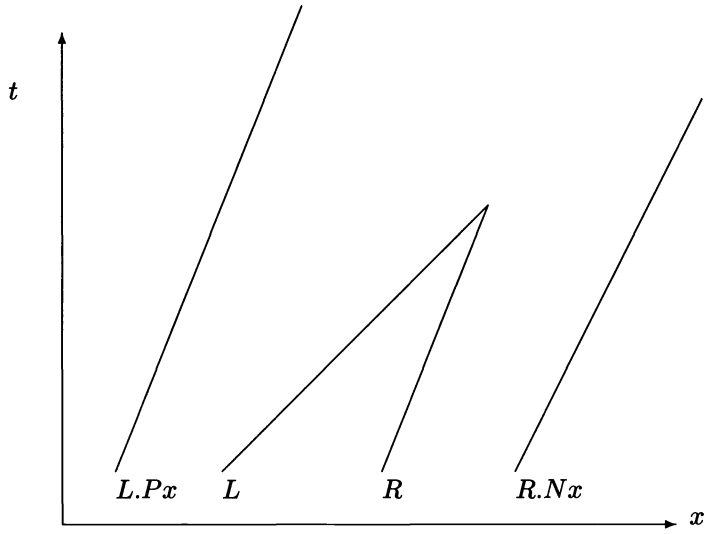


FIG. 4. *The "distance" from* $(L.s^L, L.c^L)$ *to* $(R.s^R, R.c^R)$ *is smaller than* $\delta/1000$.

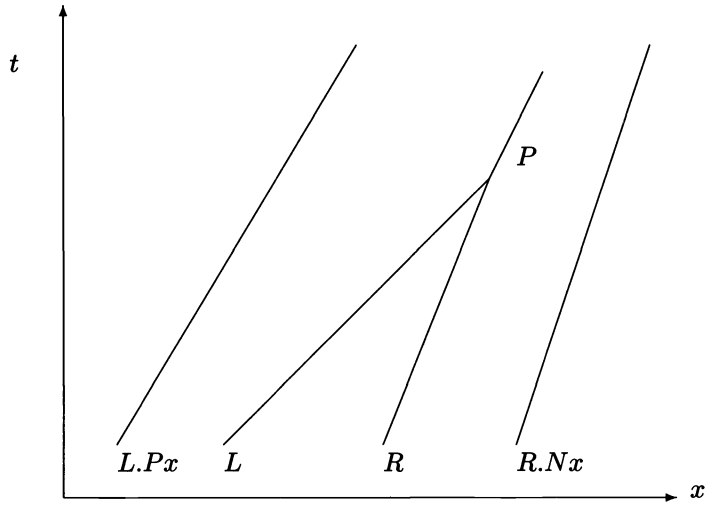


FIG. 5. *The "distance" from* $(L.s^L, L.c^L)$ *to* $(R.s^R, R.c^R)$ *is larger than* $\delta/1000$ *and smaller than* δ.

5. Examples. We have implemented and tested the front tracking method described above on a Sun4/260. In order to compare the scheme with standard methods, we have implemented the random choice scheme, cf. [9], [11], and the Godunov scheme, cf. [11]. The random choice scheme was implemented using a van der Corput sequence of random numbers, cf. Colella [5]. For (2) satisfying the assumption listed in §2, the Godunov scheme is identical with the upwind scheme, cf. [16] for details concerning the implementation of this scheme.

We will report the results of two test problems. In the first problem we study the case of one single polymer component, i.e., $n = 1$. This example illustrates both the

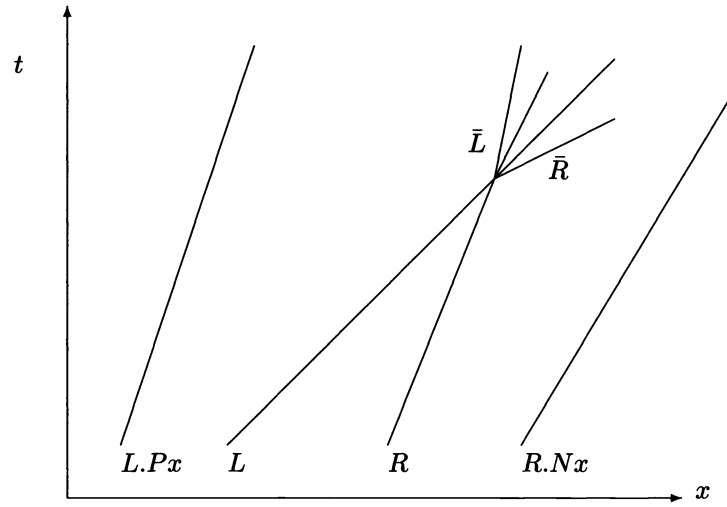


FIG. 6. *The "distance" from* $(L.s^L, L.c^L)$ *to* $(R.s^R, R.c^R)$ *is larger than* δ.

approximation of smooth parts of the solution and the fact that strong discontinuities can develop from a "smooth" initial function.

In the second example we consider an initial value problem which is particularly interesting from an engineering point of view. The problem models a situation where water containing a "polymer slug" is used to displace the oil in a reservoir. Since the polymer is quite expensive, it is important to be able to design an optimal "slug." To do this, a reliable simulator is indispensable.

In both our experiments we have used the fractional flow function

$$f(s, c_1, \cdots, c_n) = \frac{s^2}{s^2 + (1-s)^2(\frac{1}{2} + \frac{1}{4}\sum_{i=1}^{n} c_i)}$$

and the adsorbtion functions

$$a_i(c_1, \cdots, c_n) = \frac{c_i}{1 + (c_i - 1)\beta_i} \qquad i = 1, \cdots, n,$$

where $\beta_i = (i/(n+1))^2$.

For the random choice scheme and the upwind scheme we have applied a uniform mesh with $\Delta x = 1/N$. The initial function is approximated on each grid block by evaluating the initial function in the center of the block. We have used a Courant number of 0.9 for both methods. Observe that the usual time step restriction for the random choice scheme can be relaxed when only positive wave speeds are involved.

For the front tracking method we have used the same initial approximation as we used for the standard methods, and the approximation parameter δ is chosen to be $1/N$.

We compare the methods with respect to computational cost, L^1-error and conservation of mass. Reference solutions of the problems have been computed using the front tracking method with $N = 500$. The relative error is measured in the following

manner

$$L^1\text{-error}\% = \left(\frac{\|s - s_\Delta\|}{\|s\|} + \sum_{i=1}^{n} \frac{\|c_i - (c_i)_\Delta\|}{\|c_i\|} \right) \frac{100}{n+1}$$

where (s, c) denotes the reference solution and (s_Δ, c_Δ) denotes the approximate solution generated by one of the methods. The L^1-norm at time t is defined by

$$\|s\| = \int_0^{1+tV} |s(x,t)|\, dx,$$

where V is the largest speed occurring in the problem, i.e.,

$$V = \max_{(s,c)\in\mathcal{S}} (\lambda^s, \lambda^1, \cdots, \lambda^n).$$

The approximate solutions are computed from $t = 0$ to $t_{\max} = 3$.

We calculate the relative error in the mass balance for the front tracking scheme and the random choice scheme. This is done by adding absolute values of the relative error in the mass balance for each of the conservative variables.

5.1. Example 1. In the first test problem we have used $n = 1$ and the initial function

$$s(x,0) = c_1(x,0) = \begin{cases} 1/2, & x < 0 \\ 1/2 + x(1-x), & x \in [0,1] \\ 1/2, & x > 1. \end{cases}$$

In Fig. 7 the s-component of the approximate solutions is represented by dotted lines. Since the approximations are piecewise constant, we have plotted them as such, although a piecewise linear interpolation would seem more accurate. The reference solution is represented by a solid line. The CPU time given in the figures for each of the methods are measured in seconds. We observe that the front tracking method and the upwind scheme produce fairly accurate solutions using very little computing time. The random choice solution is also acceptable, but it is very time consuming to compute this solution. In Tables 1, 2, and 3, the L^1-errors and the CPU time of each method for several values of N are presented. For the front tracking method and the random choice scheme we also present the relative error in the mass balance. Note that the upwind scheme is conservative.

We observe that in order to reach an accuracy of 0.12 percent, the front tracking method uses 47.1 CPU seconds. The same accuracy is reached by the random choice scheme using 2228.75 CPU seconds, and the upwind scheme uses more than 258 seconds to reach this accuracy. For this problem the error in the mass balance is very small both for the front tracking method and the random choice scheme.

5.2. Example 2. In our second example we have used $n = 4$ polymer components and the initial functions

$$(s, c_1, c_2, c_3, c_4)(x,0) = \begin{cases} (0.9, 0.01, 0.01, 0.01, 0.01), & x < 0 \\ (0.9, 1, 1, 1, 1), & x \in [0, 0.2] \\ (0.1, 0, 0, 0, 0), & x > 0.2. \end{cases}$$

In Fig. 8 the s-component of the approximate solutions is plotted. The c_4 component is plotted in Fig. 9. The c_1, c_2, and c_3 components have a behavior similar

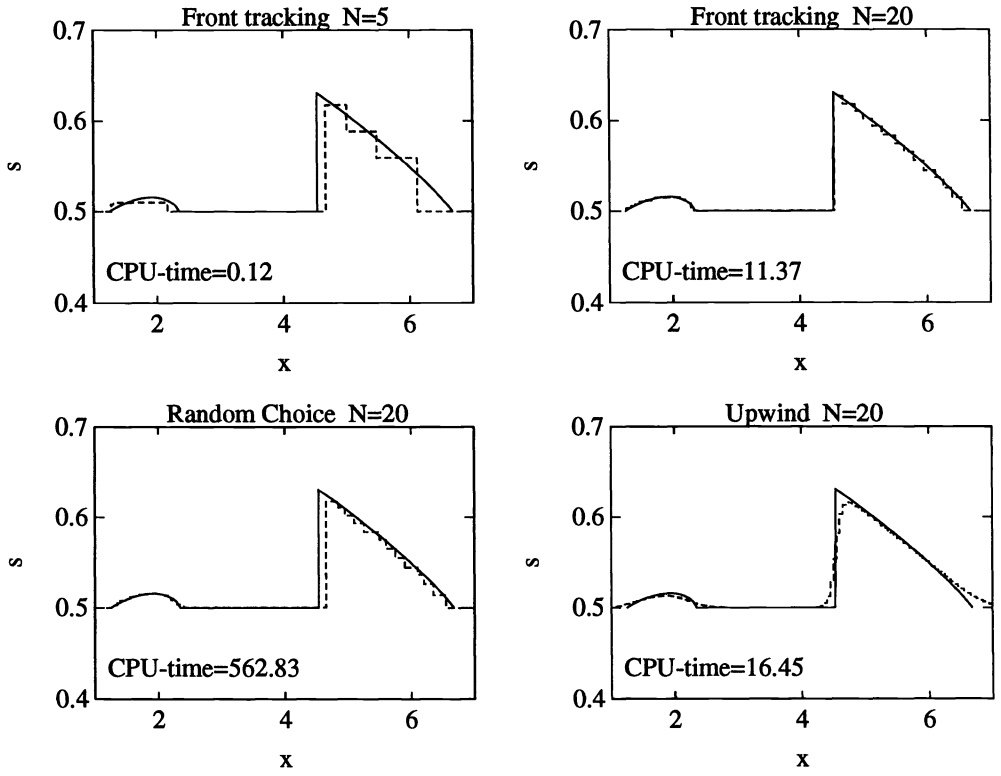


FIG. 7. *The s variable in Example* 1. *Solid line: reference solution; dotted line: approximate solution (piecewise constant).*

TABLE 1
Front tracking, Example 1.

N	L^1-error %	CPU time	Mass balance %
3	1.380	0.117	0.008
5	1.077	0.117	0.002
10	0.427	2.783	0.033
20	0.220	11.367	0.013
30	0.147	25.950	0.015
40	0.120	47.100	0.021
50	0.103	75.950	0.017

TABLE 2
Random choice, Example 1.

N	L^1-error %	CPU time	Mass balance %
5	1.099	30.667	0.462
10	0.365	147.917	0.093
20	0.342	562.834	0.153
30	0.186	1237.450	0.032
40	0.122	2228.750	0.027
50	0.087	3946.350	0.010

TABLE 3
The upwind scheme, Example 1.

N	L^1−error %	CPU-time
10	1.177	7.60000
20	0.706	16.316
40	0.408	37.033
80	0.233	91.700
160	0.138	258.050
320	0.088	806.934

to that of the c_4 component. Again we see from the figures that the front tracking scheme produces an accurate solution in a few CPU-seconds. From Tables 4, 5, and 6 we see that in order to reach an accuracy of 0.47 percent, the front tracking scheme uses about 5.2 CPU seconds. The random choice scheme uses more than 4635 CPU seconds and the upwind scheme uses more than 3659 CPU seconds to reach such an accuracy. Also for this test problem, the errors in the mass balance for the solutions generated by the front tracking method are negligible for all $N \geq 5$. But for the random choice scheme the relative error in the mass-balance was 3.2 percent for $N = 50$. This illustrates the well-known problems of mass conservation for the random choice scheme.

TABLE 4
Front tracking, Example 2.

N	L^1-error %	CPU time	Mass balance %
5	0.552	2.417	0.146
10	0.472	5.167	0.092
20	0.386	13.934	0.055
30	0.321	30.900	0.016
40	0.270	56.817	0.051
50	0.233	96.367	0.041
60	0.183	176.866	0.034

TABLE 5
Random choice, Example 2.

N	L^1−error %	CPU-time	Mass-balance %
5	2.527	36.167	11.059
10	2.154	124.017	5.509
20	1.084	509.000	8.969
30	0.681	1349.033	3.560
40	0.661	2665.767	4.219
50	0.487	4635.383	3.214

TABLE 6
The upwind scheme, Example 2.

N	L^1-error %	CPU time
10	5.360	10.516
20	4.280	27.134
40	3.215	86.834
80	2.244	278.433
160	1.471	1001.350
320	0.931	3659.766

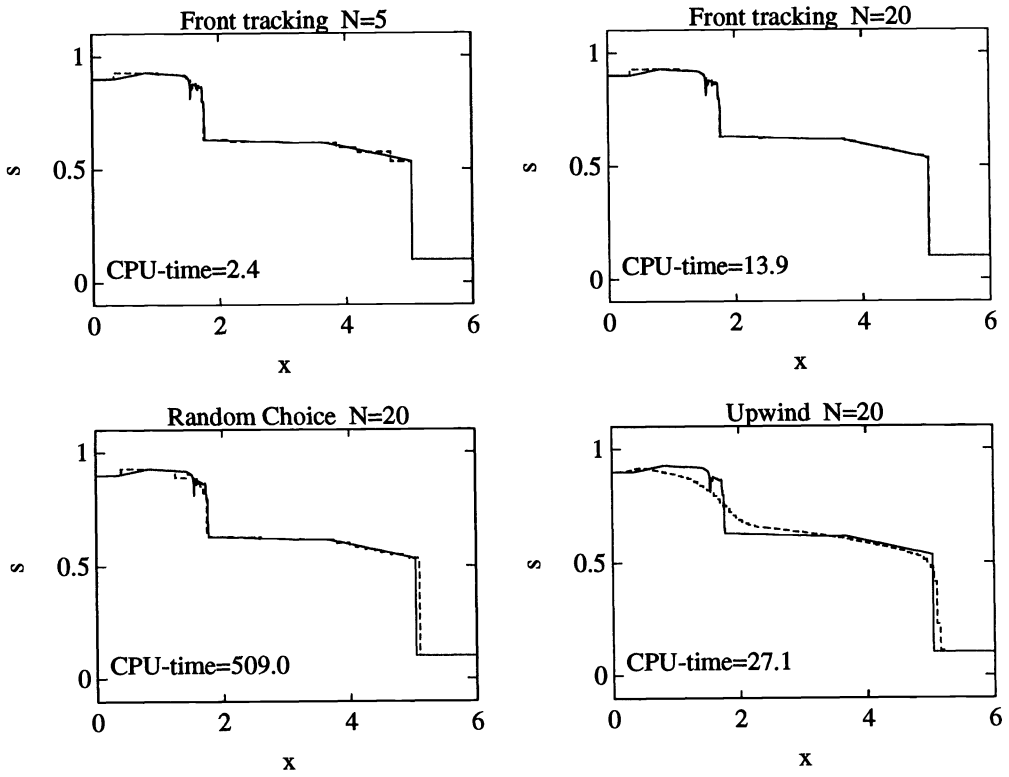


FIG. 8. *The s variable in Example 2. Solid line: reference solution; dotted line: approximate solution (piecewise constant).*

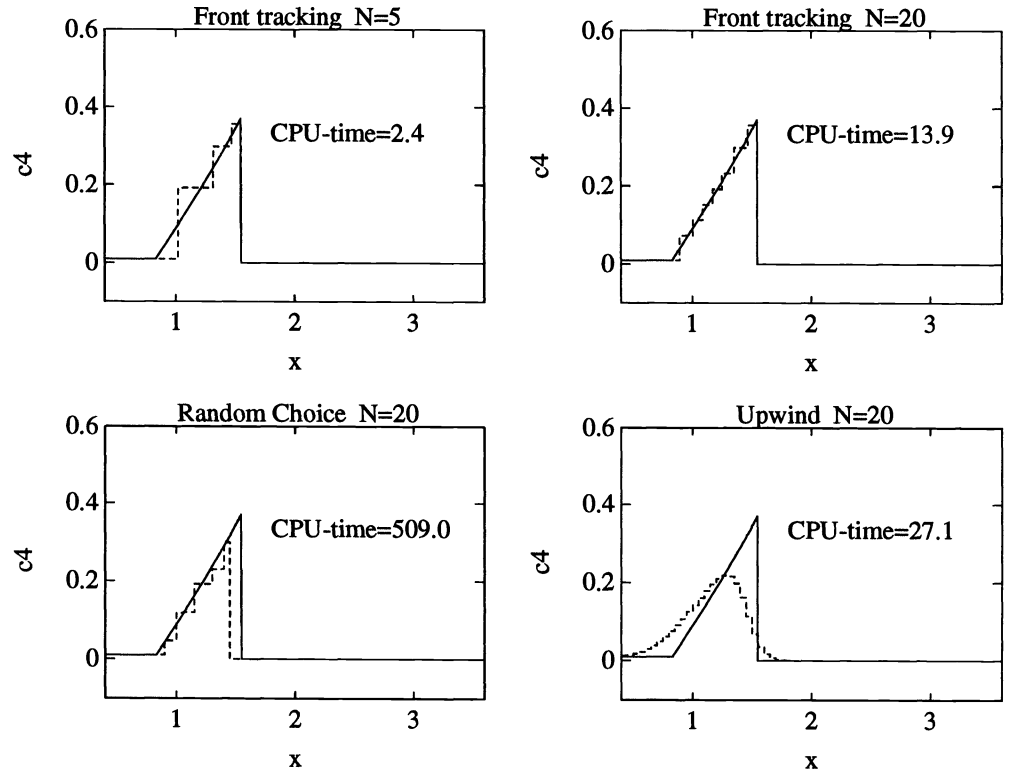


FIG. 9. *The c_4 variable in Example 2. Solid line: reference solution; dotted line: approximate solution (piecewise constant).*

6. Conclusion. We have implemented and tested a front tracking method for a nonstrictly hyperbolic system of conservation laws. In our test cases the method is faster than both the upwind scheme and the random choice scheme. The front tracking method generates a qualitatively good approximation using little CPU time, and it is well suited for computing the long time behavior of an initial value problem, cf. [22]. The method is nonconservative, but the relative error in the mass balance is small and typically less than 0.5 percent for all $N \geq 5$. In the case of coinciding shock and rarefaction curves, the method is conservative.

Both the random choice scheme and the front tracking method are based on the solution of Riemann problems. In the random choice scheme only one point on the solution path from the left to the right state is used to define the approximate solution on a grid block. Thus, most of the information from the solution of the Riemann problem is discarded. The basic idea of our front tracking method is to use all the information from the solution of the Riemann problem. This leads to an efficient scheme.

There are three major drawbacks to the front tracking method presented here. First, the method is restricted to the class of systems where the solution of the Riemann problem is known. This clearly reduces the applicability of the method. Second, the method is more complicated to implement than standard numerical methods for hyperbolic conservation laws. And third, generalizations to several space dimensions are highly nontrivial.

A version of the scheme was analyzed in [21] for strictly hyperbolic systems of conservation laws with "small" data. But the case of more general systems and "large" data needs further analysis from a mathematical point of view.

REFERENCES

[1] P. Charrier and B. Tessieras, *On front tracking methods applied to hyperbolic systems of nonlinear conservation laws*, SIAM J. Numer. Anal., 23 (1986), pp. 461–472.

[2] I-L. Chern and P. Colella, *A conservative front tracking method for hyperbolic conservation laws*, Lawrence Livermore National Laboratory, Livermore, CA, 1987, preprint.

[3] I-L. Chern, J. Glimm, O. McBryan, B. Plohr, and S. Yaniv, *Front tracking for gas dynamics*, J. Comput. Phys., 62 (1986), pp. 83–110.

[4] A. I. Chorin, *Random choice solutions of hyperbolic systems*, J. Comput. Phys., 22 (1976), pp. 517–533.

[5] P. Colella, *Glimm's method for gas dynamics*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 76–110.

[6] C. M. Dafermos, *Polygonal approximations of solutions of the initial value problem for a conservation law*, J. Math. Anal. Appl., 38 (1972), pp. 33–41.

[7] P. Daripa, J. Glimm, B. Lindquist, and O. McBryan, *Polymer floods: A case study of nonlinear wave analysis and instability control in tertiary oil recovery*, SIAM J. Appl. Math., 48 (1988), pp. 353–373.

[8] I. Gelfand, *Some problems in the theory of quasilinear equations*, Amer. Math. Soc. Transl. Ser., 2 (1963), pp. 295–381.

[9] J. Glimm, *Solutions in the large for nonlinear hyperbolic systems*, Comm. Pure Appl. Math., 18 (1965), pp. 697–715.

[10] J. Glimm, B. Lindquist, O. McBryan, and L. Padmanabhan, *A front tracking reservoir simulator, five-spot validation studies and the water coning problem*, in Frontiers in Applied Mathematics, Vol. 1, Society for Industrial and Applied Mathematics, Philadelphia, 1983.

[11] S. K. Godunov, *A finite difference method for the numerical computation of discontinuous solutions of the equations of fluid dynamics*, Math. USSR-Sb., 47 (1959), pp. 271–290.

[12] G. W. Hedstrom, *Some numerical experiments with Dafermos' method for nonlinear hyperbolic equations*, Lecture Notes in Math., Vol. 267, Springer-Verlag, Berlin, New York, 1972, pp. 117–138.

[13] J. M. Hyman, *Numerical methods for tracking interfaces*, Physica, 12D (1984), pp. 396–407.

[14] E. Isaacson, *Global solution of a Riemann problem for a nonstrictly hyperbolic system of conservation laws arising in enhanced oil recovery*, Rockefeller University, New York, NY, preprints.

[15] E. ISAACSON AND B. TEMPLE, *Analysis of a singular hyperbolic system of conservation laws*, J. Differential Equations, 65 (1986), pp. 250–268.

[16] T. JOHANSEN, A. TVEITO, AND R. WINTHER, *A Riemann solver for a two-phase multicomponent process*, SIAM J. Sci. Statist. Comput., 10 (1989) pp. 846–879.

[17] T. JOHANSEN AND R. WINTHER, *The solution of the Riemann problem for a hyperbolic system of conservation laws modelling polymer flooding*, SIAM J. Math. Anal., 19 (1988), pp. 541–566.

[18] ———, *The Riemann problem for multicomponent polymer flooding*, SIAM J. Math. Anal., 20 (1989), pp. 908–929.

[19] P. D. LAX, *Hyperbolic systems of conservation laws and the mathematical theory of shock waves*, CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 11, Society for Industrial and Applied Mathematics, Philadelphia, 1973.

[20] G. A. POPE, *The application of fractional flow theory to enhanced oil recovery*, Soc. Pet. Engrg. J., 20 (1980), pp. 191–205.

[21] N.H. RISEBRO, *A front tracking alternative to the random choice method*, preprint, Department of Mathematics University of Oslo, Oslo, Norway, 1989.

[22] N. H. RISEBRO AND A. TVEITO, *A front tracking method for conservation laws in one dimension*, Department of Mathematics University of Oslo, Oslo, Norway, 1990.

[23] J. SMOLLER, *Shock waves and reaction-diffusion equations*, Springer-Verlag, New York, 1982.

[24] B. K. SWARTZ AND B. WENDROFF, AZTEC: *A front tracking code based on Godunov's method*, Appl. Numer. Meth., 2 (1986), pp. 385–397.

[25] B. TEMPLE, *Global solution of the Cauchy problem for a class of* 2×2 *nonstrictly hyperbolic conservation laws*, Adv. in Appl. Math., 3 (1982), pp. 335–375.

[26] ———, *Systems of conservation laws with coinciding shock and rarefaction curves*, Contemp. Math., 17 (1983), pp. 143–151.

[27] A. TVEITO AND R. WINTHER, *Existence, uniqueness and continuous dependence for a system of hyperbolic conservation laws modelling polymer flooding*, SIAM J. Math. Anal., 22 (1991), pp. 905–933.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1420–1437, November 1991
© 1991 Society for Industrial and Applied Mathematics
011

THE PARALLEL MULTIPOLE METHOD ON THE CONNECTION MACHINE®*

FENG ZHAO† AND S. LENNART JOHNSSON‡

Abstract. This paper reports on a fast implementation of the three-dimensional nonadaptive Parallel Multipole Method (PMM) on the Connection Machine system model CM–2. The data interactions within the decomposition tree are modeled by a hierarchy of three-dimensional grids forming a pyramid in which parent nodes have degree eight. The base of the pyramid is embedded in the Connection Machine as a three-dimensional grid. The standard grid embedding feature is used. For 10 or more particles per processor the communication time is insignificant. The evaluation of the potential field for a system with 128k particles takes 5 seconds, and a system of a million particles about 3 minutes. The maximum number of particles that can be represented in 2G bytes of primary storage is $\sim$ 50 million. The execution rate of this implementation of the PMM is at about 1.7 Gflops/sec for a particle-processor-ratio of 10 or greater. A further speed improvement is possible by an improved use of the memory hierarchy associated with each floating-point unit in the system.

Key words. N-body algorithm, parallel computing, Parallel Multipole Method, hypercube architecture, grid embedding, binary-reflected Gray code

AMS(MOS) subject classifications. 65C20, 70-08, 70F10, 70F15

1. Introduction.

Many physical phenomena can be modeled as an ensemble of N particles interacting in a gravitational (or Coulombic) field, known as the N-body problem. Efficient computation of the gravitational (or Coulombic) potentials and forces exerted on each other by N particles has been of great interest. Recently, a class of fast algorithms known as tree-like particle methods has been developed. These methods, notably the Barnes–Hut tree algorithm [1] and the Multipole Method [8], [21], compute the potentials (or the forces) by partitioning them into two parts:

$$\Phi_{total} = \Phi_{near-field} + \Phi_{far-field}, \tag{1}$$

where $\Phi_{near-field}$ is the potential due to nearby particles and $\Phi_{far-field}$ is the potential due to faraway particles. Since $\Phi_{near-field}$ can be computed directly and relatively fast, we are mostly concerned with the fast computation of $\Phi_{far-field}$.

The tree-like particle methods compute $\Phi_{far-field}$ by recursive decomposition of the computational domain. The decomposability of $\Phi_{far-field}$ is the direct consequence of the assumed additivity of $\Phi_{far-field}$ and the spatial locality in the particle distribution. The same tree data structure can be used for both the Barnes–Hut and the Greengard–Rokhlin algorithms.

An example of additive potential fields is a Newtonian particle system. The Newtonian gravitational field $\overline{E}_{x_0}(x)$ on a particle with unit mass at location $\overline{x}$ due to a particle with mass m at location $\overline{x}_0$ is given by

$$\overline{E}_{x_0}(x) = G\frac{m}{\| \overline{x} - \overline{x}_0 \|^3}(\overline{x} - \overline{x}_0),$$

where $\| \overline{x} - \overline{x}_0 \|$ is the Euclidean distance between locations $\overline{x}$ and $\overline{x}_0$.

* Received by the editors January 10, 1990; accepted for publication (in revised form) January 23, 1991.

† Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

‡ Thinking Machines Corporation, 245 First Street, Cambridge, Massachusetts 02142, and the Division of Applied Sciences, Harvard University, Cambridge, Massachusetts 02138.

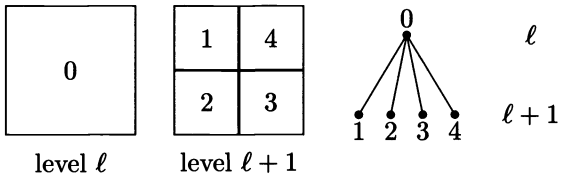


FIG. 1. *Recursive domain decomposition and the decomposition tree in two spatial dimensions.*

Greengard and Rokhlin [8] have shown that the Multipole Method solves the N-body problem in linear time on sequential machines. The focus of this paper is the implementation of the nonadaptive Multipole Method on the Connection Machine. We first review the method and its computational structure, then discuss the efficient implementation of the required communication, and conclude with experimental results on performance.

2. The Parallel Multipole Method. The Parallel Multipole Method (PMM) is the parallel version of the Multipole Method. It achieves significant speedup on parallel machines [21], [7]. In [21] we presented a data parallel version of the PMM, and showed that its parallel complexity is $O(\log N)$, asymptotically. The asymptotic growth rate was verified by an implementation on the Connection Machine.

2.1. Decomposition tree. The computational domain of the Multipole Method is a square in two spatial dimensions and a cube in three spatial dimensions. The domain is recursively divided into subdomains, i.e., subsquares in two dimensions and subcubes in three dimensions. A domain subject to decomposition is the *parent* of its subdomains after the decomposition. These subdomains are the *children* of the parent domain. The recursive decomposition process continues until some prespecified condition is met. Clearly, there is no reason to further decompose a subdomain containing only one, or no particles. However, as has been shown both analytically [6] and in several implementations [21], [7], [9] the direct method is faster than the Multipole Method for particle systems with sufficiently few particles. To minimize the computational complexity the decomposition of a subdomain should stop at a level where, for instance, using the direct method instead yields a lower computational complexity. Subdomains that are not further decomposed are *leaves*.

The recursive decomposition is most conveniently represented by a *decomposition tree*, within which a node corresponds to a subdomain in the decomposition process. The root of the tree is the original domain under consideration. Each parent has four children in two dimensions, and eight children in three dimensions. With a uniform distribution of particles over the domain the height of the decomposition tree is at most $\log_d N$, where $d = 4$ for two spatial dimensions, and $d = 8$ for three spatial dimensions. The root is at level 0, and the leaves at level $h \leq \log_d N$. The implementation of the PMM described here is nonadaptive. The complexity estimates, as well as the implementation, assume a uniform distribution of particles in the domain. The decomposition tree in two dimensions is illustrated in Fig. 1.

The Multipole Method involves computations at all levels of the decomposition tree. In the nonadaptive case the decomposition tree is balanced. This tree is represented by a hierarchy of grids, or a pyramid. A point within the grid at level ℓ represents a subdomain corresponding to a node of the decomposition tree at level ℓ. In three dimensions the grid size at level ℓ is $2^\ell \times 2^\ell \times 2^\ell$. The computations consist

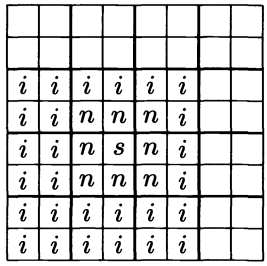


FIG. 2. *The near-field and the interactive-field in two dimensions.*

of interactions between parent/child nodes in the pyramid, i.e., between adjacent grid levels, and among grid points representing subdomains within a local neighborhood at each level of the pyramid. For the Multipole Method it is advantageous to distinguish among three regions relative to each grid point at each grid level [8], [6], [21], [19]: the *near-field*, the *interactive-field*, and the *far-field*. The *near-field* consists of those subdomains that share a boundary, or an edge, or a corner with the considered subdomain. In two dimensions the near-field consists of eight subdomains, i.e., eight adjacent grid points. In three dimensions the number of subdomains (grid points) in the near-field is 26. The near-field is represented by a 9-point stencil in two dimensions and by a 27-point stencil in three dimensions. The *far-field* of a subdomain is the entire domain excluding the subdomain and its near-field. In three spatial dimensions the number of subdomains (grid points) in the far-field is $2^{3\ell} - 27$ for a subdomain at level ℓ, $\ell > 1$. The *interactive-field* of a subdomain is the part of the far-field that is contained in the near-field of its parent. The number of subdomains in the interactive-field in two dimensions is $27 = 6^2 - 3^2$, and in three dimensions the interactive field consists of $189 = 6^3 - 3^3$ subdomains. The concepts of near-field and interactive-field are illustrated in two dimensions in Fig. 2 [21]. For the subdomain s, its near-field is the union of subdomains labeled n and its interactive-field is the union of those labeled i.

