SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 1–17, January 1989
© 1989 Society for Industrial and Applied Mathematics

ALGORITHMS WITH CONIC TERMINATION FOR NONLINEAR OPTIMIZATION*

DONG C. LIU† AND JORGE NOCEDAL†

Abstract. This paper describes the implementation of algorithms for unconstrained optimization which have the property of minimizing conic objective functions in a finite number of steps, when line searches are exact. This work extends the algorithms of Davidon ["Conjugate directions for conic functions" in *Nonlinear Optimization* 1981, M.J.D. Powell, ed., Academic Press, London, 1981] and Gourgeon and Nocedal [*SIAM J. Sci. Statist. Comput.*, 6 (1985), pp. 253-267] to general nonlinear objective functions, paying much attention to the practical behavior of the new methods. Three types of algorithms are described; they are extensions of the conjugate gradient method, the BFGS method, and a limited-memory BFGS method. The numerical results show that new methods are very effective in solving practical problems.

Key words. conjugate gradient, quasi-Newton, minimization, nonlinear optimization

AMS(MOS) subject classifications. 65, 49

1. Introduction.

Fletcher and Reeves [7] derived the conjugate gradient method for function minimization by first considering minimization of a quadratic. The formulas for generating the displacements were then applied to general objective functions, with only two changes: all references to the Hessian of the quadratic were replaced by gradient differences, and a line search was introduced to ensure progress towards the solution.

In this paper we will follow a similar procedure, but use a conic model function instead of a quadratic. We first present three algorithms for minimizing conic functions: a conjugate gradient method, a BFGS method, and a limited storage BFGS method. We then adapt these methods to general problems by making a few changes: all references to data specific to conic functions are expressed in terms of function and gradient values, and a line search is introduced at each step. We thus obtain three new types of algorithms for nonlinear optimization, and our numerical tests reveal that they are very effective in solving practical problems.

The use of conic models for optimization was first proposed by Davidon [4]. Shortly afterwards, Sorensen [19] developed an algorithm that updates collinear scalings, for which he was able to establish a superlinear rate of convergence. Closely related to this work are the numerical investigation of Grandinetti [12] and the algorithms of Ariyawansa [1]. Attention then turned to the problem of designing algorithms that minimize a conic objective function in a finite number of steps. Davidon [5] and Gourgeon and Nocedal [11] considered this problem and developed analogues of the conjugate gradient method for solving it. This paper starts where these last two papers ended. We now study how to implement algorithms with conic termination on general objective functions, paying much attention to the practical behavior of the new methods. Even though this paper is largely self-contained, some familiarity with

* Received by the editors December 30, 1987; accepted for publication (in revised form) July 14, 1988. This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contracts W-31-109-Eng-38 and DE-FG02-87ER25047, and by National Science Foundation grant DCR-86-02071. This work was performed when the authors were at Argonne National Laboratory, Argonne, Illinois 60439.

† Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60208.

Davidon [4], [5] or Gourgeon and Nocedal [11] will make our presentation easy to follow.

The use of conic functions introduces numerous practical difficulties, and as a result the algorithms we present are rather complex. For the sake of clarity we will first describe the algorithms in very general terms. We then consider, one by one, all practical issues of implementation and indicate the required changes in the algorithms. Thus the algorithms evolve throughout the various sections of this paper, until they are given in final form in §6.

This paper presents the first numerical investigation of algorithms with conic termination when applied to general nonlinear problems. The numerical results presented in §7 indicate that the new conjugate gradient method is more effective than the standard conjugate gradient method (but requires slightly more storage). On the other hand, the conic BFGS and conic Limited BFGS methods do not appear to be more efficient than the corresponding standard algorithms. Throughout the paper $B > 0$ means that the symmetric matrix B is positive definite.

2. Conic functions. In this section we describe a few basic properties of conic functions; for a more complete discussion see Davidon [4]. Given a reference point $x_0 \in \mathbf{R}^n$, we will write any $x \in \mathbf{R}^n$ as $x = x_0 + s$. With respect to this reference point we define the conic function

$$f(x) = f(x_0 + s) = f_0 + \frac{g_0^T s}{1 - a^T s} + \frac{1}{2}\frac{s^T A s}{(1 - a^T s)^2}, \tag{2.1}$$

where $g_0, a \in \mathbf{R}^n$ and A is an $n \times n$ positive definite and symmetric matrix. We call a the *horizon vector* and denote the domain of f by D, i.e., $D = \{x : 1 - a^T s \neq 0\}$. Since the term $s/(1 - a^T s)$ appears once in the second term on the right-hand side of (2.1), and twice in the third term, it is clear that by letting

$$w = \frac{s}{1 - a^T s}, \tag{2.2}$$

the conic function becomes a quadratic in the variable w,

$$f(x) = f(x_0 + s) = f_0 + g_0^T w + \frac{1}{2} w^T A w \equiv h(w). \tag{2.3}$$

We can express s in terms of w:

$$s = \frac{w}{1 + a^T w}. \tag{2.4}$$

To simplify the formulas we define

$$\gamma(x) = 1 - a^T s = \frac{1}{1 + a^T w}, \tag{2.5}$$

so that $w = s/\gamma$. We call $H = \{x : 1 - a^T s = 0\}$ the *singular hyperplane*, and note that if $\gamma(x)\gamma(y) < 0$ then x and y lie on opposite sides of H. (Davidon calls H the *horizon* of f.)

We will need to relate the derivative of f to that of h. Since

$$f(x) = f(x_0 + s) = f(x_0 + \frac{w}{1 + a^T w}) = h(w),$$

it follows from the chain rule that

$$h'(w) = \gamma(x)(I - as^T)g(x), \tag{2.6}$$

where g denotes the gradient of f. The conic (2.1) is useful as a model function for minimization only if it has a unique minimizer. This is ensured by the conditions

$$A > 0 \quad \text{and} \quad a^T A^{-1} g_0 \neq 1 \tag{2.7}$$

(cf. Gourgeon and Nocedal [11, p.255]). If (2.7) holds, f will be called a *normal* conic function.

3. Basic algorithms. The new algorithms presented in this paper are designed to minimize conic functions in n steps. We note that the problem of minimizing a normal conic function is simplified if we introduce the change of variables (2.2) and consider instead the minimization of the strictly convex quadratic function h given by (2.3). Conditions (2.7) ensure that these two minimization problems are equivalent; see Gourgeon and Nocedal [11]. Therefore we will begin by reviewing algorithms that minimize quadratic objective functions, and which form the core of our new methods.

Let us consider the minimization of h, and let the reference point x_0 be our initial estimate of the solution. Thus $s_0 = 0$, and the starting point in the minimization of h will be $w_0 = 0$. We will denote the difference of gradients of h by y:

$$y_k = h'_{k+1} - h'_k, \tag{3.1}$$

where $h'_k = h'(w_k)$. The first method we consider is the conjugate gradient method.

CG Method. Let $\beta_0 = 0, \delta_{-1} = 0$, and $w_0 = 0$. For $k = 0, 1, \cdots$,

$$w_{k+1} = w_k + \sigma_k \delta_k, \tag{3.2}$$

where σ_k is a steplength and δ_k is the search direction, given by

$$\delta_k = -h'_k + \beta_k \delta_{k-1} \tag{3.3}$$

with

$$\beta_k = \frac{(h'_k)^T y_{k-1}}{y_{k-1}^T \delta_{k-1}}, \quad \text{for } k > 0. \tag{3.4}$$

A discussion on the precise choice of the steplength σ_k for this method, as well as for the next two methods, will be deferred to §5.4. We now describe the BFGS quasi-Newton method (see, for example, Dennis and Schnabel [6]).

BFGS Method. Given $B_0 > 0$, let $w_0 = 0$. For $k = 0, 1, \cdots$,

$$w_{k+1} = w_k + \sigma_k \delta_k, \tag{3.5}$$

where

$$\delta_k = -B_k^{-1} h'_k. \tag{3.6}$$

The matrix B_k is updated by the formula

$$B_{k+1} = B_k - \frac{B_k u_k u_k^T B_k}{u_k^T B_k u_k} + \frac{y_k y_k^T}{u_k^T y_k}, \tag{3.7}$$

where

$$u_k \equiv w_{k+1} - w_k. \tag{3.8}$$

The third method we consider is a limited-memory quasi-Newton method. Various implementations of the BFGS method with limited memory have been proposed, including the algorithms of Shanno [16], Nocedal [14] and Buckley and LeNir [2]. We will consider here the algorithm of Nocedal [14], which is the simplest and therefore the easiest to generalize. Instead of using the $O(n^2)$ storage locations required by the matrix B_k, one stores separately a few BFGS corrections to the initial matrix B_0. The algorithm uses the inverse BFGS update formula, given by

$$B_{k+1}^{-1} = \left(I - \frac{u_k y_k^T}{u_k^T y_k}\right) B_k^{-1} \left(I - \frac{y_k u_k^T}{u_k^T y_k}\right) + \frac{u_k u_k^T}{u_k^T y_k}. \tag{3.9}$$

If we define

$$r_k = \frac{1}{u_k^T y_k}, \tag{3.10}$$

and

$$V_k = \left(I - r_k y_k u_k^T\right), \tag{3.11}$$

then we can write

$$B_{k+1}^{-1} = V_k^T B_k^{-1} V_k + r_k u_k u_k^T. \tag{3.12}$$

We now give a general description of the method.

Limited BFGS Method. The method differs from BFGS only in the way the matrix is updated. Let us suppose that B_0^{-1} is sparse and that we update it m times using the BFGS formula (3.12). We obtain

$$\begin{aligned} B_m^{-1} &= \left(V_{m-1}^T \cdots V_0^T\right) B_0^{-1} \left(V_0 \cdots V_{m-1}\right) \\ &+ r_0 \left(V_{m-1}^T \cdots V_1^T\right) u_0 u_0^T \left(V_1 \cdots V_{m-1}\right) \\ &+ r_1 \left(V_{m-1}^T \cdots V_2^T\right) u_1 u_1^T \left(V_2 \cdots V_{m-1}\right) \\ &+ \vdots \\ &+ r_{m-1} u_{m-1} u_{m-1}^T. \end{aligned} \tag{3.13}$$

At this stage we decide not to increase the storage any further. To obtain B_{m+1}^{-1} we delete the oldest correction to leave space for the new one. Thus (3.13) becomes

$$\begin{aligned} B_{m+1}^{-1} &= \left(V_m^T \cdots V_1^T\right) B_0^{-1} \left(V_1 \cdots V_m\right) \\ &+ r_1 \left(V_m^T \cdots V_2^T\right) u_1 u_1^T \left(V_2 \cdots V_m\right) \\ &+ r_2 \left(V_m^T \cdots V_3^T\right) u_2 u_2^T \left(V_3 \cdots V_m\right) \\ &+ \vdots \\ &+ r_m u_m u_m^T. \end{aligned} \tag{3.14}$$

Note that (3.14) only differs from (3.13) in that all the indices have been shifted by one, except for the index in B_0^{-1}.

To obtain the subsequent matrices we proceed as in (3.14), increasing the indices by one at each step. To be more precise, for $k \leq m$, B_k^{-1} is computed by (3.13), replacing m by k; and for $k \geq m$, B_k^{-1} is computed by (3.14), except that the indices $1, \cdots, m$ should be replaced by $(k-m), \cdots, (k-1)$. The number m is chosen by the user and its effect on the performance of the algorithms is discussed by Nocedal [14]; see also Buckley and LeNir [2].

It is well known that, with exact line searches, the CG, BFGS, and Limited BFGS algorithms minimize the quadratic objective h in at most n steps (see, for example, Fletcher [8] and Nocedal [14]).

4. Minimization of conic functions. The three algorithms that we have just described can be translated into algorithms for the minimization of the conic function f. Since f and h are related through (2.2) and (2.3), we now transform the iterates, given so far in terms of the variable w, back into the original variable s.

While minimizing h, the displacement at the kth step is

$$\begin{aligned} w_{k+1} - w_k &= \sigma_k \delta_k \\ &\equiv u_k. \end{aligned} \tag{4.1}$$

From (2.2) and (2.5)

$$\begin{aligned} u_k &= \frac{s_{k+1}}{\gamma_{k+1}} - \frac{s_k}{\gamma_k} \\ &= \frac{s_{k+1} - s_k}{\gamma_{k+1}} - \frac{s_k}{\gamma_k} + \frac{s_k}{\gamma_{k+1}} \\ &= \frac{s_{k+1} - s_k}{\gamma_{k+1}} - s_k \left(\frac{1 - a^T s_{k+1} - 1 + a^T s_k}{\gamma_k \gamma_{k+1}} \right) \\ &= \frac{1}{\gamma_{k+1}} \left(I + \frac{s_k a^T}{\gamma_k} \right) (s_{k+1} - s_k). \end{aligned} \tag{4.2}$$

We now express γ_{k+1} in terms of quantities evaluated at x_k. From (2.5) and (4.1)

$$\begin{aligned} \gamma_{k+1} &= \frac{1}{1 + a^T w_{k+1}} \\ &= \frac{1}{1 + a^T w_k + a^T u_k} \\ &= \frac{1}{\frac{1}{\gamma_k} + a^T u_k} \\ &= \frac{\gamma_k}{1 + \gamma_k a^T u_k}. \end{aligned} \tag{4.3}$$

Noting that the inverse of $\left(I + s_k a^T / \gamma_k\right)$ is $\left(I - s_k a^T\right)$, we obtain, from (4.2) and (4.3),

$$s_{k+1} = s_k + \frac{\gamma_k}{1 + \gamma_k a^T u_k} \left(I - s_k a^T\right) u_k. \tag{4.4}$$

This formula defines the iterates in terms of the original variable s, or equivalently, in terms of x. We can now describe a general class of algorithms for minimizing the conic function f, defined by (2.1). We assume that the horizon vector a is known.

Algorithm A.
Given x_0 (and $B_0 > 0$ for the BFGS and Limited BFGS methods), set $s_0 = 0$ and $k = 0$.
1. Compute

$$\gamma_k = (1 - a^T s_k), \tag{4.5}$$

$$h'_k = \gamma_k (I - a s_k^T) g_k. \tag{4.6}$$

2. For the CG method, when $k = 0$, let $\delta_k = -h'_k$; otherwise compute

$$\beta_k = \frac{(h'_k)^T y_{k-1}}{y_{k-1}^T \delta_{k-1}}, \tag{4.7}$$

$$\delta_k = -h'_k + \beta_k \delta_{k-1}. \tag{4.8}$$

For the BFGS and Limited BFGS methods compute

$$\delta_k = -B_k^{-1} h'_k. \tag{4.9}$$

3. Find a steplength σ_k and compute

$$u_k = \sigma_k \delta_k, \tag{4.10}$$

$$d_k = \frac{\gamma_k}{1 + \gamma_k a^T u_k} (I - s_k a^T) u_k, \tag{4.11}$$

and set

$$s_{k+1} = s_k + d_k. \tag{4.12}$$

4. For the BFGS method, update B_k by means of (3.7); for the Limited BFGS method, update B_k as described in §3.
5. If the solution has been obtained, stop; otherwise set $k \leftarrow k + 1$, and go to step 1.

Note that, if $a = 0$, we recover the standard CG, BFGS, and Limited BFGS methods described in the previous section. Due to the quadratic termination property of these three algorithms, and the fact that minimizing h and f is equivalent, it follows that Algorithm A minimizes f in at most n steps, if line searches are exact. We call this property *conic termination.* In step 2 we have used the Hestenes–Stiefel formula for β_k; however, we could have used instead the Fletcher–Reeves [7] or Polak–Ribière (see Polak [15]) formulas.

We will now describe other, more concise expressions for d_k. Let us begin by considering the CG method. From (4.11), (4.10), and (4.8) we have

$$d_k = \frac{\gamma_k}{1 + \gamma_k a^T u_k} \left(I - s_k a^T\right) \sigma_k \left(-h'_k + \beta_k \delta_{k-1}\right). \tag{4.13}$$

We can express δ_{k-1} in terms of d_{k-1}, using (4.1), (4.2), and (4.12):

$$\delta_{k-1} = \frac{1}{\sigma_{k-1} \gamma_k} \left(I + \frac{s_{k-1} a^T}{\gamma_{k-1}}\right) d_{k-1}. \tag{4.14}$$

Substituting this into (4.13), using (4.6) and (4.12), gives

$$d_k = \frac{-\gamma_k^2\sigma_k}{1+\gamma_k a^T u_k}\left(I - s_k a^T\right)\left(I - a s_k^T\right) g_k + \frac{\sigma_k \beta_k \gamma_k}{(1+\gamma_k a^T u_k)\sigma_{k-1}\gamma_{k-1}} d_{k-1}. \tag{4.15}$$

For the BFGS and Limited BFGS methods we have, from (4.11), (4.10), (4.9), and (4.6),

$$d_k = \frac{-\gamma_k^2\sigma_k}{1+\gamma_k a^T u_k}\left(I - s_k a^T\right) B_k^{-1}\left(I - a s_k^T\right) g_k. \tag{4.16}$$

We should point out, in addition, that (4.15) and (4.16) can be simplified further by setting $g_k^T s_k = 0$ at each step. This is justified by the fact that when Algorithm A, with exact line searches, is applied to a normal conic function then $g_k^T d_j = 0$ for all $j < k$ (cf. Gourgeon and Nocedal [11, Thm. 5.1]).

For future reference, we note that there is a simple expression for the term $u_k^T y_k$, required by the BFGS update formulas (3.7) and (3.9). From (3.1), (4.6), (4.2), and (4.12) we have

$$\begin{aligned} y_k^T u_k &= g_{k+1}^T\left(I - s_{k+1}a^T\right)\left(I + \frac{s_k a^T}{\gamma_k}\right)(x_{k+1} - x_k) - \frac{\gamma_k}{\gamma_{k+1}} g_k^T (x_{k+1} - x_k) \\ &= \left[\frac{\gamma_{k+1}}{\gamma_k} g_{k+1} - \frac{\gamma_k}{\gamma_{k+1}} g_k\right]^T (x_{k+1} - x_k). \end{aligned} \tag{4.17}$$

We note that, while deriving these formulas, we have not assumed that line searches are exact.

To make Algorithm A practical for the minimization of general functions we need to introduce a line search in (4.12). The vector d_k will now define only the search direction, and not the total step. Thus the iteration is

$$x_{k+1} = x_k + \lambda_k d_k, \tag{4.18}$$

where the steplength λ_k is determined by a line search on f. We also need to eliminate all dependence on the horizon vector a. Davidon [4] has shown that γ can be computed without knowledge of a, using only function and gradient values. This then enables us to compute a using function and gradient information at three collinear points (Davidon [5]). We now give the two formulas that are fundamental to our algorithm development.