2.2. The algorithm. The Multipole Method can be specified in terms of three functions G, Φ and Ψ, three translation operators T_1, T_2, and T_3, and a set of recursive equations [8], [21], [19]. G is the potential function in explicit Newtonian formulation. T_1, T_2, and T_3 are higher order translation operators on the functions Φ and Ψ[1]

$$T_1 : (\Phi, y^{(0)}) \rightarrow (\Phi, y'^{(0)})$$

$$T_2 : (\Phi, y^{(0)}) \rightarrow (\Psi, y'^{(0)})$$

$$T_3 : (\Psi, y^{(0)}) \rightarrow (\Psi, y'^{(0)}),$$

where $y^{(0)}$ and $y'^{(0)}$ are the reference centers for the corresponding functions Φ and Ψ.

Φ represents the far-field and Ψ the local field. More precisely, Φ_i^ℓ is the contribution to the potential field in the far-field region of subdomain i at level ℓ due

[1] T_1, T_2, and T_3 are defined in Lemmas 3.2.1 through 3.2.3 in [21].

to the particles in subdomain i, as computed by a multipole expansion centered at the center of subdomain i. Ψ_i^ℓ represents the contribution to the potential field in subdomain i at level ℓ due to particles in the far-field region of subdomain i. For the computation of Ψ_i^ℓ it is convenient to introduce $\tilde{\Psi}_i^\ell$, which represents the contribution to the potential field in subdomain i at level ℓ due to the particles in the far-field of its parent domain. In order to obtain Ψ_i^ℓ, it is necessary to add to $\tilde{\Psi}_i^\ell$ the contributions of the particles in the interactive-field of subdomain i at level ℓ. The interactive-field of subdomain i at level ℓ is part of its far-field.

The Multipole Method computes $\Phi_{far-field}$ of (1) in two distinct phases. The computations performed at intermediate levels of the decomposition tree reduce the far-field interactions via an upward pass and a downward pass. The tree is traversed twice. In the upward pass the far-field interactions Φ_i^ℓ are computed by shifting the multipole expansions of child nodes to the center of the parent node, and adding the result. The downward pass through the tree distributes the results of far-field interactions, aligns them properly, and combines them to form the local expansion Ψ. The computation of $\Phi_{near-field}$ of (1) performed at leaf nodes reduces near-field interactions. In the recursive formulation by Katzenelson [19], the upward pass consists of the computation

(2) Compute Φ_i^h for all nodes i at the leaf level h

$$\Phi_n^\ell = \sum_{i \in \{children(n)\}} T_1(\Phi_i^{\ell+1}), \tag{3}$$

and the downward pass consists of the computation

$$\tilde{\Psi}_i^\ell = T_3(\Psi_{parent(i)}^{\ell-1}) \tag{4}$$

$$\Psi_i^\ell = \tilde{\Psi}_i^\ell + \sum_{j \in \{interactive-field(i)\}} T_2(\Phi_j^\ell). \tag{5}$$

The first term in Ψ_i^ℓ computes the local field due to the particles in the far-field of the parent node by shifting the center of the expansion from the center of the parent domain to that of the current subdomain. The second term converts the far-field expansions to a local expansion.

At the end of this recursion it remains to compute the potential field due to particles in the near-field of subdomain i at level h. This contribution to the potential field is computed by a direct evaluation of the Newtonian interaction with nearby particles

$$\Phi_{i\ near-field}^h = \sum_{j \in \{near-field(i)\}} G_j(i). \tag{6}$$

The complete solution, therefore, is given by

$$\Phi_{i\ total}^h = \Psi_i^h + \Phi_{i\ near-field}^h.$$

In the upward pass (3), the shifting operations of T_1 for different subdomains within each level are independent of each other and can be performed concurrently, as long as the computation at levels below is completed. Similarly, the shifting operations of T_3 in the downward pass (4) can also be performed concurrently for all

subdomains at a given level. The conversion of the far-field multipole expansions to local expansions represented by the operator T_2 can be performed concurrently across all levels of the grids. The reduction operation in the upward pass can be performed concurrently not only for different parent nodes, but also for each parent node by performing the reduction in a binary-tree-like manner with trees of height two in two dimensions and height three in three dimensions. The distribution operation in the downward pass has the same characteristics with respect to concurrency as the reduction in the upward pass. The computations of (6) at different leaf nodes are entirely independent of each other, and can proceed concurrently. Moreover, the computations within any level of the grid are uniform across all grid points. The PMM exploits the high degree of independence and the regularity of the Multipole Method, and achieves significant speedup on a large number of particles with a large number of processors. It requires $O(\log N)$ time asymptotically for N particles distributed with a fixed number of particles per processor [7], [21].

The PMM algorithm performs the computations implied by (2), (3), (4), (5), and (6). Expression (2) is evaluated concurrently for all leaf nodes. In each such node a multipole expansion due to all particles within the leaf node is formed. The multipole expansions formed at the leaf nodes are shifted and reduced in parallel during the upward pass through the tree, as described by (3). The far-field interactions are then accumulated and distributed in parallel during the downward pass through the decomposition tree, as defined by (4) and (5). The far-field interaction for each of the particles is obtained by evaluating the multipole expansion from (5) at the particle position. The near-field interaction at the leaf nodes is computed by direct evaluation of the Newtonian gravitational field. This evaluation is performed concurrently for all leaf nodes. The sum of the far-field interaction and the near-field interaction gives the answer.

The algorithm for PMM on the Connection Machine is given in Fig. 3.

3. Embedding of hierarchical grids. The Connection Machine system model CM-2 has 64k bit-serial processors divided evenly among 4096 processor chips. These chips are interconnected as a 12-dimensional Boolean cube. The Connection Machine model CM-2 can also be equipped with hardware floating-point units (fpu's). Two processor chips share such a unit. Each Connection Machine processor is associated with 8k bytes of storage with 256k bit memory chips, or 32k bytes with 1M bit memory chips. The storage per floating-point unit is 64k or 256k 32-bit words. The total storage is 512M bytes, or 2G bytes. The programming systems on the Connection Machine support the notion of *virtual processors*. A program is written for a number of virtual processors consistent with the application. A virtual processor may, for instance, represent the data associated with a grid point and carry out all the computations associated with it. The virtual processors are mapped to *real* processors. The number of virtual processors assigned to each real processor is the *virtual processor ratio*. Virtual processors time-share a real processor. Each virtual processor assigned to a real processor is assigned a distinct segment of real processor memory.

Interprocessor communication can take place either by using the general routing facility, or via embedded grids. The Boolean cube network interconnecting the processor chips contains regular grids of up to twelve dimensions as subgraphs. Address encoding by a binary-reflected Gray code [20] provides a mechanism for grid emulation [18], [14]. The dimensionality and shape of the grid can be altered dynamically. The Gray code has the property that adjacent integers are at *Hamming* distance one, i.e., $Hamming(i, i \pm 1)=1$. By using this encoding successive integers are mapped into

(1) **global-initialization**
  set-up arrays for Φ, Ψ, and intermediate results;
    precompute difference vectors among tree nodes;
    precompute shifting arrays by theorem-3-2-2 in [21] for
      local-field-reduction;
    precompute mathematical tables.

(2) **far-field-reduction**
  init-expansion at leaf nodes
    forall nodes i at leaf-level h **do**
      compute Φ_i^h by theorem-3-2-1 in [21];
    end
  multigrid-reduction
    for level ℓ **from** h **to** 1 **step** -1 **do**
      forall nodes i at level ℓ **do**
        compute Φ_i^ℓ by lemma-3-2-1 in [21];
        reconfigure CM grid;
        reduce Φ_i^ℓ with + op;
      end
    end

(3) **local-field-reduction**
  multigrid-distribution
    for level ℓ **from** 1 **to** h **do**
      forall nodes i at level ℓ **do**
        compute $\sum_{j \in \{interactive-field(i)\}} T_2(\Phi_j^\ell)$ by lemma-3-2-2 in [21];
        compute $\tilde{\Psi}_i^\ell$ by lemma-3-2-3 in [21];
        compute Ψ_i^ℓ;
        distribute Ψ_i^ℓ with identity op;
        reconfigure CM grid;
      end
    end

(4) **local-interaction**
  forall nodes i at leaf-level h **do**
    for $j \in \{near - field(i)\}$ **do**
      for each particle in node i **do**
        for each particle in node j **do**
          compute $G_j(i)$;
        end
      end
    end
    compute $\Phi_{i\ near-field}^h$;
  end

(5) **final-evaluation**
  forall nodes i at leaf-level h **do**
    for each particle p in node i **do**
      evaluate Ψ_i^h at p;
      compute $\Phi_{i\ total}^h$ for p;
    end
  end

FIG. 3. *The Parallel Multipole Method on the Connection Machine.*

adjacent nodes of a Boolean cube. Each node of a Boolean cube of 2^n nodes has n neighbors. The address of a neighbor can be obtained by complementing a bit in the address.

For a lattice with several axes each axis can be encoded in a binary-reflected Gray code. The address space is partitioned with $\lceil \log_2 N_i \rceil$ bits assigned to the encoding of the N_i elements along axis i. All edges of the lattice are mapped to distinct cube edges. The separate binary-reflected Gray code encoding of each array axis results in an effective utilization of the nodes in the Boolean cube, if the length of the array axes is a power of two. In general, the utilization of the nodes in the Boolean cube offered by the Gray code encoding is $\lceil \log_2 \Pi_{i=1}^{d} N_i \rceil / \Pi_{i=1}^{d} \lceil \log_2 N_i \rceil$. For grids of arbitrary shape the utilization may be as low as $1/2^d$. Any embedding of grids into Boolean cubes preserving adjacency will have this processor utilization [10]. In order to increase the processor utilization for grids of arbitrary shape it is necessary to allow some grid edges to be mapped into paths of length greater than one. The *dilation* of an edge is the length of the path into which it is mapped, and the dilation of the embedding is the maximum dilation of any edge. The *expansion* of the embedding is the ratio between the number of nodes of the Boolean cube required for the embedding and the number of nodes in the grid being embedded. Any two-dimensional grid can be embedded into a Boolean cube with dilation two and minimum expansion [3]. Any three-dimensional grid can be embedded with at most dilation seven and minimal expansion [5]. Grids with k dimensions can be embedded with a dilation of at most $4k + 1$ and minimal expansion [4]. Several dilation two embeddings of two- and three-dimensional grids are given in [11], [12].

In the Connection Machine programming systems data represented by arrays configure the address space as a grid. Configuring the address space as a grid implies that the array indices are encoded in a binary-reflected Gray code. Each axis is encoded separately. The encoding is transparent to the programmer. The Connection Machine address field has three parts $(off\text{-}chip|on\text{-}chip|memory)$. The off-chip address field has 12 bits, the lowest order bit of which encodes the pair of Connection Machine processor chips sharing a floating-point unit (fpu). The on-chip address field encodes the processors on a Connection Machine processor chip. The memory address field encodes the bits of the memory local to a processor. Only the off-chip part of the address field is subject to the binary-reflected Gray code encoding in the lattice emulation mode. The on-chip and memory fields are encoded in binary code.

With a nonadaptive recursive decomposition of the domain it suffices to consider grids with axis lengths being powers of two. For such grids the following two properties [13], [14], [16] of the binary-reflected Gray code are important for the embedding of a hierarchy of grids: $Hamming(i, i \oplus 2^j) = 2, j > 0$, and $Hamming(i, i + 3)=1$ for $i \bmod 2 = 0$. If the computations performed on a hierarchy of grids take place at only one grid level at a time, then the different grids can be mapped onto the same set of processors without loss of efficiency. By embedding the finest grid in a Boolean cube by a binary-reflected Gray code adjacent nodes in a coarser grid consisting of every other grid point along an axis are at distance two. The same property is true for all coarser grids obtained in the same manner. Hence, even though successively coarser grids consist of points with indices differing by increasing powers of two, the points are always in proximity when the grid is mapped to a Boolean cube by a binary-reflected Gray code. This property is easily proven [13] and apparent from Fig. 4.

Communication between adjacent nodes in the finest grid is nearest neighbor communication in the Boolean cube. Communication between adjacent nodes in coarser

Integer	Gray code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

FIG. 4. *A 4-bit binary-reflected Gray code.*

grids implies communication between processors at distance two in the Boolean cube. All distance two communications can be arranged such that there is no contention for channels [13]. Then, the communication time for bit-serial pipelined communication, which is the communication mode on the Connection Machine system, is dominated by the message length. The path length is an additive term.

Subselecting grid points for coarse grids by choosing the grid at level ℓ to consist of all points such that $i \bmod 2^{h-\ell} = 0$ for each axis causes the grid points to be allocated to different subcubes. In [13], [16] an exchange step was introduced to move the selected points into subcubes identified by address bits. After the exchange the subselected points are embedded by a binary-reflected Gray code in a half size subcube. For instance, in the example above, if 2 and 3, 6 and 7, 10 and 11, and 14 and 15 are exchanged, then the grid points with the same lowest order bit are Gray coded in a code with one less bit, i.e., at Hamming distance one. Locations with the lowest order bit zero contain all the even grid points, locations with the lowest order bit one contain all the odd grid points. In general, at each step $k, k \geq 0$ exchanging grid points between locations that differ only in bit k, if the parity of bits $k+1$ through $n-1$ for an n-bit code is odd, guarantees that the subselected grid points are at distance one. This scheme converts the binary-reflected Gray code to a binary code (see [14], [15]). This exchange algorithm implicitly and recursively makes use of the fact that *Hamming* $(i, i+3)$=1 for $i \bmod 2 = 0$. If the interaction between adjacent grid levels consists of a reduction/distribution operation the exchange step need not be performed. In the one-dimensional case and a reduction on pairs of grid points the reduction operation on nodes $i \bmod 4 = 0$ and $i \bmod 4 = 1$ can be rooted in grid point $i \bmod 4 = 0$, and the reduction on $i \bmod 4 = 2$ and $i \bmod 4 = 3$ rooted in $i \bmod 4 = 3$. This strategy is used in our implementation of the PMM. It was previously used in [16], [17].

4. Implementation.

4.1. Overview. The first implementation of the PMM on the Connection Machine [21] mapped the nodes at all levels of the decomposition tree to processors by a binary encoding of the node addresses. The nodes of the decomposition tree were labeled in a breadth first order. At the time of the first Connection Machine imple-

mentation, the grid communication mechanism was not available. Communication between processors was by way of the router. The binary encoding does not preserve locality in the index space. The Hamming distance between consecutive integers may be as high as the number of bits required for the encoding. With this mapping of the grid points to processors more than 60 percent of the execution time is spent in communication.

The objective of the implementation of the PMM described here was to yield high performance on three-dimensional problems with a large number of particles, say, 1,000,000 or more. In order to exploit the locality of reference present in the nonadaptive recursive subdivision of the domain, the grid emulation feature of the Connection Machine programming systems was used. The address space was configured as a three-dimensional grid. The different grids in the hierarchy were mapped to processing units such that coarser grids were contained in subcubes identified by higher order bits. The reduction/distribution operations were performed such that the grids at any level were embedded by a binary-reflected Gray code, as described above.

In the *field-wise* programming model a word is allocated serially, i.e., in successive memory locations of a processor. The field-wise programming model is used by all programming languages on the Connection Machine. In this model each processor has its own memory. But, a floating-point unit shared by 32 Connection Machine processors can access their memories in bit-slices. Hence, with a word stored with one bit per Connection Machine processor a word can be accessed by a floating-point unit in a single cycle. This type of data representation and data access corresponds to the *slice-wise* programming model of the Connection Machine system. In this model the Connection Machine consists of 2,048 processing units interconnected as an 11-dimensional cube with two communication channels between each pair of units. The slice-wise view of the Connection Machine offers the potential for exploiting the memory hierarchy introduced through the registers within the floating-point unit. The registers and associated buses on the floating-point unit increase the effective memory bandwidth. The floating-point units are not complete processors and some operations have to be performed on the Connection Machine processors. At the time of this implementation the supported programming systems did not contain features allowing access to the internal features of the floating-point units. Many functions were implemented using a mixture of the field-wise bit-serial model and the slice-wise 32-bit wide model.

In three dimensions the near-field consists of 26 subcubes, and the interactive-field consists of 189 ($=6^3 - 27$) subcubes. In order to simplify the implementation, a $7 \times 7 \times 7$ neighborhood of subcubes (consisting of 343 ($=7^3$) subcubes), symmetrized with respect to each subcube by including a few more "null" subcubes from the far-field and the near-field, is considered, and an interactive-field mask is used to subselect from the enlarged neighborhood the subcubes of the interactive-field. The dominating operations are reduction, distribution, and convolution as defined by the near-field (a 27-point kernel with respect to grid points) and the interactive-field. The implementation is restricted to one leaf node per floating-point unit. The leaf node is represented in the field-wise model, i.e., by 32 Connection Machine processors.

In our implementation computation-intensive static data are precomputed. For example, the translation operator T_2 requires that first a set of coefficients ($b_{i,j,k}$ [21]) be computed before the convolution against the Φ's contributed by the interactive-field is carried out. For each subdomain this operation is repeated for every member

of its interactive-field (in our case, for up to 189 times). This computation accounts for most of the execution time within the tree. The coefficients $(b_{i,j,k})$ are obtained from expansions on difference vectors between centers of different subcubes that can be predetermined, and thus can be computed ahead of time. The precomputation makes use of the fact that there exists a subset of kernels from which the coefficients can be computed with a modest effort [19].

4.2. Reduction and distribution functions. In the upward pass (3) of the PMM a reduction operator is applied recursively on a three-dimensional grid data structure. The downward pass (5) accumulates the data and distributes the results. The reduction operator and the distribution operator are very general. The same type of operators are extensively used in Multigrid type algorithms. We have generalized these two operators to generic functions, called `multigrid-reduce` and `multigrid-distribute`. The `multigrid-reduce` reduces a set of data on a source grid to a smaller set of data on a destination grid, with a reduce operator. Likewise, the `multigrid-distribute` distributes data to a larger grid.

The subselection of grid points in reduction can be achieved by reshaping each axis. By adding a "dummy" axis for each array axis a plane is created for each axis. The shape of this plane is changed throughout the recursion such that the total number of grid points in the plane is preserved for every step. For instance, an axis with 32 points is represented as a 1×32 array, and for successively coarser grids is represented as a 2×16, 4×8, 8×4, 16×2, and 32×1 array. This reconfiguration can take place dynamically. The first axis requires 0, 1, 2, 3, 4, and 5 bits, respectively. The second axis requires 5, 4, 3, 2, 1, and 0 bits. If the encoding of the elements along the first axis is assigned bits starting with the lowest order bit, then successive points along the second axis in the 2×16 grid consists of all even points in the first grid, and so on.

$$\underbrace{(a_4a_3a_2a_1a_0)}_{axis-1} \qquad \underbrace{(a_4a_3a_2a_1}_{axis-1}\ \underbrace{a_0}_{axis-0}\ ) \qquad \underbrace{(a_4a_3a_2}_{axis-1}\ \underbrace{a_1a_0}_{axis-0}\ )\ldots$$

Applying reshaping to the on-chip part of the axis yields no performance advantage. The lowest order off-chip address bit is treated in the same manner as the on-chip bits, since it defines the processor chips sharing memory. An example of the representation of one reshaping step of a $16 \times 16 \times 16$ grid is the following: (1 16 1 16 1 16) $\rightarrow$ (2 8 2 8 2 8). The second, fourth, and sixth dimensions are reduced. The first, third, and fifth dimensions are used to collect grid points not part of the reduced grid.

4.3. Direct interaction. The computation of the direct interaction with particles in the near-field at the base of the pyramid is implemented in the field-wise model, except for the data motion. The interaction between a pair of subdomains is defined by the interaction of every particle in one domain with every particle in the other domain. The communication is implemented by first performing an exchange of particle information between the domains, then rotating the received information through the Connection Machine processors representing the subdomain. After a complete revolution the interaction between all particles in one domain with all particles in the other domain has been evaluated. The rotation within a domain is implemented using some of the slice-wise features of the Connection Machine. The measured performance of this implementation is 1.46 Gflops/sec on a 64k processor Connection Machine, which is very close to the maximum possible within the high level languages at a virtual processor ratio of one.

5. Results. The first implementation of the PMM on the Connection Machine [21] required 94 seconds on an 8k CM–2 for the evaluation of the potential field due to 16,000 particles. More than half of the total time was spent in communication. The measured performance of the new implementation of the PMM on the Connection Machine is 1.7 Gflops/sec at a particle-processor-ratio of 10. The communication time is insignificant. With the new implementation, the computation of the potential field for 16,000 particles requires 5 seconds on an 8k CM–2.

5.1. Timing results. In the experiments reported here we compute the potential field using third-order multipole expansions. The measured root-mean-squared error E_{rms} is less than 10^{-2}. Only uniform particle distributions were used. The performance measurements were carried out on a 16k CM–2 at a *virtual processor* ratio of 1. There was one leaf node, or one grid point in the base of the pyramid, per floating-point unit associated with a cluster of 32 CM processors. With ten or more particles per CM processor the execution time is entirely dominated by the computation of the local interaction, which is made by a direct evaluation.

TABLE 1
Timing results for potential field evaluation by the Parallel Multipole Method.

	Running Time			
Particle-processor-ratio	1	2	3	4
init-expansion (ms)	26.6	36.6	46.6	57.0
far-field (ms)	21.7	21.7	21.5	21.5
local-field (sec)	1.78	1.86	1.76	1.79
multipole-exp-eval (ms)	7.07	14.2	21.2	28.3
local-direct (sec)	0.846	2.99	6.45	11.2
Total (sec)	2.68	4.92	8.30	13.1
Particle-processor-ratio	10	20	30	40
init-expansion (ms)	116	218	316	435
far-field (ms)	21.5	21.5	21.8	21.5
local-field (sec)	1.79	1.76	1.81	1.75
multipole-exp-eval (ms)	71.1	149	222	291
local-direct (sec)	66.8	264	590	1050
Total (sec)	68.8	266	592	1052

Explanation of the entries in Table 1:

- *init-expansion*: form multipole expansions at leaf nodes;
- *far-field*: upward-pass through the pyramid;
- *local-field*: downward-pass through the pyramid;
- *multipole-exp-eval*: evaluate the resulting multipole expansions at each particle position;
- *local-direct*: compute interactions with nearby particles directly.

5.2. Complexity and performance analysis.

5.2.1. Time complexity. Using the timing results from Table 1, we can express the time complexity for each of the subroutines of the PMM in terms of the particle-processor-ratio r and with one leaf node per floating-point unit

$$\begin{aligned} T_{init-expansion} &= 16.6 + 10r \quad (ms) \\ T_{far-field} &= 21.6 \quad (ms) \end{aligned}$$

$$\begin{aligned} T_{local-field} &= 1.79 \quad (sec) \\ T_{multipole-exp-eval} &= 7.1r \quad (ms) \\ T_{local-direct} &= 0.197r + 0.649r^2 \quad (sec). \end{aligned}$$

The total running time of the PMM is

$$T_{total}(r) = 1.84 + 0.214r + 0.649r^2 \qquad (sec).$$

In order to gain insight into the relative importance of communication T_{comm} and computation T_{comp} for the PMM, detailed timings were conducted and summarized in Table 2.

TABLE 2
Communication and computation time as functions of the number of particles per processor.

	Running Time	
Time component	T_{comm}	T_{comp}
init-expansion (ms)	17.7	$10r - 1.1$
far-field (ms)	3.8	17.8
local-field (sec)	0.36	1.43
multipole-exp-eval (ms)	0	$7.1r$
local-direct (sec)	$0.197r$	$0.649r^2$
Total (sec)	$0.38 + 0.197r$	$1.45 + 0.017r + 0.649r^2$

The results of Table 2 show that for $r = 1$ the time for communication is 21.5 percent of the total time. The T_{comp} component grows much faster with r than the T_{comm} component. For $r = 10$, T_{comm} is only 3.4 percent of the total time. The computation in *far-field* and *local-field* is essentially a three-dimensional convolution, and the one in *local-direct* is Newtonian interaction.

5.2.2. Speed. The *local-field* computations require about $(343 + (343/8)) \times ((40/8) + 32)$ floating-point operations in our implementation. The computations labeled *local-direct* require $14 \times 32 \times 32 \times 40r^2$ floating-point operations. Since these two parts are dominant in terms of floating-point operations, we use them as an approximation for the total number of operations. With W floating-point processors and one leaf node of the decomposition tree per floating-point unit, the speed of the PMM is determined by

$$\begin{aligned} Speed_{PMM}(r) &= \frac{((343 + \frac{343}{8}) \times (\frac{40}{8} + 32) + 14 \times 32 \times 32 \times 40r^2)W}{T_{total}(r)} \\ &= \frac{(14277 + 573440r^2)W}{1.84 + 0.214r + 0.649r^2} \quad (flop/sec). \end{aligned}$$

On a 64k CM–2 $W = 2048$. The speed of the PMM for different r is shown in Table 3. For $r = 1$, $Speed_{PMM}(1) = 445$ Mflops/sec. For $r = 10$, $Speed_{PMM}(10) = 1.71$ Gflops/sec, which is close to the asymptotic speed 1.81 Gflops/sec. The performance for the arithmetic operations $+, -, \times$ is 1.5–2.4 Gflops/sec in *Lisp, with the lower figure achievable for a virtual processor ratio of one. All of our timings were made at a virtual processor ratio of one. Some code segments are implemented in lower level code, which explains a peak performance in excess of the *Lisp performance. But, an improvement in the arithmetic performance by a factor of 2–3 is expected from a complete change to the slice-wise, 32-bit wide model.

TABLE 3
Speed of the Parallel Multipole Method with respect to the number of particles per processor.

Particle-processor-ratio	1	2	3	4	10	20
Speed (gflops/sec)	0.445	0.971	1.27	1.44	1.71	1.77

5.2.3. Storage requirements. With a multipole expansion of degree p, the total number of terms retained in the expansion is

$$m = \tfrac{1}{6}(p+1)(p+2)(p+3).$$

For a pyramid of h levels the state and the problem description for a grid point at the base (leaf node) *and* its ancestors require $(h-2)m$ words of storage for Φ, $2m$ words for Ψ, $\lceil (h-2)(343/32) \rceil$ words for the interactive-field masks, $6(h-2)$ words for cell coordinates, and about another 100 words for miscellaneous variables. In the downward pass $343m$ words are required to store the intermediate results. Each particle takes 10 words of storage (the position, the velocity, and the acceleration require three words each plus one word for the mass). Thus, the total storage required for a grid point at the base (leaf node) and its ancestors is

$$S \doteq 88 + 343m + \left\lceil (h-2)\frac{343}{32} \right\rceil + h(m+6) + 10R \quad (words),$$

where R is the particle-node-ratio. For $p = 3$ (and therefore $m = 20$), a three level tree requires $S \doteq 7037 + 10R$ $(words)$, a four level tree $S \doteq 7074 + 10R$ $(words)$, and a five level tree $S \doteq 7111 + 10R$ $(words)$ for a grid point at the base and its ancestors. Clearly, the storage requirement per leaf node is nearly independent of the depth of the tree.

In the field-wise mode each of the 32 processors sharing a floating-point unit allocates the same storage. Our implementation uses the model of mixed field-wise and slice-wise data representation. Each CM processor represents different particles. The part of storage for $343m$ words is allocated slice-wise. The rest of the memory is allocated field-wise, that is, replicated in each CM processor, if there are fewer than 32 leaf nodes per floating-point unit. The actual storage allocated for each leaf node in our implementation is

$$S_{mixed} \doteq 343m + 32\left[88 + \left\lceil (h-2)\frac{343}{32} \right\rceil + h(m+6)\right] + 10R \quad (words),$$

where R is again the particle-node-ratio with one leaf node per floating-point unit.

For a Connection Machine system the maximum depth of the decomposition tree that is of interest is three, four, or five, which corresponds to 512, 4096, or 32768 leaf nodes, respectively. In a 64k processor Connection Machine system the number of leaf nodes per floating-point unit is 2 for a four level tree, and 16 for a five level tree. Tables 4 and 5 summarize the number of particles that can be represented with 64k words and 256k words of storage per floating-point unit as a function of the height of the decomposition tree for the slice-wise data representation and for the mixed field-wise and slice-wise data representation. Figure 5 shows the total number of particles that can be represented in the slice-wise representation. With the increase in the height of the decomposition tree the number of leaf nodes increases and the total number of particles that can be represented in fixed size storage decreases. The

TABLE 4

Estimated maximum number of particles per leaf node and total particle system size for some Connection Machine systems (slice-wise data representation).

Storage per fpu (words)	Tree height	Connection Machine configuration					
		8k			16k		
		Particles			Particles		
		leaf	fpu	Total	leaf	fpu	Total
64k	$h = 3$	2,570	5,150	1.32 M	5,850	5,850	3.00 M
	$h = 4$	—	—	—	112	894	458 K
256k	$h = 3$	12.4 K	24.8 K	6.35 M	25.5 K	25.5 K	13.1 M
	$h = 4$	931	14.9 K	3.81 M	2,570	20.6 K	10.5 M
	$h = 5$	—	—	—	—	—	—
		32k			64k		
		Particles			Particles		
		leaf	fpu	Total	leaf	fpu	Total
64k	$h = 3$	—	—	—	—	—	—
	$h = 4$	930	3,720	3.81 M	2,570	5,140	10.5 M
256k	$h = 3$	—	—	—	—	—	—
	$h = 4$	5,850	23.4 K	23.9 M	12.4 K	24.8 K	50.8 M
	$h = 5$	108	3,460	3.54 M	927	14.8 K	30.4 M

TABLE 5

Estimated maximum number of particles per leaf node and total particle system size for some Connection Machine systems (mixed field-wise and slice-wise data representation).

Storage per fpu (words)	Tree height	Connection Machine configuration					
		8k			16k		
		Particles			Particles		
		leaf	fpu	Total	leaf	fpu	Total
64k	$h = 3$	2,310	4,620	1.18 M	5,300	5,300	2.71 M
	$h = 4$	—	—	—	48	381	195 K
256k	$h = 3$	12.1 K	24.3 K	6.21 M	25.0 K	25.0 K	12.8 M
	$h = 4$	910	14.6 K	3.73 M	2,510	20.0 K	10.3 M
	$h = 5$	—	—	—	—	—	—
		32k			64k		
		Particles			Particles		
		leaf	fpu	Total	leaf	fpu	Total
64k	$h = 3$	—	—	—	—	—	—
	$h = 4$	781	3,120	3.20 M	2,250	4,500	9.21 M
256k	$h = 3$	—	—	—	—	—	—
	$h = 4$	5,700	22.8 K	23.3 M	12.1 K	24.2 K	49.5 M
	$h = 5$	108	3,460	3.54 M	902	14.4 K	29.6 M

difference in the two models of storage allocation is the reason for the difference in the size of the particle systems between Tables 4 and 5.

In our implementation that uses the mixed data representation on a 16k CM with 64k words of storage per floating-point unit, the maximum number of particles per leaf node is $\sim 5,300$ for a three level tree. With 256k words of memory per floating-point unit the maximum number of particles per leaf node for a three level tree is $\sim 25,000$. The measured maximum number of particles for a three level tree using the mixed field-wise and slice-wise representation is about $4{,}160 (= 130 \times 32)$. The difference between the measured and the predicted maximum number of particles is due to the allocation of stack variables not accounted for in the expression for the required storage.

5.3. Optimizing the number of leaf nodes. The points in the grid of finest granularity represent the leaf nodes of the decomposition tree. Each leaf node holds

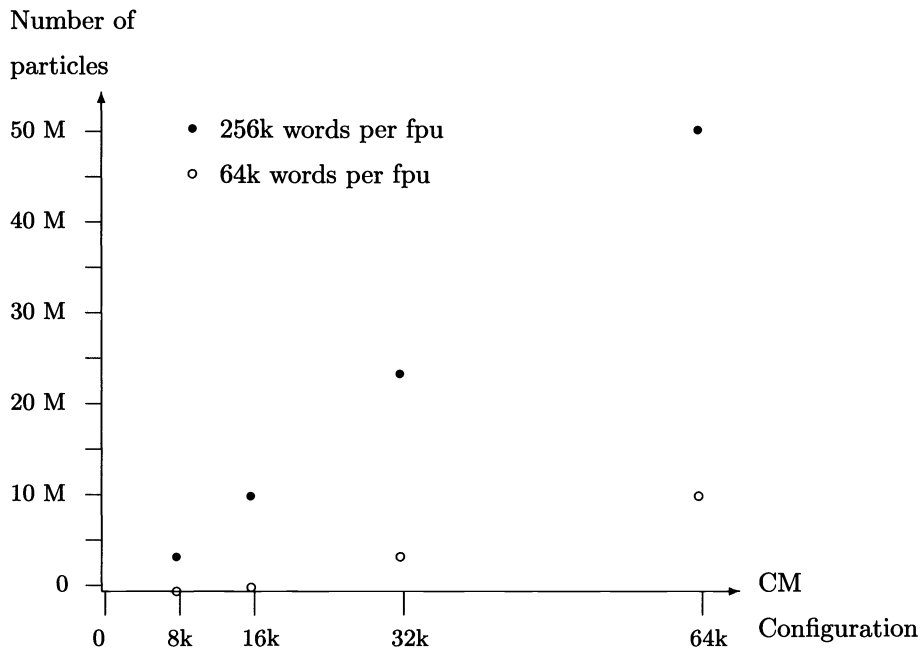


FIG. 5. *The maximum number of particles in primary storage for various Connection Machine systems and a four level decomposition tree (slice-wise data representation).*

a number of particles. The interaction among the particles in the same leaf node is computed by direct evaluation of the Newtonian gravitational field. The associated computational complexity is quadratic in the number of particles per leaf node. With a leaf node together with all its particles assigned to a single floating-point unit, the computation of the direct interactions within leaf nodes requires no communication. The computations are local and sequential for each leaf node, and "embarrassingly" parallel across different leaf nodes.

The decomposition tree of the nonadaptive Multipole Method corresponds to data interaction in the form of a pyramid with the same height h as the decomposition tree. The number of grid points at the base of the pyramid is 2^{3h}. If the number of grid points at the base is greater than the number of floating-point units 2^n, then a unit has to perform the computations of several grid points, that is, the grid points of a subpyramid of height $h - \frac{n}{3}$. The optimal value of h with respect to performance depends upon the relative overheads of different methods. Greengard and Rokhlin [2], [6] compared the nonadaptive Multipole Method with the direct evaluation of the Newtonian field on a VAX-8600. The cross-over point below which the direct method is faster was found to be in the range of 200 – 400 particles for a variety of two-dimensional particle distributions including highly nonuniform distributions, where the maximum number of particles per leaf is 30 in the Multipole Method. In [9] the nonadaptive algorithm is compared with the direct method for three-dimensional problems. For uniform distributions the cross-over point on a VAX-8600 is in the range of 1000 – 2000 particles with an accuracy of 10^{-6}. In the three-dimensional implementation of the Multipole Method reported in [21] the cross-over point is at about 1000 particles. The experiment was carried out on a Symbolics Lisp Machine

with the results accurate to 10^{-4}. In both [9] and [21] the Multipole Method restricts the number of particles in a leaf to be fewer than 8. The local direct interaction at the leaf node includes the direct interaction within the same leaf node as well as that with adjacent leaf nodes. The further subdivision of a subdomain with a few thousand particles reduces the computation of the direct interactions with adjacent subdomains which are smaller in size after the subdivision. It also reduces the computation time for interactions within the same subdomain until there are too few particles in a subdomain. Hence, the stopping point for the number of particles per subdomain in the recursive subdivision with respect to minimum execution time is somewhere below the cross-over points referenced above. The optimum stopping criterion is implementation dependent.

The stopping criterion for the subdivision of a domain represented by several floating-point processors also has to acknowledge the difference in communication needs between the methods. In the parallel implementation of the two-dimensional Multipole Method reported in [7] a comparison was made with a parallel implementation of the direct method. The two implementations were made on a shared memory machine, the Encore Multimax. The speedup for the direct method was almost identical to the number of processors, when the particle to processor ratio in the experiments ranged from 40 to 1,250. The speedup of the Multipole Method was about three fourths of the number of processors for 40 particles per processor, but increased to become almost identical to the number of processors for a larger number of particles per processor. We have not compared the Parallel Multipole Method with the direct method, or any other tree method for the Connection Machine. The expected speedup for the direct method is linear in the number of processors as long as the particle-to-processor ratio is high since the communication is linear and the computations are quadratic in the number of particles. For the Multipole Method the communications and the computations are of the same order, and the speedup is likely to be less. Assigning several processors to a subdomain favors stopping the recursive subdivision at an earlier stage.

6. Summary. The goal of the PMM implementation described here was the efficient use of the Connection Machine architecture. For the implementation reported in [21] about 60 percent of the total execution time was due to interprocessor communication. With a nonadaptive recursive partitioning of the domain the data interaction is defined by a pyramid in which each parent node has degree eight in three dimensions. The base of the pyramid was embedded in the Connection Machine as a three-dimensional grid by making use of the lattice emulation feature of the Connection Machine programming systems. With other performance enhancements the speedup of the new implementation over the old one is approximately a factor of 20. The new implementation achieves a performance in excess of 1.7 Gflops/sec for a particle-processor-ratio of 10 or higher.

The new implementation can be sped up further by more efficient coding of the arithmetic operations. All operations are currently performed as scalar operations. However, each floating-point processor can be operated as a vector processor with a performance increase of up to a factor of 10 for operations such as matrix-vector multiplication and convolution. A further speed enhancement by a factor of three is expected. By a slice-wise data representation slightly larger particle systems can be represented in the primary storage.

With 512M bytes of storage a system of about 6 million particles can be simulated without external memory. With 2G bytes of storage systems up to about 50 million

particles can be simulated. The potential field evaluation for one million particles requires about 3 minutes. A field calculation for 50 million particles would require approximately 100 hours, if the number of particles per leaf node is 12,000 with the current implementation, but can be cut to about 2.5 hours if there is sufficient memory to generate a tree structure with the same number of particles per leaf node as was used in the one million particle simulation. Future work on the PMM includes dynamic load-balancing on particle systems with highly nonuniform distributions.

7. Acknowledgments. The authors would like to thank Ajit Agrawal and members of the Computational and Mathematical Sciences Groups at TMC for their assistance in implementing the PMM, and Anne Greenbaum of the Courant Institute of Mathematical Sciences and the anonymous referee for valuable comments.

REFERENCES

[1] J. BARNES AND P. HUT, *A hierarchical $o(n \log n)$ force calculation algorithm*, Nature, 324 (1986), pp. 446–449.

[2] J. CARRIER, L. GREENGARD, AND V. ROKHLIN, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 669–686.

[3] M. Y. CHAN, *Dilation-2 embeddings of grids into hypercubes*, in 1988 International Conference on Parallel Processing, The Pennsylvania State University Press, University Park, PA, 1988.

[4] ———, *The embedding of grids into optimal hypercubes*, Tech. Report, Computer Science Dept., University of Texas, Dallas, TX, 1988.

[5] ———, *Embeddings of 3-dimensional grids into optimal hypercubes*, Tech. Report, Computer Science Dept., University of Texas, Dallas, TX, 1988. Also in the Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, March, 1989.

[6] L. GREENGARD, *The Rapid Evaluation of Potential Fields in Particle Systems*, Ph.D. thesis, Department of Computer Science, Yale University, New Haven, CT, April 1987; published by MIT Press, Cambridge, MA, 1988.

[7] L. GREENGARD AND W. D. GROPP, *A parallel version of the fast multipole method*, in Parallel Processing for Scientific Computing, G. Rodrigue, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 213–222.

[8] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, J. Comp. Phys., 73 (1987), pp. 325–348.

[9] ———, *On the efficient implementation of the fast multipole method*, Tech. Report YALEU/DCS/RR-602, Department of Computer Science, Yale University, New Haven, CT, February 1988.

[10] I. HAVEL AND J. MÓRAVEK, *B-valuations of graphs*, Czech. Math. J., 22 (1972), pp. 338–351.

[11] C.-T. HO AND S. L. JOHNSSON, *On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two*, in Proceedings of the 1987 International Conference on Parallel Processing, IEEE Computer Society, 1987, pp. 188–191.

[12] ———, *Embedding meshes in Boolean cubes by graph decomposition*, J. Parallel Distributed Comput., 8 (1990) pp. 325–339.

[13] S. L. JOHNSSON, *Odd-even cyclic reduction on ensemble architectures and the solution tridiagonal systems of equations*, Tech. Report YALE/DCS/RR-339, Department of Computer Science, Yale University, October 1984.

[14] ———, *Communication efficient basic linear algebra computations on hypercube architectures*, J. Parallel Distributed Comput., 4 (1987) pp. 133–172.

[15] ———, *Optimal Communication in Distributed and Shared Memory Models of Computation on Network Architectures*, in VLSI and Parallel Computation, pp. 223–389, Morgan Kaufman, 1990.

[16] S. L. JOHNSSON AND C.-T. HO, *Optimizing tridiagonal solvers for alternating direction methods on Boolean cube multiprocessors*, SIAM J. Sci. Statist. Comput., 11 (1990) pp. 563–592.

[17] ———, *Multiple tridiagonal systems, the alternating direction method, and Boolean cube configured multiprocessors*, Tech. Report YALEU/DCS/RR-532, Department of Computer Science, Yale University, New Haven, CT, June 1987.

[18] S. L. JOHNSSON AND P. LI, *Solution set for AMA/CS* 146, Tech. Report 5085:DF:83, California Institute of Technology, Pasadena, CA, May 1983.

[19] J. KATZENELSON, *Computational structure of the n-body problem*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 787–815.

[20] E. M. REINGOLD, J. NIEVERGELT, AND N. DEO, *Combinatorial Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[21] F. ZHAO, *An o(n) algorithm for three-dimensional n-body simulations*, Tech. Report AI-995, MIT, Artificial Intelligence Laboratory, Cambridge, MA, October 1987.

[22] F. ZHAO AND S. L. JOHNSSON, *The parallel multipole method on the Connection Machine*, Tech. Report CS89-6, Thinking Machines Corp., October 1989, Technical Report, YALEU/DCS/RR-749, Department of Computer Science, Yale University, October 1989.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1438–1456, November 1991
© 1991 Society for Industrial and Applied Mathematics
012

AXIALLY SYMMETRIC ACOUSTIC WAVE PROPAGATION THROUGH FLOWS WITH VORTICITY*

NOEL J. WALKINGTON†

Abstract. Periodic solutions of the linearized axially symmetric Euler equations are sought. The axially symmetric case is of considerable practical interest, since these equations are used to calculate the sound field around commercial aircraft engine inlets, which are circular. It is known that these equations are well posed in plane case; however, the proof relies on the two-dimensional nature of the problem. The axially symmetric problem does not trivially follow, since periodic variations in the θ direction give rise to a three-dimensional velocity field.

Key words. acoustic propagation, rotational flows

AMS(MOS) subject classifications. 76Q05, 65N30

1. Introduction.

The problem under consideration is acoustic propagation governed by the linearized Euler equations in domains with cylindrical symmetry. To fix ideas, we will consider "the duct transmission problem." This problem considers a bounded domain, $\Omega \subset \mathbb{R}^2_+$, with boundary $\Gamma_0 \cup \Gamma_1 \cup \Gamma_2 \cup \Gamma$ (see Fig. 1). Γ_0 is the center line of a duct with cross section Ω, and Γ represents the duct wall. A mean flow is assumed to enter the duct through the boundary Γ_1 and exit at Γ_2 (the duct terminations). The acoustics problem assumes that a sound source is located at Γ_1 and that the sound in the duct is to be calculated.

Previous attempts, [1], [13], to construct numerical procedures to solve the problem in this generality indicate that instabilities may arise, causing the algorithms to fail. In [12] the author proved that the plane problem was well posed, and a robust finite element scheme was proposed. Further references to the plane problem may be found in [12]. We also mention that if the velocity is assumed irrotational, the well-posedness in two or three dimensions is elementary, and standard algorithms to solve these equations perform very satisfactorily (see, e.g., [8]).

The approach in [12] uses two simplifications encountered in two-dimensional fluid mechanics that do not hold in three dimensions. The *curl* of the two-dimensional momentum equation yields a homogeneous equation for the (scalar) vorticity (cf. equation (4)), while the corresponding operation in three dimensions yields a nonhomogeneous vector equation in the vorticity. Moreover, the nonhomogeneous term involves derivatives that would not be integrable in the setting of [12]. Also, the Helmholtz decomposition of a plane velocity field is particularly simple, stating that the velocity may be represented by two scalar "potentials." In three dimensions the Helmholtz decomposition gives a representation in terms of a scalar potential and the *curl* of a vector. When the axially symmetric equations are linearized, it is necessary to assume that the perturbations are periodic in the theta direction, and this gives rise to a three-dimensional velocity field. In particular, the theta component of the momentum equation is not trivial, so that the simplifications used in [12] do not hold for this problem. In this paper we show how the results in [12] can be extended to the axially symmetric problem. The main ideas are to find a Helmholtz–type decomposition for an axially symmetric velocity field that retains the simplicity of the plane decomposition, and to construct weighted Sobolev spaces with weights subtly chosen

* Received by the editors July 9, 1990; accepted for publication October 24, 1990. This work was supported by National Science Foundation grant DMS-9002768.
† Department of Mathematics, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

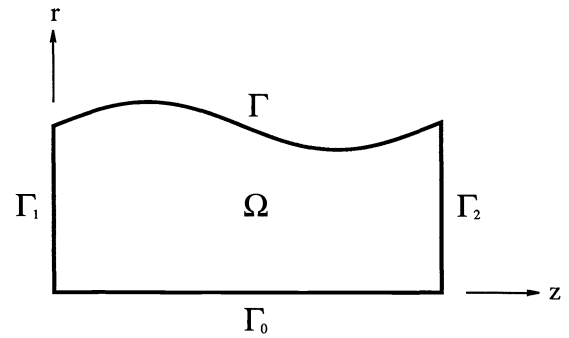


FIG. 1. *Duct geometry.*

to render the problem well posed. Once the problem is suitably formulated, it is clear that the proofs of the plane case will extend to the current situation. Since the axially symmetric equations involve a considerable number of terms, and the proofs of the existence theorems are similar in spirit to those in [12], but more detailed, we merely quote the theorems on well-posedness and indicate the ideas involved in their proofs.

In the following we will assume that $\Omega \subset \mathbb{R}^2_+$ is bounded, Lipschitz, and simply connected, where $\mathbb{R}^2_+$ denotes the upper half plane. A typical point of Ω is denoted (r, z). The boundary of Ω is denoted $\partial\Omega$, and $\partial\Omega^+ = \Gamma_1 \cup \Gamma \cup \Gamma_2$ is the closure of $\partial\Omega \cap \{r > 0\}$. $\partial\Omega^+$ could be considered the physical boundary of the circular duct which Ω represents. It will turn out that certain functions will have a trace on $\partial\Omega^+$ but not on the axis $r = 0$. Weighted $L^2(\Omega)$ spaces will be used, we use the notation $\|.\|^2_{0,r} = \int_\Omega r(.)^2$ and $\|.\|^2_{0,1/r} = \int_\Omega (1/r)(.)^2$ to indicate norms with weights r and $1/r$, respectively, and denote the corresponding function spaces $L^2_r(\Omega)$ and $L^2_{1/r}(\Omega)$. The corresponding weighted Sobolev norms will be denoted $\|.\|_{1,r}$ and $\|.\|_{1,1/r}$, respectively, and $\|.\|_\infty$ will denote the (unweighted) L^∞ norm. The symbols $\nabla = \mathbf{e}_r(\partial/\partial r) + \mathbf{e}_z(\partial/\partial r)$ and $curl = \mathbf{e}_r(\partial/\partial z) - \mathbf{e}_z(\partial/\partial r)$ will be utilized.

The Lax–Milgrim theorem will be used to solve the axially symmetric acoustics problem, so Galerkin approximations can easily be constructed using finite element techniques. However, the error analysis is complicated by the need to work with functions in $H^1_{1/r}(\Omega)$ for which there are no standard approximation results. Mercier and Raugel [6] give a complete development of the approximation theory for the spaces $H^1_r(\Omega)$ and $L^2_{1/r}(\Omega)$, and, in particular, they discuss the approximation of functions which lie in $L^2_{1/r}(\Omega)$ with derivatives in $L^2_r(\Omega)$ (e.g., ϕ in §3). However, their results do not extend to $H^1_{1/r}(\Omega)$. Similar results for $H^1_r(\Omega)$ have also been established by Zhou [14] and French [4]. Bernardi and Maday [2] discuss regularity of elliptic problems and their approximation in weighted Sobolev spaces with weights r^α with $\alpha \in (-1, 1)$, the case $\alpha = -1$ being excluded. The acoustics equations also involve two first-order equations for which there is no "standard" method of approximation. We use the technique of [12] which is compatible with the other equations to which they are coupled. Other methods of solving scalar first-order equations are discussed in Falk and Richter [3]. The development of error estimates for $1/r$–weighted Sobolev spaces and the first-order equations is beyond the scope of the current paper, and will be considered elsewhere.

In the next section the axially symmetric acoustics equations are presented and a Helmholtz-type decomposition of the axially symmetric acoustic velocity field in-

troduced. Section 3 introduces a variational formulation of the acoustics problem, and a proof of existence and uniqueness is sketched. Some numerical experiments are presented in §4.

2. Equations and boundary conditions. Euler's equations for an ideal compressible fluid in cylindrical coordinates take the form:

$$\frac{\partial \rho}{\partial t} + \frac{1}{r}\left[\frac{\partial}{\partial r}(\rho r v_r) + \frac{\partial}{\partial \theta}(\rho v_\theta) + \frac{\partial}{\partial z}(\rho r v_z)\right] = 0, \tag{1}$$

$$\frac{\partial v_r}{\partial t} + \frac{1}{2}\frac{\partial}{\partial r}|\mathbf{v}|^2 + \frac{1}{\rho}\frac{\partial p}{\partial r} = v_\theta \omega_z - v_z \omega_\theta,$$

$$\frac{\partial v_z}{\partial t} + \frac{1}{2}\frac{\partial}{\partial z}|\mathbf{v}|^2 + \frac{1}{\rho}\frac{\partial p}{\partial z} = v_r \omega_\theta - v_\theta \omega_r,$$

$$\frac{\partial v_\theta}{\partial t} + \frac{v_r}{r}\frac{\partial (r v_\theta)}{\partial r} + \frac{v_\theta}{r}\frac{\partial v_\theta}{\partial \theta} + v_z \frac{\partial v_\theta}{\partial z} + \frac{1}{\rho r}\frac{\partial p}{\partial \theta} = 0, \tag{2}$$

$$\frac{p}{\rho^\gamma} = \kappa^\gamma,$$

where ρ and p represent the fluid density and pressure, respectively, and $\mathbf{v}$ is the velocity vector, $\mathbf{v} = \mathbf{e}_r v_r + \mathbf{e}_z v_z + \mathbf{e}_\theta v_\theta$. The components of the vorticity vector are:

$$\omega_r = \frac{1}{r}\left[\frac{\partial v_z}{\partial \theta} - \frac{\partial (r v_\theta)}{\partial z}\right], \qquad \omega_\theta = \frac{\partial v_r}{\partial z} - \frac{\partial v_z}{\partial r}, \qquad \omega_z = \frac{1}{r}\left[\frac{\partial (r v_\theta)}{\partial r} - \frac{\partial v_r}{\partial \theta}\right].$$

The r and z momentum equations may be combined to give to give a Bernoulli-type equation:

$$\frac{\partial}{\partial t}\left[\frac{\partial}{\partial r}\left(\frac{v_r}{r}\right) + \frac{\partial}{\partial z}\left(\frac{v_z}{r}\right)\right] + \nabla\cdot\left[\frac{1}{r}\nabla\left(\frac{1}{2}|\mathbf{v}|^2 + \frac{\gamma\kappa}{\gamma - 1}p^{\frac{\gamma-1}{\gamma}}\right)\right] \tag{3}$$

$$= \frac{\partial}{\partial r}\left[\frac{1}{r}(v_\theta \omega_z - v_z \omega_\theta)\right] + \frac{\partial}{\partial z}\left[\frac{1}{r}(v_r \omega_\theta - v_\theta \omega_r)\right],$$

and a vorticity transport equation:

$$\frac{\partial a_\theta}{\partial t} + v_r \frac{\partial a_\theta}{\partial r} + v_z \frac{\partial a_\theta}{\partial z} = \frac{1}{\rho r}\left[\frac{\partial}{\partial r}(v_\theta \omega_r) + \frac{\partial}{\partial z}(v_\theta \omega_z) + a_\theta \frac{\partial}{\partial \theta}(\rho v_\theta)\right], \tag{4}$$

where

$$a_\theta = \frac{\omega_\theta}{\rho r}.$$

Equation (3) is obtained by weighting each of the r and z momentum equations with $1/r$ and taking their "divergence" ($\nabla = \mathbf{e}_r \frac{\partial}{\partial r} + \mathbf{e}_z \frac{\partial}{\partial z}$), and (4) is obtained by taking their *curl*, (see Serrin [10]).

2.1. Acoustic equations. Equations (1)–(4) are linearized about a steady mean flow with zero theta variation ($\partial/\partial\theta = 0$) and zero theta component ($V_\theta = 0$). The acoustic (i.e., perturbation) quantities are assumed to be periodic both in time and in the theta direction. We adopt the following notation.

NOTATION.

1. Lowercase dependent variables will indicate acoustic (i.e., perturbation) quantities, and uppercase variables will refer to the mean flow.
2. Since the mean flow is real and the acoustic variables are complex, the scalars associated with them will be taken as the real numbers and complex numbers, respectively.
3. A boldface quantity will refer to a vector with components in $\mathbf{e}_r$ and $\mathbf{e}_z$ (e.g., $\mathbf{v} = v_r\mathbf{e}_r + v_z\mathbf{e}_z$ and the theta component of the acoustic velocity is considered separately).

The perturbations may then be written as:

$$\begin{aligned}
\mathbf{v}(r,\theta,z,t) &= \mathbf{V}(r,z) + \mathbf{v}(r,z)e^{ikt+im\theta},\\
\rho(r,\theta,z,t) &= R(r,z) + \rho(r,z)e^{ikt+im\theta},\\
p(r,\theta,z,t) &= P(r,z) + p(r,z)e^{ikt+im\theta}, \text{ etc.},
\end{aligned}$$

where $m \geq 1$ is an integer (the spinning mode number), and k is the frequency.

In order to extend the ideas utilized in the plane case, we wish to describe the velocity in terms of "potentials." In Appendix A we prove the following theorem.

THEOREM. *Let $\Omega \subset \mathbb{R}^2_+$ be a Lipschitz, simply connected domain that is rectangular near $r = 0$ (see Appendix A for the precise definition). If $\mathbf{v} \in L^2_{1/r}(\Omega)^2$, then there exist functions $\phi \in H^1_{1/r}(\Omega)$ and $\psi \in H^1_r(\Omega)$ with $\psi|_{\partial\Omega^+} = 0$ such that $\mathbf{v} = \nabla\phi + r\,curl\psi$.*

The assumption $\mathbf{v} \in L^2_{1/r}(\Omega)^2$ is consistent with the idea that it is necessary to have $\mathbf{v}|_{r=0} = 0$ in order to have single-valued velocities on the axis. The hypothesis on Ω is required to rule out the situation where portions of the boundary are tangent to the axis. In the sequel it will be assumed that the acoustic velocity may be written as

1. $\mathbf{v} = \nabla\phi + r\,curl\psi$,
2. $v_\theta = \tilde{v}_\theta + \frac{im}{r}\phi$.

The $(im/r)\phi$ term appearing in the equation for v_θ is a technicality, and may be motivated from the fact that the full three-dimensional gradient of a scalar ϕ would have this term for a theta component and $\nabla\phi$ for the in-plane components. This sum may also be thought of as a decomposition of v_θ into a regular part plus a singular part, since the gradient of $\tilde{v}_\theta$ will turn out to be integrable, while the gradient of ϕ/r will not.

Substituting the above into (1)–(4) results in the following equations for the perturbation quantities:

$$\begin{aligned}
-\tfrac{ik}{C^2}p - \tfrac{1}{r}\nabla.\left[r(R\nabla\phi + Rr\,curl\psi + \tfrac{p}{C^2}\mathbf{V})\right] + \tfrac{m^2}{r^2}R\phi - \tfrac{im}{r}R\tilde{v}_\theta &= 0,\\
\nabla.\left[\tfrac{1}{r}\nabla h\right] = -\tfrac{\partial}{\partial r}\left[\tfrac{1}{r}(V_z\omega_\theta + v_z\Omega_\theta)\right] + \tfrac{\partial}{\partial z}\left[\tfrac{1}{r}(V_r\omega_\theta + v_r\Omega_\theta)\right],&\\
ika_\theta + \mathbf{V}.\nabla a_\theta + \mathbf{v}.\nabla A_\theta = \tfrac{im}{r}A_\theta(\tilde{v}_\theta + \tfrac{im}{r}\phi),&\\
ik\tilde{v}_\theta + \tfrac{1}{r}\mathbf{V}.\nabla(r\tilde{v}_\theta) + \tfrac{im}{r}\left[\tfrac{p}{R} + ik\phi + \mathbf{V}.\nabla\phi\right] = 0,&\\
\nabla.[r\nabla\psi] = \omega_\theta,&
\end{aligned} \tag{5}$$

where

$$
\begin{aligned}
\mathbf{v} &= \nabla\phi + r\,curl\psi, \\
p &= R(h - ik\phi - \mathbf{V}.\mathbf{v}), \\
\omega_\theta &= r(Ra_\theta + \frac{p}{C^2}A_\theta), \\
\rho &= \frac{1}{C^2}p, \qquad C^2 = \frac{\gamma P}{R}.
\end{aligned} \tag{6}
$$

The latter relations (6) may be used to eliminate p, $\mathbf{v}$, and ω_θ from (5) so that the unknowns become $(\phi, \psi, h, a_\theta, \tilde{v}_\theta)$. The last term of the theta momentum equation (5_4) in square brackets simplifies to $r\mathbf{V} \cdot curl\psi$, so it will be integrable after division by r. If the theta component of velocity was not written as the sum indicated above, the term in square brackets would be p/R, and would no longer be integrable after division by r.

2.2. Boundary conditions. Boundary conditions appropriate for the duct transmission problem are,

$$
\left.\begin{aligned}
(\phi, h, a_\theta, \tilde{v}_\theta) &= (\hat{\phi}, \hat{h}, \hat{a_\theta}, \hat{v}_\theta) \\
\psi &= 0
\end{aligned}\right\} \ on\ \Gamma_1,
$$

$$
\left.\begin{aligned}
\frac{\partial\phi}{\partial n} &= 0 \\
\psi &= 0 \\
\frac{\partial h}{\partial n} + (V_r\omega_\theta + v_z\Omega_\theta)n_r - (V_z\omega_\theta + v_r\Omega_\theta)n_z &= 0
\end{aligned}\right\} \ on\ \Gamma, \tag{7}
$$

$$
\left.\begin{aligned}
(R\mathbf{v} + \frac{1}{C^2}\mathbf{V}p).\mathbf{n} &= l_1 \\
\psi &= 0 \\
\frac{\partial h}{\partial n} + (V_r\omega_\theta + v_z\Omega_\theta)n_r - (V_z\omega_\theta + v_r\Omega_\theta)n_z &= 0
\end{aligned}\right\} \ on\ \Gamma_2,
$$

where $\mathbf{n} = n_r\mathbf{e}_r + n_z\mathbf{e}_z$ in the outward normal vector to Ω, and $\partial/\partial n$ denotes the normal derivative. The quantities $(\hat{\phi}, \hat{h}, \hat{a_\theta}, \hat{v}_\theta)$ and l_1 are specified to model the presence of an acoustic source at Γ_1 and the acoustic termination at Γ_2.

3. Weak problem. Inspection of (5) indicates that one reasonable choice of spaces for the unknowns is

$$
\begin{aligned}
\phi - \hat{\phi} &\in \{\phi \in L^2_{1/r}(\Omega) \mid \nabla\phi \in L^2_r(\Omega)^2, \qquad \phi|_{\Gamma_1} = 0\}, \\
\psi &\in \{\psi \in L^2_r(\Omega) \mid \nabla\psi \in L^2_r(\Omega)^2, \qquad \psi|_{\partial\Omega^+} = 0\}, \\
h - \hat{h} &\in \{h \in L^2_{1/r}(\Omega) \mid \nabla h \in L^2_{1/r}(\Omega)^2, \qquad h|_{\Gamma_1} = 0\}, \\
a_\theta - \hat{a}_\theta &\in \{a_\theta \in L^2_r(\Omega) \mid \mathbf{V} \cdot \nabla a_\theta \in L^2_r(\Omega), \qquad a_\theta|_{\Gamma_1} = 0\}, \\
\tilde{v}_\theta - \hat{v}_\theta &\in \{\tilde{v}_\theta \in L^2_r(\Omega) \mid \mathbf{V} \cdot \nabla(r\tilde{v}_\theta) \in L^2_{1/r}(\Omega), \qquad \tilde{v}_\theta|_{\Gamma_1} = 0\},
\end{aligned}
$$

where it is to be understood that the functions $\hat{\phi}$, $\hat{h}$, etc., defined on Γ_1 are extended to all of Ω in such a way that they vanish on the rest of the boundary. When $m = 0$, there is no reason why ϕ or h should vanish on the axis, so the above choice of spaces

would be inappropriate in this instance. Indeed, the existence theorem below fails when $m = 0$, so in the sequel it will be assumed that the spinning mode number $m \geq 1$. When $m = 0$, (5) are very similar in structure to the plane problem, and the ideas used in [12] extend to this case.

ASSUMPTION 1. *The mean flow* $\mathbf{V}$ *is sufficiently smooth to guarantee that there exist constants* $K_{\mathcal{A}}$ *and* $K_{\mathcal{V}}$ *such that*

$$\|f\|_{0,r} \leq K_{\mathcal{A}} \|\mathbf{V}.\nabla f\|_{0,r} \qquad \forall\ f \in \{f \in C^1(\bar{\Omega}) \mid f|_{\Gamma_1} = 0\},$$

and

$$\|f\|_{0,r} \leq K_{\mathcal{V}} \|\mathbf{V}.\nabla(rf)\|_{0,1/r} \qquad \forall\ f \in \{f \in C^1(\bar{\Omega}) \mid f|_{\Gamma_1} = 0\}.$$

This assumption guarantees the existence of Poincaré inequalities (see [11]) for the indicated spaces, and depends upon Γ_1 being "orthogonal" to the streamlines of the mean flow, and the mean flow velocity, $\mathbf{V}$, being "essentially parallel" near the axis. By "essentially parallel" we mean that there are positive constants c and C such that if $r(s)$ it the radial coordinate of the streamline passing through Γ_1 at $r = r(0)$, then $cr(0) \leq r(s) \leq Cr(0)$ (s is the streamwise variable for the mean flow: $\partial/\partial s = \mathbf{V}.\nabla$, $s = 0$ on Γ_1). When the mean flow is smooth, this condition will guarantee the existence of the Poincaré constants $K_{\mathcal{A}}$ and $K_{\mathcal{V}}$.

Let the Hilbert space $\mathcal{H}$ be defined as the product of the five function spaces indicated above, and let the bilinear form $\mathbf{a} : \mathcal{H} \times \mathcal{H} \to \mathbb{C}$ be given by

$$\begin{aligned}
&\mathbf{a}[(\phi,\psi,h,a_\theta,\tilde{v}_\theta),(\eta,\xi,g,b,u)] = \\
&\int_\Omega \Bigg\{ r\left[-\frac{ik}{C^2}p\bar{\eta} + (R\nabla\phi + Rr\,curl\psi + \frac{p}{C^2}\mathbf{V}).\nabla\bar{\eta} + \frac{m^2}{r^2}R\phi\bar{\eta} - \frac{im}{r}R\tilde{v}_\theta\bar{\eta}\right] \\
&\quad + \frac{1}{r}\left[\nabla h.\nabla\bar{g} - r[(Ra_\theta + \frac{p}{C^2}A_\theta)\mathbf{V} + RA_\theta\mathbf{v}].curl\bar{g}\right] \\
&\quad + r\left[ika_\theta + \mathbf{V}.\nabla a_\theta + \mathbf{v}.\nabla A_\theta - \frac{im}{r}A_\theta(\tilde{v}_\theta + \frac{im}{r}\phi)\right]\mathbf{V}.\nabla\bar{b} \\
&\quad + \frac{1}{r}\left[ikr\tilde{v}_\theta + \mathbf{V}.\nabla(r\tilde{v}_\theta) + im(h - r\mathbf{V}.curl\psi)\right]\mathbf{V}.\nabla(r\bar{u}) \\
&\quad + r\left[\nabla\psi.\nabla\bar{\xi} + (Ra_\theta + \frac{p}{C^2}A_\theta)\bar{\xi}\right]\Bigg\}.
\end{aligned}$$

In the above, $\mathbf{v}$ and p can be written in terms of $(\phi,\psi,h,a_\theta,\tilde{v}_\theta)$ using (6). A variational formulation of equations (5) and (7) may then be posed as the following problem.