Recall that $\gamma_0 = 1$, since $s_0 = 0$. Given γ_k, we can compute γ_{k+1} by

$$\gamma_{k+1} = \frac{-g_k^T (x_{k+1} - x_k)}{f_k - f_{k+1} + \rho_{k,k+1}} \gamma_k, \tag{4.19}$$

where

$$\rho_{k,k+1} = +\left[(f_k - f_{k+1})^2 - g_{k+1}^T (x_{k+1} - x_k) g_k^T (x_{k+1} - x_k)\right]^{1/2}; \tag{4.20}$$

see also Gourgeon and Nocedal [11, p. 257]. It can be shown that conditions (2.7) guarantee that $\rho_{k,k+1}$ is real.

The horizon vector a can be computed using function and gradient values at any three collinear points. In particular, consider the iterates x_k and x_{k+1} and let x_t be given by $x_t = x_k + \lambda_t d_k$, where $\lambda_t \neq \lambda_k$. Then (cf. Gourgeon and Nocedal [11, Lem. 3.3])

$$a = \frac{(\lambda_t - \lambda_k)\gamma_k^2 g_k + \lambda_k \gamma_t^2 g_t - \lambda_t \gamma_{k+1}^2 g_{k+1}}{(\lambda_t - \lambda_k)\gamma_k^2 s_k^T g_k + \lambda_k \gamma_t^2 s_t^T g_t - \lambda_t \gamma_{k+1}^2 s_{k+1}^T g_{k+1}}, \tag{4.21}$$

where, as before, $s_i = x_i - x_0$, $i = k, t, k+1$; and where γ_t is defined by (4.19),(4.20), with $k+1$ substituted by t.

Due to formulas (4.19), (4.20), and (4.21), Algorithm A requires only function and gradient values, and can be generalized to more general objective functions.

5. Minimization of general objective functions. We now make the transition from conic objective functions to general nonlinear functions. From now on f denotes a twice continuously differentiable function with gradient g. Algorithm A, with γ and a given by (4.19)–(4.21), can be applied to f, except that some restrictions have to be imposed on the iterates. At the beginning of the algorithm no value for a is given; it is, in fact, not needed, since a is multiplied by $s_0 = 0$. Thus the first search direction for the CG method is $d_0 = -g_0$, and for the BFGS and Limited BFGS methods $d_0 = -B_0^{-1} g_0$. Once d_0 is known, a will be computed by means of (4.21), and the conic model of the objective function will be available, as we shall now discuss.

5.1. Building a conic model. Suppose that x_k is the current iterate. For Algorithm A to be well defined, it is necessary that, for the new iterate x_{k+1}, the value $\rho_{k,k+1}$ be real. In fact one can show that $\rho_{k,k+1}$ will be real if and only if there exists a normal conic function that interpolates $f_k, g_k, f_{k+1}, g_{k+1}$ (cf. Davidon [4]). The algorithm should ensure this condition so that an adequate conic model can be built at each iteration. It is easy to see that this can be accomplished by an appropriate line search. Indeed it is clear that (4.20) will be real if the new iterate x_{k+1} is chosen so that $f_{k+1} < f_k$ and $g_{k+1}^T(x_{k+1} - x_k)$ is sufficiently small. Moreover, if d_k is a descent direction, then the condition $g_{k+1}^T(x_{k+1} - x_k) > 0$ is sufficient to ensure a real value of $\rho_{k,k+1}$.

Similarly, the formula (4.21) for a also requires γ_t, and therefore, we need to make sure that $\rho_{k,t}$ is real. Since the point x_t can lie anywhere along the line joining x_k and x_{k+1}, we will consider choosing it from among the trial points generated by the line search. If none of these trial points results in a real value of $\rho_{k,t}$, then we will need to perform an additional search. Let us assume that the objective function f satisfies the condition: $g(x_k + \lambda d_k)^T d_k \geq 0$, for all λ sufficiently large. Let us also assume that d_k is a descent direction. Then, if $x_t = x_k + \lambda_t d_k$, we see from the formula

$$\rho_{k,t} = + \left[(f_k - f_t)^2 - g_t^T(x_t - x_k) g_k^T(x_t - x_k)\right]^{1/2} \tag{5.1}$$

that $\rho_{k,t}$ will be real, when λ_t is large enough. Thus the additional search consists of trying increasingly large values of λ_t, until a real value of $\rho_{k,t}$ is obtained. We should note that this additional search was never needed in the test runs reported in §7.

While minimizing a conic function the horizon vector a can be computed once and for all, during the first iteration. In the case of a general function f we have several options. The value for a can be recomputed at each step using (4.21), or it can be held fixed for a number of iterations and only recomputed when judged necessary. In the

second case, however, it becomes unclear how the new values of γ should be computed, since we could use either (4.5) or (4.19),(4.20). When the objective function is a conic these two options are equivalent, but in the case of general objective functions they give rise to algorithms with different numerical performances.

We prefer formulas (4.19),(4.20), since they use recent function and gradient values, and provide an up-to-date estimate of γ. We note also that (4.19),(4.20) imply that, if $\rho_{k,k+1}$ is real, if $\gamma_k > 0$, if $f_{k+1} < f_k$, and if d_k is a descent direction, then $\gamma_{k+1} > 0$. Since these conditions will be enforced by the algorithm, the sequence $\{\gamma_k\}$ will be strictly positive. In contrast, the use of (4.5) may result in a negative value of γ_{k+1}.

5.2. Discarding the conic model. The approach we have used consists of creating a conic model of the objective function and generating a step u_k with respect to it. This step u_k is then translated into the search direction d_k, and a line search along d_k ensures that f decreases. The conic model can, however, be inadequate and if the step u_k were to be used, the algorithm could fail to generate a descent direction. Fortunately it is easy to detect these problems before they occur, and to take appropriate action.

Indeed from (4.3) we see that, for a conic function, the ratio

$$\frac{\gamma_k}{1 + \gamma_k a^T u_k} \tag{5.2}$$

is equal to γ_{k+1}. Suppose that at the kth step $\gamma_k > 0$, but the displacement u_k is such that $\gamma_k/(1 + \gamma_k a^T u_k)$ is negative, or is very close to zero. This means that the conic model is generating a step which either crosses the singular hyperplane H or takes us very near H. We take this as an indication that the model is inaccurate, for it predicts that exceedingly large values of the function have been encountered along the current step. We will therefore discard the conic model: the reference point is moved to the current iterate, and the algorithm is started again from the beginning. When this is done we will say that the algorithm has been *reinitialized.*

Note that since $u_k = \sigma_k \delta_k$, we can force (5.2) to be positive by choosing σ_k sufficiently small. We will not do this, but instead choose σ_k consistent with the algorithm we are employing, and use the ratio (5.2) as a tool for measuring the adequacy of our model.

It could also happen that, at an iterate x_k, $\|h_k'\| = 0$ but $\|g_k\| \neq 0$, which means that the conic model is incorrectly identifying x_k as the solution point. Again in this case the algorithm should be reinitialized, after which the gradients of h and f coincide, since $s = 0$.

Therefore the algorithm will enforce the two conditions

$$\frac{\gamma_k}{1 + \gamma_k a^T u_k} > tol_1, \tag{5.3}$$

$$\|h_k'\| > tol_2 \|g_k\|, \tag{5.4}$$

where tol_1 and tol_2 are small positive tolerances. Condition (5.3) is not enforced if the algorithm has just been reinitialized, for in this case the value of a is not yet available. We have observed in our tests that the need to reinitialize arises very infrequently.

5.3. Descent. We have assumed so far that the search directions $\{d_k\}$ generated by the algorithm are descent directions. This can, in fact, be ensured by an appropriate line search, as we shall now discuss.

For the new CG method we have, from (4.15)

$$\begin{aligned} g_k^T d_k &= -\frac{\gamma_k^2 \sigma_k}{1+\gamma_k a^T u_k}\|(I - a s_k^T) g_k\|^2 \\ &\quad + \frac{\sigma_k \beta_k \gamma_k}{(1+\gamma_k a^T u_k)\sigma_{k-1}\gamma_{k-1}} g_k^T d_{k-1}. \end{aligned} \tag{5.5}$$

Since $g_k^T d_{k-1}$ can be made arbitrarily small by the line search, and since the first term on the right-hand side of (5.5) is negative, due to (5.3), (5.4), and (4.6), we obtain the descent condition $g_k^T d_k < 0$.

Similarly, for the new BFGS and Limited BFGS methods we see from (4.16) that, if $B_k > 0$, conditions (5.3) and (5.4) ensure that $g_k^T d_k < 0$. It is well known (cf. Gill, Murray, and Wright [10]) that if B_0 is symmetric and positive definite, then all subsequent matrices $\{B_k\}$ will be symmetric and positive definite if $u_k^T y_k > 0$, for all $k \geq 0$. We now show that this last condition holds whenever $\rho_{k,k+1}$ is real.

It is not difficult to see that γ_{k+1} can also be expressed as

$$\gamma_{k+1} = \frac{f_{k+1} - f_k + \rho_{k,k+1}}{g_{k+1}^T (x_{k+1} - x_k)} \gamma_k. \tag{5.6}$$

From this expression and (4.19) we obtain both

$$\begin{aligned} g_{k+1}^T (x_{k+1} - x_k) &= \frac{\gamma_k}{\gamma_{k+1}} (f_{k+1} - f_k + \rho_{k,k+1}), \\ g_k^T (x_{k+1} - x_k) &= -\frac{\gamma_{k+1}}{\gamma_k} (f_k - f_{k+1} + \rho_{k,k+1}). \end{aligned}$$

Substituting these relations into (4.17) we obtain

$$y_k^T u_k = 2\rho_{k,k+1}. \tag{5.7}$$

Thus by forcing $\{\rho_{k,k+1}\}$ to be real, one ensures that the matrices B_k remain positive definite.

5.4. The line search. We would like the line search to have the following properties: (i) it should decrease the value of the objective function; (ii) the new iterate x_{k+1} should allow us to build a normal conic model, i.e., $\rho_{k,k+1}$ should be real; (iii) it should ensure that all search directions are descent directions.

We now give consideration to each of these requirements. To obtain a sufficient function reduction, and to prevent the steps from being too short, we impose the Wolfe conditions on the steplength λ_k:

$$f(x_k + \lambda_k d_k) - f(x_k) \leq \alpha \lambda_k g_k^T d_k, \tag{5.8}$$

$$|g(x_k + \lambda_k d_k)^T d_k| \leq \eta |g_k^T d_k|, \tag{5.9}$$

where $0 < \alpha < \frac{1}{2}$ and $\alpha < \eta < 1$. In our code we use a line search algorithm developed by Moré that is guaranteed to find a steplength satifying (5.8),(5.9). This line search evaluates both the function and the directional derivative at each trial point.

Let us now consider the second requirement. Even though it is always possible to find a point x_{k+1} with $\rho_{k,k+1}$ real (as discussed in §5.1), a line search based on (5.8),(5.9) is not guaranteed to find such a point. Conditions (5.8),(5.9) ensure that there exists a convex quadratic function whose gradients at x_k and x_{k+1} are g_k and

g_{k+1} , respectively. However, these conditions do not guarantee that there exists a normal conic function whose function and gradient values at x_k and x_{k+1} are (f_k, g_k) and (f_{k+1}, g_{k+1}). Of course, if α and η are small enough, it follows from (4.20) that requirement (ii) will be satisfied. Due to this, we experimented initially with a special line search procedure, in which the values of α and η were adaptively reduced, if necessary. It turns out, however, that in practice this more elaborate procedure was never needed, and that a line search, with termination criteria (5.8),(5.9), and with values $\alpha = 10^{-4}$, and $\eta = 0.1$ (or $\eta = 0.9$), always satisfied requirement (ii).

Let us now consider the last requirement. We noted in §5.3 that the descent condition for the new BFGS and Limited BFGS methods is guaranteed by condition (ii). For the new CG method we could use (5.5) before terminating the line search, to test whether a descent direction will be generated. If not, we could continue the line search, knowing that eventually the descent property will hold. We do not do this, because it requires a substantial amount of work, and because in our tests demanding (5.8),(5.9) was almost always sufficient to ensure descent.

We therefore chose the following line search strategy, which is economical and uses the fully developed line search algorithm of Moré. We find a steplength λ_k satisfying (5.8),(5.9), and set $x_{k+1} = x_k + \lambda_k d_k$. If $\rho_{k,k+1}$ is not real we set $a = 0$ in the algorithm before generating the next search direction. Therefore in the case where a conic model cannot easily be built, we fall back on the algorithms based on quadratics which are known to perform well in practice. (In our tests $\rho_{k,k+1}$ was always positive and there was never a need to set $a = 0$.) For the CG method, if the search direction generated from x_{k+1} is not a descent direction, we used the standard CG iteration based on a quadratic model (this only occurred twice in our tests).

We have experimented with various choices for the parameter σ_k and for the initial trial steplength $\lambda_k^{(0)}$ in the line search. Note that both parameters give us control of the size of the step. After some experimentation, we opted for setting $\sigma_k = 1$, at each step, and to follow Shanno and Phua [18] and let

$$\lambda_k^{(0)} = \lambda_{k-1} \frac{g_{k-1}^T d_{k-1}}{g_k^T d_k}, \tag{5.10}$$

except for restart iterations, in which case we set $\lambda_k^{(0)} = 1$. This strategy was used in the three classes of algorithms.

6. The new algorithms. We now consolidate the discussion of the previous sections into a general algorithm. For the sake of clarity we first describe the generation of the step and the test of its accuracy with respect to the conic model, and then translate the step into the original variables. Of course, in the practical implementation of the algorithm, all formulas are given directly in terms of the nonlinear objective f. Several options have been left open to allow for variations of the algorithm, some of which will be tested in the next section. In particular we do not specify how often a is computed.

Algorithm B.

1. (Initialization)
   The initial point x_0 and the parameters tol_1, tol_2, ϵ_1, and ϵ_2 are given. For the BFGS and Limited BFGS methods an initial matrix $B_0 > 0$ is also given.
   (a) Compute f_0, g_0 and set $k = 0, \gamma_0 = 1, \lambda_0^{(0)} = 1$.
   (b) Compute $\delta_0 = d_0 = -g_0$ for the CG method; $\delta_0 = d_0 = -B_0^{-1} g_0$ for the BFGS

and Limited BFGS methods. Set $h'_0 = g_0$.

2. (Generate a search direction from the conic model)
   If $k = 0$ set $u_0 = \delta_0$ and go to 3.b.
   (a) If (5.4) is violated go to 7.
   (b) For the CG method compute δ_k by means of (4.7),(4.8). For the BFGS and Limited BFGS methods compute δ_k by (4.9).
   (c) Set $\sigma_k = 1$. Compute $u_k = \sigma_k \delta_k$.
   (d) If (5.3) is violated go to 7.
3. (Step in the original space of variables)
   (a) For the CG method compute d_k by means of (4.11) or (4.15), and $\lambda_k^{(0)}$ by means of (5.10). If d_k is not a descent direction set $a = 0, \gamma_k = 1$, and recompute d_k. If d_k is still not a descent direction go to 7. For the BFGS and Limited BFGS methods compute d_k using (4.16).
   (b) Perform a line search, using the initial trial value $\lambda_k^{(0)}$, to obtain λ_k satisfying (5.8),(5.9). Compute $x_{k+1} = x_k + \lambda_k d_k$, $s_{k+1} = x_{k+1} - x_0$, and evaluate f_{k+1} and g_{k+1}.
   (c) If $\rho_{k,k+1}$, given by (4.20), is not real, set $a = 0, \gamma_{k+1} = 1$, and do not perform 3.(d) (i.e., the next step will use a quadratic model). Otherwise, compute γ_{k+1} using (4.19).
   (d) Compute a (only when desired).
   Find $x_t = x_k + \lambda_t d_k$, where λ_t is such that $\rho_{k,t}$ is real (see discussion in §5.1). Compute a by means of (4.21).
   (e) Compute h'_{k+1} using (4.6); and y_k using (3.1).
4. (Termination test)
   If $\|g_{k+1}\| \leq \epsilon_1$ or $\lambda_k \|d_k\| \leq \epsilon_2$ stop.
5. (Matrix update)
   Compute $y_k^T u_k$ using (4.17). For the BFGS method obtain B_{k+1} using (3.7). For the Limited BFGS method obtain B_{k+1} as described in §3.
6. Set $k \leftarrow k + 1$.
   For the BFGS and Limited BFGS methods go to 2. Go to 7 for the CG method if $k = n$; otherwise go to 2.
7. (Restart)
   Let $x_0 = x_k$ and go to 1a.

A careful examination of the formulas reveals that the new CG method can be implemented using six n-vectors, whereas the traditional CG method requires only four n-vectors. If the Fletcher–Reeves formula for β is used, then the new CG method requires five n-vectors, and the traditional CG method requires three n-vectors. The two additional n-vectors are needed to store the horizon vector a and the reference point x_0. Similarly, the new BFGS and Limited BFGS methods require two n-vectors more than the corresponding algorithms based on quadratics. However, for the new BFGS method it is possible to eliminate one of the extra storage locations by moving the reference point at each step, so that it coincides with the current iterate x_k. This was suggested by Davidon [5], who points out that conic termination is preserved, provided a is recomputed at each step.

The algorithms we have just described are a few from among a great number of algorithms with conic termination. Our algorithmic approach uniquely determines Algorithm A. However, as discussed in §4, several formulas for d_k can be derived by

making simplifications due to the properties of conic functions, and several strategies for updating a are possible. When adapting an algorithm with quadratic termination for general nonlinear minimization, several options also arise. For example, three different formulas for β can be used in the CG method: the Hestenes–Stiefel, Polak–Ribière, and Fletcher–Reeves formulas. When conic models are used, the class of possible formulas is much larger.

7. Numerical results. We have experimented with various strategies for recomputing the horizon vector a. For any of the three methods, a can be held fixed for a certain number of iterations, unless (5.3) or (5.4) is violated, or unless the descent property is lost. One can also recompute a at every step, or use an automatic criterion for restarting, if the conic approximation is not judged accurate enough.

In this section we present numerical results obtained with our preferred implementations of the three new classes of algorithms. We incorporate neither automatic criteria for refreshing a nor automatic restarts for the standard CG method, in order to simplify the comparison of the two approaches. We now describe results obtained with the following six algorithms, all of which use the line search of Moré.