WEAK PROBLEM. *Find* $(\phi,\psi,h,a_\theta,\tilde{v}_\theta) \in \mathcal{H}$ *such that*

$$\mathbf{a}[(\phi - \hat{\phi},\psi,h - \hat{h},a_\theta - \hat{a}_\theta,\tilde{v}_\theta - \hat{v}_\theta),(\eta,\xi,g,b,u)] = F(\eta,\xi,g,b,u) \tag{8}$$

for all $(\eta,\xi,g,b,u) \in \mathcal{H}$, *where the linear functional* $F : \mathcal{H} \to \mathbb{C}$ *is defined by,*

$$F(\eta,\xi,g,b,u) = \int_{\Gamma_2} l_1\bar{\eta}.$$

3.1. Solution of the weak problem. In this section we outline the proof of existence and uniqueness for the weak problem presented above.

ASSUMPTION 2.

1. $R, C \in L^\infty(\Omega)$.
2. There are constants R_0, $C_0 > 0$ such that $R \geq R_0$ and $C \geq C_0$ almost everywhere on Ω.
3. $\left\|\frac{1}{r}A_\theta\right\|_\infty < \infty$ and $\|\nabla A_\theta\|_\infty < \infty$.
4. $\mathbf{V} \in L^\infty(\Omega)^2$ and $M = \left\|\frac{1}{C}\,|\mathbf{V}|\right\|_\infty < 1$.

Remark. If $\nabla A_\theta \in L^\infty(\Omega)^2$, it is not unreasonable to expect $1/rA_\theta \in L^\infty(\Omega)$, since if Ω_θ and R are smooth

$$\frac{\partial A_\theta}{\partial r} = \frac{\partial}{\partial r}\left(\frac{\Omega_\theta}{Rr}\right) = -\frac{A_\theta}{r} + \frac{1}{Rr}\frac{\partial \Omega_\theta}{\partial r} - \frac{A_\theta}{R}\frac{\partial R}{\partial r}.$$

LEMMA 1. *If the mean flow satisfies Assumptions* 1 *and* 2, *then the sesquilinear form* $\mathbf{a} : \mathcal{H} \times \mathcal{H} \to \mathbb{C}$ *is continuous. If* $l_1 \in L^2_r(\Gamma_2)$, *then* $F : \mathcal{H} \to \mathbb{C}$ *is continuous.*

Remark. The hypotheses on l_1 is slightly stronger than necessary; it need only belong to a certain dual space. In practice, the given hypothesis is convenient since the dual space has a nontrivial characterization.

In order to demonstrate coercivity of $\mathbf{a}$, it is convenient to split it into the sum of a principal part, $\mathbf{a}_1$, and a part, $\mathbf{a}_2$, that can be made arbitrarily small by requiring the frequency, k, and the mean flow vorticity, A_θ, to be small. Let

$$\begin{aligned}
&\mathbf{a}_1[(\phi,\psi,h,a_\theta,\tilde{v}_\theta),(\eta,\xi,g,b,u)] = \\
&\int_\Omega \Bigg\{ R\Bigg[r\left(\nabla\phi - \frac{1}{C^2}(\mathbf{V}.\nabla\phi)\mathbf{V} + r\,curl\psi - \frac{r}{C^2}(\mathbf{V}.curl\psi)\mathbf{V} + \frac{1}{C^2}h\mathbf{V}\right) \\
&\qquad\qquad \cdot\nabla\overline{\eta} + \frac{m^2}{r}\phi\overline{\eta} - im\tilde{v}_\theta\overline{\eta}\Bigg] \\
&\quad + \frac{1}{r}\left[\nabla h.\nabla\overline{g} - Rra_\theta\mathbf{V}.curl\overline{g}\right] + r(\mathbf{V}.\nabla a_\theta)(\mathbf{V}.\nabla\overline{b}) \\
&\quad + \frac{1}{r}\left[\mathbf{V}.\nabla(r\tilde{v}_\theta) + im(h - r\mathbf{V}.curl\psi)\right]\mathbf{V}.\nabla(r\overline{u}) + r\left[\nabla\psi.\nabla\overline{\xi} + Ra_\theta\overline{\xi}\right]\Bigg\},
\end{aligned}$$

and

$$\begin{aligned}
&\mathbf{a}_2[(\phi,\psi,h,a_\theta,\tilde{v}_\theta),(\eta,\xi,g,b,u)] = \\
&\int_\Omega \Bigg\{ -r\frac{ik}{C^2}\left[p\overline{\eta} + \phi\mathbf{V}.\nabla\overline{\eta}\right] - A_\theta\left[\frac{p}{C^2}\mathbf{V} + R\mathbf{v}\right].curl\overline{g} \\
&\quad + r\left[ika_\theta + \mathbf{v}.\nabla A_\theta - \frac{im}{r}A_\theta(\tilde{v}_\theta + \frac{im}{r}\phi)\right]\mathbf{V}.\nabla\overline{b} + ik\tilde{v}_\theta\mathbf{V}.\nabla(r\overline{u}) + \frac{p}{C^2}A_\theta r\overline{\xi}\Bigg\}
\end{aligned}$$

LEMMA 2. *If the mean flow satisfies Assumptions* 1 *and* 2, *then there are constants* Δ_1 *and* $\Delta_2 > 0$ *such that for each* $(\phi,\psi,h,a_\theta,\tilde{v}_\theta) \in \mathcal{H}$

$$\sup_{(\eta,\xi,g,b,u)\in\mathcal{H}-\{0\}} \frac{\mathbf{a}_1[(\phi,\psi,h,a_\theta,\tilde{v}_\theta),(\eta,\xi,g,b,u)]}{\|(\eta,\xi,g,b,u)\|_{\mathcal{H}}} \geq \Delta_1 \|(\phi,\psi,h,a_\theta,\tilde{v}_\theta)\|_{\mathcal{H}},$$

and for each $(\eta,\xi,g,b,u)\epsilon\mathcal{H}$

$$\sup_{(\phi,\psi,h,a_\theta,\tilde{v}_\theta)\in\mathcal{H}-\{0\}} \frac{\mathbf{a}_1[(\phi,\psi,h,a_\theta,\tilde{v}_\theta),(\eta,\xi,g,b,u)]}{\|(\phi,\psi,h,a_\theta,\tilde{v}_\theta)\|_{\mathcal{H}}} \geq \Delta_2 \|(\eta,\xi,g,b,u)\|_{\mathcal{H}},$$

where Δ_1 *and* Δ_2 *depend only upon* m, M, C_0, R_0, $\|R\|_\infty$, $\|\mathbf{V}\|_\infty$, *the diameter of* Ω, *and Poincaré constants.*

The decomposition of $\mathbf{a}$ was chosen in order to get Δ_1 and Δ_2 independent of k and A_θ. The first estimate in the lemma is proven by putting $(\eta, \xi, g, b, u) = (\phi, \psi + \beta, h + \gamma, a_\theta + \alpha, \tilde{v}_\theta + \nu)$ and selecting β, γ, α, and ν judiciously. The second estimate follows by selecting $(\phi, \psi, h, a_\theta, \tilde{v}_\theta) = (\eta + \alpha, \xi + \beta, g + \gamma, b, u + \nu)$ with an appropriate choice of β, γ, α, and ν. The proof is similar in spirit to that in [12], where the plane case is considered.

LEMMA 3. *If the mean flow satisfies Assumptions* 1 *and* 2, *then*

$$|\mathbf{a}_2[(\phi, \psi, h, a_\theta, \tilde{v}_\theta), (\eta, \xi, g, b, u)]| \leq \delta \ \|(\phi, \psi, h, a_\theta, \tilde{v}_\theta)\|_{\mathcal{H}} \ \|(\eta, \xi, g, b, u)\|_{\mathcal{H}}$$

where

$$\delta = C \max\left[k, \left\|\frac{1}{r} A_\theta\right\|_\infty, \|\nabla A_\theta\|_\infty\right] . \max[1, k],$$

and C *depends only upon* m, M, C_0, R_0, $\|R\|_\infty$, $\|\mathbf{V}\|_\infty$, *the diameter of* Ω, *and Poincaré constants.*

The proof of this lemma follows from the Hölder and Poincaré inequalities. The above lemmas show that all the hypotheses for the generalized Lax–Milgrim theorem [7] are satisfied by the weak problem.

THEOREM. *If the mean flow satisfies Assumptions* 1 *and* 2, *and the frequency,* k, *and the mean flow vorticity,* A_θ, *are sufficiently small to guarantee* $\delta < \min[\Delta_1, \Delta_2]$ (*where* δ, Δ_1, *and* Δ_2 *are the constants from Lemmas* 2–3), *then the weak problem has a unique solution.*

4. Numerical experiments. Since the generalized Lax–Milgrim theorem was used to solve the weak form of equations (5), standard error estimates show that if $\mathcal{H}^h$ is a closed subspace of $\mathcal{H}$, then the solution given by the Galerkin approximation of the weak problem will satisfy

$$\left\|(\phi, \psi, h, a_\theta, \tilde{v}_\theta) - (\phi, \psi, h, a_\theta, \tilde{v}_\theta)^h\right\| \leq C \inf_{(\eta,\xi,g,b,u)\in\mathcal{H}^h} \|(\phi, \psi, h, a_\theta, \tilde{v}_\theta) - (\eta, \xi, g, b, u)\| .$$

Standard approximation estimates will not apply to $\mathcal{H}$ since nonstandard weights are utilized, and essentially only one derivative is defined for the spaces containing a_θ and $\tilde{v}_\theta$. In this section we present the results of several numerical experiments which suggest that $\mathcal{H}$ may be well approximated using standard finite element subspaces.

When dealing with weights of the form $1/r$ care must be taken to guarantee that an approximating function is integrable with the given weight ($L^2_r(\Omega) \subset L^2(\Omega)$, but $L^2_{1/r}(\Omega) \not\subset L^2(\Omega)$). If an approximating function, f, is a polynomial that vanishes on $r = 0$, it is clear that $f \in L^2_{1/r}(\Omega)$; however, finite element subspaces are not polynomials, they are constructed by composing polynomials with the inverse of a polynomial map $F : \mathbb{R}^2 \to \mathbb{R}^2$. It is then natural to ask what conditions on F guarantee the property that polynomials vanishing on $F^{-1}(0, z)$ will still yield basis functions in $L^2_{1/r}(\Omega)$. Figure 2(a) shows that gross distortions of a parent element may destroy this property. The following elementary lemma gives one answer to this question.

LEMMA. *Let* $F : \hat{\bar{\Omega}} \to \bar{\Omega}$ *be a polynomial homeomorphism mapping the axis* $\eta = 0$ *to* $r = 0$, *i.e.,* $F(\eta, \xi) = (r(\eta, \xi), z(\eta, \xi))$, *where* $r(.,.)$ *and* $z(.,.)$ *are polynomials with* $r(0, \xi) = 0$ (*see Fig.* 2(b)). *If* $f \in L^2_{1/\eta}(\hat{\Omega})$, *and for each* $\epsilon > 0$ *there is* $\delta > 0$ *such*

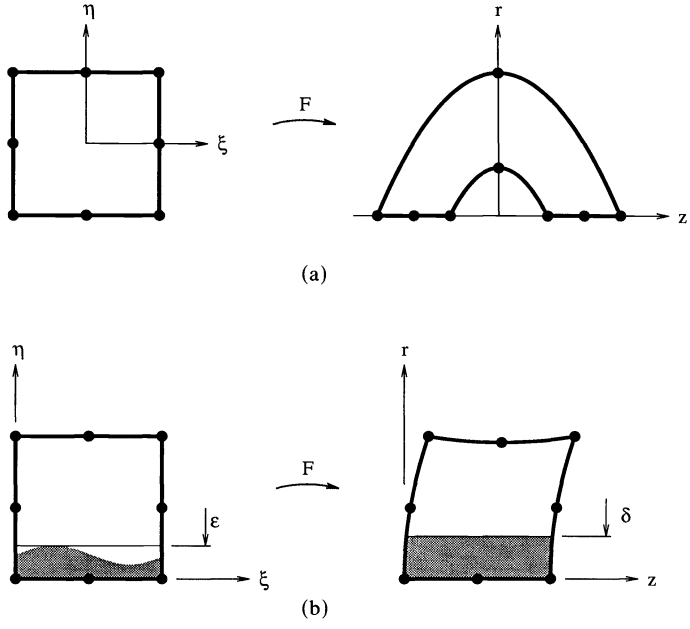


FIG. 2. (a) *Gross distortion of an element,* (b) *"reasonable" element distortion.*

that $F^{-1}\{(r,z) \in \Omega \mid 0 < r < \delta\} \subset \{(\eta,\xi) \in \hat{\Omega} \mid 0 < \eta < \epsilon\}$, *then* $f \circ F^{-1} \in L^2_{1/r}(\Omega)$. This lemma shows that for reasonable meshes, a standard finite element basis function vanishing on $r = 0$ will be in $L^2_{1/r}(\Omega)$. It is not always necessary to explicitly implement the boundary condition on $r = 0$, since if the integral $\int_\Omega \phi_i^2/r$ occurs in the weak problem, where ϕ_i is a basis function that is not zero on the axis, numerical quadrature will automatically produce a large positive number in the (i,i)-diagonal, causing the coefficient of this basis function to vanish. This reasoning applies to the variable ϕ in the weak form of (5); however, it is necessary to force h to be zero on the axis, since the term $\int_\Omega h^2/r$ does not explicitly appear in the weak problem.

While a boundary condition will guarantee that $h \in L^2_{1/r}(\Omega)$ when standard finite element basis functions are used, there is no guarantee that $\nabla h \in L^2_{1/r}(\Omega)^2$; in fact, this is not true for the standard finite element basis functions vanishing on the axis. However, the term $\int_\Omega |\nabla h|^2/r$ does appear in the weak problem, so one may suspect that the numerical quadrature will "take care" of this problem by analogy with the argument in the previous paragraph. A more rational approach would be to modify the basis functions so that their gradients are in $L^2_{1/r}(\Omega)$. This is easy to do since polynomial basis functions vanishing on $\eta = 0$ are of the form $\eta p(\eta,\xi)$, where p is also a polynomial. One can then adjust the factor of η to be η^γ, with $\gamma > 1$. Both standard finite element basis functions and modified basis functions with $\gamma = \frac{3}{2}$ were used to approximate h for the uniform duct solution reported below. We refer to the standard finite element basis functions as nonconforming approximations, and the modified basis functions as a conforming approximations.

In order to construct suitable basis functions for a_θ and $\tilde{v}_\theta$, it will be assumed that the streamlines of the mean flow trace out a coordinate system as indicated in Fig. 3. Subspaces can then be constructed by aligning "standard" elements within a

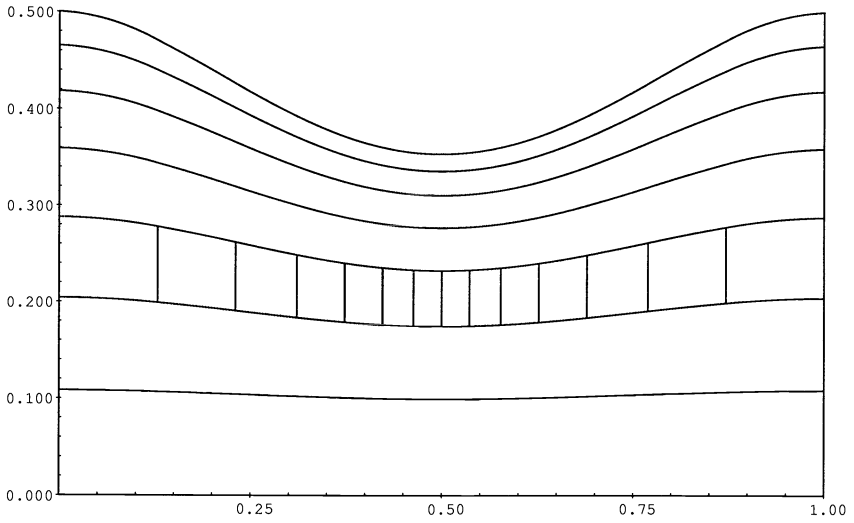


FIG. 3. *Streamlines of a typical stream tube.*

"stream tube," as indicated in the figure, and requiring interelement continuity within each tube. No continuity is required between adjacent stream tubes. All examples reported were calculated using eight noded isoparametric quadrilateral elements [7] to approximate $\mathcal{H}$.

PRESSURE CALCULATIONS. Since the acoustic pressure is the quantity measured in most applications, it was calculated as the solution to

$$p \in L^2_r(\Omega) \qquad \int_\Omega r\, p\bar{q} = \int_\Omega r\, R[h - ik\phi - \mathbf{V}.(\nabla\phi + r\, curl\psi)]\bar{q} \qquad \forall\, q \in L^2_r(\Omega).$$

This weak problem was approximated using four noded quadrilateral elements that were aligned with the eight noded elements used to approximate $\mathcal{H}$.

A more natural choice would be to require $p \in L^2_{1/r}(\Omega)$, since it is expected that $p|_{r=0} = 0$, and to chose the weight in the above weak problem to be $1/r$. Unfortunately, the right-hand side would not be continuous with this choice, since $\nabla\phi \in L^2_r(\Omega)^2$. (See also the remark at the end of Appendix A.)

4.1. Uniform duct results. The solution of (5) for a uniform duct with a uniform mean flow is given in Appendix B. A judicious choice of wave amplitudes could be made to yield $\psi\ |_{\Gamma_1 \cup \Gamma_2} = 0$; however, this is unnecessary. The condition $\psi\ |_{\partial\Omega^+} = 0$ may be relaxed to

$$\psi|_{\Gamma_1} = \hat{\psi}, \qquad \psi|_{\Gamma} = 0, \qquad \frac{\partial\psi}{\partial n}|_{\Gamma_2} = l_2,$$

provided the boundary condition for h on Γ_2 is similarly changed,

$$\frac{\partial h}{\partial n} + (V_r\omega_\theta + v_z\Omega_\theta)n_r - (V_z\omega_\theta + v_r\Omega_\theta)n_z = l_3$$

(where $\hat{\psi}$, l_2, and l_3 are specified) and the results of §3 will still hold. This enables any combination of waves to be specified on Γ_1.

For the first example, the mean flow was specified by $R = C = 1$, $A_\theta = 0$, $\mathbf{V} = -0.5\mathbf{e}_z$ in a duct of radius $a = 0.5$ and length $l = 1.0$. Boundary data corresponding

to the rotational solution with the first root of $J'_m(0.5r)$ was specified, with $k = 1$ and spinning mode number $m = 2$.

Tables 1 and 2 show the errors for this first example using the conforming and nonconforming elements, respectively. The conforming elements exhibit optimal L^2 rates convergence for each component except $\tilde{v}_\theta$. Since both a_θ and $\tilde{v}_\theta$ are governed by first-order equations one may expect their rates of convergence to be similar; in fact, when the mean flow is parallel, the spaces for a_θ and $\tilde{v}_\theta$ coincide, as do the principal parts of their equations. However, when the mean flow is parallel, the terms which couple the equation for a_θ with the gradients of other quantities vanish, unlike the equation for $\tilde{v}_\theta$. Since derivatives converge at lower rates, this could explain the difference in the L^2 rates for a_θ and $\tilde{v}_\theta$.

TABLE 1
Norms for example 1, *nonconforming elements.*

Mesh	ϕ	ψ	h	a	$\tilde{v}_\theta$	p
$\|\cdot\|_{0,r}$ or $\|\cdot\|_{0,1/r}$						
Exact norm	0.36328	2.5450	6.1206	69.067	2.1166	0.0
Error 5×10	2.9065E-3	1.1288E-3	7.1781E-3	3.5651E-2	8.2225E-2	2.2590E-3
Error 10×20	2.0912E-4	1.4166E-4	8.4602E-4	4.4501E-3	2.1013E-2	2.8105E-4
Rate	3.80	2.99	3.08	3.00	1.97	3.01
1–Seminorm						
Exact norm	0.49884	13.257	31.888	69.067	2.1166	-
Error 5×10	1.0626E-2	7.3602E-2	4.4013E-1	1.0996E-1	1.0868E-1	-
Error 10×20	2.0776E-3	1.8438E-2	1.0767E-1	2.6194E-2	2.7800E-2	-
Rate	2.35	2.00	2.03	2.07	1.97	-

TABLE 2
Norms for example 1, *conforming elements.*

Mesh	ϕ	ψ	h	a	$\tilde{v}_\theta$	p
$\|\cdot\|_{0,r}$ or $\|\cdot\|_{0,1/r}$						
Exact norm	0.36328	2.5450	6.1206	69.067	2.1166	0.0
Error 5×10	9.1529E-3	1.2288E-3	1.4897E-2	3.5651E-2	8.6690E-2	2.4661E-3
Error 10×20	2.2850E-3	1.4166E-4	3.2893E-3	4.4510E-3	2.2165E-2	3.2924E-4
Rate	2.00	3.12	2.18	3.00	1.98	2.91
1–Seminorm						
Exact norm	0.49884	13.257	31.888	69.067	2.1166	-
Error 5×10	1.2760E-2	7.3602E-2	9.0706E-1	1.0996E-1	1.1464E-1	-
Error 10×20	2.6491E-3	1.8438E-2	3.8706E-1	2.6194E-2	2.9333E-2	-
Rate	2.27	2.00	1.23	2.07	1.97	-

Comparing the results of the nonconforming and conforming elements shows that when conforming basis functions are used for h, the rates of convergence for both ϕ and h are significantly degraded. The results for ψ and a_θ did not change since, when the flow is parallel, the two equations for ψ and a_θ decouple from the others. This suggests that the conforming elements do not have optimal approximating properties for $H^1_{1/r}(\Omega)$. In practice, it is unusual to know the normal derivatives to specify on Γ_2 as indicated in equations (7). In lieu of this, it is common to specify a reflection (impedance) condition. A typical condition would be

$$\left.\begin{aligned} (R\mathbf{v} + \frac{1}{C^2}p\mathbf{V}).\mathbf{n} &= z_1\phi + z_2 h \\ \psi &= 0 \\ \frac{\partial h}{\partial n} + (V_r\omega_\theta + v_z\Omega_\theta)n_r - (V_z\omega_\theta + v_r\Omega_\theta)n_z = l_2 &= 0 \end{aligned}\right\} \text{ on } \Gamma_2.$$

The results of §3 do not immediately apply to these boundary conditions; however, if $Re(z_1) \leq 0$, the proofs can be modified to accommodate such boundary conditions. Typical values for z_1 and z_2 that satisfy the above requirements may be obtained by requiring that one of the *positively* propagating irrotational waves pass through Γ_2 without reflection. Setting

$$z_1 = R\delta_n, \qquad z_2 = \frac{RM}{C}$$

(see Appendix B) will allow the corresponding irrotational wave to pass through Γ_2.

The second example has the same mean flow as the previous one, and the irrotational wave corresponding to the second root of $J_m'(0.5r)$ was specified at Γ_1 with a frequency $k = 1.0$, and spinning mode number $m = 1$. The impedance boundary condition, was utilized on Γ_2. The results are shown in Table 3. The data for ψ, h, a_θ, and $\tilde{v}_\theta$ are not reported since they are all zero for this example, and the approximate solution for each of them was identically zero. Since $h = 0$ for this example, the conforming and nonconforming elements gave identical results. Again, the expected rates of convergence are achieved for ϕ and p.

TABLE 3
Norms for Example 2.

Mesh	$\|\phi\|_{0,1/r}$	$\|\phi\|_{1,r}$	$\|p\|_{0,r}$
Exact Norm	0.02136	0.05929	0.06312
Error 5×10	2.8518E-4	2.9709E-3	1.5024E-3
Error 10×20	3.7023E-5	7.2954E-4	3.9207E-4
Rate	2.94	2.03	1.94

4.2. Nonuniform duct results. No exact solutions for nonuniform ducts are available; however, qualitative behavior of the solution and numerical algorithm may be obtained from contour plots of the approximate solutions. All of the nonuniform duct results are for a converging–diverging duct, symmetric about the throat. The duct has radius $r = 0.5$ at the inlet and outlet and a throat radius $r = 0.5/\sqrt{2}$, (half the area); see Fig. 3. The duct radius is described by a degree-four polynomial, and the duct length is $l = 1.0$.

Three mean flows were used, two being variants of the first. The first mean flow was constructed by integrating the one-dimensional nozzle equations (see [10]) to obtain the density, speed of sound, and z component of velocity. The r component of velocity at each position z is proportional to r^2, varying from zero at $r = 0$ to the value required to make the mean flow tangential at the duct wall. A quadratic variation was chosen to guarantee that $\nabla A_\theta \in L^\infty(\Omega)$. Since highly sheared mean flows are known to cause difficulty with other algorithms [1], two perturbations of the first mean flow were considered, each obtained by scaling the velocity by a weight to simulate a steep boundary layer at the throat. The weights considered were:

1. $w_1(r,z) = \cos\left[2\pi\frac{z}{l}(1-\frac{z}{l})\left(\frac{r}{\mathcal{R}(z)}\right)^2\right]$,
2. $w_2(r,z) = \left[1-4(1-\epsilon)\frac{z}{l}(1-\frac{z}{l})\left(\frac{r}{\mathcal{R}(z)}\right)^3\right]^{1/7}$,

where $\mathcal{R}(z)$ is the duct radius at z, $0 \leq z \leq l = 1.0$. These weights were chosen to simulate problems that may occur if experimental or computational values were used

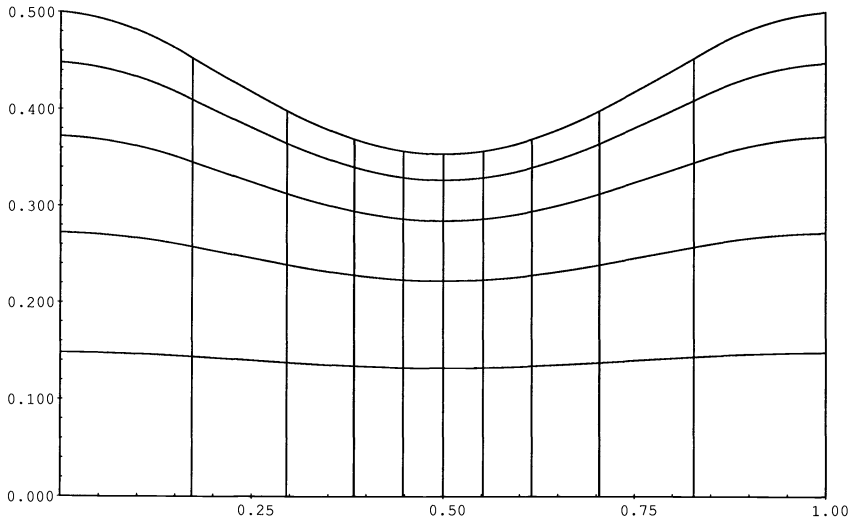


FIG. 4. *A 5 × 10 mesh for the nonuniform duct.*

for a mean flow. The first weight goes to zero at the upper wall in the throat; therefore, it is doubtful that Assumption 1 is valid in this situation. The $\epsilon > 0$ appearing in the second weight is necessary to guarantee that the vorticity A_θ is finite. When ϵ is small, Assumption 2 is almost violated; results are reported for $\epsilon = 10^{-6}$. The mean flow variables R, C, $\mathbf{V}$, and A_θ were all calculated at the nodal locations. Their values (and their derivatives) within each element were calculated using the finite element interpolation functions.

The mean flow varies rapidly near the throat at the upper wall; therefore, meshes were chosen which had a higher density of elements in this region. The locations of nodes in the z direction were obtained with a transformation of the form

$$\frac{z}{l}(\xi) = \frac{1}{2}\left[1 + \frac{\xi(1+\xi^2)}{2}\right], \qquad -1 \le \xi \le 1,$$

where equal increments of ξ were used to generate the nodal values. The locations of the nodes in the r direction at $z = 0$ were obtained with the transformation

$$\frac{r}{\mathcal{R}(0)}(\eta) = \eta\left[\frac{4 - 1.5\eta}{2.5}\right], \qquad 0 \le \eta \le 1;$$

again, equal increments of η yield the desired locations. To complete the mesh, the streamlines through each node at $z = 0$ were calculated using a numerical integration scheme, and the nodes within the duct were located where the streamlines cross the specified z locations. The streamlines for each of the mean flows are identical; therefore, the same meshes could be used for all three cases. A 5×10 mesh is pictured in Fig. 4.

For each of the mean flows, an irrotational wave corresponding to the first root of $J_m'(0.5r)$ was specified at $z = 0$ with $m = 1$ and $k = 1.0$. The corresponding impedance condition was specified at $z = 1.0$. Numerical experiments show that there is little qualitative difference between the conforming and nonconforming elements, the norms differing in the third significant figure. Accordingly, we present only the nonconforming solutions. However, the solution for the mean flow with weight w_2

was found to depend upon the machine representation of floating point numbers. The norms calculated on a VAX mainframe differed from those calculated on a DEC-2100 workstation in the third significant figure. Since this is the only example for which such differences exist, one presumes that the accuracy of the very large values of A_θ at the throat influence the solution.

A pressure contour plot of each solution for a 10×20 grid is given in Fig. 5, and the norms of each component of the three solutions is given in Tables 4 and 5, for a 5×10 and a 10×20 mesh. Qualitatively, the solutions appear to be converging, and do no exhibit any of the instabilities reported in [1]. The 1-seminorm (energy norm) for each component either changed negligibly or decreased, unlike unstable algorithms, which develop oscillations. The huge difference between the norms for the two meshes in the example with w_2 was observed to accumulate in a gradual manner with refinement through intermediate meshes (6×12, 7×14, etc.). The algorithm appears to be quite robust in the sense that the singularity at the throat doesn't "pollute" the whole solution; in fact its effect is only along the stream tube adjacent to it. This indicates that the algorithm reflects the qualitative behavior of Euler's equations for which disturbances in the vorticity propagate along stream lines.

TABLE 4
$L^2_r(\Omega)$ or $L^2_{1/r}(\Omega)$ norms of the nonuniform duct solutions.

Mesh	ϕ	ψ	h	a	$\tilde{v}_\theta$	p
No weight						
5×10	0.1259	0.02375	0.1075	2.0698	0.02414	0.06991
10×20	0.1259	0.02365	0.1071	2.0644	0.02405	0.06989
Weight w_1						
5×10	0.1358	0.03723	0.1859	20.298	0.1673	0.06461
10×20	0.1354	0.04019	0.1907	17.154	0.1123	0.06831
Weight w_2						
5×10	0.1539	0.1654	1.6101	1.5575E+3	0.4053	0.3196
10×20	0.1266	0.02127	0.1772	0.2313E+3	0.08285	0.07661

TABLE 5
1-seminorms of the nonuniform duct solutions.

Mesh	ϕ	ψ	h	a	$\tilde{v}_\theta$
No weight					
5×10	0.1566	0.1845	0.9491	3.6649	0.05316
10×20	0.1566	0.1839	0.9472	3.6731	0.05293
Weight w_1					
5×10	0.1530	0.3694	3.5209	18.233	0.1742
10×20	0.1531	0.3681	3.4152	16.734	0.1453
Weight w_2					
5×10	0.2977	3.1765	164.76	2.6755E+4	1.0510
10×20	0.1538	0.5181	33.80	0.5235E+4	0.1655

A. Velocity representation. In this section we recall some properties of weighted spaces and discuss the representation of the velocity $\mathbf{v} = \nabla\phi + r\,curl\psi$. In particular, we are interested in characterizing the velocity vectors that arise in this manner. We assume that $\Omega \subset \mathbb{R}^2_+$ is bounded, and denote by $\partial\Omega_+$ the closure of that part of the boundary away from the line $r = 0$.