ALGORITHM CG:
The Polak–Ribière [15] conjugate gradient algorithm.
ALGORITHM L_BFGS:
The standard limited storage BFGS algorithm, storing four corrections.
ALGORITHM BFGS:
The standard BFGS algorithm.
ALGORITHM C_CG:
The new conic conjugate gradient algorithm, based on (4.11). The horizon vector a was recomputed every n steps.
ALGORITHM L_C_BFGS:
The new limited storage BFGS conic algorithm, storing four corrections, based on (4.16). The horizon vector was computed during the first iteration and kept fixed throughout the run.
ALGORITHM C_BFGS:
The new BFGS conic algorithm, based on (4.16). The horizon vector was computed during the first iteration and kept fixed throughout the run.

For algorithms CG and C_CG the line search enforces at least two interpolations before the final steplength is accepted. The reason for doing this is discussed by Shanno [17].

All the algorithms were coded in double-precision FORTRAN 77. The runs were made on an Alliant FX/8 at the Argonne National Laboratory. Double-precision arithmetic on this machine has a unit roundoff of 10^{-16}. The tolerances that appear in (5.3), (5.4), and Step 4 of Algorithm B were set as follows: $tol_1 = 10^{-6}, tol_2 = 10^{-4}$, and $\epsilon_1 = \epsilon_2 = 10^{-7}$. For all methods, the line search parameters in (5.8),(5.9) were $\alpha = 10^{-4}$ and $\eta = 0.1$.

We have run the six algorithms on the test problems collected by Moré, Garbow, and Hillstrom [13], using their standard starting points, and on a problem described by Cragg and Levy [3]. In the tables that follow, P is the problem number and its name is given in Table 7.1. The problem dimensions appear in the columns headed N, and the number of iterations and function (or gradient) evaluations performed appear

Table 7.1
Set of test problems.

P	Name	P	Name
1	Helical Valley	2	Biggs Exp6
3	Wood	4	Trigonometric
5	Rosenbrock	6	Extended Rosenbrock
7	Extended Powell	8	Brown and Dennis
9	Oren's Power	10	Beale
11	Penalty I	12	Watson
13	Cragg and Levy	14	Variably Dimensioned
15	Broyden Tridiagonal		

Table 7.2
CG *versus* C_CG.

		CG		C_CG		$+ = \log_2(\text{NFc_cg}/\text{NFcg})$						
P	N	IT	NF	IT	NF	F	-.9		-.6		0	.3
1	3	40	102	28	77				+		|	
2	6	79	202	70	176					+	|	
3	4	41	108	29	81				+		|	
4	5	26	60	24	54					+	|	
4	10	38	81	38	81						+	
4	15	42	86	40	88						+	
4	20	62	136	56	121					+	|	
5	2	37	131	35	125					+	|	
6	8	25	81	27	81						+	
6	20	23	76	28	92						|	+
7	8	45	117	52	132						|	+
7	16	37	91	37	94						+	
7	20	62	147	41	93			+			|	
8	4	29	92	29	97						|	+
9	8	13	43	12	42						+	
9	16	18	56	17	54						+	
9	20	19	53	18	51						+	
10	2	7	24	9	25						+	
11	10	6	15	5	16						|	+
11	50	7	16	7	16						+	
12	4	37	88	33	77					+	|	
12	5	56	133	71	164						|	+
13	4	61	170	29	98		+				|	
14	4	F		9	39	+					|	
15	4	F		31	187	+					|	
Wins		8		14								

Table 7.3
L_BFGS *versus* L_C_BFGS.

		L_BFGS		L_C_BFGS		+ = $\log_2$(NFl_c_bfgs/NFl_bfgs)						
P	N	IT	NF	IT	NF	F	-1.5		-.6		0	.6
1	3	28	58	24	58						+	
2	6	23	55	23	53						+	
3	4	41	139	23	61				+		|	
4	5	15	35	18	42						|	+
4	10	34	67	30	62					+	|	
4	15	42	82	37	77					+	|	
4	20	60	119	52	116						+	
5	2	20	58	23	61						+	
6	8	20	58	25	66						|	+
6	20	20	58	25	66						|	+
7	8	38	91	36	91						+	
7	16	38	91	36	91						+	
7	20	41	98	36	91					+	|	
8	4	18	79	21	79						+	
9	4	16	51	7	25				+		|	
9	8	24	75	10	34				+		|	
9	16	33	105	14	44			+			|	
9	20	42	134	16	48			+			|	
10	2	11	25	15	38						|	+
11	10	4	7	4	11						|	+
11	50	5	9	5	12						|	+
12	4	13	30	18	41						|	+
12	6	51	143	52	118					+	|	
13	4	33	86	32	84						+	
14	4	5	16	6	24						|	+
15	4	F		20	105	+					|	
Wins		9		13								

under IT and NF. An F in the iteration column designates failure to converge. The starting matrix B_0 was always chosen to be the identity matrix.

In the right column of Table 7.2 NFcg and NFc_cg denote the number of function evaluations required by the CG and C_CG methods, and we follow Gay [9] and plot the logarithm of the quotient of these two quantitites. We can see from this table that the new conjugate gradient method performs rather well.

The new BFGS and Limited BFGS methods are less successful. After testing various versions of these methods we concluded that overall the new methods required approximately the same number of iterations, but more function evaluations than the corresponding standard methods. In Tables 7.3 and 7.4 we present the results of a special implementation in which the horizon vector was computed only during the first iteration and was kept fixed throughout the run. Thus the information about a was updated only indirectly through γ. We can see from these tables that the new methods, with this implementation, are comparable in efficiency to the corresponding standard methods.

Tables 7.2 to 7.4 present the results obtained using the standard starting points

Table 7.4
BFGS *versus* C_BFGS.

		BFGS		C_BFGS		+ = $\log_2$(NFc_bfgs/NFbfgs)						
P	N	IT	NF	IT	NF	F	−2	-.9	0		.9	
1	3	23	51	24	60				|	+		
2	6	24	53	30	71				|		+	
3	4	37	101	25	63			+	|			
4	5	15	35	17	40				|	+		
4	10	21	45	21	49				+			
4	15	27	57	29	62				+			
4	20	39	80	39	85				+			
5	2	20	58	23	61				+			
6	8	51	133	33	83			+	|			
6	20	91	235	34	83				|			
7	8	28	73	63	141				|			+
7	16	36	93	65	136				|		+	
7	20	40	99	67	141				|		+	
8	4	16	68	16	43			+	|			
9	4	15	48	7	25			+	|			
9	8	27	89	10	34				|			
9	16	39	133	14	44				|			
9	20	78	222	16	48				|			
10	2	10	23	15	38				|		+	
11	10	4	7	4	11				|		+	
11	50	5	9	5	12				|	+		
12	4	13	30	18	41				|		+	
12	6	25	59	34	75				|	+		
13	4	39	93	32	73				|			
14	4	5	16	6	24				|		+	
15	4	F		18	125	+			|			
Wins		16		10								

given by Moré, Garbow, and Hillstrom [13] and Cragg and Levy [3]. We have performed tests using other starting points that are further away from the solution, and have observed that the relative performance of the methods is very similar to the one given in these tables.

7.1. Final remarks. In our view, the new conjugate gradient method is a natural extension of the standard method, and our tests indicate that it is one of the most effective conjugate gradient methods available. Even though the algorithm is complex due to its many safeguards, they are very rarely needed. The new BFGS and Limited BFGS methods do not appear to be superior to the corresponding standard methods, and our formulation of them may not be the best. The algorithmic approach we have used is well suited for deriving conjugate gradient methods, but may not be as useful for quasi-Newton methods, since the matrix update and the computation of a appear to be independent of each other. Therefore it is possible that a different formulation of the conic BFGS method may prove superior to the standard BFGS method.

Acknowledgement. We thank William Davidon for several helpful conversations and suggestions.

REFERENCES

[1] K. A. ARIYAWANSA, *Conic approximations and collinear scalings in algorithms for unconstrained minimization*, Ph.D. thesis, University of Toronto, Canada, 1983.

[2] A. BUCKLEY AND A. LENIR, *QN-like variable storage conjugate gradients*, Math. Programming, 27 (1983), pp. 155-175.

[3] E. E. CRAGG AND A. V. LEVY, *Study on a supermemory gradient method for the minimization of functions*, J. Optim. Theory Appl., 4 (1969), pp. 191-205.

[4] W. C. DAVIDON, *Conic approximations and collinear scalings for optimizers*, SIAM J. Numer. Anal., 17 (1980), pp. 268-281.

[5] ———, *Conjugate directions for conic functions*, in Nonlinear Optimization 1981, M.J.D. Powell, ed., Academic Press, London, 1981.

[6] J. E. DENNIS AND R. B. SCHNABEL, *Numerical methods for unconstrained optimization and nonlinear equations*, Prentice-Hall, NJ, 1983.

[7] R. FLETCHER AND C. M. REEVES, *Function minimization by conjugate gradients*, Computer J., 7 (1964), pp. 149-154.

[8] R. FLETCHER, *Practical methods of optimization*, Vol. 1, Unconstrained Optimization, Wiley, Chichester, 1980.

[9] D. GAY, *Subroutines for unconstrained minimization using a model/ trust-region approach*, ACM Trans. Math. Software, 9 (1983), pp. 503-524.

[10] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, London, 1981.

[11] H. GOURGEON AND J. NOCEDAL, *A Conic Algorithm for Optimization*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 253-267.

[12] L. GRANDINETTI, *Some investigations in a new algorithm for nonlinear optimization based on conic models of the objective function*, J. Optim. Theory Appl., 43 (1984), pp. 1-21.

[13] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM, *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 17-41.

[14] J. NOCEDAL, *Updating quasi-Newton matrices with limited storage*, Math. Comput., 35 (1980), pp. 773-782.

[15] E. POLAK, *Computational Methods in Optimization: A Unified Approach*, Academic Press, London, 1981.

[16] D. F. SHANNO, *Conjugate gradient methods with inexact searches*, Math. Oper. Res., 3 (1978), pp. 244-256.

[17] ——— , *Quadratic termination of conjugate gradient methods*, in Extremal Methods and Systems Analysis, A. V. Fiacco and K. O. Kortanek, eds., Springer Verlag, New York, 1980.

[18] D. F. SHANNO AND K. H. PHUA, *Minimization of unconstrained multivariate functions*, ACM Trans. Math. Software, 6 (1980), pp. 618-622.

[19] D. SORENSEN, *The Q-superlinear convergence of a collinear scaling algorithm for unconstrained optimization*, SIAM J. Numer. Anal., 17 (1980), pp. 88-114.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 18-26, January 1989
© 1989 Society for Industrial and Applied Mathematics

ON PARALLEL JACOBI ORDERINGS*

FRANKLIN T. LUK† AND HAESUN PARK‡

Abstract. This paper presents a systematic description of five well-known Jacobi orderings, using the tool of a caterpillar track. The following surprising result is obtained: under certain assumptions, the orderings are all mathematically equivalent. Three new caterpillar-track orderings are then derived, but they, in turn, are shown to be equivalent to the known schemes.

Key words. parallel Jacobi methods, caterpillar track, caterpillar tractor tread, odd-even ordering, systolic arrays

AMS(MOS) subject classifications. 65F15, 68A10

1. Introduction. The matrix-based approach to signal processing is rapidly gaining popularity (cf. the collection of papers in [3]). Four matrix operations constitute the fundamental computing requirements [14]: matrix-matrix multiplication; the QR decomposition; the singular value decomposition (SVD); and the generalized singular value decomposition (GSVD). Although these operations are costly, VLSI and parallel processing technology will soon enable us to complete the tasks in real time. A signal processing machine, called a Systolic Linear Algebra Parallel Processor, is being built at the Naval Ocean Systems Center in San Diego [5]. The machine will be hardwired to implement parallel Jacobi-like methods [7], [9] for computing the SVD and the GSVD.

The Jacobi approach is to solve a "big" $n \times n$ problem as a sequence of "small" 2×2 subproblems. To define the method, we specify an "ordering" of the $n(n-1)/2$ distinct index pairs to denote the planes in which the transformations are to take place. For serial computing, we refer to such a sequence of $n(n-1)/2$ distinct index pairs as a "sweep." A usual choice is the cyclic by-row ordering ($n=4$):

$$(1,2)(1,3)(1,4)(2,3)(2,4)(3,4).$$

Now, if successive Jacobi transformations involve disjoint planes, e.g., $(1,4)(2,3)$ in the above ordering, then these transformations can occur simultaneously. For an efficient "parallel" implementation, we want to perform as many noninteracting transformations as possible at each time "stage." An example is the following ordering of Brent and Luk [1] ($n=8$):

stage 1 $(1,2)(3,4)(5,6)(7,8)$
stage 2 $(1,4)(2,6)(3,8)(5,7)$
stage 3 $(1,6)(4,8)(2,7)(3,5)$
stage 4 $(1,8)(6,7)(4,5)(2,3)$
stage 5 $(1,7)(8,5)(6,3)(4,2)$
stage 6 $(1,5)(7,3)(8,2)(6,4)$
stage 7 $(1,3)(5,2)(7,4)(8,6)$.

For parallel computing, we refer to the set of disjoint transformations at one stage as a "compound rotation" and define a "sweep" as the minimal sequence of compound

* Received by the editors July 3, 1986; accepted for publication (in revised form) April 4, 1988. This work was supported in part by the Office of Naval Research under contract N00014-85-K-0074.

† School of Electrical Engineering, Cornell University, Ithaca, New York 14853.

‡ Department of Computer Science, Cornell University, Ithaca, New York 14853. Present address, Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455.

rotations that includes all $n(n-1)/2$ distinct pairs. Our goal is to seek a "good" ordering that satisfies these three criteria:

(a) Each sweep is completed in n or fewer stages. Since at most $\lfloor n/2 \rfloor$ transformations can be done in parallel, a minimum of n stages is needed if n is odd ($n-1$ stages if n is even).

(b) Only nearest-neighbor connections are required for data communication.

(c) Data movement between stages is systematic.

However, we know of only one "good" ordering, viz. [1], that completes one sweep in $n-1$ stages when n is even. Many other examples (see, e.g., [12], [15], [16]) require n stages for all values of n.

As a way of finding the relation among "good" orderings, we use the tool of a caterpillar track [12], which neatly illustrates the systematic manner by which the orderings are generated. We discover that the five orderings given in [1], [4], [12], [15], [16] can all be visualized as caterpillar-track orderings. These orderings have been proposed for calculating the symmetric eigenvalue decomposition [1], [4], [12], [13], [16]; the Schur decomposition [6], [15]; the SVD decomposition [1], [2], [9]; the GSVD decomposition [7]; the CS decomposition [11]; and the QR decomposition [8]. We then consider all possible "good" caterpillar-track orderings, and derive three new ones. However, they in turn are shown to be equivalent to the five known schemes.

This paper is organized as follows. Section 2 presents a description of five well-known Jacobi orderings. The *surprising* result that (under certain assumptions) they are all mathematically equivalent is derived in § 3. Three new caterpillar-track orderings are presented and discussed in § 4. The last section contains concluding remarks and Fig. 14 summarizes the equivalent relationships.

2. Various orderings. We present five different orderings of n indices for Jacobi methods. With two exceptions, they all require n stages to complete a sweep. The exceptions are the Brent–Luk ordering for even n (respectively, $n-1$) stages, and the Chen–Irani ordering for odd n (respectively, $n+1$ stages). Hence the latter is not a "good" ordering for odd n. To illustrate each ordering, we show one sweep of the case $n=6$ for even n, and of the case $n=5$ for odd n.

(i) Modi and Pryce [12] describe a "mobile" scheme in terms of a "caterpillar track." There are $2n$ places on the track to store the n indices in every other position. The indices move one step per stage in a counterclockwise direction, as shown in Fig. 1. Each index on the lower track is paired with the closest index to its right on the upper track. The compound rotations for each stage are shown in Fig. 2.

For $n=5$ we get the track as in Fig. 3 and the corresponding compound rotations as in Fig. 4.

(ii) Whiteside et al. [16] present an ordering in terms of a "caterpillar tractor tread." In this ordering, the n indices are stored on a caterpillar tractor tread of length n that rolls along a roadbed of processors. Each processor has access to the pair of

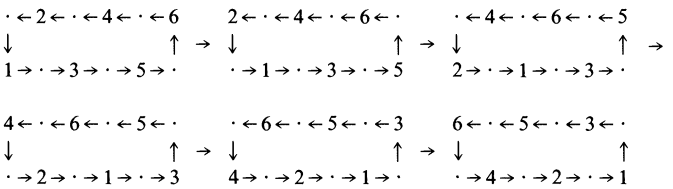


FIG. 1. *Caterpillar-track for* $n=6$.

1. (1, 2)(3, 4)(5, 6)
2. 2(1, 4)(3, 6)5
3. (2, 4)(1, 6)(3, 5)
4. 4(2, 6)(1, 5)3
5. (4, 6)(2, 5)(1, 3)
6. 6(4, 5)(2, 3)1

FIG. 2. *Caterpillar-track ordering for* $n = 6$.

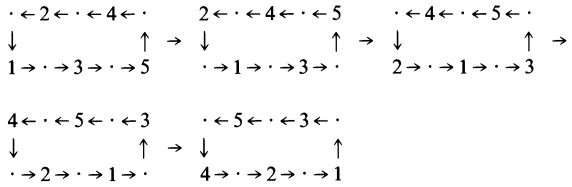


FIG. 3. *Caterpillar track for* $n = 5$.

1. (1, 2)(3, 4)5
2. 2(1, 4)(3, 5)
3. (2, 4)(1, 5)3
4. 4(2, 5)(1, 3)
5. (4, 5)(2, 3)1

FIG. 4. *Caterpillar-track ordering for* $n = 5$.

indices directly above it, on the lower and upper parts of the tread. The upper tread of the tractor advances one step per time stage and the lower tread stays stationary on the ground. Figure 5 gives an example for $n = 6$. Corresponding to the tread, the compound rotations for each time stage are exactly as those given in Fig. 2.

For n odd, the tread proceeds as in Fig. 6. Again the compound rotations are the same as those in Fig. 4.

(iii) Stewart [15] introduces the "odd-even" ordering. The paired indices are exchanged for the next stage, and pairings start with either the first or the second index at alternate time stages. So the ordering proceeds in exactly the same fashion as the caterpillar-track ordering portrayed in Figs. 2 and 4.

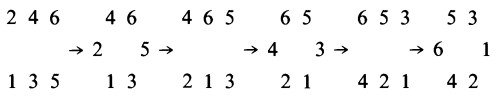


FIG. 5. *Caterpillar tractor tread for* $n = 6$.

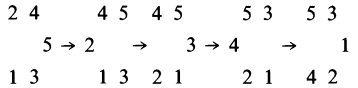


FIG. 6. *Caterpillar tractor tread for* $n = 5$.

(iv) Brent and Luk [1] present an ordering that is often used in round-robin chess tournaments. For n even, this ordering requires only $n-1$ stages to complete a sweep. See Fig. 7 for an example and Fig. 8 for the corresponding compound rotations.