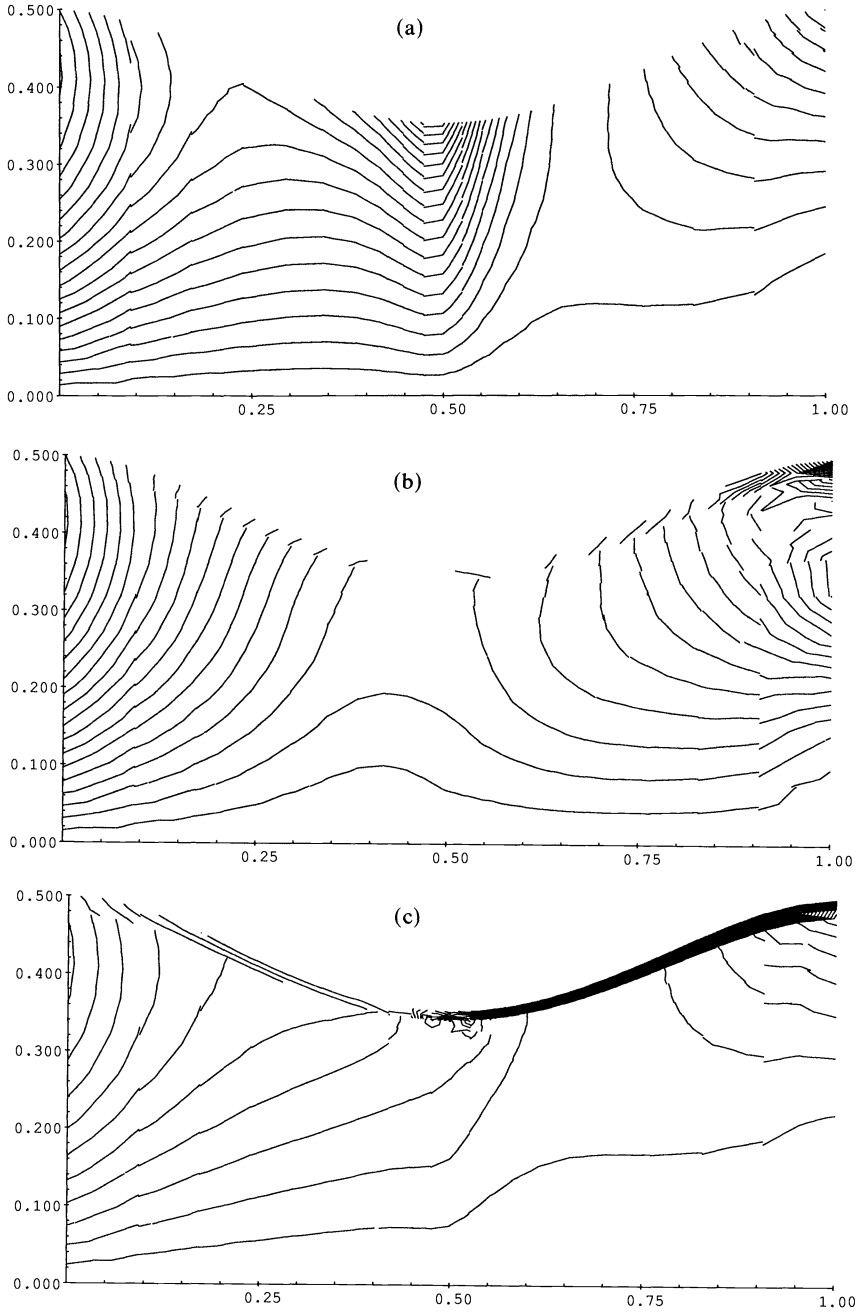


FIG. 5. *Nonuniform duct pressure contour plots:* (a) *no weight,* (b) *weight* w_1, *and* (c) *weight* w_2.

Consider the spaces defined by completing $C^1(\bar{\Omega})$ and $C_0^\infty(\Omega)$ in the norm

$$\|\psi\|_{1,r}^2 \equiv \int_\Omega r(|\psi|^2 + |\nabla\psi|^2),$$

and denote them as Ψ and Ψ_0, respectively. On $r = 0$ we do not have boundary values, i.e., the set of functions in $C^\infty(\bar{\Omega})$ whose support is disjoint from $\partial\Omega_+$ yield Ψ, when completed with $\|.\|_{1,r}$ This follows from the fact that the weight r satisfies

$\int_0^\rho 1/r\,dr = \infty$ (see [9]). An elementary exercise shows that there is a Poincaré inequality for Ψ of the form $\|\psi\|_{0,r} \leq const\ \|\nabla\psi\|_{0,r}$.

Next, define the space Φ to be the completion of smooth functions on $\bar{\Omega}$ whose support is bounded away from the line $r = 0$, in the norm

$$\|\phi\|_{1,1/r} \equiv \int_\Omega \frac{1}{r}(|\phi|^2 + |\nabla\phi|^2).$$

Unlike the space Ψ, functions in Φ will have a trace at $r = 0$ and it will be zero. That is if $(0, R) \times (z_1, z_2) \subset \Omega$ then the map from $(0, R)$ to $L^2(z_1, z_2)$ defined by $r \to \phi(r, .)$ is continuous almost everywhere when $\phi \in \Phi$; moreover, it has a limit of zero at $r = 0$. When the boundary of Ω is sufficiently smooth, there is a Poincaré inequality for Φ of the form $\|\phi\|_{0,1/r} \leq const\ \|\nabla\phi\|_{0,1/r}$.

Recall that when Euler's equations were linearized, it was assumed that

$$\mathbf{v}(r, \theta, z) = \mathbf{V}(r, z) + \mathbf{v}(r, z) \exp^{i(m\theta + kt)}.$$

In order to render $\mathbf{v} = (v_r, v_z)$ single valued at $r = 0$, it is natural to require $\mathbf{v}|_{r=0} = 0$. Since $\mathbf{v}$ is not expected to be a smooth function, a reasonable approach is to require

$$\int_\Omega \frac{1}{r} |\mathbf{v}|^2 < \infty.$$

We will show that such velocity fields may be represented as $\mathbf{v} = \nabla\phi + r\,curl\psi$, where $\phi \in \Phi$ and $\psi \in \Psi_0$. In order to avoid a lot of technical details with partitions of unity and corners, etc., we introduce the following definition.

DEFINITION. $\Omega \subset \mathbb{R}^2_+$ is "rectangular near $r = 0$" if, for some $\epsilon > 0$, the intersection of Ω with the strip $0 < r < \epsilon$ is a rectangle of the form $(0, \epsilon) \times (z_1, z_2)$ (or the finite union of such rectangles).

NOTATION. Ω^ϵ is the domain obtained by intersecting Ω with the half plane $\epsilon < r$, and $\partial\Omega^\epsilon_+$ is the intersection of $\partial\Omega_+$ with the half plane $\epsilon \leq r$. $\Psi_0(\Omega^\epsilon)$ is the completion of the set of functions $\{\psi \in C^1(\bar{\Omega}^\epsilon) \mid \psi|_{\partial\Omega^\epsilon_+} = 0\}$ with the norm $\|.\|_{1,r}$.

LEMMA A.1. *If Ω is rectangular near $r = 0$, then there is a bounded linear operator $\mathcal{P}_\epsilon : \Psi_0(\Omega^\epsilon) \to \Psi_0$ that extends the functions on Ω^ϵ to all of Ω. Moreover, the bound on $\mathcal{P}_\epsilon$ can be chosen independently of ϵ, i.e.,*

$$\|\mathcal{P}_\epsilon(\psi)\|_{\Psi_0} \leq C\,\|\psi\|_{\Psi_0(\Omega^\epsilon)},$$

where C is independent of ϵ.

This lemma may be proved using an extension defined by

$$\mathcal{P}_\epsilon(\psi)(r, z) = \begin{cases} \psi(r, z) & \epsilon \leq r \\ -3\psi(2\epsilon - r, z) + 4\psi\left(\frac{3}{2}\epsilon - \frac{r}{2}, z\right) & 0 < r < \epsilon. \end{cases}$$

This extension will be in $C^1(\bar{\Omega})$ if $\psi \in C^1(\bar{\Omega}^\epsilon)$. The assumption that Ω is rectangular near $r = 0$ is used to take care of the zero boundary condition requirements on $\partial\Omega_+$. A calculation shows that the constant can be chosen independently of ϵ.

LEMMA A.2. *Suppose that Ω is a Lipschitz, simply connected domain that is rectangular near $r = 0$. Let $\mathbf{w} \in L^2_r(\Omega)^2$ satisfy $\nabla.\mathbf{w} = 0$ in $\mathcal{D}(\Omega)'$, and suppose $\mathbf{w}.\mathbf{n} = 0$ on $\partial\Omega_+$ in the sense that*

$$\int_\Omega \mathbf{w}.\nabla\eta = 0$$

for all smooth functions η defined on $\bar{\Omega}$ whose support is bounded away from $r = 0$. Then there exists $\psi \in \Psi_0$ such that $\mathbf{w} = curl\psi$ almost everywhere.

Proof. The restriction, $\mathbf{w}^\epsilon$, of $\mathbf{w}$ to Ω^ϵ is in $L^2(\Omega^\epsilon)^2$ and satisfies $\mathbf{w}^\epsilon.\mathbf{n} = 0$ on $\partial\Omega^\epsilon_+$. The Helmholtz decomposition for the plane problem (see Girault and Raviart [5]) then guarantees the existence of a $\hat{\psi}^\epsilon \in H^1(\Omega^\epsilon)$ satisfying $curl\hat{\psi}^\epsilon = \mathbf{w}^\epsilon$ and whose trace vanishes on $\partial\Omega^\epsilon_+$. The extensions of these functions to all of Ω given by the previous lemma will be denoted by ψ^ϵ. Since $\|\nabla\psi^\epsilon\|_{0,r} \le C\left\|\nabla\hat{\psi}^\epsilon\right\|_{0,r} = C\|\mathbf{w}^\epsilon\|_{0,r} \le C\|\mathbf{w}\|_{0,r}$, it follows from the Poincaré inequality that the net $\{\psi^\epsilon\}$ is bounded in Ψ_0, so has a weak limit ψ. If $\mathbf{u} \in L^2_r(\Omega)^2$, then

$$\begin{aligned}\int_\Omega r(curl\psi - \mathbf{w}).\mathbf{u} &= \lim_{\epsilon\to 0}\int_\Omega r(curl\psi^\epsilon - \mathbf{w}).\mathbf{u}\\ &= \lim_{\epsilon\to 0}\int_{\Omega\setminus\Omega^\epsilon} r(curl\psi^\epsilon - \mathbf{w}).\mathbf{u}\\ \left|\int_\Omega r(curl\psi - \mathbf{w}).\mathbf{u}\right| &\le \lim_{\epsilon\to 0}(1+C)\|\mathbf{w}\|_{0,r}\|\mathbf{u}\|_{L^2_r(\Omega\setminus\Omega^\epsilon)}\\ &= 0.\end{aligned}$$

It follows that $\mathbf{w} = curl\psi$ almost everywhere □

THEOREM. *Suppose that Ω is a Lipschitz, simply connected domain that is rectangular near $r = 0$. If $\mathbf{v} \in L^2_{1/r}(\Omega)^2$, then there exist functions $\phi \in \Phi$ and $\psi \in \Psi_0$ such that $\mathbf{v} = \nabla\phi + r\,curl\psi$.*

Proof. Let $\phi \in \Phi$ satisfy the weak problem:

$$\int_\Omega \frac{1}{r}\nabla\phi.\nabla\eta = \int_\Omega \frac{1}{r}\mathbf{v}.\nabla\eta$$

for all $\eta \in \Phi$. Define $\mathbf{w} = (1/r)[\mathbf{v} - \nabla\phi]$. It is clear that $\mathbf{w}$ satisfies the hypotheses for the previous lemma, so $\mathbf{w} = curl\psi$. □

Remark. When solving the acoustics problem we sought

$$\phi \in \{\phi \in L^2_{1/r}(\Omega) \mid \nabla\phi \in L^2_r(\Omega)^2\}$$

which may be a slightly larger space than Φ, clearly the velocities we obtain will then lie somewhere in between $L^2_{1/r}(\Omega)^2$ and $L^2(\Omega)^2$.

B. Uniform duct solution. The solution of equations (5) in a uniform duct of radius a, with mean flow, $\mathbf{V} = (0, V)$, and constant speed of sound, C, and density, R, may be found by assuming exponential axial variation. As in the plane case, there are five solutions. Two solutions correspond to the usual irrotational forward and backward propagating waves. The third solution travels in the direction of the flow and is rotational. The final two solutions are not physical in the sense that the acoustic velocity and pressure are both zero. One decays in the forward direction and the other decays in the backward direction.

B.1. Irrotational solutions. Let $\alpha = \alpha_n$ correspond to a root of $J'_m(ar) = 0$, then put

$$\lambda = \lambda^\pm_n = \frac{i\hat{k}M \mp \delta_n}{1 - M^2}, \qquad \delta_n = \sqrt{(1-M^2)\alpha_n^2 - \hat{k}^2}.$$

The irrotational solution may then be written as

$$\phi = \frac{-1}{R(ik+\lambda V)} J_m(\alpha r)\, e^{\lambda z}, \qquad \psi = h = a = \tilde{v}_\theta = 0.$$

Other pertinent quantities are

$$p = J_m(\alpha r)\, e^{\lambda z}, \qquad v_r = \tfrac{-\alpha}{R(ik+\lambda V)} J_m'(\alpha r) e^{\lambda z},$$
$$v_\theta = \tfrac{-im}{Rr(ik+\lambda V)} J_m(\alpha r)\, e^{\lambda z}, \qquad v_z = \tfrac{-\lambda}{R(ik+\lambda V)} J_m(\alpha r)\, e^{\lambda z}.$$

B.2. Nonphysical solutions. Letting $\alpha = \alpha_n$ correspond to a root of $J_0(ar) = 0$ the nonphysical solutions may be written as

$$\phi = \mp r J_1(\alpha r)\, e^{\pm \alpha z}, \qquad \psi = J_0(\alpha r)\, e^{\pm \alpha z},$$

and

$$p = a = v_r = v_\theta = v_z = 0, \qquad \tilde{v}_\theta = -\frac{im}{r}\phi, \qquad h = ik\phi.$$

B.3. Rotational solution. Let $\lambda = -ik/V$ and $\alpha = \alpha_n$ be a root of $J_m'(ar) = 0$, then the following quantities can be written down explicitly:

$$p = 0,$$
$$v_r = \frac{-ik}{V}\alpha J_m'(\alpha r)\, e^{-\frac{ik}{V}z}, \qquad v_\theta = \frac{mk}{rV} J_m(\alpha r)\, e^{-\frac{ik}{V}z},$$
$$v_z = \alpha^2 J_m(\alpha r)\, e^{-\frac{ik}{V}z}, \qquad a = \frac{-\alpha}{Rr}\left[\left(\frac{k}{V}\right)^2 + \alpha^2\right] J_m'(\alpha r)\, e^{-\frac{ik}{V}z}.$$

ϕ and ψ may be calculated as the solution to

$$\phi'(r) + \lambda r\psi(r) = \lambda\alpha J_m'(\alpha r), \qquad -r\psi'(r) + \lambda\phi(r) = \alpha^2 J_m(\alpha r),$$

where the prime denotes d/dr. Then $\phi(r,z) = \phi(r)e^{\lambda z}$ and $\psi(r,z) = \psi(r)e^{\lambda z}$. If $\psi'(0)$ is bounded, appropriate boundary conditions are $\psi(a) = 0$ and $\phi(0) = \alpha^2 J_m(0)/\lambda$. $\psi'(0)$ may be calculated as $(\lambda^2 - \alpha^2)\alpha J_m'(0)$.

Given ϕ and ψ we calculate $\tilde{v}_\theta = v_\theta - (im/r)\phi$, and

$$h = \left[ik\phi + Vv_z + \frac{p}{R}\right] = \left[ik\phi + V\alpha^2 J_m(\alpha r)e^{\lambda z}\right].$$

A series expansion for ϕ and ψ, in $rJ_1(\beta_n r)$ and $J_0(\beta_n r)$ may be written down explicitly, where β_n are the roots of $J_0(ar) = 0$. However, we chose to numerically approximate the solution by guessing two values of $\psi(0)$, integrating from $r = 0$ to $r = a$ and taking the appropriate linear combination to satisfy the boundary condition $\psi(a) = 0$. In order to calculate the solution for small values of r, asymptotic values of the solution (and its derivatives) can be calculated near the axis.

B.3.1. Asymptotic values. Assuming $r\psi(r)$ and $r\psi'(r) \to 0$ as $r \to 0$, we obtain

$$\psi'(0) = (\lambda^2 - \alpha^2)\alpha J_m'(0), \qquad \tilde{v}_\theta(0) = 0,$$

$$\phi(0) = \begin{cases} 0 & \text{if } m \neq 1 \\ \lambda\alpha/2 & \text{if } m = 1 \end{cases} \qquad a_\theta(r) \sim \frac{(\lambda^2 - \alpha^2)}{R} e^{\lambda z} \times \begin{cases} \alpha/2r & \text{if } m = 1 \\ \alpha^2/4 & \text{if } m = 2 \\ 0 & \text{otherwise.} \end{cases}$$

Asymptotic estimates for the derivatives at $r = 0$ can be calculated similarly, and are required to evaluate the weighted Sobolev norms.

When $m = 1$ it is clear that a_θ is not bounded near the axis, and it is not difficult to show that in this instance the solution is not in the Hilbert space $\mathcal{H}$ constructed in §3. The series solution of for ϕ and ψ alluded to above also exhibits problems if $m = 1$. In particular, it is necessary to expand $J_m'(\alpha r)$ in terms of $rJ_0(\beta_n r)$, which will not converge at $r = 0$ when $m = 1$ since $J_1'(0) = \frac{1}{2}$. Also, the span of the $rJ_0(\beta_n r)$ will form a subspace of $H^1(0, a)$ which has zero trace at $r = 0$, so the series could never converge to $J_1'(\alpha r)$ in $H^1(0, a)$.

REFERENCES

[1] A. L. Abrahamson, AIAA-83-0666, 1983.

[2] C. Bernardi and Y. Maday, *Properties of some weighted Sobelov spaces and applications to spectral approximations*, SIAM J. Numer. Anal., 26 (1989), pp. 769–829.

[3] R. S. Falk and G. R. Richter, *Analysis of a continuous finite element method for hyperbolic equations*, SIAM J. Numer. Anal., 24 (1987), pp. 257–278.

[4] D. A. French, *The finite element method for a degenerate elliptic equation*, SIAM J. Numer. Anal., 24 (1987), pp. 788–815.

[5] V. Girault and R. A. Raviart, *Finite Element Approximation of the Navier–Stokes Equations*, Springer-Verlag, Berlin, 1979.

[6] B. Mercier and G. Raugel, *Résolution d'un problème aux limites dans un ouvert axisymétrique par élèments finis en r, z et séries de Fourier en θ*, RAIRO Modél. Math. Anal. Numer., 16 (1982), pp. 405–461.

[7] J. T. Oden and G. F. Carey, *Finite Elements, Vol. 4: Mathematical Aspects*, Prentice Hall Inc., Englewood Cliffs, NJ, 1983.

[8] A. V. Parret, *Applications of finite and wave envelope element approximations to acoustic radiation from turbofan engine inlets in flight*, Ph.D. thesis, University of Missouri-Rolla, MO, 1984.

[9] E. H. Poulson, *Boundary value problems connected with some improper Dirichlet integrals*, Math. Scand., 8, (1960), pp. 5–14.

[10] J. Serrin, *Mathematical principles of classical fluid mechanics*, Hanbuch der Physik, Vol. VIII/I, Springer-Verlag, Berlin, 1959.

[11] R. E. Showalter, *Hilbert Space Methods for Partial Differential Equations*, Pitman, San Francisco, CA, 1977.

[12] N. J. Walkington, *Acoustic wave propagation through flows with vorticity*, SIAM J. Numer. Anal., 25 (1988), pp. 533–549.

[13] N. J. Walkington, T. M. Wicks, and W. Eversman, *Fundamental investigations of the finite element solutions for acoustic propagation in ducts*, NASA Langley Report NAG-1-198.

[14] S. Z. Zhou, *The linear finite element method for a singular two-dimensional boundary value problem*, SIAM J. Numer. Anal., 20 (1983), pp. 976–984.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1457–1470, November 1991
© 1991 Society for Industrial and Applied Mathematics
013

PARALLELIZATION OF ROBUST MULTIGRID METHODS: ILU FACTORIZATION AND FREQUENCY DECOMPOSITION METHOD*

PETER BASTIAN† AND GRAHAM HORTON†

Abstract. The parallelization of two multigrid methods that are robust for systems of linear equations arising from the discretization of certain singularly perturbed elliptic partial differential equations is presented. The multigrid method with ILU smoother and the Frequency Decomposition Method, based on a multiple coarse grid correction, were implemented on a MIMD computer with distributed shared memory using a ring configuration for the first and a hypercube configuration for the second method. The speedups were determined for various grid sizes and numbers of processors. An objective comparison of both methods against the multigrid method with highly parallelizable red-black Gauß–Seidel smoother is made for the anisotropic equation and shows the superiority of both methods over the standard approach. The speedup of the ILU smoother is modelled and the influence of computer architecture on the speedup is discussed.

Key words. singularly perturbed partial differential equation, robust multigrid method, incomplete factorization, MIMD computer, distributed shared memory

AMS(MOS) subject classification. 65F10

1. Introduction.

In this paper we study the parallel solution of systems of linear equations arising from the discretization of partial differential equations which depend on one or more parameters. The equation is assumed to be basically of elliptic type, except for certain extremal values of the parameters. As model equations for this class of singularly perturbed problems we consider the anisotropic equation

$$-\alpha \frac{\partial^2 u}{\partial x^2} - \beta \frac{\partial^2 u}{\partial y^2} = f(x,y) \tag{1}$$

on $\Omega = [0,1] \times [0,1]$, with

$$u(x,y) = b(x,y) \text{ or } \frac{\partial u}{\partial n} = b(x,y) \quad \text{on } \partial\Omega \tag{2}$$

and $\alpha, \beta \geq 0$, and n the exterior normal. Further we consider the convection-diffusion equation

$$-\epsilon \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + c_1 \frac{\partial u}{\partial x} + c_2 \frac{\partial u}{\partial y} = f(x,y) \tag{3}$$

on $\Omega = [0,1] \times [0,1]$, $u(x,y) = b(x,y)$ on $\partial\Omega$, $\epsilon \geq 0$, $c_1, c_2 \in \mathbf{R}$.

The anisotropic equation is of elliptic type for $\alpha, \beta > 0$ and of parabolic type for $\alpha = 0$ or $\beta = 0$. The convection-diffusion equation is of elliptic type for $\epsilon > 0$ and of hyperbolic type and first order if $\epsilon = 0$. These two equations can serve as model equations for the numerical solution of the Navier–Stokes equations, where the influence of grids with high aspect ratio or a flow with high Reynolds number on the rate of convergence can be studied.

* Received by the editors August 11, 1989; accepted for publication November 12, 1990. This research was supported in part by the Stiftung Volkswagenwerk within its program "Entwicklung von Berechnungsverfahren für Probleme der Strömungstechnik."

† Institut für Mathematische Maschinen und Datenverarbeitung III, Universität Erlangen-Nürnberg, 8520 Erlangen, Federal Republic of Germany (graham@immd3.informatik.uni-erlangen.de).

A standard finite difference discretization of (1) with Dirichlet boundary conditions on an equidistant grid $\Omega_l = \{(ih_l, jh_l) | 0 \leq i,j \leq N_l, h_l = 1/N_l, N_l = 2^{l+1}\}$ yields a system of linear equations

$$A_l u_l = f_l, \tag{4}$$

where the index $l \in \mathbf{N}$ indicates the grid level within the multigrid process. A_l is a block tridiagonal matrix if the grid points are ordered lexicographically first in x then in y direction:

$$A_l = h_l^{-2} \begin{pmatrix} T_l & -\beta I & & & \\ -\beta I & T_l & -\beta I & & \\ & \ddots & \ddots & \ddots & \\ & & -\beta I & T_l & -\beta I \\ & & & -\beta I & T_l \end{pmatrix}. \tag{5}$$

The diagonal blocks T_l have the form

$$T_l = \begin{pmatrix} 2(\alpha+\beta) & -\alpha & & & \\ -\alpha & 2(\alpha+\beta) & -\alpha & & \\ & \ddots & \ddots & \ddots & \\ & & -\alpha & 2(\alpha+\beta) & -\alpha \\ & & & -\alpha & 2(\alpha+\beta) \end{pmatrix}. \tag{6}$$

Equation (4) will serve as the model equation throughout the paper.

If $K_l(\alpha,\beta)$ is the iteration matrix of an iterative method for the solution of the discrete problem (4), then the method $K_l(\alpha,\beta)$ is called robust for the anisotropic equation if and only if

$$\|K_l(\alpha,\beta)\| \leq \zeta < 1 \quad \forall \alpha,\beta \geq 0, \qquad l \in \mathbf{N} \tag{7}$$

with $\|\cdot\|$ the spectral norm.

We assume a central difference discretization of the diffusive terms of the convection-diffusion equation (3) and stable upwind differences for the convective terms. The robustness of a method for a discretization of (3) can be defined similarly.

The first method we consider is the multigrid method with ILU smoother. This method is suitable for a finite difference or finite volume discretization of a wide range of two-dimensional partial differential equations (see [11]) including (1) and (3). Several variants of the method have been published, for example, the semi-implicit procedure (SIP) of Stone [10], 7- and 9-point ILU, or Wittum's ILU_β. Wittum showed in [12] that the multigrid method with his ILU_β smoother is robust for the anisotropic equation. The ILU_β and the SIP method have both been successfully used in Navier–Stokes solvers (see [3], [13]). The parallelization of the ILU method presented in §3 is applicable to all of the above-mentioned variants and is described only for the basic 5-point version.

The second method we consider is the frequency decomposition multigrid algorithm introduced by Hackbusch in [5]. The range of applicability of the method in this form is not as large as for the first method. It is able to solve (1) but not (3). The motivation behind the method is as follows. Let $K_l(\alpha,\beta)$ denote the iteration matrix of an iterative algorithm for the solution of (4), then the error in the nth iterate $v_l^n = u_l - u_l^n$ is given by $v_l^n = K_l(\alpha,\beta) v_l^{n-1}$. With the eigenvectors $e_l^{\nu\mu}$ of A_l given by

$$(e_l^{\nu\mu})_{i,j} = \sin(\nu\pi i h_l)\sin(\mu\pi j h_l), \qquad 0 \leq \mu,\nu \leq N_l - 1 \tag{8}$$

we define the low frequencies $V_{00} = \text{span}\{e_l^{\nu\mu} | \nu, \mu \in \mathcal{L}\}$, the high frequencies in x-direction $V_{10} = \text{span}\{e_l^{\nu\mu} | \nu \in \mathcal{H}, \mu \in \mathcal{L}\}$, the high frequencies in y-direction $V_{01} = \text{span}\{e_l^{\nu\mu} | \nu \in \mathcal{L}, \mu \in \mathcal{H}\}$, and the high frequencies in both directions $V_{11} = \text{span}\{e_l^{\nu\mu} | \nu, \mu \in \mathcal{H}\}$ with $\mathcal{L} = \{1, \cdots, N_{l-1} - 1\}$ and $\mathcal{H} = \{N_{l-1}, \cdots, N_l - 1\}$. Standard multigrid theory assumes the smoother to reduce the high frequency errors $v \in V_{10} \cup V_{01} \cup V_{11}$ and the coarse grid correction to reduce the low frequency errors $v \in V_{00}$. Fourier analysis of the two-grid method with a damped Jacobi smoother or a red-black Gauß–Seidel smoother applied to (4) yields that the smoother fails to reduce the errors in V_{10} (V_{01}) if $\alpha \to 0$ ($\beta \to 0$). A first way to achieve robustness is to incorporate a more complex smoothing algorithm that works for all high frequencies, like the ILU method. Another approach is the frequency decomposition method, where four different coarse grid corrections are constructed to reduce the error in each of the spaces V_ι, $\iota \in \{00, 10, 01, 11\}$ on coarser grids. As smoother, a simple damped Jacobi is sufficient.

In the next section the principles of parallel multigrid algorithms will be reviewed briefly. Sections 3 and 4 describe the ILU iteration and the frequency decomposition method, respectively, together with the parallelization and speedup results for each method. In the fifth section the two methods will be compared with a parallel multigrid method using a highly parallelizable red-black Gauß–Seidel smoother.

2. Parallelization of multigrid methods. When we assume that a two-dimensional partial differential equation is discretized on a rectangular grid with $(N_l - 1) \times (N_l - 1)$ unknowns then each processor is assigned to a subset of the unknowns (data partitioning). In an one-dimensional arrangement of n processors called a ring configuration of length n, processor p, $p \in \{0, \cdots, n-1\}$, is assigned to the grid points $\{(i,j) | \max(1, pN_l/n) \le i < (p+1)N_l/n, 1 \le j < N_l\}$. In a $n \times m$ array configuration each processor is assigned to $N_l/n \times N_l/m$ unknowns. However, if the sidelength of the grid is not divisible by the number of processors in the appropriate direction, then some processors will be assigned more unknowns than others, which results in an unequal load balance. This is one source for loss of efficiency.

All components of the multigrid method require only local operations, i.e., computations at grid point (i,j) need only values at the grid points $(i \pm 1, j \pm 1)$. In addition most operations (defect calculation, restriction, ...) can be carried out in parallel at each point. However, the number of points that can be computed simultaneously varies between different smoothing procedures: This number is $(N_l - 1)^2$ for Jacobi smoothing, $(N_l - 1)^2/2$ for red-black smoothing and at most $N_l - 1$ for ILU and lexicographic Gauß–Seidel smoothers.

The speedup $S(n)$ of a parallel algorithm is defined as

$$S(n) = \frac{T_{Mono}}{T_{Multi}(n)} \tag{9}$$

with T_{Mono} the time needed for the execution of the program on a single computer and $T_{Multi}(n)$ the time taken on n processors. This is the factor by which computation time is reduced through the use of a parallel processor. The efficiency $E(n)$ of a parallel implementation is then defined as

$$E(n) = \frac{S(n)}{n}. \tag{10}$$

The speedup for all operations carried out on a fixed level l within the multigrid method depends heavily on the number of unknowns per processor, because a larger

proportion of computing time is spent on communication and the effects of unequal load distribution are more pronounced if the number of grid points per processor is small. This means that a high speedup can be achieved on the fine grids (assuming a large number of grid points per processor on the fine grids), whereas the speedup deteriorates on the coarser grids. Nevertheless, the overall speedup can be expected to be very high because most of the work is spent on the finest levels.

2.1. The DIRMU system. DIRMU is a multiprocessor consisting of 26 standalone microcomputers (8086/87) developed at the University of Erlangen–Nürnberg [6]. The processors communicate via distributed shared memories whereby each processor can be connected to the memories of up to seven other processors. In addition to this communication memory, called multiport memory, each processor has a private memory where the program and local data are stored. All variables shared with neighbouring processors are stored in the multiport memories in order to avoid explicit transportation of data. The distributed operating system DIRMOS supplies information about various possible connection structures like ring, array, tree, or hypercube to a parallel program. In this way it is possible to use a connection structure that best fits the problem. DIRMOS also allows multiple users to run parallel programs on different parts of the system simultaneously and independently of each other.

2.2. Transputers. In order to compare the speedups on different computer architectures, the ILU method was also implemented on an INMOS transputer system consisting of seven T800 transputers connected in a ring. In contrast to DIRMU, the transputer system is a message passing architecture, where the time for sending a message consists of a fixed message setup time and a portion that depends on the length of the message.

3. The incomplete LU factorization. In general, we allow a 5-point finite difference star with variable coefficients denoted by

$$A_l(i,j) = \begin{bmatrix} & A^N_{i,j} & \\ A^W_{i,j} & A^C_{i,j} & A^E_{i,j} \\ & A^S_{i,j} & \end{bmatrix}. \tag{11}$$

The discretization on a rectangular equidistant grid then yields a matrix with a sparsity pattern as for matrix A_l in (4). As long as we regard ILU as a single grid method we will omit the index l. ILU is an iterative method which uses an approximate factorization of A into a product of lower and upper triangular matrices L and U with prescribed sparsity pattern. In the 5-point version the entries of L and U are allowed to be nonzero on the five nonzero diagonals of A. L and U are then defined in star notation as follows:

$$L(i,j) = \begin{bmatrix} & 0 & \\ L^W_{i,j} & L^C_{i,j} & 0 \\ & L^S_{i,j} & \end{bmatrix}, \qquad U(i,j) = \begin{bmatrix} & U^N_{i,j} & \\ 0 & U^C_{i,j} & U^E_{i,j} \\ & 0 & \end{bmatrix}. \tag{12}$$

The matrix A is replaced by $LU - C$, where C is the error in the approximation. The iterative algorithm can be formulated without explicit calculation of C using the defect $d^n = f - Au^n$:

$$u^{n+1} = u^n + U^{-1}L^{-1}d^n = u^n + v^n, \tag{13}$$

where v^n is obtained from solving $LUv^n = d^n$ in two steps:

$$L\delta = d^n \tag{14}$$

$$Uv^n = \delta. \tag{15}$$

This approach is preferable to the use of C for three reasons; storage space for the diagonals of C is saved, the defect is available, since it must be calculated at every iteration for convergence testing, and the iteration is more numerically stable. The matrices L and U are triangular and therefore easily invertible once computed. If A is a tridiagonal matrix, the incomplete LU factorization is equal to the exact LU factorization and this explains why the method can be applied with great success to discretizations of the singularly perturbed problems (1) and (3), since the discretization matrices of these equations tend towards tridiagonal matrices for the limit values of the parameters in the singularly perturbed sense. Existence and stability aspects of the ILU decomposition process are covered in [9].

The entries of L and U are computed recursively from the entries of A in the following manner:

$$L^S_{i,j} = A^S_{i,j}, \qquad L^W_{i,j} = A^W_{i,j} \tag{16}$$

$$L^C_{i,j} = A^C_{i,j} - \frac{A^W_{i,j}A^E_{i-1,j}}{L^C_{i-1,j}} - \frac{A^S_{i,j}A^N_{i,j-1}}{L^C_{i,j-1}} \tag{17}$$

$$U^C_{i,j} = 1, \quad U^E_{i,j} = \frac{A^E_{i,j}}{L^C_{i,j}}, \qquad U^N_{i,j} = \frac{A^N_{i,j}}{L^C_{i,j}}. \tag{18}$$

The computation of the vectors δ and v^n is then given by forward and back substitution:

$$\delta_{i,j} = \frac{d_{i,j} - L^W_{i,j}\delta_{i-1,j} - L^S_{i,j}\delta_{i,j-1}}{L^C_{i,j}} \qquad i = 1(1)N-1;\ j = 1(1)N-1 \tag{19}$$

$$v^n_{i,j} = \delta_{i,j} - U^E_{i,j}v_{i+1,j} - U^N_{i,j}v_{i,j+1} \qquad i = N-1(-1)1;\ j = N-1(-1)1. \tag{20}$$

3.1. Parallelization. As described, the method consists of two parts: decomposition of A into L and U and the inversion of L and U by forward and backward substitution. First we will consider the decomposition process. The key to the parallelization lies in the observation that when Dirichlet or von Neumann boundary conditions are given, the "west" coefficients of the leftmost variables $A^W_{1,j}$ and the "east" coefficients of the rightmost variables $A^E_{N-1,j}$ are equal to 0. This means that the term $A^W_{i,j}A^E_{i-1,j}(L^C_{i-1,j})^{-1}$ drops out of (17) and therefore $L^C_{1,j}$ depends only on $L^C_{1,j-1}$. We therefore have the following data dependencies: $L^C_{i,j}$ can be computed as soon as $L^C_{i-1,j}$ and $L^C_{i,j-1}$ have been computed except on the west boundary, where $L^C_{1,j}$ depends only on $L^C_{1,j-1}$ and on the south boundary where $L^C_{i,1}$ depends only on $L^C_{i-1,1}$. Regarding the grid it follows that L^C at all locations (i,j) with $i+j=k$, k fixed, can be computed independently of each other from L^C at the locations $i+j=k-1$.

The two-dimensional grid is mapped onto a ring of processors as described in §2. The parallel algorithm then proceeds as follows: Processor 0 starts with its segment of the first row of grid points. After completion, a signal is sent to processor 1 which then starts to process its section of the first row. Meanwhile processor 0 has started with the computation of its part of the second row. All processors are running when

processor $n-1$ begins with the computation of its part of the first row. Algorithm 3.1, which is executed simultaneously by each processor, shows the parallel computation of the coefficients $L^C_{i,j}$, where the variables $x\min(p)$ and $x\max(p)$ contain the leftmost and rightmost grid point column stored by processor p. On DIRMU, the execution of routine signal_row(j) by processor p indicates the completion of its segment of row j by incrementing a counter in the multiport memory of its right neighbour. The procedure wait_row(j) waits for this signal in order to be able to proceed with the computation of row j. Note that in the DIRMU, implementation grids which are to be accessed by neighbouring processors are located in the multiport memory, the others in private memory.

ALGORITHM 3.1. *Parallel computation of the L^C entries.*

```
PROCEDURE computeLC (A, L);
BEGIN
    FOR j := 1 TO N − 1 DO
        wait_row(j);
        FOR i := x min(p) TO x max(p) DO
            L^C_{i,j} = A^C_{i,j} − A^W_{i,j}(L^C_{i−1,j})^{−1} A^E_{i−1,j} − A^S_{i,j}(L^C_{i,j−1})^{−1} A^N_{i,j−1};
        END;
        signal_row(j);
    END;
END computeLC ;
```

Zero entries in A^W and A^E are reproduced in L^W and U^E according to (16) and (18). Therefore, the forward substitution to compute δ leads to the same data dependencies as the computation of L^C and therefore can be parallelized using the same ideas. The data dependencies for the backward substitution are different. Here again the values of v^n at locations $i+j=k$ can be computed simultaneously but are dependent on values of v^n with $i+j=k+1$, which means that the parallel algorithm starts in the upper right corner of the domain instead of the lower left.

The extension of the ILU method to 9-point difference stars is straightforward. For the computation of the matrices L, U and the vector δ we have the following data dependencies: computations at grid point (i,j) need values at grid points $(i-1,j)$, $(i-1,j-1)$, $(i,j-1)$, and $(i+1,j-1)$. We can use the same basic algorithm with the exception that at processor boundaries the values at grid point $(i+1,j-1)$, located in the right neighbour are needed. In the memory coupled system this poses no problem, but in the message passing system an additional data transfer is necessary. In the backward substitution process for computing v^n the computations at grid point (i,j) need values at locations $(i+1,j)$, $(i+1,j+1)$, $(i,j+1)$, and $(i-1,j+1)$.

3.2. Speedup results. In order to better understand the behaviour of the parallel method we have developed a model for the speedup. Having an accurate model of the speedup has several advantages:

- The model can be used to predict speedups for large problems whose memory requirements exceed the capacity of the DIRMU system.
- Estimated speedups for larger processor configurations can be obtained.
- The influence of the multiprocessor architecture can be examined by variation of the model parameters.

Here a simplified version of the model is given:

TABLE 1
Predicted (P) and Measured (M) ILU efficiencies for the DIRMU multiprocessor.

h	Processors						
	2	4	6	8	12	16	20
1/32 P	0.92	0.84	0.73	0.71			
1/32 M	0.93	0.83	0.73	0.69			
1/64 P		0.93	0.89	0.86	0.77	0.75	0.63
1/64 M		0.95	0.91	0.86	0.77	0.73	0.62
1/128 P					0.90	0.87	0.81
1/128 M					0.93	0.88	0.82

TABLE 2
Predicted ILU efficiencies for large DIRMU configurations.

h	Processors					
	4	8	16	32	48	64
1/256	0.98	0.97	0.94	0.88	0.79	0.78
1/512	0.99	0.98	0.97	0.94	0.91	0.89
1/1024	0.99	0.99	0.98	0.97	0.94	0.93

$$S(n, N_l) = \frac{(N_l - 1)^2 * T_{OP}}{T_{startup} + T_{parallel}} \tag{21}$$

$$T_{startup} = (n-1) * T_{SY} + \left(N_l - 1 - \frac{N_l - 1}{n}\right) * T_{OP} \tag{22}$$

$$T_{parallel} = (N_l - 2) * T_{SY} + (N_l - 1) * \frac{N_l - 1}{n} * T_{OP}. \tag{23}$$

$T_{startup}$ is the time the rightmost processor must wait until it can begin its computation. $T_{parallel}$ is then the time taken until the parallel computations are complete. T_{SY} (synchronization time) is the time needed for a signal_row and wait_row pair, and T_{OP} (operation time) is the time needed for the arithmetical computations at a single grid point. We can obtain the following estimate for the resulting efficiency:

$$E(n, N_l) \geq \frac{1}{\frac{T_{SY}}{T_{OP}}\phi(\phi - 1) + \phi} \tag{24}$$

where $\phi = \frac{n}{N_l - 1} + 1$.

This estimate, which was pointed out to the authors by Wittum, shows how the ratio T_{SY}/T_{OP} will be an important factor in the resulting speedup. The smaller this ratio the higher will be the speedup for given values of n and N_l. Alternatively, it can be seen that a value of T_{SY}/T_{OP} that is not small will require a large number of grid points per processor to achieve a high efficiency. In a message passing system T_{SY} will include the message setup time and the time required to transport the value of a boundary point $L^C_{x\,\max(p),j}$ from one processor to the next, while in a memory coupled system T_{SY} is the time that passes from incrementing a variable in shared memory until the neighbouring processor recognizes that the variable has changed its value.

Tables 1 through 3 show the results for the DIRMU implementation. Table 1 shows a comparison of the measured efficiency E of the ILU method and the pre-

TABLE 3
Multigrid efficiency on the DIRMU. Multigrid data : ILU smoother, $\nu_1 = 1$, $\nu_2 = 0$, $\gamma = 1$, $h = 1/64$.

Processors	Efficiency
5	0.88
6	0.82
7	0.76
8	0.73
9	0.67
10	0.64
11	0.65
12	0.58
13	0.54
14	0.50
15	0.46
16	0.44

dictions for various configuration lengths and grid sizes obtained by an enhanced speedup model which includes specific hardware parameters of the DIRMU multiprocessor. Table 2 shows the predictions obtained for larger problems and configurations. Finally, Table 3 shows the efficiency of the multigrid method with ILU smoothing and $h = 1/64$ on the finest grid.

The model is found to predict the measured efficiency very well for the grid and configuration sizes available on DIRMU. It is seen that $h < 1/64$ gives "reasonable" grid sizes for DIRMU, with efficiencies better than 75 percent for up to 14 processors for $h = 1/64$, and up to 23 processors for $h = 1/128$. The results are typical for a parallel algorithm with data-distribution, where the efficiency decreases when the number of processors is increased, owing to the growing communication and synchronization requirements relative to the number of parallelizable operations. For the same reason large grids can be more efficiently processed than smaller ones.

The model was considered to be accurate enough to enable predictions for larger problems. Here efficiencies better than 75 percent can be expected with up to 64 processors for a grid of side length 256, and the efficiency is better than 90 percent for grids with side length 512 or larger.

Table 4 shows the efficiencies measured for the transputer implementation. Note that T_{OP} on the transputer system is about sixty times shorter than that of the DIRMU system, and that T_{SY} is about three times as small. The difference in the ratio of T_{SY} and T_{OP} means that the transputer system needs grids with twice the side length in order to achieve efficiencies comparable to DIRMU.

Note that it is possible to reduce the number of messages by grouping the grid lines [7]. With this modification, each processor computes k line segments, for some small k, before passing data on to the next processor. In this way, the efficiency for message-passing parallel computers with a high message setup overhead can be greatly enhanced. This has been shown both by an appropriate model and by direct simulation [8].

4. The frequency decomposition method. The reduction of high frequency errors on coarser grids is made possible by the introduction of new prolongation

TABLE 4
Efficiency for the ILU method on the transputer system.

h	Processors					
	2	3	4	5	6	7
1/32	0.89	0.79	0.76	0.67	0.62	0.58
1/64	0.94	0.88	0.87	0.83	0.79	0.73
1/128	0.97	0.94	0.93	0.90	0.87	0.85

operators p_{00}, p_{10}, p_{01}, and p_{11}, which are constructed such that an error function v on the coarse grid is mapped to the space V_ι by p_ι. p_{00} is identical to the usual bilinear interpolation operator. The restriction operators r_ι are constructed from p_ι by adjoint mappings: $r_\iota = \frac{1}{4} p_\iota^T$, which results in the usual full weighting operator for r_{00}. Now we can describe the two grid algorithm. Assuming that we enter the cycle with the nth iterate u_l^n, first a smoothing algorithm is applied ν_1 times on the fine grid. Then the defect $d_l = f - Au^n$ is calculated. Now the four restricted defects $d_{l-1}^\iota = r_\iota d_l$ can be computed and the four defect equations $A_{l-1}^\iota v_{l-1}^\iota = d_{l-1}^\iota$ have to be solved on the coarse grid. The coarse grid matrices are $A_{l-1}^\iota = r_\iota A_l p_\iota$ (Galerkin product). After the solution of the coarse grid equations the fine grid solution is corrected by $u_l^{n+1} = u_l^n + \sum_\iota p_\iota v_{l-1}^\iota$. Note that the four coarse grid corrections (restriction of the defect, solution of the coarse grid equations, prolongation of the corrections) can be computed independently of each other. As in the standard multigrid method the multigrid cycle is obtained by an approximate solution of the coarse grid equations with a recursive call (γ times) of the method.

As described above, the algorithm produces four matrices A_{l-1}^ι on level $l-1$. These produce 16 coarse grid matrices $A_{l-1}^{\iota\kappa}$ on level $l-2$, and in general 4^k coarse grid equations result at level $l-k$. Hackbusch showed that not all coarse grid equations are necessary and gave a criterion that results in the necessary coarse grid tree of Fig. 1. This criterion states that a coarse grid matrix $A_{l-k}^{\iota_1 \iota_2 \cdots \iota_k}$ can be omitted if

$$\exists \iota, \kappa \in \{\iota_1, \iota_2, \cdots, \iota_k\} : \ \iota \neq \kappa \wedge \iota, \kappa \neq 00. \tag{25}$$

Note that most of the nodes have two descendants (type II nodes in the following) and only some nodes have four descendants (type I nodes). The number of coarse grid equations on level $l-k$ is thus reduced to $3 \cdot 2^k - 2$. For further details of the method we refer to [5]. This gives us the following algorithm. The parameter κ is used to omit the unnecessary coarse grids.

ALGORITHM 4.1. *The frequency decomposition multigrid algorithm using the necessary coarse grids only.*

```
PROCEDURE fdm (l, A_l, u_l, f_l, κ);
BEGIN
    IF l = 0 THEN u_l := A_l^{-1} f_l
    ELSE
        u_l := S_l^{ν1}(u_l, f_l);                    (* pre-smoothing *)
        d_l := f_l − A_l u_l; v_l := 0;               (* coarse grid correction *)
        IF κ = 00 THEN
            I := {00, 10, 01, 11}
        ELSE
            I := {00, κ}
```

```
        END;
        FOR ι ∈ I DO
            d^ι_{l-1} := r_ι d_l;
            v^ι_{l-1} := 0;
            IF κ = 00 THEN λ := ι ELSE λ := κ END;
            FOR k := 1 TO γ DO fdm(l − 1, r_ι A_l p_ι, v^ι_{l-1}, d^ι_{l-1}, λ) END;
            v_l := v_l + p_ι v^ι_{l-1};
        END;
        u_l := u_l + v_l;
        u_l := S_l^{ν2}(u_l, f_l);                          (* post-smoothing *)
    END;
END fdm ;
```

It can be shown that the complexity of one application of Algorithm 4.1 is $O(n_l)$ if $\gamma = 1$ and $O(n_l \log n_l)$ if $\gamma = 2$ (n_l denotes the number of unknowns on the finest grid). For further details we refer to [5].

4.1. Parallelization. The frequency decomposition multigrid algorithm allows parallelism to be exploited in two ways [2]. First, each processor can be assigned to a subset of the unknowns as described in §2. A high speedup can be expected because simple Jacobi smoothing is sufficient. Second, task partitioning can be used on the coarser grids, where the speedup normally deteriorates (one task is one coarse grid correction).

Therefore we use a two-level strategy. The fine grids, where the number of grid points per processor is assumed to be large, are distributed amongst all processors of an array configuration as described in §2. On the coarser grids, the configuration is split into halves and each subconfiguration is assigned to a different coarse grid correction, thus avoiding the problem of the decreasing number of unknowns per processor usually associated with parallel multigrid methods. The level where the subdivision process starts can be prescribed by the parameter l_{subdiv}.

The subdivision strategy is as follows: If a $2n \times m$ array configuration is used on level l and a type II node is encountered, then the configuration is subdivided into two equal halves of size $n \times m$. Note that the whole defect grid function is needed in each of the two halves, which seems to require an explicit data transport, and that the situation is reversed when the two corrections have been computed. If a type I node is encountered, the configuration is also split into two equal halves, where first the 00 and 10 corrections are computed in parallel followed by the parallel computation of the 01 and 11 corrections. Now the strategy is applied recursively, whereby subconfigurations can be further divided.

The parallel computation of the 00 and 10 corrections, each on one half of the configuration, results in an unequal load balance, as can be seen from Fig. 1, where the dashed subtree (00 correction) has more nodes than the dotted subtree (10 correction). This can be partially balanced by using a different parameter γ in the multigrid cycle for type I and type II nodes. If $\gamma = 1$ is used when calling a type I node, and $\gamma = 2$ is used when calling a type II node, the work spent in the two subtrees is almost equal.

In order to make the exact solver on level 0 as efficient as possible, the level 0 grids should be located in a single processor. To achieve this, a configuration of the form $2^{n'} \times 2^{m'}$ has to be used on the finest level. Now, after $n' + m'$ subdividing steps, $2^{n'+m'}$ configurations of size 1×1 are reached. This restriction results in a lower bound for the parameter l_{subdiv}: $l_{subdiv} \geq n' + m'$.

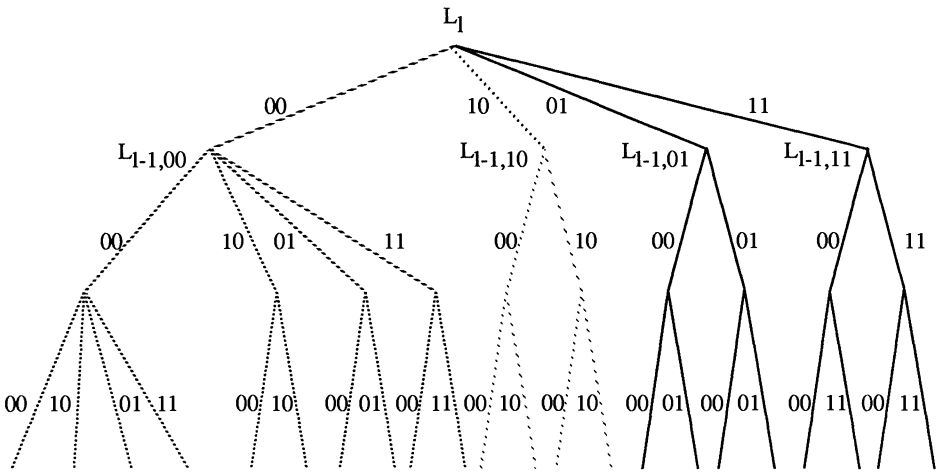


FIG. 1. *Necessary coarse grid tree. The dashed and dotted subtrees are computed in parallel.*

Using a hypercube configuration, it turns out that the explicit transfer of data mentioned above can be omitted in the DIRMU implementation. An array configuration, where the number of processors in both directions is a power of 2 can be mapped onto a hypercube configuration using so-called gray codes. If we assume a $2^{n'+1} \times 2^{m'}$ configuration that is subdived into two halves of size $2^{n'} \times 2^{m'}$ then in each row of processors the same data transports would be necessary. Therefore it is sufficient to consider one row of processors (this is a ring of length $2^{n'}$). Instead of taking the processors $\{0, \cdots, 2^{n'} - 1\}$ and $\{2^{n'}, \cdots, 2^{n'+1} - 1\}$ as the two halves we take the processors

$$Left = \{p \mid 0 \le p < 2^{n'+1},\ p \bmod 4 = 0 \ \vee\ p \bmod 4 = 3\}$$
$$Right = \{p \mid 0 \le p < 2^{n'+1},\ p \bmod 4 = 1 \ \vee\ p \bmod 4 = 2\}.$$

The following can be shown by inspection of the gray codes.

- The two halves form a ring configuration.
- All required data during the restriction and prolongation operations is located in neighbouring processors. This means that no data transports are necessary in the memory coupled DIRMU system.
- This process can be applied recursively.

4.2. Speedup results. The speedups and efficiencies have been determined for 2, 4, 8, and 16 processors (this is due to the hypercube configuration used) and various grid sizes. Table 5 shows the results for the V-cycle, the mixed cycle described above, and for the W-cycle. Index l indicates the grid fineness. The speedups obtained for the mixed cycle are very good even for coarse grids. However, it should be noted that numerical tests have shown that the W-cycle is necessary to achieve a convergence rate independent of h_l.

TABLE 5
Speedups for different cycle forms.

cycle	l	2 processors		4 processors		8 processors		16 processors	
		$S(2)$	$E(2)$	$S(4)$	$E(4)$	$S(8)$	$E(8)$	$S(16)$	$E(16)$
V	3	1.82	0.91	3.26	0.81	5.43	0.68		
	4	1.90	0.95	3.58	0.89	6.28	0.79	11.00	0.69
	5			3.59	0.90	6.80	0.85	12.48	0.78
	6							13.95	0.87
Mixed	3	1.93	0.97	3.59	0.90	5.82	0.73		
	4	1.98	0.99	3.80	0.95	7.04	0.88	11.89	0.74
	5			3.92	0.98	7.63	0.95	14.39	0.90
	6							15.47	0.97
W	3	1.86	0.93	3.21	0.80	5.24	0.66		
	4	1.88	0.94	3.50	0.88	5.78	0.72	9.47	0.59
	5			3.56	0.89	6.49	0.81	11.04	0.69
	6							12.29	0.77

5. Comparison. Speedup alone is not sufficient as an objective criterion for the comparison of iterative methods on a parallel computer. In addition, we have to consider the convergence rate and the operation count, as these factors also play a key role in the overall efficiency of a solution procedure. We therefore define the normalized parallel computation time T_{np} to be

$$T_{np} = \frac{T_{cycle}}{-\ln \rho \cdot S(n)}, \tag{26}$$

where T_{cycle} is the time needed for one iteration on a single computer, ρ is the average convergence rate and $S(n)$ is the speedup of the method on n processors. T_{np} is the time needed for n processors to reduce the error in the solution by a factor of $1/e$. Alternatively, the operation count for one iteration can be used instead of T_{cycle}, in order to provide a measure that is independent of differences in implementation between the methods under comparison.

Table 6 shows the convergence rates and values of T_{np} obtained on DIRMU for 8 and 16 processors for the frequency decomposition method, and for standard multigrid methods with ILU and Gauß–Seidel red-black smoothing applied to the anisotropic equation with $h = 1/64$ on the finest grid. Two pre-smoothing steps were used for the red-black and frequency decomposition methods, and one for the ILU method. No post-smoothing was done. In addition V-cycles were used for the red-black and ILU measurements, and W-cycles for the frequency decomposition tests. Full-weighting was used as restriction operator in all cases.

Table 6 shows that as soon as the two factors α and β differ by a factor of 10 or more, the two robust methods are superior to the standard parallel algorithm with red-black smoothing. In the case of strongly anisotropic coefficients, the ILU method is several hundred times as fast and the frequency decomposition method is about sixteen times as fast. Even for the Poisson equation ($\alpha = \beta = 1$) the ILU method is comparable to red-black smoothing.

The multigrid method with ILU smoother has been found to be superior to the frequency decomposition method also for other problems we tested. Table 7 shows values of T_{np} for ILU and red-black smoothing applied to a discretization of (3), where upwind differencing has been used for the convective terms and central differencing has been used for the diffusive terms. The flow direction was 45 degrees ($c_1 = c_2 = 1$).

TABLE 6
Normalized parallel computation times for the anisotropic equation, $h = 1/64$, $\beta = \alpha^{-1}$.

α	GS − rb			FDM			ILU		
	ρ	$T_{np}(8)$	$T_{np}(16)$	ρ	$T_{np}(8)$	$T_{np}(16)$	ρ	$T_{np}(8)$	$T_{np}(16)$
1	0.108	0.9	0.5	0.086	5.9	3.4	0.121	0.86	0.73
$\frac{1}{2}$	0.393	2.23	1.1	0.196	8.8	5.2	0.150	0.96	0.81
$\frac{1}{10}$	0.938	32.5	16.2	0.295	11.8	6.9	0.135	0.91	0.77
10^{-2}	0.977	89.4	44.7	0.050	4.8	2.8	$8 \cdot 10^{-4}$	0.26	0.22
10^{-5}	0.977	89.4	44.7	0.051	4.8	2.8	$4 \cdot 10^{-15}$	0.05	0.05

TABLE 7
Normalized parallel computation times for the convection-diffusion equation, $h = 1/64$.

ϵ	GS − rb		ILU	
	$T_{np}(8)$	$T_{np}(16)$	$T_{np}(8)$	$T_{np}(16)$
1	3.71	1.85	4.23	3.57
0.1	5.02	2.51	5.35	4.51
0.01	7.97	3.99	3.97	3.35
0.001	14.1	7.05	2.27	1.92

Again the ILU method is superior to red-black smoothing. The frequency decomposition method in the present form could not be applied to the convection-diffusion equation with strong convection because the stable upwind differences on the finest grid are gradually changed to unstable central differences on the coarser grids by the Galerkin products.

6. Concluding remarks. The results presented here show that the multigrid method with ILU smoother can be effectively parallelized and is greatly superior to the multigrid method with red-black smoother often used on parallel machines. The efficient solution properties of the method in the case of vanishing ellipticity are provided by the ILU relaxation, which in this case would also be appropriate as a preconditioner for a conjugate gradient scheme, which is known to be highly parallelizable (see, for example, [1]). In the elliptic case, however, multigrid is known to be an optimal order method, having a rate of convergence independent of the grid size. It has to be emphasized that the ILU algorithm was not changed to achieve the parallelization. The method is especially suited for the anisotropic equation and the convection-diffusion equation with strong convection, which are important problems in practice.

The second method considered also showed superiority over the standard approach for the anisotropic equation but is applicable only to a smaller range of problems than the ILU method.

The parallelization of the ILU method can be easily extended to the three-dimensional case, but it should be noted that the method is not robust for the three-dimensional anisotropic equation. Whether the frequency decomposition method remains robust in the sense of (7), (which requires a bounded rate of convergence also for $h \to 0$), for three-dimensional problems is not known yet. Further work will include the incorporation of the ILU method into a parallel solver for the incompressible Navier–Stokes equations.

REFERENCES

[1] O. AXELSSON AND V. EIJKHOUT, *Robust vectorizable preconditioners for three-dimensional elliptic difference equations with anisotropy*, in Algorithms and Applications on Vector and Parallel Computers, North–Holland, Amsterdam, 1987, pp. 279–306.

[2] P. BASTIAN, *Die Frequenzzerlegungsmethode als robustes Mehrgitterverfahren: Implementierung und Parallelisierung*, Diplomarbeit, IMMD III, Universität Erlangen-Nürnberg, FRG, 1989.

[3] C. BECKER, J. H. FERZIGER, M. PERIC, AND G. SCHEUERER, *Finite volume multigrid solution of the two-dimensional incompressible Navier-Stokes equations*, in Robust Multigrid Methods, Proceedings of the Fourth GAMM Seminar, Notes on Numerical Fluid Mechanics, Vol. 23, Vieweg Verlag, Braunschweig, FRG, 1988.

[4] W. HACKBUSCH, *Multigrid Methods and Applications*, Springer-Verlag, Berlin, Heidelberg, 1985.

[5] ———, *The frequency decomposition multigrid method*, in Robust Multigrid Methods, Proceedings of the Fourth GAMM Seminar, Notes on Numerical Fluid Mechanics, Vol. 23, Vieweg Verlag, Braunschweig, FRG, 1988.

[6] W. HÄNDLER, E. MAEHLE, AND K. WIRL, *The DIRMU testbed for high-performance multiprocessor configurations*, Proceedings of the First International Conference on Supercomputing Systems, St. Petersburg, FL, 1985.

[7] G. HORTON, *Parallelisierung eines Mehrgitterverfahrens mit ILU Glättung*, Diplomarbeit, IMMD III, Universität Erlangen-Nürnberg, FRG, 1989.

[8] R. KNIRSCH, *Parallele Lösung eines nichtlinearen Gleichungssystemes mit dem SUPRENUM Simulator*, Studienarbeit, IMMD III, Universität Erlangen-Nürnberg, FRG, 1989.

[9] J. A. MEIJERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of Computation, 31 (1977), pp. 148-162.

[10] H. L. STONE, *Iterative solution of implicit approximations of multidimensional partial differential equations*, SIAM J. Numer. Anal., 5 (1968), pp. 530–558.

[11] P. WESSELING, *Theoretical and practical aspects of a multigrid method*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 387–407.

[12] G. WITTUM, *On the robustness of ILU smoothing*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 699–717.

[13] ———, *Multigrid methods for Stokes– and Navier–Stokes equations*, Numer. Math., 54 (1989).

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1471–1479, November 1991
© 1991 Society for Industrial and Applied Mathematics
014

TIMELY COMMUNICATIONS

Under the "timely communications" policy for the SIAM Journal on Scientific and Statistical Computing, papers that have significant timely content and do not exceed five pages automatically will be considered for a separate section of the journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

EIGENDECOMPOSITION OF DOMAIN DECOMPOSITION INTERFACE OPERATORS FOR CONSTANT COEFFICIENT ELLIPTIC PROBLEMS*

TONY F. CHAN† AND THOMAS Y. HOU‡

Abstract. In this paper the authors derive the exact eigendecomposition of the interface operators arising in domain decomposition methods for general five-point discretizations to constant coefficient elliptic equations on rectangular domains. The special case of convection-diffusion problems is studied in some detail, including both central and upwind differencing for the convection term and flows normal and tangential to the interface. It is shown that preconditioners based on the diffusion operator alone may give very slow convergence when the cell Reynolds number is of order $O(1)$.

Key words. domain decomposition, interface operator, eigendecomposition, convection-diffusion

AMS(MOS) subject classifications. 65N20, 65F10

1. Introduction.

In nonoverlapping domain decomposition methods, the key idea is to reduce the differential operator on the whole domain to an operator governing the solution on a reduced set of variables on the interfaces between the subdomains and perhaps a set of cross-points on which a coarse grid discretization is available [2], [13]. The understanding of the properties of the reduced operator is fundamental to domain decomposition methods.

In this paper, we follow the approach in [4], [5] and derive the exact eigendecomposition of the interface operators arising in domain decomposition methods for general five-point discretizations to constant coefficient elliptic equations on rectangular domains. We study in some detail the special case of convection-diffusion problems, including both central and upwind differencing for the convection term and flows normal and tangential to the interface.

One of our motivations is to study the effect of the size of the convection term on the performance of preconditioners based on the diffusion operator only [1], [2], [4], [5], [9], [11]. In particular, we show that convergence can be slow when the cell Reynolds number is of order $O(1)$.

Another possible use of the eigendecompositions is as direct preconditioners for variable coefficient problems (e.g., by averaging coefficients) on more general domains (e.g., by approximating irregular domains by domains with regular geometries sharing the same interface [6]). Although we do not have theoretical justifications for such

* Received by the editors June 24, 1988; accepted for publication (in revised form) May 15, 1991.

† Department of Mathematics, University of California, Los Angeles, Califoria 90024. The work of this author was supported in part by the Department of Energy under contract DE-FG03-87ER25037, by the Army Research Office under contract DAAL03-88-K-0085, and by the National Science Foundation under contracts NSF-DMS87-14612 and ASC 9003002.

‡ Courant Institute, 251 Mercer Street, New York, New York 10012. The work of this author was supported in part by the Air Force Office of Scientific Research under University Research Initiative grant AFOSR 86-0352, the National Science Foundation under contract DMS-9003202, and a Sloan fellowship.

a procedure, the motivation is that by taking into account the convection term in the preconditioners, the rate of convergence will be less sensitive to the cell Reynolds number. For some numerical experiments along this line, see [8]. We note that some of the known preconditioners in the literature, in particular the Neumann–Dirichlet preconditioner [1] and Schwarz-type methods [3], do implicitly account for the first-order terms.