Our presentation here looks different from the original description in [1] that is illustrated in Fig. 9. However, one version is just the other applied to the inverted list of indices $(n, n-1, \cdots, 2, 1)$. For n odd, we simply add a dummy index ϕ to return to the even case. See Figs. 10 and 11 for illustrations.

(v) The final scheme is due to Chen and Irani [4]. They consider only the case of an even n. Like the "odd-even" ordering, pairings of indices start with the first or the second index at alternate time stages. However, the indices are exchanged after every *two* stages, according to the permutation

$$\sigma = (2, 4, 1, 6, 3, 8, 5, \cdots, n-5, n, n-3, n-1),$$

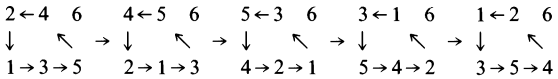


FIG. 7. *Round-robin exchanges for* $n = 6$.

1. $(1, 2)(3, 4)(5, 6)$
2. $(2, 4)(1, 5)(3, 6)$
3. $(4, 5)(2, 3)(1, 6)$
4. $(5, 3)(4, 1)(2, 6)$
5. $(3, 1)(5, 2)(4, 6)$

FIG. 8. *Brent–Luk ordering for* $n = 6$.

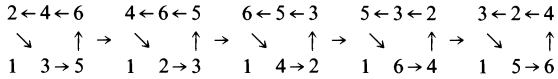


FIG. 9. *Original round-robin exchanges for* $n = 6$.

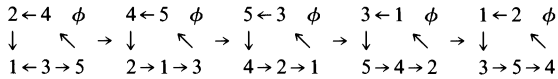


FIG. 10. *Round-robin exchanges for* $n = 5$.

1. $(1, 2)(3, 4)5$
2. $(2, 4)(1, 5)3$
3. $(4, 5)(2, 3)1$
4. $(5, 3)(4, 1)2$
5. $(3, 1)(5, 2)4$

FIG. 11. *Brent–Luk ordering for* $n = 5$.

whose action is better explained by the following figure:

$$\begin{array}{l} 2 \leftarrow 4 \leftarrow 6 \leftarrow \cdots \leftarrow n \\ \downarrow \qquad\qquad\qquad\qquad \uparrow \\ 1 \rightarrow 3 \rightarrow 5 \rightarrow \cdots \rightarrow n-1 \end{array}$$

Figure 12 presents the compound rotations. We may extend the ordering to the odd n case by appending a dummy index ϕ. See Fig. 13.

3. Equivalence. From the descriptions and examples in § 2, we see that the five orderings are intimately related. This section is devoted to the proof of their mathematical equivalence. For any n, let us choose the same initial stage for each ordering:

$$(1,2)(3,4)(5,6)\cdots(n-1,n) \qquad \text{if } n \text{ is even,}$$

$$(1,2)(3,4)(5,6)\cdots(n-2,n-1)n \qquad \text{if } n \text{ is odd.}$$

DEFINITION. Consider the two sequences S_1 and S_2 of compound rotations generated in one sweep of the orderings O_1 and O_2. We say that O_1 and O_2 are "identical" if $S_1 = S_2$, and that they are "stage-equivalent" if S_1 is a permutation of S_2. Note that two stage-equivalent orderings may not share the same numerical properties.

Since the caterpillar track makes it easy to visualize the orderings geometrically, we centralize our discussion on this theoretical machine. Modi and Pryce [12] have given the following result.

LEMMA 1. *The caterpillar track generates the same compound rotation and completes one sweep in every n stages.* □

Consequently, when proving stage-equivalence of some ordering to the caterpillar-track ordering, we need only consider any n consecutive stages of the track. We say that two stages are "equal" if they correspond to the same compound rotation.

LEMMA 2. *The caterpillar-track ordering is identical to the odd-even ordering.*

Proof. Suppose that n is even (the proof is similar for odd n). We use induction on the stage number. By assumption, the first stages are identical. Suppose that we get

1. (1, 2)(3, 4)(5, 6)
2. 1(2, 3)(4, 5)6
3. (2, 4)(1, 6)(3, 5)
4. 2(4, 1)(6, 3)5
5. (4, 6)(2, 5)(1, 3)
6. 4(6, 2)(5, 1)3

FIG. 12. *Chen-Irani ordering for* $n = 6$.

1. $(1, 2)(3, 4)(5, \phi)$
2. $1(2, 3)(4, 5)\phi$
3. $(2, 4)(1, \phi)(3, 5)$
4. $2(4, 1)(\phi, 3)5$
5. $(4, \phi)(2, 5)(1, 3)$
6. $4(\phi, 2)(5, 1)3$

FIG. 13. *Chen-Irani ordering for* $n = 5$.

the same $(2k+1)$st stages:

$$\begin{array}{l} \cdot \leftarrow a_2 \leftarrow \cdot \leftarrow a_4 \leftarrow \cdots \leftarrow \cdot \leftarrow a_n \\ \downarrow \qquad\qquad\qquad\qquad\qquad\qquad \uparrow \\ a_1 \rightarrow \cdot \rightarrow a_3 \rightarrow \cdot \rightarrow \cdots \rightarrow a_{n-1} \rightarrow \cdot \end{array} \qquad (a_1, a_2)(a_3, a_4) \cdots (a_{n-1}, a_n).$$

Then the $(2k+2)$nd and $(2k+3)$rd stages are the following:

$$\begin{array}{l} a_2 \leftarrow \cdot \leftarrow a_4 \leftarrow \cdot \leftarrow \cdots \leftarrow a_n \leftarrow . \\ \downarrow \qquad\qquad\qquad\qquad\qquad\qquad \uparrow \\ \cdot \rightarrow a_1 \rightarrow \cdot \rightarrow a_3 \rightarrow \cdots \rightarrow \cdot \rightarrow a_{n-1} \end{array} \qquad a_2(a_1, a_4) \cdots (a_{n-3}, a_n) a_{n-1}, \text{ and}$$

$$\begin{array}{l} \cdot \leftarrow a_4 \leftarrow \cdot \leftarrow a_6 \leftarrow \cdots \leftarrow \cdot \leftarrow a_{n-1} \\ \downarrow \qquad\qquad\qquad\qquad\qquad\qquad \uparrow \\ a_2 \rightarrow \cdot \rightarrow a_1 \rightarrow \cdot \rightarrow \cdots \rightarrow a_{n-3} \rightarrow . \end{array} \qquad (a_2, a_4)(a_1, a_6) \cdots (a_{n-3}, a_{n-1}). \qquad \square$$

LEMMA 3. *The caterpillar tractor tread ordering is identical to the odd-even ordering.*

Proof. The proof is similar to that of Lemma 2. $\square$

We now generalize the concept of a caterpillar track.

DEFINITION. An (o, e) caterpillar track is a caterpillar track that moves o steps at every odd time stage and e steps at every even time stage, both in the counterclockwise direction.

Thus, the original caterpillar track is a $(1, 1)$ caterpillar track. It can be shown that if $-2 \leqq o$, $e \leqq 3$, then we can implement the (o, e) caterpillar-track ordering on a multiprocessor array with only nearest-neighbor connections. In this section we discuss two specific choices: $(o, e) = (2, 2)$, and $(-1, 3)$. What follow are two obvious properties.

FACT 1. The $(2, 2)$ caterpillar track repeats itself in every n stages, and its ith stage equals the $(2i-1)$st stage of the $(1, 1)$ caterpillar track, for $i = 1, 2, \cdots, n$.

FACT 2. Consider the $(-1, 3)$ caterpillar track. Its ith stage equals the ith stage of the $(1, 1)$ caterpillar track when i is odd, and equals the $(i-2)$nd stage of the $(1, 1)$ track when i is even.

Note by Lemma 1 that stages $n+2$, $n+4, \cdots, 2n-1$ of the $(1, 1)$ track equal stages $2, 4, \cdots, n-1$ of the $(1, 1)$ track, respectively. If n and i are both even, then the index $i-2+n$ is even and stage i of the $(-1, 3)$ track equals stage $i-2+n$ of the $(1, 1)$ track. From these observations and Facts 1 and 2, we get the following two lemmas.

LEMMA 4. *For any odd n, the* $(2, 2)$ *caterpillar-track ordering is stage-equivalent to the* $(1, 1)$ *caterpillar-track ordering.* $\square$

LEMMA 5. *For any even n, the* $(-1, 3)$ *caterpillar-track ordering is stage-equivalent to the* $(1, 1)$ *caterpillar-track ordering.* $\square$

Let us relate the $(2, 2)$ and $(-1, 3)$ tracks to the two remaining orderings.

LEMMA 6. *For any odd n, the* $(2, 2)$ *caterpillar track generates the same ordering as the Brent–Luk scheme.*

Proof. We use induction for the proof. Stages 1 are the same by assumption. Suppose that the kth stages are equal, i.e.,

$$\begin{array}{l} \cdot \leftarrow a_2 \leftarrow \cdot \leftarrow a_4 \leftarrow \cdots \leftarrow a_{n-1} \leftarrow \cdot \\ \downarrow \qquad\qquad\qquad\qquad \uparrow \\ a_1 \rightarrow \cdot \rightarrow a_3 \rightarrow \cdot \rightarrow \cdots \rightarrow \cdot \rightarrow a_n \end{array} \qquad \begin{array}{l} a_2 \leftarrow a_4 \leftarrow \cdots \leftarrow a_{n-1} \quad \phi \\ \downarrow \qquad\qquad\qquad \nwarrow \\ a_1 \rightarrow a_3 \rightarrow \cdots \rightarrow a_{n-2} \rightarrow a_n \end{array} .$$

Then the $(k+1)$st stages become the following:

$$\begin{array}{l} \cdot \leftarrow a_4 \leftarrow \cdot \leftarrow a_6 \leftarrow \cdots \leftarrow a_n \leftarrow \cdot \\ \downarrow \qquad\qquad\qquad\qquad \uparrow \\ a_2 \rightarrow \cdot \rightarrow a_1 \rightarrow \cdot \rightarrow \cdots \rightarrow \cdot \rightarrow a_{n-2} \end{array} \qquad \begin{array}{l} a_4 \leftarrow a_6 \leftarrow \cdots \leftarrow a_n \quad \phi \\ \downarrow \qquad\qquad\qquad \nwarrow \\ a_2 \rightarrow a_1 \rightarrow \cdots \rightarrow a_{n-4} \rightarrow a_{n-2} \end{array} . \qquad \square$$

LEMMA 7. *For any even n, the* $(-1, 3)$ *caterpillar track generates the same ordering as the Chen–Irani scheme.*

Proof. We use induction on the stage number. Stages 1 are equal by assumption. Assume that the $(2k+1)$st stages are equal, i.e.,

$$\begin{array}{l} \cdot \leftarrow a_2 \leftarrow \cdot \leftarrow a_4 \leftarrow \cdots \leftarrow \cdot \leftarrow a_n \\ \downarrow \qquad\qquad\qquad\qquad\qquad\qquad \uparrow \\ a_1 \rightarrow \cdot \rightarrow a_3 \rightarrow \cdot \rightarrow \cdots \rightarrow a_{n-1} \rightarrow \cdot \end{array} \qquad (a_1, a_2)(a_3, a_4) \cdots (a_{n-1}, a_n).$$

Then the $(2k+2)$nd and $(2k+3)$rd stages of the two schemes are the following:

$$\begin{array}{l} a_1 \leftarrow \cdot \leftarrow a_2 \leftarrow \cdot \leftarrow \cdots \leftarrow a_{n-2} \leftarrow \cdot \\ \downarrow \qquad\qquad\qquad\qquad\qquad\qquad \uparrow \\ \cdot \rightarrow a_3 \rightarrow \cdot \rightarrow a_5 \rightarrow \cdots \rightarrow \cdot \rightarrow a_n \\ \qquad\quad \downarrow \qquad\qquad\qquad\qquad\qquad\quad \downarrow \\ \cdot \leftarrow a_4 \leftarrow \cdot \leftarrow a_6 \leftarrow \cdots \leftarrow \cdot \leftarrow a_{n-1} \\ \downarrow \qquad\qquad\qquad\qquad\qquad\qquad \uparrow \\ a_2 \rightarrow \cdot \rightarrow a_1 \rightarrow \cdot \rightarrow \cdots \rightarrow a_{n-3} \rightarrow \end{array} \qquad \begin{array}{l} a_1(a_2, a_3) \cdots (a_{n-2}, a_{n-1}) a_n \\ \\ \\ (a_2, a_4)(a_1, a_6) \cdots (a_{n-3}, a_{n-1}). \end{array}$$ □

We summarize our results in a theorem.

THEOREM 1. *The* $(1, 1)$ *caterpillar track, the caterpillar tractor tread, and the odd-even orderings are all identical. For n odd, the three orderings are stage-equivalent to the Brent-Luk ordering, while for n even, they are stage-equivalent to the Chen-Irani ordering.* □

4. New orderings. First, we give the conditions under which an (o, e) caterpillar track generates a Jacobi ordering.

THEOREM 2. *In order that an* (o, e) *caterpillar track generate all the distinct index pairs, the three parameters must satisfy either one of the following two relations*:

$$\gcd(n, o+e) = 1, \quad \textit{or}$$

$$\gcd(n, o+e) = 2 \quad \textit{and } o \textit{ is odd.}$$

The track then completes a sweep in at most $2n$ *stages.*

Proof. We assume that the first stages are the same. Then stage $(2i+1)$ of the (o, e) track equals stage $1+(o+e)i$ of the regular track, for $i = 0, 1, \cdots, n-1$. But if n and $o+e$ are relatively prime, then the set $S = \{(o+e)i \bmod n, \text{ for } i = 1, 2, \cdots, n-1\}$ equals the set $\{1, 2, \cdots, n-1\}$. Now if $\gcd(n, o+e) = 2$, then the set S equals only $\{2, 4, \cdots, n-2\}$. Thus, we need an odd o so that we can use the set of even stages of the (o, e) track. If $\gcd(n, o+e) \geqq 3$, then this patchwork fails and not all index pairs can be generated. □

COROLLARY. *The* (o, o) *and* $(1, 1)$ *caterpillar-track orderings are stage-equivalent if*

$$\gcd(n, o) = 1.$$ □

We now consider only those orderings that satisfy $-2 \leqq o,\ e \leqq 3$, a condition imposed by the nearest-neighbor communication restriction. We are particularly interested in caterpillar tracks that complete one sweep in n stages. Clearly, there is no need to discuss the cases where $o = 0$, $e = 0$, or $o + e \leqq 0$.

(i) $o + e = 1, 3, 5$. According to Theorem 2, the ordering generates all possible pairs in at most $2n$ stages so long as $\gcd(n, o+e) = 1$. The caterpillar track finishes one sweep in n stages when $n = 2|o|$. For example, the $(2, -1)$ caterpillar track requires more than n stages when $n > 4$ because its fifth stage equals its second stage.

(ii) $o+e=2$. The three possibilities are: $(1,1)$, $(-1,3)$, and $(3,-1)$. The first two are described in detail in the previous sections, and the third ordering is obviously stage-equivalent to the second when n is even.

(iii) $o+e=4$. The three choices are: $(2,2)$, $(1,3)$, and $(3,1)$. The first choice is the Brent-Luk ordering, and the other two satisfy the conditions of Theorem 2 for any n that is not a multiple of 4.

(iv) $o+e=6$. From the corollary, the $(3,3)$ caterpillar track completes one sweep in n stages for any n that is not a multiple of 3.

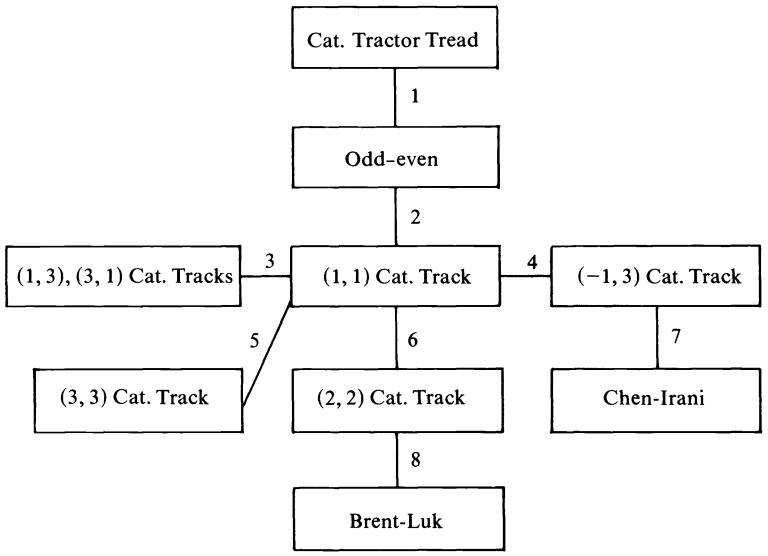
We present the equivalence properties of the three new orderings in the next two lemmas. Their proofs are like those in the previous section.

LEMMA 8. *The* $(1,3)$, $(3,1)$, *and* $(1,1)$ *caterpillar-track orderings are stage-equivalent if*

$$n \bmod 4 = 2. \qquad \square$$

LEMMA 9. *The* $(3,3)$ *and* $(1,1)$ *caterpillar-track orderings are stage-equivalent if* n *is not a multiple of* 3. $\square$

5. Concluding remarks. In this paper, we study five well-known parallel Jacobi orderings and derive three new ones via the tool of a caterpillar track. We show that all eight orderings are either identical or stage-equivalent, and that they constitute all possible "good" orderings that can be generated by a caterpillar track. Figure 14 summarizes the equivalence relations from which the convergence of various Jacobi SVD methods can be derived [10].



1. Identical by Lemma 3.
2. Identical by Lemma 2.
3. Stage-equivalent by Lemma 8 when $n \bmod 4 = 2$.
4. Stage-equivalent by Lemma 5 when n is even.
5. Stage-equivalent by Lemma 9 when $n \bmod 3 \neq 0$.
6. Stage-equivalent by Lemma 4 when n is odd.
7. Identical by Lemma 7 when n is even.
8. Identical by Lemma 6 when n is odd.

FIG. 14. *Relationships among the orderings.*

REFERENCES

[1] 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.

[2] 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 (1985), pp. 242–270.

[3] K. BROMLEY AND J. M. SPEISER, *Signal processing algorithms, architectures, and applications*, in Tutorial 31, SPIE 27th Annual International Technical Symposium, San Diego, CA, 1983.

[4] K-W. CHEN AND K. B. IRANI, *A Jacobi algorithm and its implementation on parallel computers*, in Proc. 18th Annual Allerton Conference on Communication, Control, and Computing, Monticello, IL, 1980, pp. 564–573.

[5] B. L. DRAKE, F. T. LUK, J. M. SPEISER, AND J. J. SYMANSKI, SLAPP: *A Systolic Linear Algebra Parallel Processor*, IEEE Computer, 20 (1987), pp. 45–49.

[6] P. J. EBERLEIN, *On the Schur decomposition of a matrix for parallel computation*, IEEE Trans. Comput., 36 (1987), pp. 167–174.

[7] F. T. LUK, *A parallel algorithm for computing the generalized singular value decomposition*, J. Parallel Distrib. Comput., 2 (1985), pp. 250–260.

[8] ———, *A rotation algorithm for computing the* QR*-decomposition*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 452–459.

[9] ———, *A triangular processor array for computing singular values*, J. Linear Algebra Appl., 77 (1986), pp. 259–273.

[10] F. T. LUK AND H. PARK, *A proof of convergence for two parallel Jacobi* SVD *algorithms*, IEEE Trans. Comput., 38 (1989), to appear.

[11] F. T. LUK AND S. QIAO, *Computing the* CS*-decomposition on systolic arrays*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1121–1125.

[12] J. J. MODI AND J. D. PRYCE, *Efficient implementation of Jacobi's diagonalization method on the* DAP, Numer. Math., 46 (1985), pp. 443–454.

[13] A. H. SAMEH, *On Jacobi and Jacobi-like algorithms for a parallel computer*, Math. Comput., 25 (1971), pp. 579–590.

[14] J. M. SPEISER AND H. J. WHITEHOUSE, *A review of signal processing with systolic arrays*, in Real Time Signal Processing VI, Proc. SPIE 431, 1983, pp. 2–6.

[15] 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.

[16] R. A. WHITESIDE, N. S. OSTLUND, AND P. G. HIBBARD, *A parallel Jacobi diagonalization algorithm for a loop multiple processor system*, IEEE Trans. Comput., 33 (1984), pp. 409–413.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 27-35, January 1989
© 1989 Society for Industrial and Applied Mathematics
003

VECTORIZATION OF LINEAR RECURRENCE RELATIONS*

H. A. VAN DER VORST† AND K. DEKKER†

Abstract. In this paper the usefulness of the partition method [S. C. Chen, D. J. Kuck, and A. H. Sameh, *ACM Trans. Math. Software*, 4 (1978), pp. 270-277], [H. H. Wang, *ACM Trans. Math. Software*, 7 (1981), pp. 170-183] for solving linear bidiagonal systems on vector computers is studied. It is shown that the method is numerically stable for an interesting class of problems. For different vector computers suitable variants of the method are computed and upper bounds for their performance are presented. Also reported are some actually observed performances in a FORTRAN environment.

Key words. bidiagonal linear systems, recurrence relations, vector computers, partition method

AMS(MOS) subject classifications. 65F05, 65F10, 65F50, 65N20, 65Q05, 65W05

1. Introduction. Many iterative solution methods for classes of large sparse linear systems involve the solution of linear recurrence relations of large order. For vector computers, as well as for parallel computers, this has been recognized as a severe bottleneck with respect to the speed of computation.

Several algorithms have been proposed that lead, at least partially, to vectorizable code, e.g., recursive doubling [3], [8] and cyclic reduction [5], [6]. Cyclic reduction is often considered to be a useful technique for vector computers, but in our experience the gain in computational speed on computers such as the CRAY-1 and the CYBER 205 is rather limited [7], [9].

Chen, Kuck, and Sameh [1] have proposed a parallel solver for banded triangular systems. In [11] Wang follows essentially the same approach, to be called the partition method, for the solution of tridiagonal linear systems. These methods simplify even further when applied to bidiagonal linear systems and we will study the partition method for solving such systems on vector computers. For a discussion on the actual behaviour of the partition method on certain parallel computers, see, e.g., [9].

First we will show that the method is numerically stable for an interesting class of problems. Then we will analyse the potential performance of suitable variants of the algorithm for different vector computers and we will present upper bounds for the performance. In order to get some feeling for the actually achievable speeds, we also report on some experiments with FORTRAN codes.

2. The partition algorithm.

2.1. The basic algorithm. The partition algorithm has been described for general banded triangular systems in [1] and for tridiagonal systems in [11]. In order to facilitate the presentation of our error analysis and the vectorizable variants of the algorithm, we give a brief description of the method when applied to the computation of the values x_i, defined by

$$\begin{aligned} &x_1 = y_1, \\ &x_i = y_i + a_i x_{i-1}, \qquad i = 2, 3, \cdots, n. \end{aligned} \tag{2.1}$$

Algorithm (2.1) will be referred to as the standard algorithm. The problem is written

* Received by the editors March 23, 1987; accepted for publication (in revised form) April 19, 1988.
† Faculty of Technical Mathematics and Informatics, Delft University of Technology, 2600 AJ Delft, the Netherlands.

in matrix notation as

$$Bx = y, \tag{2.2}$$

in which B is a lower bidiagonal matrix.

For simplicity we assume that n can be factored as $n = mk$ and we partition B in an $m \times m$ block matrix in which the blocks are $k \times k$ matrices:

$$B = \begin{bmatrix} 1 & & & \\ -a_2 & 1 & & 0 \\ & -a_3 & 1 & \\ & & -a_4 & 1 \ddots \\ & 0 & & \ddots \ddots \\ & & & -a_n \quad 1 \end{bmatrix} = \begin{bmatrix} D_1 & & 0 \\ -A_2 & D_2 & \ddots \\ & \ddots & \ddots \\ & 0 & -A_m \quad D_m \end{bmatrix}. \tag{2.3}$$

Note that the A_i have only one nonzero element in their $(1, k)$ positions. The vectors x and y are partitioned accordingly in segments $\bar{x}_i$ and $\bar{y}_i$ of length k (x_i denotes the ith element of x, $\bar{x}_i$ the ith segment). We define matrices D, A, and V as follows:

$$D = \begin{bmatrix} D_1 & & 0 \\ & D_2 & \\ & \ddots & \\ 0 & & D_m \end{bmatrix}, \quad A \equiv D - B, \quad V \equiv D^{-1}A. \tag{2.4}$$

Then (2.2) can be written as

$$\mathrm{D}(I - V)x = y. \tag{2.5}$$

The first step in the partition algorithm is to ignore V and to solve h from

$$Dh = y. \tag{2.6}$$

The matrix V can be viewed as the fill-in that occurs when carrying out Gaussian elimination on B only at the elements corresponding to D. This fill-in is eliminated in the second (and last) step, when solving x from

$$(I - V)x = h. \tag{2.7}$$

We readily verify that $V_i = D_i^{-1}A_i$ is a matrix having only nonzero elements in its kth column. This property is exploited in the construction of suitable vectorizable algorithms. The first step of the algorithm is vectorizable if we compute the jth components, for a fixed value of j, of all segments $\bar{h}_i$ simultaneously from the previously obtained $(j-1)$th components.

2.2. Error analysis. We denote the variables computed in finite precision floating-point arithmetic, with relative machine precision u, by a suffix $\hat{}$. A straightforward analysis as in [4], [12] shows that the actually computed vector $\hat{h}$ satisfies the system

$$\tilde{D}\hat{h} = \tilde{y}, \tag{2.8}$$

in which

$$\tilde{D} = D + \Delta D, \qquad \Delta D = (\xi^+ + \xi^* + \xi^+\xi^*)(D - I),$$

$$\tilde{y} = y + \Delta y, \qquad \Delta y = \xi^+ y,$$

where $\xi^+ = \text{diag}\,(\xi_1^+, \cdots, \xi_n^+)$, $\xi^* = \text{diag}\,(\xi_1^*, \cdots, \xi_n^*)$, $|\xi_j^+|, |\xi_j^*| \leqq u$. By following the computation of the elements of the nonzero column of each V_i we obtain

$$\hat{V}_i = (I + N_i) V_i, \tag{2.9}$$

in which N_i is a diagonal matrix defined by

$$I + N_i = \begin{bmatrix} 1 & & & 0 \\ & 1+\eta_{2i}^* & & \\ & & (1+\eta_{2i}^*)(1+\eta_{3i}^*) & \\ & & & \cdots\cdots\cdots \prod_{j=2}^{k} (1+\eta_{ji}^*) \\ & 0 & & \end{bmatrix}, \qquad |\eta_{ji}^*| \leqq u. \tag{2.10}$$

The complete fill-in may then be written as

$$\hat{V} = (I + N) V \quad \text{with } I + N = \text{diag}\,(I, I + N_2, \cdots, I + N_m). \tag{2.11}$$

In the same spirit we find for the last step of the partition algorithm:

$$\hat{\hat{x}}_i = (I + \Gamma_i^+)(\hat{\hat{h}}_i + (I + \Gamma_i^*)\hat{V}_i \hat{\hat{x}}_{i-1}), \tag{2.12}$$

in which the elements of the diagonal matrices Γ_i^+ and Γ_i^* in absolute value are smaller than u. Relation (2.12) can be rewritten for the whole system as

$$(I - (I + \Gamma^+)(I + \Gamma^*)\hat{V})\hat{x} = (I + \Gamma^+)\hat{h}, \tag{2.13}$$

which gives

$$\hat{h} = ((I + \Gamma^+)^{-1} - (I + \Gamma^*)\hat{V})\hat{x} \equiv (I - V + \Delta V)\hat{x}. \tag{2.14}$$

Combining of (2.8) and (2.14) and writing $\hat{x} = x + \Delta x$, we obtain

$$(D + \Delta D)\{I - V + \Delta V\}(x + \Delta x) = y + \Delta y.$$

Using (2.5), defining $\Delta B = D \Delta V$, we obtain

$$\Delta x = (B + \Delta B)^{-1}[(I + \Delta D D^{-1})^{-1}(\xi^+ + (\xi^* + \xi^+ \xi^*)(I - D))(I - V) - D \Delta V]x. \tag{2.15}$$

When ignoring second-order terms in u, we have from (2.14) an expression for ΔV:

$$\begin{aligned} \Delta V &= (I + \Gamma^+)^{-1} + (I + \Gamma^*)\hat{V} - (I - V) \\ &\approx (I - \Gamma^+) - (I + \Gamma^*)(I + N) V - (I - V), \end{aligned} \tag{2.16}$$

and hence

$$D \Delta V \approx -D\Gamma^+ - D\{N + \Gamma^*(I + N)\} V. \tag{2.17}$$

Since

$$\begin{aligned} N_i + \Gamma_i^*(I + N_i) &= \text{diag}\left(\gamma_{1i}^*, \gamma_{2i}^* + \eta_{2i}^*, \cdots, \gamma_{ki}^* + \sum_{j=2}^{k} \eta_{ji}^*\right) \\ &\equiv M_i = \text{diag}\,(m_{1i}, m_{2i}, \cdots, m_{ki}), \end{aligned}$$

it follows that

$$D\{N + \Gamma^*(I + N)\} V = DMV, \tag{2.18}$$

with

$$D_i M_i V_i = \begin{bmatrix} & & a_{(i-1)k+1} m_{1i} \\ & & a_{(i-1)k+1} a_{(i-1)k+2}(m_{2i} - m_{1i}) \\ 0, 0, \cdots, 0, & & \vdots \\ & & \prod_{j=1}^{k} a_{(i-1)k+j}(m_{ki} - m_{k-1,i}) \end{bmatrix}.$$

Note that $m_{ji} - m_{j-1,i} = \gamma^*_{ji} - \gamma^*_{(j-1),i} - \eta^*_{ji}$ for $j > 1$, $m_{1i} = \gamma^*_{1i}$.

Now we take for the norm $\|\cdot\|$ the infinite norm $\|\cdot\|_\infty$. For $|a_i| \leqq 1$ we have that $\|\Delta B\| \|B^{-1}\| \ll 1$ and $\|\Delta D D^{-1}\| \ll 1$. Hence (2.15), combined with (2.17) and (2.18), simplifies to

$$\|\Delta x\| \lesssim \|B^{-1}\|\{(u + u\|I - D\|)\|I - V\| + u\|D\| + 3u\|V\|\}\|x\|. \tag{2.19}$$

As a model problem we consider the situation $|a_i| = a \leqq 1$. Then we can easily verify that

$$\|I - D\| = a, \quad \|I - V\| = 1 + a, \quad \|V\| = a, \quad \|D\| = 1 + a,$$

which leads to

$$\|\Delta x\| \lesssim u\|B^{-1}\|(a^2 + 6a + 2)\|x\|. \tag{2.20}$$

For the standard recursion we obtain, in finite precision arithmetic, an error Δx_R that satisfies

$$\|\Delta x_R\| \lesssim u\|B\| \|B^{-1}\| \|x\| = u\|B^{-1}\|(a + 1)\|x\|, \tag{2.21}$$

since $\|B\| = 1 + a$ for the model problem.

Hence the partition algorithm leads to an upper bound, which is $(a^2 + 6a + 2)/(a + 1)$ larger than the upper bound for the standard recursion. For many discretized PDEs we encounter the situation that $|a_i|$ is a good deal smaller than 1. Obviously the standard recursion leads to an error which is bounded by the theoretical lowest upper bound, in terms of the condition number, that we may expect according to (2.21). Therefore we conclude that for these situations the partition algorithm is also a very stable algorithm. Since our analysis is a typical worst-case analysis, we also ran numerical experiments to compare the methods to actual situations. Here we present results for systems of different order (n) and with a constant subdiagonal (α). Note that the condition number of such a system is bounded by $\min(2n, 2/(1 - |\alpha|))$, for $|\alpha| < 1$.

We took care that rounding errors were likely to occur, e.g., by avoiding integer-valued right-hand sides y or solutions x, and by selecting different sign patterns for y as well as for x. Altogether we generated 80 different systems for each pair (n, α). From a solution computed in extended precision we derived the real value of $\|\Delta x\|_\infty / \|x\|_\infty$. For each pair (α, n) we determined $U_p = \max_{80 \text{ systems}} \|\Delta x\|_\infty / \|x\|_\infty$, for Δx and x corresponding to the partition method, and similarly U_s for the standard recursion method. We expect the values of U_p and U_s to give a more realistic impression of how rounding errors actually work out.

In Table 1 we have collected the values of the ratios U_p / U_s for various pairs (α, n). Much to our surprise we observed that this ratio is close to 1, with possibly a slight advantage for the partition method.

3. Implementation and performance aspects.

3.1. Implementation of the partition algorithm. In this section we present three different implementations for the partition algorithm and we comment on each version in view of certain aspects of vector architectures. It should be noted that all three implementations lead to exactly the same solution x, i.e., they have the same round-off pattern.

TABLE 1

The ratios of the maximum relative errors for the partition method and the standard recursion (each maximum has been taken over 80 different systems).

α	$n=100$, $k=5$, $m=20$	$n=1{,}000$, $k=20$, $m=50$	$n=10{,}000$, $k=50$, $m=200$	$n=50{,}000$, $k=100$, $m=500$
−.9999	1.28	.78	.49	.46
−.999	1.15	.54	.47	.44
−.99	1.58	.51	.49	.52
−.9	.92	.72	.89	.90
−.5	1.18	1.20	1.27	1.25
.5	1.45	1.40	1.24	1.20
.9	.78	.93	1.03	1.04
.99	.94	.51	.50	.63
.999	.95	.55	.47	.44
.9999	.77	.75	.49	.46

VERSION A. It is assumed that an element with index $(j-1)k+i$ is stored in position (i,j) of a two-dimensional array $(1 \leqq j \leqq m,\ 1 \leqq i \leqq k)$.

Step A1. Solution of h from $Dh = y$ (2.6). The elements of y and a are overwritten by those of h and V:

for $i = 2, \cdots, k$ **do**
  for $j = 1, \cdots, m$ **do**
    $y(i,j) = y(i,j) + a(i,j) * y(i-1,j)$
  for $j = 2, \cdots, m$ **do**
    $a(i,j) = a(i,j) * a(i-1,j)$.

Step A2. The solution x is obtained in y, by successively solving segments of x from (2.7):

for $j = 2, \cdots, m$ **do**
  for $i = 1, \cdots, k$ **do**
    $y(i,j) = y(i,j) + a(i,j) * y(k,j-1)$.

Both the innermost loops in Version A are vectorizable. The one in Step A1 has a stride k, whereas the one in Step A2 is a SAXPY (or linked triad), since $y(k,j-1)$ is a constant within that loop. Hence Version A leads to a vectorizable code for computers such as the CRAY-1, CRAY X-MP, and the FUJITSU FACOM VP-series; however, the performance may be degraded by memory bank conflicts due to stride problems in Step A1. For the CYBER 205 the vectorization of Step A1 is complicated and at least two gather and two scatter operations are necessary. Of course, the performance on all vector computers is also seriously degraded when the vector length k or m (or both) is short.

The situation with respect to stride problems is somewhat relieved by Version B.

VERSION B. It is assumed here that an element with index $(j-1)k+i$ is stored in position (j, i) of a two-dimensional array $(1 \leqq j \leqq m,\ 1 \leqq i \leqq k)$.

Step B1. Solution of h from $Dh = y$ (2.6). The elements of y and a are overwritten by those of h and V:

for $i = 2, \cdots, k$ **do**
  for $j = 1, \cdots, m$ **do**
    $y(j, i) = y(j, i) + a(j, i) * y(j, i-1)$
  for $j = 2, \cdots, m$ **do**
    $a(j, i) = a(j, i) * a(j, i-1)$.

Step B2. The solution x is obtained in y, by successively solving segments of x from (2.7):

for $j = 2, \cdots, m$ **do**
  for $i = 1, \cdots, k$ **do**
    $y(j, i) = y(j, i) + a(j, i) * y(j-1, k)$.

Instead of solving the successive segments of x from (2.7), we can solve first for the elements x_{k*j}, $j = 2, \cdots, m$. When this has been completed all the other elements of x can be computed independently. This leads to Version C.

VERSION C. Storage scheme as in Version B.

Step C1. Step B1.

Step C2. The elements x_{j*k}, $j = 2, \cdots, m$ are solved from (2.7):

for $j = 2, \cdots, m$ **do**
  $y(j, k) = y(j, k) + a(j, k) * y(j-1, k)$.

Step C3. The other elements of x are obtained, in array y, from (2.7):

for $i = 1, \cdots, k-1$ **do**
  for $j = 2, \cdots, m$ **do**
    $y(j, i) = y(j, i) + a(j, i) * y(j-1, k)$.

In Version C the solution of (2.7) is now largely vectorized with stride 1, however, at the cost of the scalar recursion in Step C2. The innermost loops in Steps C1 and C3 perform better when m is large; however, m should not be too large in order to keep the CPU-time for Step C2 relatively low. In the performance analysis for the CYBER 205, shown in § 3.2(d), we give optimal values of m for given n.