2. Formulation. Since we are primarily interested in deriving interface preconditioners, we need only consider the simple case of a domain split into two subdomains with one interface. For example consider the following problem: $Lu = f$ on Ω with boundary conditions $u = u_b$ on $\partial\Omega$, where L is a linear elliptic operator and the domain Ω is decomposed into two subdomains Ω_1 and Ω_2 by an interface Γ. If we order the unknowns for the internal points of the subdomains first and those in the interface Γ last, then the discrete solution vector $u = (u_1, u_2, u_3)$ satisfies the linear system $Au = b$, which can be expressed in block form as

$$\begin{pmatrix} A_{11} & & A_{13} \\ & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

By block-Gaussian elimination, the reduced operator on Γ (i.e., the Schur complement of A_{33} in A) is given by

$$C = A_{33} - A_{31}A_{11}^{-1}A_{13} - A_{32}A_{22}^{-1}A_{23}. \tag{2.1}$$

3. General five-point stencil. We consider the general second-order elliptic equations with constant coefficients

$$Lu = \alpha\frac{\partial^2 u}{\partial x^2} + 2\beta\frac{\partial^2 u}{\partial x \partial y} + \gamma\frac{\partial^2 u}{\partial y^2} + \delta\frac{\partial u}{\partial x} + \eta\frac{\partial u}{\partial y} + \rho u = g \quad \text{on} \quad \Omega \tag{3.1}$$

with boundary condition $u = 0$ on $\partial\Omega$, where Ω is a two-dimensional rectangle.

Without loss of generality, we may assume that $\beta = 0$, $\alpha > 0$, and $\gamma > 0$.

We use a uniform mesh with grid size h on Ω and with n internal grid points in the x-direction, i.e., $h = 1/(n+1)$. We assume that the interface Γ is parallel to the x-axis and that Ω_1 has m_1 internal grid points and Ω_2 has m_2 internal grid points in the y-direction, i.e., the heights of Ω_1 and Ω_2 are given by $l_1 = (m_1 + 1)h$ and $l_2 = (m_2 + 1)h$. Let $x_i = ih$, $y_k = kh$, and denote the approximation to $u(x_i, y_k)$ by $u_{i,k}$. Suppose that equation (3.1) is approximated by a discrete finite-difference scheme of the following form:

$$au_{i-1,k} + bu_{i,k} + cu_{i+1,k} + du_{i,k+1} + eu_{i,k-1} = g_{i,k} \tag{3.2}$$

with boundary conditions given by $u_{0,k} = u_{i,0} = u_{n+1,k} = u_{i,n+1} = 0$, where we assume that the coefficients a, c, d, and e are nonzero. The idea now is to generalize the result in [3], [5], [7] to the more general five-point stencil formula (3.2), by deriving an exact eigendecomposition of the corresponding capacitance matrix C. This result is summarized in the following theorem.

THEOREM 3.1. *Define $\tilde{W} = DW$, where D is defined by*

$$D = diag\left[1, \sqrt{\frac{a}{c}}, \cdots, \sqrt{\frac{a}{c}}^{n-1}\right]$$

and W is an orthogonal matrix whose columns are given by

$$w_j = \sqrt{2h}(\sin j\pi h, \sin 2j\pi h, \cdots, \sin nj\pi h)^T.$$

Then we have $\tilde{W}^{-1}C\tilde{W} = diag(\lambda_1, \cdots, \lambda_n)$, where the eigenvalues λ_j are given by

$$\lambda_j = -\left(\frac{1+\gamma_j^{m_1+1}}{1-\gamma_j^{m_1+1}} + \frac{1+\gamma_j^{m_2+1}}{1-\gamma_j^{m_2+1}}\right)\sqrt{\frac{1}{4}[b+\sqrt{ac}(2-\sigma_j)]^2 - de}, \tag{3.3}$$

with $\sigma_j \equiv 4\sin^2\left(\frac{j\pi h}{2}\right)$, and

$$\gamma_j = \frac{\left((b+\sqrt{ac}(2-\sigma_j)) + \sqrt{(b+\sqrt{ac}(2-\sigma_j))^2 - 4de}\right)^2}{4de}.$$

Proof. First we will show that $C\tilde{w}_j = \lambda_j\tilde{w}_j$, where $\tilde{w}_j$ is the jth column of matrix $\tilde{W}$.

As pointed out in [4], [5], the term $-A_{31}A_{11}^{-1}A_{13}\tilde{w}_j$ can be computed by first solving the discrete equation (3.2) on Ω_1 with homogeneous right-hand side and the boundary condition $u = \tilde{w}_j$ on Γ, and homogeneous boundary conditions elsewhere, and then taking the solution on the first row of grid point above Γ multiplied by d. A similar observation applies to the term $-A_{32}A_{11}^{-1}A_{23}\tilde{w}_j$.

Let us now consider the term $-A_{31}A_{11}^{-1}A_{13}\tilde{w}_j$ first. As we mentioned above, this requires the solution of the discrete equation

$$au_{i-1,k} + bu_{i,k} + cu_{i+1,k} + du_{i,k+1} + eu_{i,k-1} = 0 \quad \text{on} \quad \Omega_1 \tag{3.4}$$

with boundary conditions

$$u_{i,k} = \tilde{w}_j \quad \text{on} \quad \Gamma \quad \text{and} \quad u_{i,k} = 0 \quad \text{on} \quad \partial\Omega_1/\Gamma. \tag{3.5}$$

Consider the solution vector of (3.4) of the form

$$u_{i,k} = d_k\sqrt{2h}(\sqrt{a/c})^i \sin j\pi ih, \tag{3.6}$$

where $0 \le i \le n+1$ and $0 \le k \le m_1+1$.

Substituting (3.6) into (3.4), we have

$$(dd_{k+1} + ed_{k-1})\sqrt{2h}\left(\sqrt{\frac{a}{c}}\right)^i \sin j\pi ih \tag{3.7}$$

$$+d_k\sqrt{2h}\left(\sqrt{\frac{a}{c}}\right)^i\left[b\sin j\pi ih + a\left(\sqrt{\frac{a}{c}}\right)^{-1}\sin j\pi(i-1)h + c\sqrt{\frac{a}{c}}\sin j\pi(i+1)h\right] = 0$$

with boundary conditions

$$d_0 = 1 \quad \text{and} \quad d_{m_1+1} = 0. \tag{3.8}$$

It follows that the d_k satisfy the following difference equation

$$dd_{k+1} + [b+\sqrt{ac}(2-\sigma_j)]d_k + ed_{k-1} = 0 \tag{3.9}$$

with boundary conditions (3.8) and where $\sigma_j \equiv 4\sin^2\left(\frac{j\pi h}{2}\right)$. The roots of the corresponding characteristic polynomial for (3.9) are

$$r_{\pm} = \frac{-(b+\sqrt{ac}(2-\sigma_j)) \pm \sqrt{(b+\sqrt{ac}(2-\sigma_j))^2 - 4de}}{2d}.$$

The general solution of (3.7) is then given by

$$d_k = c_1 r_+^k + c_2 r_-^k. \tag{3.10}$$

Boundary condition (3.8) implies $c_1 + c_2 = 1$ and $c_1 r_+^{m_1+1} + c_2 r_-^{m_1+1} = 0$, which gives

$$c_1 = -\frac{r_-^{m_1+1}}{r_+^{m_1+1} - r_-^{m_1+1}}, \qquad c_2 = \frac{r_+^{m_1+1}}{r_+^{m_1+1} - r_-^{m_1+1}}.$$

Therefore we have

$$-A_{31}\, A_{11}^{-1}\, A_{13}\, \tilde{w}_j = dd_1 \tilde{w}_j, \tag{3.11}$$

where

$$d_1 = \left(\frac{r_- - r_+ \gamma_j^{m_1+1}}{1 - \gamma_j^{m_1+1}}\right), \qquad \gamma_j \equiv \frac{r_-}{r_+}. \tag{3.12}$$

Similarly,

$$-A_{32}\, A_{22}^{-1}\, A_{23}\, \tilde{w}_j = ed_1^* \tilde{w}_j, \tag{3.13}$$

where

$$d_1^* = \frac{r_-^* - r_+^* (\gamma_j^*)^{m_2+1}}{1 - (\gamma_j^*)^{m_2+1}}, \qquad \gamma_j^* = \frac{r_-^*}{r_+^*},$$

$$r_{\pm}^* = \frac{-[b+\sqrt{ac}(2-\sigma_j)] \pm \sqrt{(b+\sqrt{ac}(2-\sigma_j))^2 - 4de}}{2e}.$$

One can easily see that $\gamma_j^* = \gamma_j$.

Finally, it can be verified directly:

$$A_{33}\tilde{w}_j = [b+\sqrt{ac}(2-\sigma_j)]\tilde{w}_j. \tag{3.14}$$

Thus (3.11), (3.13), and (3.14) together give $C\tilde{w}_j = \lambda_j \tilde{w}_j$, where

$$\lambda_j = [b+\sqrt{ac}(2-\sigma_j)] + dd_1 + ed_1^*, \tag{3.15}$$

which after some simplifications gives (3.3). □

4. Convection-diffusion equations. We now summarize results for a few special cases corresponding to various finite-difference discretizations of convection-diffusion equation. We use the standard five-point central and upwind differencing for the convection term. We also consider flows normal and tangential to the interface.

4.1. Central differencing for $\Delta u + \eta(\partial u/\partial y) = 0$. This is the normal flow case. The coefficients of the stencil are given by

$$a = 1, \quad b = -4, \quad c = 1, \quad d = 1 + \frac{\eta h}{2}, \quad e = 1 - \frac{\eta h}{2}.$$

The eigenvalues λ_j depend on the value of ηh. They have the following form:

$$\lambda_j = -\left[\frac{(1+\gamma_j^{m_1+1})}{(1-\gamma_j^{m_1+1})} + \frac{(1+\gamma_j^{m_2+1})}{(1-\gamma_j^{m_2+1})}\right]\sqrt{\sigma_j + \frac{{\sigma_j}^2}{4} + \left(\frac{\eta h}{2}\right)^2},$$

where

$$\gamma_j = \frac{(1 + \frac{\sigma_j}{2} - \sqrt{\sigma_j + \sigma_j^2/4 + (\eta h/2)^2}\quad)^2}{1 - (\eta h/2)^2}.$$

It is interesting to note that although $\gamma_j \to \infty$ as $|\frac{\eta h}{2}| \to 1$, λ_j remain finite in the limit.

4.2. Central differencing for $\Delta u + \delta(\partial u/\partial x) = 0$. This is the tangential flow case. The coefficients of the stencil are given by

$$a = 1 - \frac{\delta h}{2}, \quad b = -4, \quad c = 1 + \frac{\delta h}{2}, \quad d = 1, \quad e = 1.$$

The eigenvalues λ_j depend on the value of δh. They have the following form:

$$\lambda_j = -\left[\frac{(1+\gamma_j^{m_1+1})}{(1-\gamma_j^{m_1+1})} + \frac{(1+\gamma_j^{m_2+1})}{(1-\gamma_j^{m_2+1})}\right]\sqrt{(2-\mu)\mu\sigma_j + \mu^2\frac{{\sigma_j}^2}{4} + (2-\mu)^2 - 1},$$

where

$$\mu = \left(1 + \frac{\delta h}{2}\right)\sqrt{\frac{1 - \frac{\delta h}{2}}{1 + \frac{\delta h}{2}}},$$

and

$$\gamma_j = \frac{\left(\mu(2-\sigma_j) - 4 + \sqrt{(\mu(2-\sigma_j)-4)^2 - 4}\right)^2}{4}.$$

We note that in this case, the Fourier transform matrix $\tilde{W}$ has a singularity as the cell Reynolds number $\delta h/2$ approaches 1 because the matrix D in the Theorem 3.1 becomes singular. This difficulty can be removed by using upwind differencing.

4.3. Upwind differencing for $\Delta u + \eta(\partial u/\partial y) = 0$, $\eta > 0$. The stencil coefficients are as follows

$$a = 1, \quad b = -4 - \eta h, \quad c = 1, \quad d = 1 + \eta h, \quad e = 1.$$

The eigenvalues λ_j are:

$$\lambda_j = -\left[\frac{(1+\gamma_j^{m_1+1})}{(1-\gamma_j^{m_1+1})} + \frac{(1+\gamma_j^{m_2+1})}{(1-\gamma_j^{m_2+1})}\right]\sqrt{\sigma_j + \frac{\sigma_j^2}{4} + \sigma_j\eta h/2 + (\eta h)^2/4},$$

where γ_j are given by

$$\gamma_j = \frac{\left[1 + \frac{\sigma_j}{2} + \eta h/2 - \sqrt{\sigma_j + \frac{\sigma_j{}^2}{4} + \eta h(\sigma_j/2 + \eta h/4)}\ \right]^2}{1 + \eta h}.$$

An important feature about the upwind discretization is that the eigenvalues λ_j are well defined for all values of the cell Reynolds number ηh ($\eta > 0$). This property is not shared by the central differencing discretization.

4.4. Upwind differencing for $\Delta u + \delta(\partial u/\partial x) = 0, \delta > 0$. The stencil coefficients are

$$a = 1, \quad b = -4 - \delta h, \quad c = 1 + \delta h, \quad d = 1, \quad e = 1.$$

The eigenvalues λ_j are:

$$\lambda_j = -\left[\frac{(1+\gamma_j^{m_1+1})}{(1-\gamma_j^{m_1+1})} + \frac{(1+\gamma_j^{m_2+1})}{(1-\gamma_j^{m_2+1})}\right]\sqrt{\sigma_j + \frac{\sigma_j^2}{4} - \epsilon_j(2 + \sigma_j - \epsilon_j)},$$

where the γ_j and ϵ_j are given by

$$\gamma_j = \left[-1 - \frac{\sigma_j}{2} + \epsilon_j + \sqrt{\sigma_j + \frac{\sigma_j{}^2}{4} - \epsilon_j(2 + \sigma_j - \epsilon_j)}\ \right]^2,$$

and

$$\epsilon_j = (1 - \frac{\sigma_j}{2})(\sqrt{1 + \delta h} - 1) - \frac{\delta h}{2}.$$

Note that ϵ_j tends to zero as δh goes to zero and the eigenvalues are well defined for all values of the cell Reynolds number δh ($\delta > 0$).

5. Effect of convection term on diffusion-based preconditioners. Consider the equation $\Delta u + \eta(\partial u/\partial y) = 0$ discretized using both central and upwind differencing. Let $C(\eta)$ denote the interface matrix as a function of η. Consider preconditioning $C(\eta)$ by $C(0)$, the interface operator corresponding to the diffusion operator only. In Fig. 1, we plot the condition numbers $K(C^{-1}(0)C(\eta))$ (the ratio of the maximum eigenvalue to the minimum eigenvalue) versus the coefficient η for $h = 0.02$, $m_1 = 50$, $m_2 = 100$. Note that the critical cell Reynolds number corresponds to $\delta = 100$ for the central differencing case, and the eigenvalues of $C^{-1}(0)C(\eta)$ are real in this case. We see that as δ approaches the critical value, the condition number can grow larger than 10.

Next we consider the equation $\Delta u + \delta(\partial u/\partial x) = 0$ with the same values for h, m_1, and m_2. In this case, the eigenvalues of $C^{-1}(0)C(\delta)$ are complex. Instead of a plot of the condition number, we show in Figs. 2 and 3 the distribution of the eigenvalues of $C^{-1}(0)C(\delta)$ for several values of δ for upwind and central differencing, respectively. These plots show how rapidly the spectrum spreads from unity as δ increases. The clustering around the value 1 can be easily seen, but even this effect weakens as δ increases. For many nonsymmetric iterative methods, such as conjugate-gradient–like methods and Chebychev methods, such information on the spectrum plays a crucial role in determining the convergence rates [10], [12]. We note that for $\delta > 65$, the formula for the eigendecomposition of the interface operator for the

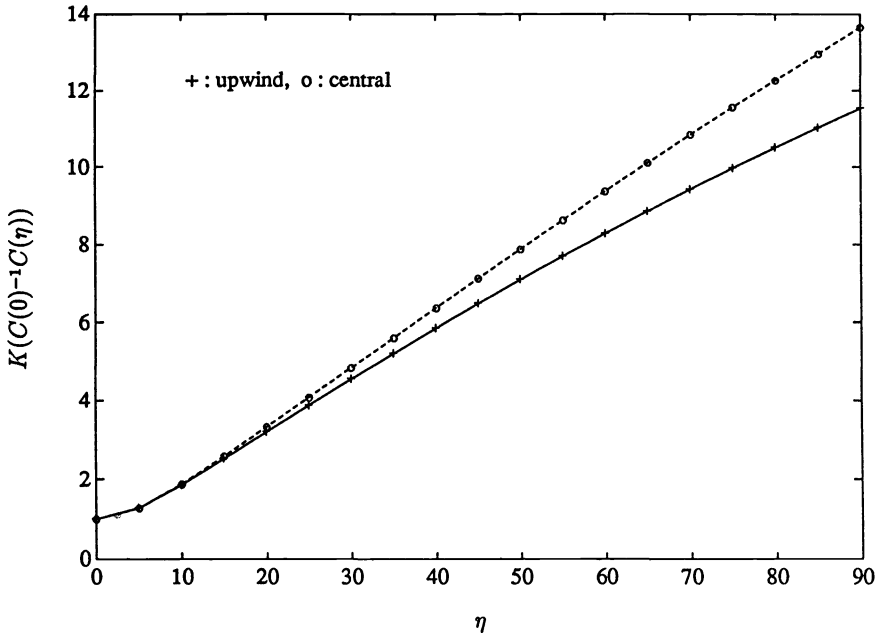


FIG. 1. $K(C(0)^{-1}c(\eta))$ *versus* η *for* $\triangle u + \eta(\partial u/\partial y) = 0$.

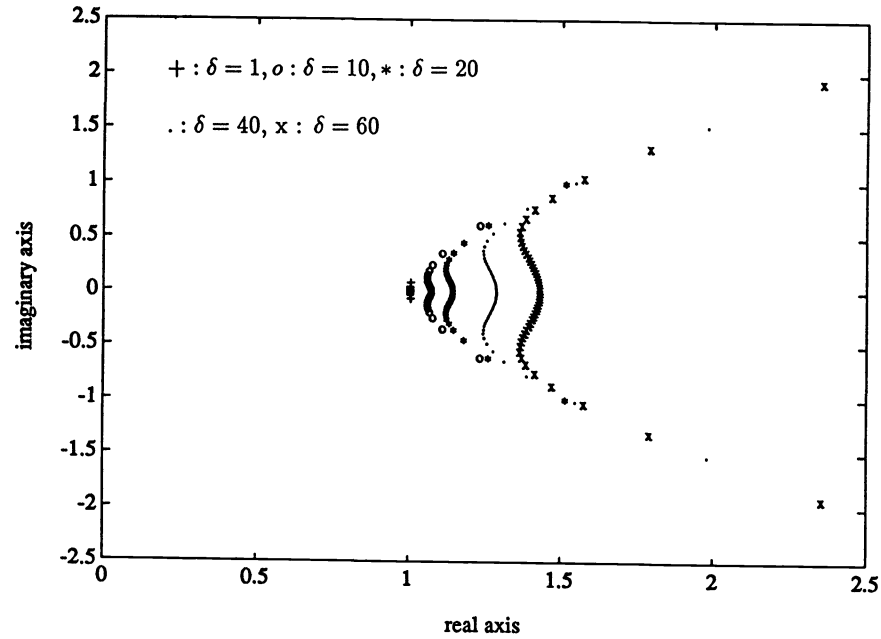


FIG. 2. *Eigenvalue distributions for upwind difference for* $\triangle u + \delta(\partial u/\partial x) = 0$.

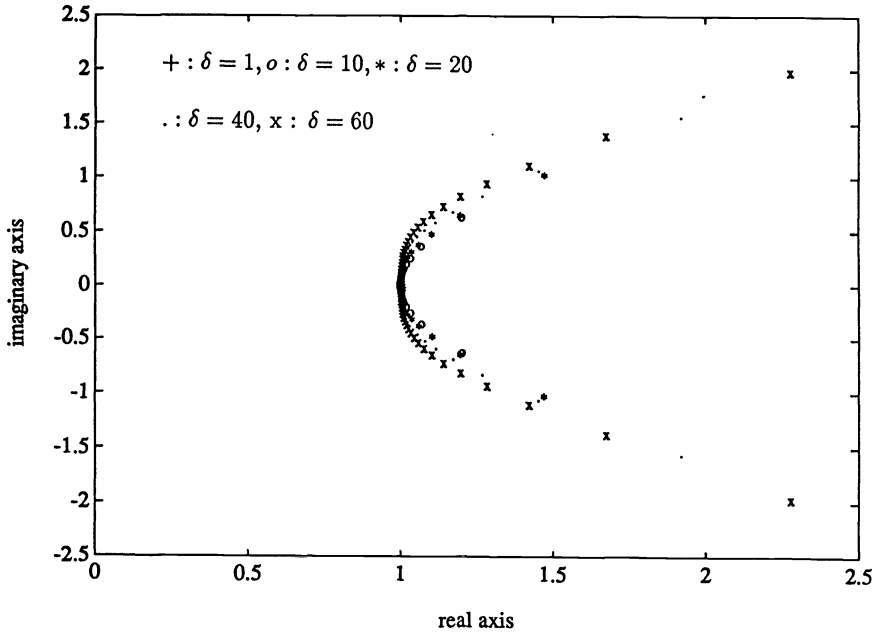


FIG. 3. *Eigenvalue distributions for central difference for* $\triangle u + \delta(\partial u/\partial x) = 0$.

central differencing case becomes unstable and hence the eigenvalues are not plotted in Figs. 2 and 3.

These calculations show that the effectiveness of the diffusion-based preconditioners (as reflected in either the condition number or the spread of the spectrum of the preconditioned interface system) can deteriorate appreciably as the size of the cell Reynolds number increases to $O(1)$.

Acknowledgment. The authors would like to thank Professor David Keyes for many helpful suggestions and for catching some errors in an earlier version of the paper.

REFERENCES

[1] P. E. BJØRSTAD AND O. B. WIDLUND, *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23 (1986), pp. 1097–1120.

[2] J. H. BRAMBLE, J. E. PASCIAK, AND A. H. SCHATZ, *The construction of preconditioners for elliptic problems by substructuring*, I, Math. Comp., 47 (1986), pp. 103–134.

[3] X. C. CAI, *Some domain decomposition algorithms for nonself-adjoint elliptic and parabolic partial differential equations*, Ph.D. thesis, Courant Institute, New York University, New York, NY, 1989.

[4] T. F. CHAN, *Analysis of preconditioners for domain decomposition*, SIAM J. Numer. Anal., 24 (1987), pp. 382–390.

[5] T. F. CHAN AND D. C. RESASCO, *A framework for the analysis and construction of domain decomposition preconditioners*, in Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations, 1987, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 217–230.

[6] ———, *Analysis of domain decomposition preconditioners on irregular regions*, in Proceedings of the Sixth IMACS International Symposium on Computer Methods for Partial Differential

Equations, R. Vichnevetsky and R. S. Stepleman, eds., June 23–26, 1987, Lehigh University, Bethelem, PA, pp. 317–322.

[7] ———, *A domain-decomposed fast Poisson solver on a rectangle*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s14–s26.

[8] T. F. CHAN AND D. E. KEYES, *Interface preconditionings for domain-decomposed convection-diffusion operators*, in Domain Decomposition Methods for Partial Differential Equations, T. F. Chan, R. Glowinski, J. Périaux, and O. B. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 245–262.

[9] M. DRYJA, *A capacitance matrix method for Dirichlet problem on polygonal region*, Numer. Math., 39 (1982), pp. 51–64.

[10] R. W. FREUND AND N. M. NACHTIGAL, QMR: *a quasi-minimal residual method for non-Hermitian linear systems*, Tech. Report 90.51, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, December, 1990.

[11] G. H. GOLUB AND D. MAYERS, *The use of pre-conditioning over irregular regions*, in Computing Methods in Applied Sciences and Engineering VI, R. Glowinski and J. L. Lions, eds., North–Holland, 1984, pp. 3–14.

[12] Y. SAAD AND M. H. SCHULTZ, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[13] B. F. SMITH, *An optimal domain decomposition preconditioner for the finite element solution of linear elasticity problems*, Tech. Report 482, Courant Institute, 1989; to appear in Proceedings of Copper Mountain Conference on Iterative Methods, April 1990.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1480–1485, November 1991
©1991 Society for Industrial and Applied Mathematics
015

SOLUTION OF LINEAR SYSTEMS OF ORDINARY DIFFERENTIAL EQUATIONS ON AN INTEL HYPERCUBE*

L. LUSTMAN†, B. NETA†, AND C. P. KATTI‡

Abstract. In this paper there is developed and tested a parallel scheme for the solution of linear systems of ordinary initial value problems based on the box scheme and a modified recursive doubling technique. The box scheme may be replaced by any stable integrator. The algorithm can be modified to solve boundary value problems. Software for both problems is available upon request.

Key words. initial value problems, parallel processing, hypercube, box scheme, recursive doubling

AMS(MOS) subject classifications. 65L, 65W

1. Introduction.

We consider the solution of linear problems on a hypercube. By a hypercube we intend "a distributed memory MIMD computer with communication between processors ... via a network having the topology of a p-dimensional cube, with the vertices considered as processors and the edges as communication links" [4]. See also Fox [1], [2] and Fox and Otto [3]. Our method of solution is based on the box scheme to discretize the system of initial value problems:

$$y' = Ay + f(x),$$
$$y(a) = y_0,$$

where y and f are n-dimensional vectors and A is an $n \times n$ matrix. We obtain, in parallel, fundamental solutions on subintervals. The resulting system of equations is solved by a modified version of the recursive doubling technique (see Stone, [7]). Another technique that parallelizes the solution by subinterval decomposition has been proposed by Skeel [6].

In the next section, the general problem is stated and some information on the INTEL Hypercube is given. The algorithm for initial value problems is described in §3, and the efficiency of the algorithm is discussed in §4, where we detail the numerical experiments performed with our algorithm. In the last section we present our conclusions.

2. The general problem.

The numerical solution of ordinary differential systems is an intrinsically sequential procedure: given the data at a point x (or at several points $x, x-h, \cdots, x-Kh$), one advances to the following point $x+h$. In order to parallelize this procedure, we make the basic remark that for a linear system $y' = A(x)y$ on an interval $[a, b]$, the solution at the right endpoint is a linear function of the values at the left endpoint:

$$y(b) = Y_{[a,b]}y(a).$$

*Received by the editors September 24, 1990; accepted for publication (in revised form) July 2, 1991. This research was conducted for the Office of Naval Research and was funded by the Naval Postgraduate School. Preliminary results were obtained by the last two authors, whose research was partially supported by the National Science Foundation, through grants INT-8519159 and INT-8613396.

†Naval Postgraduate School, Department of Mathematics, Monterey, California 93943.

‡Jawaharlal Nehru University, School of Computer and Systems Sciences, New Delhi 110067, India.

Here $Y_{[a,b]}$ is the value at $x = b$ of Y, the fundamental solution on the interval, which is defined by:

$$Y' = A(x)Y, \qquad Y(a) = I, \quad \text{the identity matrix.}$$

To solve a problem on the interval $[x_{\min}, x_{\max}]$, we propose to assign several contiguous subintervals:

$$[x_0, x_1], [x_1, x_2], \cdots, [x_{N-1}, x_N],$$

with $x_0 = x_{\min}$, $x_N = x_{\max}$ to the N processors, and let each of them compute in parallel the corresponding fundamental solution. This is a task that may require a large number of sequential steps, for the numerical evaluation of $Y_{[x_i, x_{i+1}]}$. After these quantities are ready, one may obtain $y(x_i)$ from the initial data $y(x_{\min})$ by matrix multiplication, as obviously

$$Y_{[a,b]} Y_{[b,c]} = Y_{[a,c]}.$$

Let us remark at the outset that this elementary procedure may be extended to inhomogeneous equations, with the following initial data:

$$y' = A(x)y + f(x),$$
$$y(x_{\min}) = \text{given},$$

or two-point boundary data:

$$y' = A(x)y + f(x), \qquad x_{\min} \le x \le x_{\max},$$
$$B_1\ y(x_{\min}) + B_2\ y(x_{\max}) = \text{given}.$$

Such extensions necessitate only the linearity of the equations and initial or boundary conditions. We shall discuss these general algorithms, as well as the steps necessary to obtain computational efficiency.

In order to address efficiency matters, we must also briefly present the machine on which the algorithm is run. The iPSC/2 Intel Hypercube is a MIMD (multiple instruction–multiple data) machine, consisting of several processors in hypercube connection. Each such processor—also called a node—executes its own program, on data in its own memory. The nodes are controlled by another processor—the host—which loads the programs into the nodes and starts them running. Host and nodes communicate by message passing; these messages are strings of arbitrary length, with an arbitrary "message type" (an integer), which may be sent from any processor to any other processor. At any moment a processor may send a message, find whether messages of a certain type are pending, or receive messages. The communication may be performed synchronously, i.e., the processing stops until a message is sent or received, or asynchronously, where processing and communication overlap.

It is seen, therefore, that an algorithm is optimal on such a machine if it may be set as several parallel processes, each working on its own data, with a minimum of interprocess communication. We shall see that our ordinary differential equation solvers fit very well the Intel architecture.

3. The algorithm for the initial value problem.

Step 1. Using N processors to solve the linear inhomogeneous system with initial conditions:

$$y' = A(x)y + f(x),$$
$$y(x_{\min}) = g,$$

divide the required interval into N subintervals:

$$[x_0, x_1], [x_1, x_2], \cdots, [x_{N-1}, x_N], \quad \text{with } x_0 = x_{\min}, \quad x_N = x_{\max}.$$

The algorithm will produce numerical approximations for $y(x_j)$, $j = 1, \cdots, N$.

Step 2. Do in parallel:

Processor j, working on the interval $[x_{j-1}, x_j]$ solves numerically the following two systems:

$$Y_j' = A(x)Y_j,$$
$$Y_j(x_{j-1}) = I, \quad \text{the identity matrix,}$$

and

$$\phi_j' = A(x)\phi_j + f(x),$$
$$\phi_j(x_{j-1}) = 0.$$

In our program this is done using the box scheme (see, e.g., Neta and Katti, [5]). The matrix Y_j is the fundamental solution on the subinterval, whereas ϕ_j incorporates the inhomogeneous effect of the forcing function f. When this step is completed, one may recursively compute $y(x_j)$ from:

$$\begin{aligned} y(x_1) &= Y_1(x_1)g + \phi_1(x_1), \\ y(x_2) &= Y_2(x_2)y(x_1) + \phi_2(x_2), \\ &\vdots \\ y(x_N) &= Y_N(x_N)y(x_{N-1}) + \phi_N(x_N). \end{aligned}$$

The last step of the algorithm is an efficient performance of the recursion above, assuming that $N = 2^m$, as usual on a hypercube.

Step 3. (This is a modification of the recursive doubling due to Stone [7])

(3a) For $1 \le j \le N$, initialize:

$$y_j = \phi_j(x_j), \qquad M_j = Y_j(x_j).$$

Also initialize: $y_1 = g + M_1 y_1$, $k = 1$.

(3b) For all $j > k$ compute:

$$y_j^* = y_j + M_j y_{j-k}, \qquad M_j^* = M_j M_{j-k}.$$

(3c) For all $j > k$ replace M, y by M^*, y^*:

$$y_j = y_j^*, \qquad M_j = M_j^*.$$

(3d) Set $k = 2k$. If $k < N$, repeat steps (3b)–(3c) above. Otherwise the algorithm ends with y_j the numerical approximations to the solutions at x_j.

4. The efficiency of the algorithm. We begin our discussion with an investigation of the communication overhead.

In Step 3 there will be interprocessor communication, as processor j obtains data from processor $j - k$. It is obvious that the algorithm requires only one additional buffer per processor, to hold M^* and y^*—under the assumption that the matrix

multiplications are performed in the order shown. It is also possible to perform steps (3b)–(3c) in parallel, but then care must be exercised to avoid data corruption by message passing.

One option is to use just one buffer, and accept data only when ready. We shall call this the "send on request" scheme. The other option is to broadcast data as soon as it is ready. This we shall call the "multiple buffer" scheme. Yet another possibility is to use as temporary buffers the memory provided by the hypercube communication technology. For example, processor 1 sends data to processor 5, in a message with message type 1 (the identity of the sender). Processor 5, executing Step (3b) with $k = 1$, expects data from processor 4, so it will accept only a message with message type 4. The data from processor 1 are left in the communication buffers, to be read when processor 5 reaches the stage $k = 4$. This version, the "message type" scheme, is clearly the simplest to program.