Note that the recursion in Step C2 could again be vectorized by the partition algorithm. Since in most applications m is much smaller than n, we have in our experiments optimized this step by loop unrolling it once (see also [7], [10]).

3.2. Global performance analysis for different vector computers. In this section we will consider the performance of the algorithms presented in § 3.1 for a number of different vector computers. We have selected the CRAY-1, FUJITSU FACOM VP100, the CRAY X-MP/48, and the CYBER 205, since they all differ with respect to their load and store possibilities. Other vector computers may be analysed along similar lines as those selected.

The idea behind this analysis is to get an impression of what is potentially achievable by a specific architecture. The reported actual performance then shows the gap between the abstract architecture and the way we see the real architecture through a given FORTRAN compiler.

(a) CRAY-1. This is a vector machine with only one path to memory. Algorithm Version A (as well as Version B) leads to completely vectorizable FORTRAN code. The innermost loop in Step A1 requires four vector loads and two vector stores, assuming that $a(i, j)$ are kept in a vector register. Since $y(k, j-1)$ is a constant within the innermost loop in Step A2, this loop requires two vector loads and one store. Hence the computational speed on the CRAY-1 will asymptotically be five flops per nine clockcycles, or $(5/9) * 80 = 44.4$ Mflops. Since the original recursion (2.1) requires only $2n$ flops, instead of the $5n$ for Wang's algorithm, the *effective* Mflops-rate is $\sim(2/9) * 80$, 17.8 Mflops, asymptotically.

With the FORTRAN compiler CFT77 1.3 (ID = CFT7713), for $m = k = 305$ we have effectively observed a rate of 15.4 Mflops. Memory bank conflicts can seriously affect the performance, e.g., for $m = k = 300$ on an eight-bank machine the performance decreased to 8.0 Mflops.

In CAL (CRAY Assembly Language) it should be possible to keep $y(i-1,j)$ and $a(i-1,j)$ in a register from the previous pass of the outer i-loop. Then only seven clockcycles would be necessary for each five flops, and hence the *effective* Mflop-rate could asymptotically be $\sim(2/7)*80=22.9$ Mflops. We have not tried this yet.

(b) FUJITSU FACOM VP200. Since the amount of vector accesses plays a dominant role in the presented algorithms, it is interesting to see what happens when the load/store possibilities are increased. The VP200 admits two simultaneous vector load/stores. For contiguous vectors this means that, e.g., two vector elements of different vectors can be loaded each 3.75 nanoseconds (ns) (the effective clockcycle). However, when there is a constant stride, if we assume no memory bank conflicts to occur, only two vector elements can be loaded each 7.5 ns.

In the innermost loop of Step A1 in Version A, we may first load the vectors $a(i,\cdot)$ and $y(i-1,\cdot)$, and then compute their elementwise products simultaneously. Hereafter $y(i,\cdot)$ is loaded, the addition can take place and the results can be stored. Finally $a(i-1,\cdot)$ can be loaded, be multiplied by the already available $a(i,\cdot)$ in register, and this result too can be stored. Since with all these operations a constant stride is involved, we may expect asymptotically to realize three flops per six clockcycles (of 3.75 ns each).

Similarly, the innermost loop in Step A2 may be expected to perform at a rate of two flops per two clockcycles (stride one), so that the asymptotical peak performance for Version A may be expected to be about $(5/8)*267=167$ Mflops. The effective Mflops-rate for the $2n$ operations of the original recursion would then be asymptotically $(2/5)*167=67$ Mflops. This agrees nicely with the experimentally observed 61 Mflops for the case $m=k=500$.

We may readily verify that Step A1 can be done more efficiently, by realizing that $a(i-1,\cdot)$ and $y(i-1,\cdot)$ can be held available in the registers from the previous pass of the i-loop. Then Version A can be executed at a rate of five flops per six cycles, which then might lead to an effective Mflops-rate of 89 Mflops (instead of 67 Mflops). Because of fewer stride problems, Version B may be expected to perform at a slightly higher rate than Version A.

Since in Version C stride problems are avoided at the cost of an (expensive) recursion, we may also study the performance of this variant on the VP200. It is now easily verified that Steps C1 and C3 can be executed at a total rate of five flops per five cycles, which implies an asymptotical Mflops-rate of 267 Mflops. The recursion in Step C2, however, has the poor performance of only 3.8 Mflops. Hence the $5mk$ flops of C1 and C3 at a rate of 267 Mflops and the $2m$ flops of C2 at a rate of 3.8 Mflops lead to the combined performance of $2mk/(5mk/267+2m/3.8)$ Mflops (effectively, for the original $2n$ flops). For, e.g., $k=100$ we find an asymptotic performance of 83 Mflops. This corresponds nicely with the experimentally observed 76.6 Mflops for $m=1000$ (and $k=100$). Hence, for large m, Version C leads to a speedup of about a factor 20 over the original recursion.

(c) CRAY X-MP/48 (in one-processor mode). On CRAY X-MP computers there are three simultaneous register-memory contacts possible: two loads and one store. A possible sequence of vector instructions for Version A is the following. First the vectors $y(i-1,\cdot)$ and $a(i,\cdot)$ are loaded and are elementwise multiplied. Then $y(i,\cdot)$ is loaded, the addition takes place, and the result vector is stored. Finally $a(i-1,\cdot)$ is loaded, multiplied elementwise by $a(i,\cdot)$, and the result is stored. Since the vector update in Step A2 can fully profit from chaining we find that five flops can be done in four clockcycles (asymptotically). Hence we have an *effective* asymptotical Mflop-rate of $(2/4)*105=52.5$ Mflops.

An alternative sequence is obtained as follows. In the first step the vectors $a(i-1,\cdot)$, $a(i,\cdot)$ are loaded, multiplied, and the result is stored. Then $y(i-1,\cdot)$ is loaded, $y(i,\cdot)$ is loaded, $y(i-1,\cdot)$ is elementwise multiplied by the (old) values of $a(i,\cdot)$, which is available in a vector register, the addition takes place, and the result is stored. Together with the vector update in Step A2 this leads to five flops per three clockcycles, and hence the *effective* asymptotical performance would be $(2/3) * 105 =$ 70 Mflops.

So far the best we have obtained with the FORTRAN code, for $m = k = 305$, is 44.3 Mflops (which implies an actual performance for Version A of ~110 Mflops). Part of the difference between the theoretical 70 Mflops and the experimental 44.3 Mflops can be explained by the fact that, when using all three paths to memory in a chain, only one result per ≈ 1.2 clockcycle is delivered, instead of one per clockcycle [2]. Hence we have 3.6 instead of 3 clockcycles per element, so that we then might expect, effectively, $(2/3.6) * 105 \approx 58$ Mflops, which explains about half of the gap between the theoretical and the experimental performance.

(d) CYBER 205. The CYBER 205 leads to a different performance analysis, since it has direct memory access and it requires contiguous vectors (i.e., stride = 1). Because of this stride requirement both Versions A and B are not very attractive since gather/scatter instructions are now necessary. We have not investigated this further as yet. Instead we consider the performance for Version C, in which all vectorizable statements have stride 1; this is at the cost of an extra nonvectorizable loop (C2) and with a different storage scheme (the arrays a and y must be stored by row rather than by column).

The performance of Version C can be analysed by counting the clockcycles for the three steps. Let p denote the number of vector pipes. Step C1 can be executed in $(k-1)\cdot(m-1)/p + 2\cdot m\cdot(k-1)/p + 3\cdot(k-1)\cdot 51$ clockcycles (the startup time for a vector instruction is 51 clockcycles), and Step C3 takes $2\cdot(k-1)\cdot(m-1)/p + 2\cdot(k-1)\cdot 51$ clockcycles. The $2\cdot(m-1)$ flops in the nonvectorized Step C2 take $2\cdot(m-1)\cdot s$ cycles, in which s is the number of clockcycles for a scalar flop. In our experiments we have unrolled the loop in C2 once, which led to a performance of ≈ 7 clockcycles per flop ($s \approx 7$).

Hence the total number N_c of clockcycles for Version C is given by $N_c = 5n/p + (2s - 5/p)m + (255 - 3/p)k + 3/p - 2s - 255$. Since a clockcycle is 20 ns, and since the original recursion (2.1) requires $2(n-1)$ flops, we define the *effective* Mflops-rate R_E, for Version C on a p-pipe CYBER 205, by

$$R_E = \frac{2(n-1) * 50}{N_c} \text{ Mflops.} \tag{3.1}$$

It is easily verified that R_E, as a function of $m\ (=n/k)$, is maximal for

$$m = \sqrt{\frac{255 - 3/p}{2s - 5/p}} \cdot \sqrt{n}.$$

For $s = 7$ we have $m \approx 5.3\sqrt{n}$ (for n large enough). For example, for $m = 1{,}372$ and $p = 1$, s was experimentally determined as $s \approx 6.529$, and hence $n \approx m^2 \cdot (2s - 5)/252 \approx 60{,}368$ ($k = 44$). According to (3.1) we then expect, for a one-pipe CYBER 205, a performance of $R_E = 18.648$ Mflops. Experimentally we observed an effective performance for Version C of 18.649 Mflops. The higher actual performance can be explained from the partial overlap of start-up times in step C1. We also give some experimental results for $n = 60{,}480$, $m = 945$, $k = 64$. For $p = 1$: 18.57 Mflops ($R_E = 18.56$ Mflops), for $p = 2$: 33.3 Mflops and, when computing in half precision, for $p = 1$: 33.4 Mflops

and $p = 2 : 56.4$ Mflops. Note that these rates are effective rates, so that, e.g., in the first example Version C requires about one-third of the CPU-time necessary for the recursion (2.1).

REFERENCES

[1] S. C. CHEN, D. J. KUCK, AND A. H. SAMEH, *Practical parallel band triangular system solvers*, ACM Trans. Math. Software, 4 (1978), pp. 270–277.

[2] A. DAVE, CRAY Research (UK), LTD, personal communication, 1987.

[3] P. DUBOIS AND G. RODRIGUE, *An analysis of the recursive doubling algorithm*, in High Speed Computer and Algorithm Organization, Kuck et al., eds., Academic Press, New York, 1977.

[4] G. H. GOLUB AND C. VAN LOAN, *Matrix Computations*, North Oxford Academic, Oxford, 1983.

[5] D. HELLER, *Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems*, SIAM J. Numer. Anal., 13 (1978), pp. 484–496.

[6] J. J. LABIOTTE AND R. G. VOIGT, *The solution of tridiagonal linear systems on the* CDC-STAR-100 *computer*, ICASE Report, ICASE-NASA Langley Research Center, Hampton, VA, 1974.

[7] J. J. F. M. SCHLICHTING AND H. A. VAN DER VORST, *Solving bidiagonal systems of linear equations on the* CDC CYBER 205, Report NM-R8725, CWI, Amsterdam, the Netherlands, 1987.

[8] 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.

[9] H. A. VAN DER VORST, *Parallel solution of bidiagonal systems coming from discretised* PDE's, IEEE Trans. Magnetics, 24 (1988), pp. 286–290.

[10] H. A. VAN DER VORST AND J. M. VAN KATS, *The performance of some linear algebra algorithms in* FORTRAN *on* CRAY-1 *and* CYBER 205 *supercomputers*, Tech. Report TR-17, Academisch Computer Centrum Utrecht, Utrecht, the Netherlands, 1984.

[11] H. H. WANG, *A parallel method for tridiagonal equations*, ACM Trans. Math. Software, 7 (1981), pp. 170–183.

[12] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 36-52, January 1989

© 1989 Society for Industrial and Applied Mathematics
004

CGS, A FAST LANCZOS-TYPE SOLVER FOR NONSYMMETRIC LINEAR SYSTEMS*

PETER SONNEVELD†

Abstract. A Lanczos-type method is presented for nonsymmetric sparse linear systems as arising from discretisations of elliptic partial differential equations. The method is based on a polynomial variant of the conjugate gradients algorithm. Although related to the so-called bi-conjugate gradients (Bi-CG) algorithm, it does not involve adjoint matrix-vector multiplications, and the expected convergence rate is about twice that of the Bi-CG algorithm. Numerical comparison is made with other solvers, testing the method on a family of convection diffusion equations, on various grids, and with the use of two different preconditioning methods. Upwind as well as central differencing is used in the experiments.

Key words. preconditioned conjugate gradients, Lanczos algorithm, iterative methods, CG, CGS, SPD, Bi-CG

AMS(MOS) subject classifications. 65F10, 65N20

1. Introduction. For solving large sparse linear systems with a real symmetric positive definite coefficient matrix (referred to as an SPD-matrix), the Conjugate Gradients algorithm, when used with a suitable preconditioning technique, seems to be one of the most efficient methods in the class of simple iterative techniques. However, when the system is not SPD, as with discretisations of nonself-adjoint elliptic differential equations, the algorithm is not applicable. There are some generalisations available, such as Generalised Conjugate Gradients (Axelsson [1], [2]), Orthomin (Vinsome [23]), Orthodir (Young and Jea [26]), Manteuffel-Chebyshev (Manteuffel [12], [13], Van der Vorst [21]), Bi-Conjugate Gradients (Fletcher [8]), and Induced Dimension Reduction (Wesseling and Sonneveld, [24]). Some of these methods can be regarded as variants of a so-called Petrov-Galerkin approach (Saad and Schultz [18]). Also, hybrid combinations of CG generalisations and Chebyshev have been proposed (Elman, Saad, and Saylor [7]). All these generalisations use residuals lying in a Krylov subspace associated with the coefficient matrix of the system. Other generalisations use a Krylov subspace associated with the coefficient matrix of the normal equations (Paige and Saunders [17]), or use a Krylov subspace associated with related skew symmetric operator (Concus and Golub [4], Widlund [25]). Just as with application of ordinary CG on SPD problems, these methods are frequently combined with preconditioning techniques, such as incomplete factorisations (Meijerink and Van der Vorst [15], [16]), and we could say that without preconditioning these methods are hardly competitive.

The difficulty of solving nonsymmetric linear systems by means of CF-type methods can be understood as follows. In the CG algorithm, the residual vector is minimised with respect to some suitable vector norm. As a consequence, the residual vectors during the process are constructed such that they build a system orthogonal with respect to some related inner product. An essential property of the algorithm is the three-term recurrence relationship between the residuals, yielding an algorithm that is economical in both storage requirements and computing time. If operations with the coefficient matrix $\mathbf{A}$ are to be restricted to matrix vector multiplications, then, if $\mathbf{A}$ is real and not SPD, the only way for maintaining the three-term relations as well as the minimisation property involves the solution of the normal equations with the coefficient

* Received by the editors April 24, 1984; accepted for publication (in revised form) February 2, 1988.
† Delft University of Technology, Department of Mathematics, Delft, the Netherlands.

matrix $\mathbf{A}^T\mathbf{A}$. This can be done either by using the CG algorithm (see, e.g., Kershaw [11]), or, preferably, by applying Paige's bi-diagonalisation algorithm for the solution of sparse least-square problems (Paige and Saunders [17]). If the matrix $\mathbf{A}$ arises from a discretisation of a nonself-adjoint elliptic operator, the matrix $\mathbf{A}^T\mathbf{A}$ may have a much higher spectral condition number than the original matrix $\mathbf{A}$, causing a considerably slower convergence. There are several other possibilities for generalisations of the CG algorithm to nonsymmetric systems:

(1) We can maintain the minimisation property by choosing the next search direction as a linear combination of $\mathbf{r}_n$ and s previous search directions, such that the $\mathbf{A}$-image of the new search direction is orthogonal to the $\mathbf{A}$-images of the previous ones. Minimising the Euclidean norm of the next residual along this search direction yields the ORTHOMIN procedure. Other methods of this kind are ORTHODIR and Axelsson's method, *Generalised Conjugate Gradients.*

(2) We can maintain the three-term recursion by suitable re-interpretation of the CG algorithm. This is done in *Bi-Conjugate Gradients*, as well as in some sense in the *Induced Dimension Reduction* algorithm. In the general case nothing is minimised at all, which can be regarded as a theoretical weakness. In practice however, things turn out to be far less serious than we would expect, as numerical experiments have shown.

(3) We can maintain a three-term recursion, but abandon the entire CG idea. So we leave the class of Lanczos-type methods, and enter that of the Chebyshev-like methods. A disadvantage here is the necessity for finding optimal values for some scalar parameters (as when using SOR methods). So the method often is a slow starter, and therefore is slow compared with sprinters. However, if the spectrum of the coefficient matrix is in the right half of the complex plane (which is a common requirement for an iterative method to be successful anyway), the *Manteuffel-Chebyshev* algorithm with optimal parameters will certainly converge. The convergence rate depends on a "smallest ellipse" around the spectrum, and therefore cannot be influenced by using detailed information about the distribution of eigenvalues. This may be a second disadvantage.

(4) We can maintain a three-term recursion by splitting off the symmetric part of the matrix, if this part is an SPD-matrix, and applying some imaginary axis variant of the CG algorithm on a related skew symmetric system. This procedure requires the solution of an SPD problem at each iteration step [4], [25].

In this paper the second alternative is chosen. Via a polynomial equivalent of the CG algorithm, an algorithm for the squares of Lanczos polynomials is derived, which yields the *Conjugate Gradients Squared* algorithm (CGS). This method is found to be more efficient, for a relevant class of problems, than Bi-CG and ORTHOMIN, and, in our experience, competitive with the Manteuffel-Chebyshev algorithm.

2. The polynomial equivalent of the CG method. Let $\mathbf{A}$ be an $N \times N$ SPD-matrix, and let $\mathbf{Ax} = \mathbf{b}$ be a system to be solved. Let $\mathbf{x}_0$ be an initial guess for the solution $\mathbf{x}$; then the usual CG algorithm for this problem reads

$$
\begin{aligned}
&\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0;\ \mathbf{p}_{-1} := 0;\ \rho_{-1} := 1;\\
&n := 0;\\
&\textbf{while } \textit{residual} > \textit{tolerance} \textbf{ do}\\
&\textbf{begin}\\
&\quad \rho_n := \mathbf{r}_n^T\mathbf{r}_n;\ \beta_n := \frac{\rho_n}{\rho_{n-1}};\\
&\quad \mathbf{p}_n := \mathbf{r}_n + \beta_n\mathbf{p}_{n-1};\\
&\quad \sigma_n := \mathbf{p}_n^T\mathbf{A}\mathbf{p}_n;\ \alpha_n := \frac{\rho_n}{\sigma_n};
\end{aligned}
\tag{2.1}
$$

$\mathbf{r}_{n+1} := \mathbf{r}_n - \alpha_n \mathbf{A}\mathbf{p}_n;$
$\mathbf{x}_{n+1} := \mathbf{x}_n + \alpha_n \mathbf{p}_n;$
$n := n+1$
end;

It has been proved that this algorithm theoretically gives the exact answer after at most N steps. The vectors $\mathbf{r}_n$ satisfy $\mathbf{r}_n = \mathbf{b} - \mathbf{A}\mathbf{x}_n$, and are called residuals. The vectors $\mathbf{p}_n$ are called search directions.

An important property of this algorithm is that the function $E_n = \mathbf{r}_n^T \mathbf{A}^{-1} \mathbf{r}_n$ is minimised at each step. This quantity can be regarded as the square of a norm for the residual $\mathbf{r}_n$. It has also been proved that the residuals are mutually orthogonal, and that the search directions are mutually **A**-orthogonal:

$$\mathbf{r}_n^T \mathbf{r}_m = \rho_n \delta_{nm}, \qquad \mathbf{p}_n^T \mathbf{A}\mathbf{p}_m = \sigma_n \delta_{nm}$$

where δ_{nm} is the Kronecker symbol.

From the structure of the algorithm it can be seen that the vectors $\mathbf{r}_n$ and $\mathbf{p}_n$ can be written as follows:

$$\mathbf{r}_n = \varphi_n(\mathbf{A})\mathbf{r}_0, \qquad \mathbf{p}_n = \psi_n(\mathbf{A})\mathbf{r}_0$$

where φ_n and ψ_n are polynomials of degree less than or equal to n. In fact this can be proved simply by induction if we define: $\varphi_0(\tau) \equiv 1$, $\psi_{-1}(\tau) \equiv 0$, which is consistent with the CG algorithm. Substituting

$$\mathbf{r}_n = \varphi_n(\mathbf{A})\mathbf{r}_0, \qquad \mathbf{p}_{n-1} = \psi_{n-1}(\mathbf{A})\mathbf{r}_0 \tag{2.2}$$

for some $n \geqq 0$, we find

$$\mathbf{p}_n = \varphi_n(\mathbf{A})\mathbf{r}_0 + \beta_n \psi_{n-1}(\mathbf{A})\mathbf{r}_0 = \psi_n(\mathbf{A})\mathbf{r}_0$$

with

$$\psi_n(\tau) \equiv \varphi_n(\tau) + \beta_n \psi_{n-1}(\tau) \quad \forall \tau \in \mathbb{R}.$$

Substituting this in the computation step for $\mathbf{r}_{n+1}$ we find

$$\mathbf{r}_{n+1} = \varphi_n(\mathbf{A})\mathbf{r}_0 - \alpha_n \mathbf{A}\psi_n(\mathbf{A})\mathbf{r}_0 = \varphi_{n+1}(\mathbf{A})\mathbf{r}_0$$

with

$$\varphi_{n+1}(\tau) \equiv \varphi_n(\tau) - \alpha_n \tau \psi_n(\tau) \quad \forall \tau \in \mathbb{R}.$$

Hence the validity of hypothesis (2.2) for n implies the validity for $n+1$. So (2.1) can be re-interpreted as an algorithm for generating a system of (orthogonal) polynomials.

The computation of the scalar quantities ρ_n and σ_n can be carried out in terms of the polynomials as well. Denote the set of real polynomials of degree up to N by Π^N, and define the symmetric bilinear form $(\cdot, \cdot)$ on Π^N by

$$(\varphi, \psi) = [\varphi(\mathbf{A})\mathbf{r}_0]^T \psi(\mathbf{A})\mathbf{r}_0.$$

Obviously we have $(\varphi, \varphi) \geqq 0$, for all $\varphi \in \Pi^N$. Since **A** is symmetric, we also can write

$$(\varphi, \psi) = \mathbf{r}_0^T \varphi(\mathbf{A})\psi(\mathbf{A})\mathbf{r}_0.$$

Furthermore, from the associativity law for matrix products follows

$$(\varphi\chi, \psi) = (\varphi, \chi\psi)$$

for any polynomials φ, χ, ψ. So this bilinear form satisfies all but one of the requirements for a proper inner product on the space Π^N. Only positive definiteness is not guaranteed for all polynomials, since for some nonzero polynomial φ the vector $\varphi(\mathbf{A})\mathbf{r}_0$ may be

zero; this implies $(\varphi, \varphi) = 0$. However, as long as **A** is real symmetric, $(\cdot, \cdot)$ is semi-definite in any case.

Now the polynomial equivalent of (2.1) reads

$$
\begin{aligned}
&\varphi_0 \equiv 1;\ \psi_{-1} \equiv 0;\\
&\rho_{-1} := 1;\ n := 0;\\
&\textbf{while } \textit{residual} > \textit{tolerance} \textbf{ do}\\
&\textbf{begin}\\
&\quad \rho_n := (\varphi_n, \varphi_n);\ \beta_n := \frac{\rho_n}{\rho_{n-1}};\\
&\quad \psi_n := \varphi_n + \beta_n \psi_{n-1};\\
&\quad \sigma_n := (\psi_n, \vartheta\psi_n);\ \alpha_n := \frac{\rho_n}{\sigma_n};\\
&\quad \varphi_{n+1} := \varphi_n - \alpha_n \vartheta\psi_n;\\
&\quad n := n+1\\
&\textbf{end};
\end{aligned}
\tag{2.3}
$$

where

$$\vartheta(\tau) \equiv \tau. \tag{2.4}$$

The minimisation property now simply reads

$$E_n = (\varphi_n, \vartheta^{-1}\varphi_n) = \min_{\varphi \in \Pi^N} \frac{(\varphi, \vartheta^{-1}\varphi)}{\varphi(0)^2}.$$

(Note that all algorithms of the type (2.3), with freedom in the choice of α_n and β_n, share the property $\varphi_n(0) = 1$ for all n.)

The orthogonality of $\mathbf{r}_n$ and the **A**-orthogonality of $\mathbf{p}_n$ translate into orthogonality of the polynomials φ_n and ψ_n with respect to the inner products $(\cdot, \cdot)$ and $(\cdot, \vartheta\,\cdot)$ respectively.

The polynomial CG algorithm has a meaning in its own right, i.e., apart from its relation with the vectorial algorithm. We prove the following theorem.

THEOREM. *Let* $[\cdot, \cdot]$ *be any symmetric bilinear form defined on* Π^N, *satisfying*

$$[\varphi\psi, \chi] = [\varphi, \psi\chi] \quad \forall \varphi, \psi, \chi \in \Pi^N. \tag{2.5}$$

Let the sequence of polynomials φ_n *and* ψ_n *be constructed according to algorithm* (2.3), *but using* $[\cdot, \cdot]$ *instead of* $(\cdot, \cdot)$. *Then, as long as the algorithm does not break down by zero division, the polynomials* φ_n *and* ψ_n *satisfy*

$$[\varphi_n, \varphi_m] = \rho_n \delta_{nm}, \quad [\psi_n, \vartheta\psi_m] = \sigma_n \delta_{nm} \quad \forall n, m \tag{2.6}$$

with ϑ *defined by* (2.4).

Proof. We prove by induction the following statement:

$$[\psi_{n-1}, \vartheta\psi_k] = \sigma_{n-1}\delta_{n-1,k}, \quad [\varphi_n, \psi_k] = 0 \quad \forall n \geqq 0, \quad -1 \leqq k \leqq n-1 \tag{2.7}$$

with $\sigma_{-1} = 0$. If $n = 0$, this statement is true since $\psi_{-1}(\tau) \equiv 0$. Suppose (2.7) holds for $n \leqq m$, and let $k < m$; then according to (2.3) we have

$$[\psi_m, \vartheta\psi_k] = [\varphi_m, \vartheta\psi_k] + \beta_m [\psi_{m-1}, \vartheta\psi_k].$$

Now the second right-hand term vanishes for $k < m-1$ (by hypothesis). In the first term we substitute $\psi_k = (\varphi_k - \varphi_{k+1})/\alpha_k\vartheta$, yielding

$$[\psi_m, \vartheta\psi_k] = \frac{[\varphi_m, \varphi_k] - [\varphi_m, \varphi_{k+1}]}{\alpha_k} + \beta_m \sigma_{m-1}\delta_{m-1,k} \quad \forall k \leqq m-1.$$

By hypothesis this is zero if $k < m-1$, while for $k = m-1$ we have

$$[\psi_m, \vartheta\psi_{m-1}] = -\frac{\rho_m}{\alpha_{m-1}} + \beta_m \sigma_{m-1} = 0,$$

which proves the first part of (2.7) for $n = m+1$.

Proceeding with proving the second part, we can write

$$[\varphi_{m+1}, \psi_k] = [\varphi_m, \psi_k] - \alpha_m [\psi_m, \vartheta\psi_k] \quad \forall k \leqq m.$$

If $k \leqq m-1$ the right-hand side vanishes by hypothesis. Using the algorithm and choosing $k = m$ we get

$$[\varphi_{m+1}, \psi_m] = [\varphi_m, \varphi_m + \beta_m \psi_{m-1}] - \alpha_m [\psi_m, \vartheta\psi_m] = \rho_m - \alpha_m \sigma_m = 0,$$

which proves the second part of (2.7). Hence by induction (2.7) is valid for all natural n.

Finally, writing $\varphi_k = \psi_k - \beta_k \psi_{k-1}$, for all $k \geqq 0$ we have

$$[\varphi_n, \varphi_k] = [\varphi_n, \psi_k] - \beta_n [\varphi_n, \psi_{k-1}] = 0 \quad \forall k < n,$$

which together with the first part of (2.7) proves the theorem.

In general the polynomial algorithm could possibly break down by zero division in the computation of α_n and β_n. However, as long as the algorithm runs properly, the above theorem is valid. For this reason we shall use the term *orthogonal polynomials* for φ_n and ψ_n, whether or not the bilinear forms involved are inner products.

We are now in a position to generalise the CG algorithm to nonsymmetric cases, by defining $[\cdot, \cdot]$ in an appropriate way. Let $\mathbf{Ax} = \mathbf{b}$ be a nonsingular system to be solved. Let $\mathbf{x}_0$ be an initial guess for the solution $\mathbf{x}$, with corresponding residual $\mathbf{r}_0 = \mathbf{Ax}_0 - \mathbf{b}$, and let $\tilde{\mathbf{r}}_0$ be a suitably chosen vector. Now define $[\cdot, \cdot]$ by means of

$$[\varphi, \psi] = \tilde{\mathbf{r}}_0^T \varphi(\mathbf{A}) \psi(\mathbf{A}) \mathbf{r}_0 = (\varphi(\mathbf{A}^T)\tilde{\mathbf{r}}_0)^T \psi(\mathbf{A}) \mathbf{r}_0 \tag{2.8}$$

and define the vectors $\mathbf{r}_n$, $\tilde{\mathbf{r}}_n$, $\mathbf{p}_n$, and $\tilde{\mathbf{p}}_n$ by

$$\begin{aligned} \mathbf{p}_{-1} = \tilde{\mathbf{p}}_{-1} &= \mathbf{0}, \\ \mathbf{r}_n = \varphi_n(\mathbf{A})\mathbf{r}_0, &\qquad \tilde{\mathbf{r}}_n = \varphi_n(\mathbf{A}^T)\tilde{\mathbf{r}}_0, \\ \mathbf{p}_n = \psi_n((\mathbf{A})\mathbf{r}_0, &\qquad \tilde{\mathbf{p}}_n = \psi_n(\mathbf{A}^T)\tilde{\mathbf{r}}_0 \end{aligned} \tag{2.9}$$

with φ_n and ψ_n according to (2.3). Then it can easily be shown that these vectors can be produced by the following so-called Bi-Conjugate Gradients algorithm (Bi-CG, Fletcher [8]):

$\mathbf{r}_0 := \mathbf{b} - \mathbf{Ax}_0$; $\mathbf{p}_{-1} := \tilde{\mathbf{p}}_{-1} := \mathbf{0}$;
$n := 0$; $\rho_{-1} := 1$;
while *residual* > *tolerance* **do**
begin

$$\begin{aligned} &\rho_n := \tilde{\mathbf{r}}_n^T \mathbf{r}_n; \ \beta_n := \frac{\rho_n}{\rho_{n-1}}; \\ &\mathbf{p}_n := \mathbf{r}_n + \beta_n \mathbf{p}_{n-1}; \ \tilde{\mathbf{p}}_n := \tilde{\mathbf{r}}_n + \beta_n \tilde{\mathbf{p}}_{n-1}; \\ &\sigma_n := \tilde{\mathbf{p}}_n^T \mathbf{A}\mathbf{p}_n; \ \alpha_n := \frac{\rho_n}{\sigma_n}; \\ &\mathbf{r}_{n+1} := \mathbf{r}_n - \alpha_n \mathbf{A}\mathbf{p}_n; \ \tilde{\mathbf{r}}_{n+1} := \tilde{\mathbf{r}}_n - \alpha_n \mathbf{A}^T \tilde{\mathbf{p}}_n; \\ &\mathbf{x}_{n+1} := \mathbf{x}_n + \alpha_n \mathbf{p}_n; \\ &n := n+1 \end{aligned} \tag{2.10}$$

end;

Again (as long as no breakdown occurs) we have $\mathbf{r}_n = \mathbf{b} - \mathbf{Ax}_n$, meaning that convergence of $\mathbf{r}_n$ to $\mathbf{0}$ implies convergence of $\mathbf{x}_n$ to the solution $\mathbf{x}$. In practice the vector $\tilde{\mathbf{r}}_0$ is often

chosen equal to $\mathbf{r}_0$. Then, if $\mathbf{A}$ is not too far from being SPD, the bilinear expressions $[\cdot,\cdot]$ and $[\cdot,\vartheta\cdot]$ will be positive semi-definite, and the algorithm will converge in the same way, and by the same argument as does the ordinary CG algorithm in the SPD-case.

3. Squaring the CG algorithm: The CGS algorithm. Assume that the Bi-CG algorithm is converging well in some case. Then $\mathbf{r}_n \to 0$ as n increases, which can be interpreted as a property of the matrices $\varphi_n(\mathbf{A})$: since $\mathbf{r}_n = \varphi_n(\mathbf{A})\mathbf{r}_0$ these matrices behave like contracting operators, at least as far as their effect on the start residual is concerned. Then it may be expected that the matrices $\varphi_n(\mathbf{A}^T)$ are contracting too, with the consequence that also $\tilde{\mathbf{r}}_n \to \mathbf{0}$. So we may expect the two sequences of vectors $\mathbf{r}_n$ and $\tilde{\mathbf{r}}_n$ to converge at a comparable rate. However, the convergence of the "*quasi-residuals*" $\tilde{\mathbf{r}}_n$ is not exploited. But on the other hand, we need the computation of these vectors because they are involved in the computation of the scalar quantities ρ_n and σ_n. Now the amount of work needed for the computation of the vectors $\tilde{\mathbf{r}}_n$ and $\tilde{\mathbf{p}}_n$ is about the same as that for the original CG method itself, so we observe that the computational work of the Bi-CG algorithm is twice the computational work for the CG algorithm. Apart from this observation, another disadvantage is the occurrence of the operation $\mathbf{A}^T\mathbf{v}$, which may cause very cumbersome programming codes, especially if $\mathbf{A}$ is stored with a general data structure.

In order to improve the situation, we return to the polynomial algorithm (2.3). Here the scalars are calculated by $\rho_n = [\varphi_n, \varphi_n]$, $\sigma_n = [\psi_n, \vartheta\psi_n]$. Now since $[\cdot,\cdot]$ has the property (2.5) it follows that ρ_n and σ_n could be calculated just as well by the following formulae:

$$\rho_n = [\varphi_0, \varphi_n^2], \qquad \sigma_n = [\varphi_0, \vartheta\psi_n^2].$$

This suggests the possibility of an algorithm that generates the polynomials φ_n^2 and ψ_n^2 rather than φ_n and ψ_n. Consequently, we will try to interpret the vectors $\varphi_n^2(\mathbf{A})\mathbf{r}_0$ as residuals of estimates $\mathbf{x}_n$ of the solution $\mathbf{x}$. The computation of the quantities ρ_n for instance could be done by $\rho_n = \tilde{\mathbf{r}}_0^T\mathbf{r}_n$, where $\mathbf{r}_n$ now stands for $\varphi_n^2(\mathbf{A})\mathbf{r}_0$. Then a process for generating the sequences $\tilde{\mathbf{r}}_n$ and $\tilde{\mathbf{p}}_n$ is no longer needed, which is an important simplification of the programming code. Furthermore, the contracting effect of the matrices $\varphi_n(\mathbf{A})$ is "doubled."

To arrive at an algorithm for the squared polynomials we take the squares of the computational steps for ψ_n and φ_{n+1} in (2.3):

$$\psi_n^2 = \varphi_n^2 + 2\beta_n\varphi_n\psi_{n-1} + \beta_n^2\psi_{n-1}^2, \qquad \varphi_{n+1}^2 = \varphi_n^2 - 2\alpha_n\vartheta\varphi_n\psi_n + \alpha_n^2\vartheta^2\psi_n^2$$

where ϑ again is defined by (2.4). Obviously the computation of ψ_n^2 and φ_{n+1}^2 requires the products $\varphi_n\psi_{n-1}$ and $\varphi_n\psi_n$ as well. However, $\varphi_n\psi_n$ can be expressed as $\varphi_n^2 + \beta_n\varphi_n\psi_{n-1}$, and is therefore not explicitly needed. Furthermore, multiplication of the computational step for φ_{n+1} with ψ_n yields

$$\varphi_{n+1}\psi_n = \varphi_n\psi_h - \alpha_n\vartheta\psi_n^2,$$

which also is expressible in φ_n^2, $\varphi_n\psi_{n-1}$ and ψ_{n-1}^2. Hence an algorithm with the required properties can be constructed. Define for $n \geqq 0$:

$$\begin{aligned} \Phi_n &= \varphi_n^2, \\ \Theta_n &= \varphi_n\psi_{n-1}, \\ \Psi_{n-1} &= \psi_{n-1}^2. \end{aligned} \tag{3.1}$$

Then it is easily verified that the following algorithm has the desired properties:

$\Phi_0 \equiv 1;\ \Theta_0 \equiv \Psi_{-1} \equiv 0;\ \rho_{-1} := 1;$
$n := 0;$
while *residual* $>$ *tolerance* **do**
begin

$$\rho_n := [1, \Phi_n];\ \beta_n := \frac{\rho_n}{\rho_{n-1}};$$
$$\Upsilon_n := \Phi_n + \beta_n \Theta_n;$$
$$\Psi_n := \Upsilon_n + \beta_n(\Theta_n + \beta_n \Psi_{n-1}); \tag{3.2}$$
$$\sigma_n := [1, \vartheta \Psi_n];\ \alpha_n := \frac{\rho_n}{\sigma_n};$$
$$\Theta_{n+1} := \Upsilon_n - \alpha_n \vartheta \Psi_n;$$
$$\Phi_{n+1} := \Phi_n - \alpha_n \vartheta(\Upsilon_n + \Theta_{n+1});$$
$$n := n+1$$

end;

We obtain a vectorial variant of this algorithm by substituting $\mathbf{A}$ into the polynomials, and making the appropriate parts act as linear operators on the starting residual $\mathbf{r}_0$. Defining $\mathbf{r}_n = \Phi_n(\mathbf{A})\mathbf{r}_0$, $\mathbf{q}_n = \Theta_n(\mathbf{A})\mathbf{r}_0$, and $\mathbf{p}_n = \Psi_n(\mathbf{A})\mathbf{r}_0$ for all appropriate values of n, in this way we arrive at the Conjugate Gradients Squared (CGS) algorithm.