We have implemented all three versions mentioned above. As expected, the "send on request" program has a higher communication overhead, because about twice as many messages are passed as in the other schemes. The multiple buffer scheme and the message type scheme essentially run at the same speed, although the messages arrive in a different order. The test problems show that the message type scheme is preferable, unless the data to be transferred are so bulky as to slow down communication. This certainly does not happen in this program, which transfers matrices of moderate size. Moreover, as the size of the problem (i.e., the dimension of the vector y) increases, more and more work will be done on actually solving the differential equations, and the communication overhead will be less significant.

An idea about the magnitude of the communication overhead may be obtained from the data in the following tables, which summarize several numerical experiments in solving the following system:

$$\begin{aligned} y'_i &= y_i + x y_{i+1} + f_i, \qquad 0 \le i < 10, \\ y'_{10} &= y_{10} + f_{10}, \end{aligned}$$

where f_i is adjusted so that the exact solution is:

$$y = (1, e^x, e^{-x}, e^{2x}, e^{-2x}, e^{3x}, e^{-3x}, x, \sin(x), \cos(x)).$$

The first table shows the total time spent by each processor in solving the problem, as obtained from the `mclock` system call.

Most of the work is done in computing the fundamental solutions, and communication is a relatively small quantity. Even the "send on request" scheme, which has a large number of messages transmitted, does not strongly influence the run times, which seem nearly constant on the various processors.

Another efficiency measure, critical for comparing single processor and multiprocessor versions of the same mathematical procedure, is the total running time needed for the complete solution.

We can roughly estimate this quantity as follows: let the unknown vector y be of dimension n, and assume that the numerical solution involves s steps (of size h) to reach $x_{\max}$ from $x_{\min}$. A single processor algorithm will need a time proportional to

$$sn,$$

as it evaluates n right-hand sides s times (we assume that most of the computational work is spent on obtaining the right-hand sides of the differential equations, and

TABLE 1
System of order 10, *"send on request." Total busy time in msec.*

Processor :	1	2	3	4	5	6	7	8
No. of steps per processor								
10	871	856	847	827	873	852	834	825
20	1641	1630	1621	1600	1637	1617	1608	1597
40	3183	3163	3162	3139	3188	3159	3153	3138
80	6265	6244	6244	6221	6270	6234	6240	6220

ignore matrix-vector or matrix-matrix multiplications). Our parallel algorithm, using N processors, will have a running time of:

$$\frac{s}{N}n^2,$$

because each processor executes only s/N steps; the quantity computed is the fundamental solution, an $n \times n$ matrix. Thus, it appears that there will be a gain only if $n < N$, i.e., the order of the differential system is less than the number of processors.

Even if there is no obvious gain in parallelization if all one needs is the solution of one differential problem, the algorithm proposed may become efficient when used as the first step of an inverse problem, or distributed parameter problem. In such a case, the same system is solved repeatedly with different initial conditions (say); then, after obtaining the quantities M_j, ϕ_j in the processors, one may use Step 3 of the algorithm to obtain sets of values y_j from sets of initial conditions.

The discussion and numerical experiment data of this section have been concerned only with initial value problems. It is also valid for the parallel solution of boundary value problems.

5. Conclusion. We have presented a parallel algorithm for solving ordinary initial value problems. We have shown that this algorithm is easy to program, and that machine-dependent optimization is readily achievable. Moreover, the algorithm is very flexible: as the equations are solved independently on each subinterval, it is possible to use different subinterval sizes, or different solution strategies in each subinterval, in order to control the error or balance the work among processors. The algorithm can be modified slightly to solve boundary value problems.

We have identified certain classes of practical mathematical procedures, for which our methods will be useful; these include various forms of inverse problems.

The basic limitation of our algorithm is that it applies only to linear problems. We are currently working on a method of parallelizing the solution of general, nonlinear ordinary differential systems.

Acknowledgments. The authors gratefully acknowledge comments and corrections by the editor and referees.

REFERENCES

[1] G. C. FOX, *Concurrent processing for scientific calculations*, Proc. COMPCON 1984, 1984, pp. 70–73.

[2] G. C. Fox, *Are concurrent processors general purpose computers?*, IEEE Trans. Nucl. Sci., NS-32 (1984), pp. 182–186.

[3] G. C. Fox and S. W. Otto, *Algorithm for concurrent processors*, Physics Today, 37 (1984), pp. 50–59.

[4] H. B. Keller and P. Nelson, *Hypercube implementations of parallel shooting*, Appl. Math. Comp., 31 (1986), pp. 574–603.

[5] B. Neta and C. P. Katti, *Solution of linear initial value problems on a hypercube*, Tech. Report NPS-53-89-001, Naval Postgraduate School, Monterey, CA, 1988.

[6] R. D. Skeel, *Waveform iteration and the shifted Picard splitting*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 756–776.

[7] H. S. Stone, *An efficient parallel algorithm for the solution of a tridiagonal linear system of equations*, J. Assoc. Comput. Mach., 20 (1973), pp. 27–38.

SIAM J. SCI. STAT. COMPUT.
Vol. 12, No. 6, pp. 1486–1495, November 1991
© 1991 Society for Industrial and Applied Mathematics
016

A DOMAIN DECOMPOSITION PRECONDITIONER BASED ON A CHANGE TO A MULTILEVEL NODAL BASIS*

CHARLES H. TONG†, TONY F. CHAN‡, AND C.C. JAY KUO§

Abstract. A domain decomposition method based on a simple change of basis on the interfaces and vertices is presented. It is shown that this leads to an effective preconditioner compared to the ones previously considered, such as the preconditioner by Bramble, Pasciak, and Schatz (BPS) [*Math. Comp.*, 47 (1986), pp. 103–134], and the hierarchical basis domain decomposition (HBDD) preconditioner by Smith and Widlund [*SIAM J. Sci. Statist. Comput.*, 11 (1990), pp. 1212–1226]. This domain-decomposed preconditioner is based on Bramble, Pasciak, and Xu's multilevel nodal basis preconditioner [*Math. Comp.*, to appear]. It is shown that analytically this method and the HBDD method give the same order of condition number, namely, $O(\log^2(H/h))$ for problems with smooth coefficients. Numerically this method appears to be more effective with little additional cost and for the model Poisson problem, the condition numbers appear to be $O(1)$.

Key words. domain decomposition, hierarchical basis, multilevel nodal basis, preconditioned conjugate gradient methods, Schur complement

AMS(MOS) subject classifications. 65F10, 65N30

1. Introduction.

We consider second-order, selfadjoint, uniformly elliptic partial differential equations on a two-dimensional polygonal domain Ω. The problems are solved numerically by using piecewise linear finite elements. The domain is first divided into nonoverlapping subregions Ω_i's, which are further divided into triangular finite elements. We use H to denote the diameter of a typical subregion and h to denote the diameter of its elements.

We begin with the linear system arising from a discretization of the problem and we first eliminate the variables interior to the subregions Ω_i. The resulting reduced system, the Schur complement, involves only the variables associated with Γ, the set of edges, and vertices of the subregions. This system is then solved by a preconditioned conjugate gradient method, where the preconditioner is constructed from certain problems associated with the interfaces $\Gamma_{ij} = \partial\Omega_i \cap \partial\Omega_j$ and vertices and a global coarse problem associated with the vertices.

Many preconditioners have been proposed for the subproblems associated with the edges Γ_{ij}. For example, the method by Bramble, Pasciak, and Schatz (BPS) [2] uses an operator similar to the square root of the Laplacian operator as the subproblem. Recently Smith and Widlund [8] proposed a computationally more efficient hybrid preconditioning method which involves only a simple change of basis (between nodal and hierarchical basis) with the unknowns on the edges Γ_{ij}. They show that the new method has a condition number which grows no faster than $C(1+\log(H/h))^2$, which is comparable to that of the BPS method.

* Received by the editors August 20, 1990; accepted for publication February 19, 1991. This work was supported in part by the Department of Energy under contract DE-FG-03-87-ER-25037, the Army Research Office under contract DAAL03-88-K-0085, Office of Naval Research under contract N00014-90-J-1695, and National Science Foundation under contract ASC 9003002.

† Sandia National Laboratories, Livermore, California 94551-0969 (chtong@snll-arpagw.llnl.gov).

‡ Department of Mathematics, University of California, Los Angeles, California 90024 (chan@math.ucla.edu).

§ Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, California 90089. This work was also supported by the University of Southern California Faculty Research and Innovation Fund.

The domain-decomposed preconditioner we consider in this paper is inspired by the work of Smith and Widlund [8]. In the same way that [8] uses the hierarchical basis on the edges to obtain a domain decomposition method, we use the multilevel nodal basis of Bramble, Pasciak, and Xu [3] applied to the reduced system on the interfaces (i.e., the edges *and the vertices*) to obtain our domain decomposition method. We call this preconditioner the multilevel nodal basis domain decomposition (MNBDD) preconditioner. We derive a proof, similar to the proof by Smith and Widlund [8], such that the condition number is bounded by $O(\log^2(H/h))$ for smooth coefficient problems. Numerical results are included to confirm the theoretical results. In particular, numerical experiments show that the condition number appears to be $O(1)$ for the model Poisson problem. We show that the MNBDD preconditioner also achieves convergence for variable and discontinuous problems in fewer iterations than the HBDD precondtioner, and with little extra cost.

2. The multilevel nodal basis algorithm and domain decomposition methods. In this section we provide the necessary background to define the MNBDD algorithm. We consider a second-order, selfadjoint, uniformly elliptic, bilinear form $a(u,v)$ on Ω with Dirichlet condition on $\partial\Omega$:

$$a(u,v) = (f,v) \quad \forall v \in H_0^1(\Omega), \quad u \in H_0^1(\Omega).$$

Let $V^H(\Omega)$ and $V^h(\Omega)$ be the spaces of continuous, piecewise linear functions, on the two triangulations, which vanish on $\partial\Omega$. We use elements which obey certain regularity assumptions, and obtain the following discrete variational problem:

$$a(u_h, v_h) = (f, v_h) \quad \forall v_h \in V^h(\Omega), \quad u_h \in V^h(\Omega).$$

By introducing the standard nodal basis $\{\phi_i\}$ for the space V^h, the above finite dimensional variational problem is reduced to a linear system:

$$Kx = b.$$

Here x is the vector of unknowns x_i, b is the vector of components (f, ϕ_i), and K is the stiffness matrix where $K_{ij} = a(\phi_i, \phi_j)$.

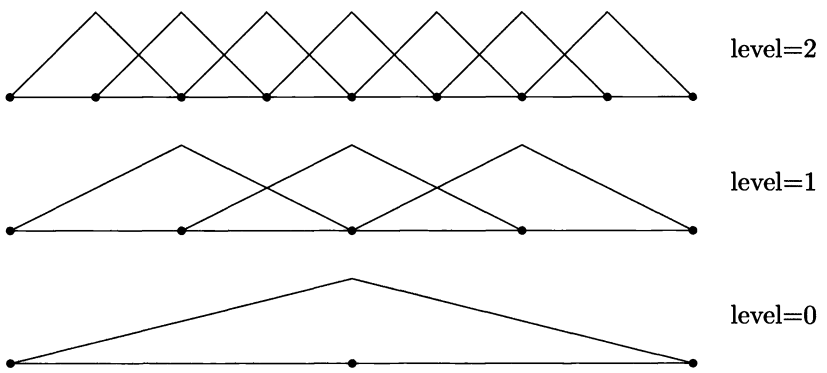


FIG. 1. *Multilevel nodal basis functions.*

2.1. The multilevel nodal basis algorithm. The multilevel nodal basis method [3] is given in terms of a set of nested sequence of finite element spaces,

$$V_0^h \subset V_1^h \subset \cdots \subset V_J^h \equiv V^h, \quad J = \log_2(H/h) \geq 1,$$

which are successive refinements by a factor of two of $V_0^h = V^H$. Here V_i^h is the set of piecewise linear finite element functions after i levels of refinement from the original coarse triangulation. In other words, V_i^h is the set of piecewise linear functions $\{\phi_i^j\}_{j=1}^{n_i}$ ($n_i = \dim V_i^h$) in $V^{2^{J-i}h}$ that satisfies

$$\phi_i^j(y_i^l) = \delta_{jl}, \quad \forall j, l = 1, \cdots, n_i,$$

where $\{y_i^l : l = 1, \cdots, n_i\}$ is the set of all interior nodal points of the triangulation on which V_i^h is defined. In short, $\{\phi_i^j\}_{j=1}^{n_i}$ is the standard nodal basis for the space V_i^h. Figure 1 shows the multilevel nodal basis functions in one dimension. The multilevel nodal basis preconditioner M of Bramble, Pasciak, and Xu [3], [6] applied to $v \in V^h$ takes the following form:

$$M^{-1}v = \alpha A_0^{-1} Q_0 v + \sum_{l=1}^{J} \sum_{i=1}^{n_l} (v, \phi_l^i)\phi_l^i, \tag{1}$$

where the operator A_0 is a discretization of the elliptic operator $-\triangle$ on V_0^h and Q_0 is the standard orthogonal L^2 projection from V_J^h to V_0^h. The parameter α is not present in the original formulation of this preconditioner. Its inclusion here is inspired by the work of Smith and Widlund [8]. It can be shown that this modified version has the same order of condition number bound as the unmodified one (i.e., $\alpha = 1$), provided that α is independent of h and H.

The operator M^{-1} involves transformations between standard nodal basis and multilevel nodal basis (the second term in the above equation), as well as the solution of the problem corresponding to the original coarse triangulation (the first term in the equation). In matrix form, this preconditioner $\hat{M}$ can be written as:

$$\hat{M}^{-1} = GD^{-1}G^T,$$

where G and G^T represent the transformation matrices from multilevel nodal basis to nodal basis and vice versa, respectively; and $D^{-1} = \text{blockdiag}[I_{n_J}, I_{n_{J-1}}, \cdots, I_{n_1}, \alpha A_0^{-1}]$ solves the elliptic problem on the coarse level while leaving the other levels unchanged. It was proved in [3] that for smooth coefficient problems, the condition number of the preconditioned system $\kappa(\hat{M}^{-1}K) \leq CJ^2$, where C is a constant independent of h and H. In addition, this algorithm requires only $O(n_J)$ operations, where n_J is the dimension of the finite element space V^h.

The matrix G transforms the input vector from multilevel nodal basis to nodal basis. The dimension of the multilevel nodal basis is $m \equiv n_J + n_{J-1} + \cdots + n_1 + n_0$, whereas that of the standard nodal basis is $n = n_J$. Thus, G is a rectangular matrix of size $n \times m$ (which is unlike the square transformation matrices for the hierarchical basis). Let $v = \{v_l^i, i = 1, \cdots, n_l\}_{l=0}^J \in R^M$, where v_l^i is the value at the nodal point y_l^i corresponding to level l, then

$$Gv = \sum_{l=0}^{J} \sum_{i=1}^{n_l} v_l^i \phi_l^i,$$

where $\phi_l^i, i = 1, 2, \cdots, n_l$ is the set of basis functions in V_l^h.

The algorithm for G as applied to a vector v (of dimension m) is as follows:

Algorithm $G : v_i \rightarrow u, i = 0, \cdots, J$
  for $l = 0, \cdots, J-1$
    $v_{l+1} := v_{l+1} + I_l^{l+1} v_l$
  end for
  $u = v_J$
end G

Here the I_l^{l+1} matrix is obtained from the choice of ϕ_i's. On a two-dimensional uniform domain using triangular elements, it corresponds to a seven-point interpolation operator.

2.2. Domain decomposition methods. Domain decomposition methods generally split the space V^h into $N+1$ subspaces

$$V^h = V_{har}^h \oplus V_0^h(\Omega_1) \oplus \cdots \oplus V_0^h(\Omega_N).$$

For each subregion Ω_i, we thus have a subspace $V_0^h(\Omega_i) = V^h \cap H_0^1(\Omega_i)$. The elements of V_{har}^h are piecewise, discrete harmonic functions, i.e., they are orthogonal, in the sense of the bilinear form $a(\cdot, \cdot)$, to all the other subspaces.

First we partition the stiffness matrix K and vector x into those corresponding to the interior of the subregions and the edges and vertices. We then have

$$Kx = \begin{bmatrix} K_I & K_{IB} \\ K_{IB}^T & K_B \end{bmatrix} \begin{bmatrix} x_I \\ x_B \end{bmatrix} = \begin{bmatrix} b_I \\ b_B \end{bmatrix}.$$

If we apply block Gaussian elimination to eliminate the interior points, we obtain the following reduced system or Schur complement for the edges and interfaces

$$S_B x_B = (K_B - K_{IB}^T K_I^{-1} K_{IB}) x_B = b_B - K_{IB}^T K_I^{-1} b_I = \hat{b}_B.$$

The Schur complement matrix S_B is generally dense. However, it is not necessary to generate this matrix since, in the conjugate gradient iteration, this matrix is needed only in terms of matrix-vector products which can be computed by solving each subregion once and collecting the solution on the interfaces and vertices.

2.3. Multilevel nodal basis domain decomposition preconditioner. In this section, we combine the ideas from previous sections to derive a new domain decomposition algorithm. The symmetric form of the preconditioned system using the multilevel nodal basis preconditioner can be written as:

$$D^{-\frac{1}{2}} G^T K G D^{-\frac{1}{2}} \tilde{x} = \tilde{b},$$

where $GD^{-\frac{1}{2}}\tilde{x} = x$ and $\tilde{b} = D^{-\frac{1}{2}} G^T b$. Let us partition the unknowns corresponding to the multilevel nodal basis $\tilde{x}$ into those on the subregion interior $\tilde{x}_I$ and those on the interface $\tilde{x}_B$ and we eliminate the subregion interior variables $\tilde{x}_I$, we obtain the reduced system:

$$\tilde{S}_B \tilde{x}_B = \tilde{b}_B,$$

where $\tilde{S}_B$ is the Schur complement of $D^{-\frac{1}{2}} G^T K G D^{-\frac{1}{2}}$ after eliminating x_I. Here we can also decompose G according to the interior and interface unknowns so that

$$G = \begin{bmatrix} G_I & G_{IB} \\ G_{BI} & G_B \end{bmatrix},$$

where G_I and G_B are the transformation (rectangular) matrices involving subregion interior points and interface points, respectively.

The above formulation requires many arithmetic operations when the Schur complement is applied to a vector during the conjugate gradient iterations. By using the following lemmas, the above system can be reduced to a simpler form.

LEMMA 2.1. *G represents a change of basis which leaves the space of variables on Γ invariant (i.e., $G_{BI} = 0$).*

Proof. G_{BI} represents the contribution of the multilevel nodal bases in the subdomain interior to the nodal basis on the interfaces during the transformation. Recall from previous sections that

$$u \equiv Gv = \sum_{l=0}^{J} \sum_{i=1}^{n_J} v_l^i \phi_l^i, \tag{2}$$

where $\{\phi_l^i, i = 1, 2, \cdots, n_l\}$ is the set of basis functions in V_l^h, and $v_l^i, i = 1, \cdots, n_l$ is the set of values defined on the nodal points x_l^i. Let $u = (u_I, u_B)^T$ and $v = (v_I, v_B)$ be the partitionings according to subdomain interior and interfaces, where $u \in R^n$ and $v \in R^m$. If we evaluate the above expression at node y_J^i on the interface Γ, we obtain

$$u_B^i = (G_B v_B)^i + (G_{BI} v_I)^i = \sum_{l=0}^{J} \left\{ \sum_{j \in \Gamma} (v_B)_l^j (\phi_l^j)_i + \sum_{j \notin \Gamma} (v_I)_l^j (\phi_l^j)_i \right\},$$

where $(\phi_l^j)_i$ is the value of the basis function at node j evaluated at node i on level l. It can be verified that

$$(\phi_l^j)_i = 0 \quad \forall j \in \Omega_k \text{ and } i \in \Gamma,$$

since all multilevel nodal basis functions at the nodes interior to the subregions vanish on the interface. In other words, the second term in the summation of (2) is identically equal to 0. Thus $G_{BI} = 0$ and $u^i = (G_B v_B)^i$. □

The following lemma is based on a similar one from Smith and Widlund's paper [8]. The proof can be obtained by a straightforward matrix manipulation.

LEMMA 2.2. *Let $G_{BI} = 0$ and $D = \text{blockdiag}(D_I, D_B)$, then*

$$\tilde{S}_B = D_B^{-\frac{1}{2}} G_B^T S_B G_B D_B^{-\frac{1}{2}}.$$

The above two lemmas imply that if we first eliminate the interior variables and then transform to the multilevel nodal basis, we will be solving the same linear system as before (transformation to multilevel nodal basis and then do the elimination). As a result, we are solving the following simpler and smaller system:

$$D_B^{-\frac{1}{2}} G_B^T S_B G_B D_B^{-\frac{1}{2}} \tilde{x}_B = \tilde{b}_B,$$

which is equivalent to solving the Schur complement system $S_B x_B = b_B$ with the preconditioner $M^{-1} = G_B D_B^{-1} G_B^T$. We call the preconditioner using this new formulation the multilevel nodal basis domain decomposition (MNBDD) preconditioner. This method offers several possible advantages over the standard multilevel nodal basis method by Bramble, Pasciak, and Xu [3]. The conjugate gradient iteration is

carried out over a much smaller set of unknowns and we will show that the condition number is smaller. The solution of the subproblems on the interfaces is easily parallelizable since they are independent. One possible drawback, however, is that now it is necessary to solve each subregion exactly in each iteration which adds more computational overhead.

Here the algorithm for G_B is similar to the one for G shown before, except now the basis functions used in the evaluation are restricted only to those on the interfaces. The operation of G_B on a vector v_B is defined by:

$$G_B v_B = \sum_{j=0}^{J} \sum_{i=1}^{m_j} (v_B)_j^i \phi_j^i,$$

where $\phi_j^i, i = 1, 2, \cdots m_j$ is the set of basis functions on Γ.

The MNBDD algorithm can be summarized as follows:

Algorithm MNBDD : input $= r$, output $= z = M_B^{-1} r$
Perform partial change of basis to the MN basis $v = G_B^T r$
solve the coarse grid problem $y = D_B^{-1} v$
Perform change of basis back to the nodal basis $z = G_B y$
end MNBDD

We need one more lemma from [8].

LEMMA 2.3. *Let K be symmetric and positive definite. Then the condition numbers of K and its Schur complement satisfy*

$$\kappa(Schur(K)) \leq \kappa(K).$$

Using Lemma 2.3, as well as the condition number bounds from [3], we arrive at the following main theorem.

THEOREM 2.4. $\kappa(M_B^{-1} S_B) \leq O(\log^2(H/h))$ *for smooth coefficient problems.*

Proof. By using Lemma 2.1 and 2.2 we obtain

$$\tilde{S}_B = D_B^{-\frac{1}{2}} G_B^T S_B G_B D_B^{-\frac{1}{2}}.$$

By using Lemma 2.3, we obtain

$$\begin{aligned} \kappa(D_B^{-\frac{1}{2}} G_B^T S_B G_B D_B^{-\frac{1}{2}}) &= \kappa(\tilde{S}_B) \\ &\leq \kappa(D^{-\frac{1}{2}} G^T K G D^{-\frac{1}{2}}), \end{aligned}$$

which is bounded by $O(\log^2(H/h))$, see [3]. □

It is also proved in [3] that the condition number bound in Theorem 2.4 depends also on the elliptic regularity of the problem. For example, for smooth coefficient problems on convex polygonal domains, the condition number is bounded by $O(\log(H/h))$; and for certain discontinuous problems, the condition number is bounded by $O(\log^3(H/h))$.

The hierarchical basis domain-decomposed (HBDD) [8] algorithm is similar to the BPS algorithm by Bramble, Pasciak, and Schatz [2]. The only difference is that the HBDD uses the hierarchical basis preconditioner for the edges while the BPS uses variants of Dryja's preconditioner [5]. The MNBDD algorithm, however, has

one important difference; namely, in addition to the use of multilevel nodal basis as preconditioners on the edges, the MNBDD algorithm also implements the multilevel nodal basis on the vertices. This introduces some redundancy on the vertices. This redundancy may be the reason for its improved performance (see next section).

3. Numerical results.

3.1. Two-subdomain example. For the two-subdomain case, since there is one edge and no vertex, it is not necessary to solve a coarse problem. We use the two-subdomain example of [8] with a different right-hand side, and we compare our results with those reported in [8]. We also include results from using the Dryja ($l_0^{1/2}$) [5] preconditioner. We use the domain $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2$, where $\bar{\Omega}_1$ and $\bar{\Omega}_2$ are unit squares aligned along an edge $\Gamma = \bar{\Omega}_1 \cap \bar{\Omega}_2$. We use the standard uniform mesh, and the usual five-point discretization for the Laplacian and the iteration counts are listed in Table 1 (where n is the number of unknowns on the interface). The right-hand side is such that the solution is $x(x-1)y(y-1)$ and the stopping criterion is when the relative 2-norm of the residual falls below 10^{-6}. The initial guess used is $u^{(0)} = 1.0$. We observe that while the iteration count for the HBDD continues to grow with larger n, the other preconditioners seem to be bounded independent of n. While the iteration counts using the HBDD preconditioners grow with n, we see that the iteration counts using MNBDD seem to gradually level off. Overall, we see that the MNBDD preconditioner performs relatively well compared to the others.

TABLE 1
Iteration count versus n.

n	No precond	Dryja	MNBDD	HBDD
8	4	4	4	4
16	8	6	7	7
32	16	6	9	8
64	27	6	9	10
128	39	6	9	12

3.2. Many-subdomain example. Next we consider the case of many subdomains. The unit square Ω is subdivided uniformly into $k \times k$ square subdomains and the same model problem is solved using uniform meshes. We compare our results with a set of experiments reported in [8]. We also compare the condition numbers as well as iteration counts between our method, Smith and Widlund's method, and Bramble, Pasciak, and Schatz's method (BPS).

3.2.1. Two-dimensional Poisson equation. A two-dimensional Poisson equation is solved in this case. The right-hand side f is constructed such that the solution is $u(x,y) = x(x-1)y(y-1)$ and the stopping criterion is 10^{-5}. Again, the initial guess is $u^{(0)} = 1.0$. The results are shown in Table 2.

Our first observation is that our condition number results for the HBDD method agree very well with those reported in [8]. We also observe that the condition numbers using the MNBDD method are much lower than the BPS and HBDD methods. The condition numbers grow very slowly with n; this is not the case with the BPS and HBDD methods.

TABLE 2
Condition numbers and iteration counts for the Poisson problem.

Grid	Number of subdomains	HBDD($\alpha = 3.6$)		BPS		MNBDD	
		κ	iter	κ	iter	κ	iter
32x32	2x2	9.62	11	11.85	11	2.24	7
32x32	4x4	7.96	11	8.75	14	2.19	8
32x32	8x8	5.30	10	6.08	12	2.10	7
64x64	2x2	12.68	13	16.47	12	2.32	8
64x64	4x4	11.84	13	13.03	15	2.28	8
64x64	8x8	8.52	12	9.79	15	2.21	8
64x64	16x16	5.41	10	6.32	13	2.11	7
128x128	4x4	16.49	15	17.92	18	2.35	8
128x128	8x8	12.54	15	14.18	16	2.35	8
128x128	16x16	8.69	13	10.21	15	2.24	8
128x128	32x32	5.42	10	6.36	13	2.11	7
256x256	4x4	21.90	17	23.45	19	2.39	8
256x256	8x8	17.30	17	19.33	18	2.43	8
256x256	16x16	12.72	15	14.79	17	2.36	8
256x256	32x32	8.69	13	10.27	15	2.24	8
256x256	64x64	5.37	10	6.37	12	2.09	7

3.2.2. Two-dimensional variable coefficient problem. Next, we use the following variable coefficient boundary value problem:

$$-\frac{\partial}{\partial x}\left(e^{-xy}\frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(e^{xy}\frac{\partial u}{\partial y}\right) = f(x,y) \quad \text{in} \quad \Omega = [0,1]^2.$$

In order to account for the variation of coefficients in this case, we use a simple diagonal scaling in the HBDD, MNBDD, and BPS preconditioners. The right-hand side is constructed such that the solution is $u(x,y) = x(x-1)y(y-1)$, and the stopping criterion is achieved when the relative 2-norm of the residual drops below 10^{-5}. The number of iterations are shown in Table 3. Here we observe a small deterioration in convergence rates. With the use of strongly varying coefficients, all the methods used show rapidly decaying convergence rates.

TABLE 3
Iteration counts for variable coefficient case.

Grid	No. subdomains	HBDD	BPS	MNBDD
32x32	2x2	14	13	9
32x32	4x4	16	17	10
32x32	8x8	15	15	11
64x64	4x4	19	19	10
64x64	8x8	18	19	11
64x64	16x16	16	16	12
128x128	4x4	21	21	11
128x128	8x8	21	21	12
128x128	16x16	19	20	12
128x128	32x32	16	17	13

3.2.3. Two-dimensional discontinuous coefficient problem. The discontinuous coefficient example used in this experiment is as follows:

$$-\frac{\partial}{\partial x}\left(\mu(x,y)\frac{\partial u}{\partial x}\right) - \frac{\partial}{\partial y}\left(\mu(x,y)\frac{\partial u}{\partial y}\right) = f(x,y) \quad \text{in} \quad \Omega = [0,1]^2,$$

where μ is a constant within each subdomain. In the following, we show the numerical results for the 16-subdomain case with μ in each subdomain as shown in Fig. 2.

10^{-1}	10^{3}	10^{-2}	10^{2}
10^{-2}	10^{2}	10^{-3}	10
10^{-3}	10	10^{-4}	1
10^{-4}	1	10^{4}	10^{-1}

FIG. 2. *Discontinuous coefficients used.*

Again, diagonal scaling is used to account for the variation of coefficients in all preconditioners used. The right-hand side is constructed such that the solution is $u(x, y) = x(x-1)y(y-1)$, and the stopping criterion is the same as in the previous cases. The number of iterations are shown in Table 4, which shows that all the methods used are effective in solving the above discontinuous coefficient problem, with the MNBDD preconditioner slightly better.

TABLE 4
Iteration counts for discontinuous coefficient case.

Grid	No. subdomains	HBDD	BPS	MNBDD
32x32	4x4	15	14	12
64x64	4x4	18	16	15
128x128	4x4	22	18	18
256x256	4x4	25	20	21

Through the numerical experiments, we have shown that the MNBDD preconditioner offers good convergence rates (better than BPS and HBDD for the problems used in our experiment), as well as low computational cost ($O(n)$ for MNBDD and HBDD and $O(n \log n)$ for BPS).

REFERENCES

[1] R. E. BANK, T. F. DUPONT, AND H. YSERENTANT, *The hierarchical basis multigrid method*, Numer. Math., 52 (1988), pp. 427–458.

[2] J. H. BRAMBLE, J. E. PASCIAK, AND A. H. SCHATZ, *The construction of preconditioners for elliptic problems by substructuring*, I, Math. Comp., 47 (1986), pp. 103–134.

[3] J. H. BRAMBLE, J. E. PASCIAK, AND J. XU, *Parallel multilevel preconditioners*, Math. Comp., 55 (1990), pp. 1–22.

[4] T. F. CHAN, R. GLOWINSKI, J. PERIAUX, AND O. B. WIDLUND, *Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[5] M. DRYJA, *A capacitance matrix method for Dirichlet problem on polygon region*, Numer. Math., 39 (1982), pp. 51–54.

[6] C.-C. J. KUO, T. F. CHAN, AND C. H. TONG, *Multilevel filtering elliptic preconditioners*, UCLA CAM Report 89-23, August 1989; SIAM J. Matrix Anal. Appl., 11 (1990), pp. 403–429.

[7] B. F. SMITH, *An optimal domain decomposition preconditioner for the finite element solution of linear elasticity problems*, preprint.

[8] B. SMITH AND O. WIDLUND, *A domain decomposition algorithm using a hierarchical basis*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 1212–1226.

[9] H. YSERENTANT, *On the multi-level splitting of finite element spaces*, Numer. Math., 49 (1986), pp. 379–412.

[10] J. XU AND J. QIN, *On some multilevel preconditioners*, SIAM J. Sci. Statist. Comput., submitted.

[11] J. XU, *Theory of multilevel methods*, Ph.D. thesis, Department of Mathematics, Cornell University, NY, 1989.