Let $\mathbf{x}_0$ be a starting estimate of the solution $\mathbf{x}$ of the nonsingular linear system $\mathbf{Ax} = \mathbf{b}$, and let $\tilde{\mathbf{r}}_0$ be suitably chosen. Then the CGS algorithm reads:

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0;$
$\mathbf{q}_0 := \mathbf{p}_{-1} := \mathbf{0};\ \rho_{-1} := 1;$
$n := 0;$
while *residual* $>$ *tolerance* **do**
begin

$$\rho_n := \tilde{\mathbf{r}}_0^T \mathbf{r}_n;\ \beta_n := \frac{\rho_n}{\rho_{n-1}};$$
$$\mathbf{u}_n := \mathbf{r}_n + \beta_n \mathbf{q}_n;$$
$$\mathbf{p}_n := \mathbf{u}_n + \beta_n(\mathbf{q}_n + \beta_n \mathbf{p}_{n-1}); \tag{3.3}$$
$$\mathbf{v}_n := \mathbf{A}\mathbf{p}_n;$$
$$\sigma_n := \tilde{\mathbf{r}}_0^T \mathbf{v}_n;\ \alpha_n := \frac{\rho_n}{\sigma_n};$$
$$\mathbf{q}_{n+1} := \mathbf{u}_n - \alpha_n \mathbf{v}_n;$$
$$\mathbf{r}_{n+1} := \mathbf{r}_n - \alpha_n \mathbf{A}(\mathbf{u}_n + \mathbf{q}_{n+1});$$
$$\mathbf{x}_{n+1} := \mathbf{x}_n + \alpha_n(\mathbf{u}_n + \mathbf{q}_{n+1});$$
$$n := n+1$$

end;

Since $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ and $\mathbf{r}_{n+1} - \mathbf{r}_n = \mathbf{A}(\mathbf{x}_n - \mathbf{x}_{n+1})$ for all n it follows that $\mathbf{r}_n = \mathbf{b} - \mathbf{A}\mathbf{x}_n$. So this algorithm produces iterates $\mathbf{x}_n$ of which the residuals satisfy

$$\mathbf{r}_n = \varphi_n^2(\mathbf{A})\mathbf{r}_0.$$

Each step requires twice the amount of work necessary for the symmetric CG method, however, the contracting effect of $\varphi_n(\mathbf{A})$ is used twice each step. The work is not more than for Bi-CG, and working with $\mathbf{A}^T$ is avoided.

4. Convergence and preconditioning. For the CG method, when applied to self-adjoint positive definite problems, the following theoretical result holds for the

residuals:

$$0 < \frac{E_n}{E_0} \leqq 4\left[\frac{\sqrt{K_A}-1}{\sqrt{K_A}+1}\right]^{2n} \tag{4.1}$$

with

$$E_n = \mathbf{r}_n^T \mathbf{A}^{-1} \mathbf{r}_n$$

where K_A denotes the spectral condition number of $\mathbf{A}$, defined by $K_A = \lambda_{\max}/\lambda_{\min}$. Since E_n can be regarded as the square of a norm for $\mathbf{r}_n$, this formula means that CG converges at least as fast as a linearly convergent process with convergence factor $\rho = (1-2/\sqrt{K_A})$. For the proof see, for instance, Axelsson [3]. More refined theoretical results are presented in Van der Vorst [22], which takes into account more information about the distribution of eigenvalues of $\mathbf{A}$. The convergence of the CG algorithm turns out to be superlinear, with a convergence rate that depends on the distribution of the (mainly smallest) eigenvalues of $\mathbf{A}$, rather than on the spectral condition number.

It is common to apply *preconditioning techniques* in order to reduce the spectral condition number (Gustafsson [9]), or to improve the distribution of the smallest eigenvalues (Van der Vorst [22]). Although Bi-CG and CGS algorithms do not require an SPD-matrix $\mathbf{A}$, it may be expected that, if the bilinear form $[\cdot, \cdot]$ does not deviate too much from a proper inner product (i.e., from being positive definite), the convergence of both methods will be largely similar to the SPD case. It can easily be proved that in absence of roundoff both Bi-CG and CGS terminate after at most N steps by zero division, since ρ_n will be zero for some $n \leqq N$. If the vector $\tilde{\mathbf{r}}_0$ is not orthogonal to any invariant subspace of the Krylov subspace associated with $\mathbf{A}$ and $\mathbf{r}_0$ (which is true in "almost all" cases with random choice for $\tilde{\mathbf{r}}_0$), this implies $\mathbf{r}_n = \mathbf{0}$; hence both processes are theoretically finite in the same way as CG is in the SPD case. An essential requirement for convergence in considerably less than N steps is that *no eigenvalues of the matrix* $\mathbf{A}$ *are in the left half of the complex plane*, i.e., that the matrix $\mathbf{I} - \alpha\mathbf{A}$ is a contraction for some positive α. In fact it can be proved that if $\mathbf{A}$ is not defective and has positive real eigenvalues, a vector $\tilde{\mathbf{r}}_0$ exists such that both Bi-CG and CGS converge at a rate comparable to that of CG (however, it is not easy to find this "suitable" choice for $\tilde{\mathbf{r}}_0$ in practice). We may also expect (and this has been verified numerically) that preconditioning will improve the behaviour of the Bi-CG and CGS algorithms, in the same way as with CG.

A global description of preconditioning will now be given. Let $\mathbf{P}_L$ and $\mathbf{P}_R$ be two linear operators mapping $\mathbb{R}^N$ onto itself, being "*sparse*" in the following sense: for arbitrary $\mathbf{x} \in \mathbb{R}^N$ the products $\mathbf{P}_L\mathbf{x}$ and $\mathbf{P}_R\mathbf{x}$ require about the same number of operations as would be required if $\mathbf{P}_L$ and $\mathbf{P}_R$ were sparse *matrices*. In practice $\mathbf{P}_L$ and $\mathbf{P}_R$ are often chosen to be inverses of sparse, triangular matrices; in that case the action of these operators can be carried out as back-substitutions, which require the same amount of work as is needed for a sparse matrix-vector multiplication.

Define $\tilde{\mathbf{x}}$ by $\mathbf{P}_R\tilde{\mathbf{x}} = \mathbf{x}$ and $\tilde{\mathbf{b}}$ by $\tilde{\mathbf{b}} = \mathbf{P}_L\mathbf{b}$; then the preconditioned version of the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ reads:

$$\mathbf{B}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \tag{4.2}$$

where $\mathbf{B} = P_L\mathbf{A}\mathbf{P}_R$ is the preconditioned matrix. The operators $\mathbf{P}_L$ and $\mathbf{P}_R$ are chosen such that $\mathbf{B}$ has better spectral properties with respect to the CG algorithm. We can regard the product $\mathbf{P}_R\mathbf{P}_L$ as an approximate inverse of $\mathbf{A}$ in some sense.

There are several possibilities for implementing the preconditioning matrices in the CGS algorithm. The variant used in our numerical experiments reads as follows:

$\mathbf{r}_0 := \mathbf{P}_L(\mathbf{b} - \mathbf{A}\mathbf{x}_0)$; $\tilde{\mathbf{r}}_0 := \mathbf{r}_0$;
$\mathbf{q}_0 := \mathbf{p}_{-1} := \mathbf{0}$;
$\rho_{-1} := 1$; $n := 0$;
while *residual* > *tolerance* **do**
begin

$$\rho_n := \tilde{\mathbf{r}}_0^T \mathbf{r}_n;\ \beta_n := \frac{\rho_n}{\rho_{n-1}};$$
$$\mathbf{u}_n := \mathbf{r}_n + \beta_n \mathbf{q}_n;$$
$$\mathbf{p}_n := \mathbf{u}_n + \beta_n(\mathbf{q}_n + \beta_n \mathbf{p}_{n-1});$$
$$\mathbf{v}_n := \mathbf{P}_L \mathbf{A} \mathbf{P}_R \mathbf{p}_n; \tag{4.3}$$
$$\sigma := \tilde{\mathbf{r}}_0^T \mathbf{u}_n;\ \alpha_n := \frac{\rho_n}{\sigma_n};$$
$$\mathbf{q}_{n+1} := \mathbf{u}_n - \alpha_n \mathbf{v}_n;$$
$$\mathbf{v}_n := \alpha_n \mathbf{P}_R(\mathbf{u}_n + \mathbf{q}_{n+1});$$
$$\mathbf{x}_{n+1} := \mathbf{x}_n + \mathbf{v}_n;$$
$$\mathbf{r}_{n+1} := \mathbf{r}_n - P_L \mathbf{A} \mathbf{v}_n;$$
$$n := n + 1$$

end;

In this algorithm, $\mathbf{x}_n$ approximates the solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ (not $\tilde{\mathbf{x}}$ in (4.2)).

An important class of preconditioning methods is based on *incomplete factorisations* of the matrix **A**. An incomplete factorisation of **A** consists of a lower triangular matrix **L** and an upper triangular matrix **U** with a prescribed (sparse) nonzero pattern and satisfying

$$\mathbf{L}\mathbf{U} = \mathbf{A} + \mathbf{E}.$$

The deviation matrix **E** is often very sparse itself, but this is not necessary in general (as in the case of *incomplete line LU factorisation*). The incomplete *LU factorisation* used most frequently can be obtained by a slightly modified Gaussian elimination procedure. Let $NZ(\mathbf{A})$ denote the set of pairs $[i, j]$ for which the entries a_{ij} of the matrix **A** are nonzero, the *nonzero pattern of* **A**. Then the incomplete Gauss algorithm reads:

for $k := 1$ **step** 1 **until** N **do**
begin
  for $i := k+1$ **step** 1 **until** N **do**
  begin

$$a_{ik} := \frac{a_{ik}}{a_{kk}};$$

    for $j := k+1$ **step** 1 **until** N **do** (4.4)
    begin
      if $[i, j] \in NZ(\mathbf{A})$ **then**
      begin
        $corr := -a_{ik}a_{kj}$;
        $\alpha_{ij} := a_{ij} + corr$
      end
    end
  end
end;

In the case of a compactly stored matrix **A**, the integers i and j do not get all values in the domain of the loops; only values of i for which $(i, k) \in NZ(\mathbf{A})$ and values of j

for which $(k,j) \in NZ(\mathbf{A})$ are necessary. Coding this is the most difficult part of the incomplete Gauss algorithm.

The upper triangular part of the resulting matrix $\mathbf{A}$ is now to be understood as the matrix U, the strictly lower triangular part together with the unit diagonal matrix as the matrix L in the factorisation. We denote the preconditioning based on the above algorithm by ILU.

Other possibilities will not be shown in detail in this paper. We mention Gustaffson's variant, presented in [9], *modified incomplete LU factorisation*, denoted by MILU, in which variant the quantity *corr* in the algorithm is added to the main diagonal of the matrix whenever (i,j) is not in the nonzero pattern. Generalisations can also be obtained by choosing other sets of pairs (i,j) instead of $NZ(\mathbf{A})$ (Meijerink and Van der Vorst [15], [16]).

With the matrices $\mathbf{L}$ and $\mathbf{U}$ as obtained with the above algorithm, we can define preconditionings as follows:

$$\begin{aligned} &\mathbf{P}_L = \mathbf{L}^{-1}, \qquad \mathbf{P}_R = \mathbf{U}^{-1}, \\ &\mathbf{P}_L = \mathbf{U}^{-1}\mathbf{L}^{-1}, \qquad P_R = \mathbf{I}, \\ &\mathbf{P}_L = \mathbf{I}, \qquad \mathbf{P}_R = \mathbf{U}^{-1}\mathbf{L}^{-1}. \end{aligned}$$

Each of these alternatives yields a matrix $\mathbf{B}$ with the same spectrum. The choice between them depends on arguments concerning the implementation. The existence and the numerical stability of ILU factorisations have been proved for cases where $\mathbf{A}$ is an M-matrix. In practice this is not a severe restriction. Van der Vorst in [21] and [22] has shown that in some important cases where $\mathbf{A}$ is not an M-matrix, a suitable preconditioning is obtained by applying the ILU algorithm to the matrix $\mathbf{A}+\mathbf{D}$ where $\mathbf{D}$ is some positive diagonal matrix.

Another type of incomplete factorisation, used as preconditioning, can be applied if $\mathbf{A}$ is a block-tridiagonal matrix. This is often the case if $\mathbf{A}$ arises from discretisations of second-order elliptic partial differential equations. It is called *incomplete line LU factorisation*, and denoted by ILLU. Let $\mathbf{L}$ and $\mathbf{U}$ be the strictly *block-lower* and strictly *block-upper* parts of $\mathbf{A}$ and let $\mathbf{D}_\mathbf{A}$ be the *block-diagonal* part. A block-diagonal matrix $\mathbf{D}$ is constructed, with the same nonzero pattern as $\mathbf{D}_\mathbf{A}$, satisfying

$$\mathbf{A} = (\mathbf{L}+\mathbf{D})\mathbf{D}^{-1}(\mathbf{U}+\mathbf{D}) + \mathbf{E}$$

where $\mathbf{E}$ is a (block-diagonal) matrix, having a nonzero pattern which is complementary to that of $\mathbf{D}_\mathbf{A}$, thus having a zero tridiagonal part. Details are described in [5], [14], and [19].

The ILLU factorisation has the advantage of apparently being less sensitive to special directions in the problem, as, for instance, the convection direction in the convection-diffusion equation.

Preconditioning with ILLU is done by choosing (for instance):

$$\mathbf{P}_L = (\mathbf{L}+\mathbf{D})^{-1}, \qquad \mathbf{P}_R = (\mathbf{U}+\mathbf{D})^{-1}\mathbf{D}.$$

Numerical experiments in this paper were carried out using ILU as well as ILLU preconditioning.

5. Numerical experiments. In this section the applicability of the CGS algorithm is demonstrated for some test examples belonging to the class of convection diffusion equations. In examples (1) and (2), the behaviour of the preconditioned CGS is compared with the author's implementation of the Bi-CG and the ORTHOMIN

algorithms. Examples (3) and (4) are taken from Van der Vorst [21], for a comparison of CGS with the Manteuffel variant of the CHEBYSHEV algorithm.

5.1. Operation counts and storage requirements. If the algorithms used in this section are implemented carefully in order to avoid double computations of the same quantities, the computational complexities are:

$$
\begin{aligned}
&W_{\text{Bi-CG}} = W_{\text{PREC}} + 1 + W_{\mathbf{P}_L} + W_{\mathbf{A}} + \text{steps} \times [15 + 2 \times (W_{\mathbf{P}_L} + W_{\mathbf{A}} + W_{\mathbf{P}_R})],\\
&W_{\text{ORTHOMIN}} = W_{\text{PREC}} + 1 + W_{\mathbf{A}} + \text{steps} \times [40 + W_{\mathbf{P}_L} + W_{\mathbf{A}} + W_{\mathbf{P}_R}],\\
&W_{\text{CGS}} = W_{\text{PREC}} + 1 + W_{\mathbf{P}_L} + W_{\mathbf{A}} + \text{steps} \times [18 + 2 \times (W_{\mathbf{P}_L} + W_{\mathbf{A}} + W_{\mathbf{P}_R})].
\end{aligned} \tag{5.1}
$$

The quantity W_{PREC} denotes the computational work for constructing the preconditioning operators $\mathbf{P}_L$ and $\mathbf{P}_R$. The quantities $W_{\mathbf{P}_L}$, $W_{\mathbf{A}}$, and $W_{\mathbf{P}_R}$ denote the computational work required for the evaluation of $\mathbf{P}_L\mathbf{x}$, $\mathbf{Ax}$, and $\mathbf{P}_R\mathbf{x}$, respectively. All quantities *work* are measured in *floating point operations* (*flops*) *per degree of freedom.* The quantity *steps* denotes the number of iteration steps. It must be noted that the ORTHOMIN procedure requires a different amount of work during the first five steps. The precise operation count reads as follows:

$$
W_{\text{ORTHOMIN}}(\text{step}_k) = 4 + 6k + W_{\mathbf{P}_L} + W_{\mathbf{A}} + W_{\mathbf{P}_R}. \tag{5.2}
$$

In the *work* an extra inner product for checking the norm reduction of the residuals is included. In ORTHOMIN the Euclidean norm of the (recursively obtained) true residual is measured each step, whereas in the other two algorithms the preconditioned residuals ($\mathbf{P}_L(\mathbf{Ax}_n - \mathbf{b})$) are considered. After termination of the process the true residual reduction was compared with the measured quantities; in all cases the differences were not significant.

Apart from storage requirements for storing the operators $\mathbf{P}_L$, $\mathbf{P}_R$, and $\mathbf{A}$, and the vectors $\mathbf{x}$ and $\mathbf{b}$, the number S of N-vectors used by the three algorithms is listed below:

$$
S_{\text{Bi-CG}} = 6, \quad S_{\text{ORTHOMIN}} = 13, \quad S_{\text{CGS}} = 6. \tag{5.3}
$$

The storage requirements for the preconditioning operators as well as the amounts of work W_{PREC}, $W_{\mathbf{P}_L}$, $W_{\mathbf{A}}$, and $W_{\mathbf{P}_R}$ are strongly dependent on the implementation. Dependent on the type of grid (5-point- or 7-point finite difference star, or a, possibly irregular, finite-element grid) several saving devices for computing time have been developed (see, for instance, Eisenstat [6]). We consider the possibility of computing the product $\mathbf{P}_L\mathbf{AP}_R\mathbf{x}$ by $\mathbf{x} - \mathbf{P}_L\mathbf{EP}_R\mathbf{x}$, in the case of ILU preconditioning, saving operations as well as storage (a similar trick exists for the ILLU preconditioning). In Table 1 these reduced amounts of work are given between brackets.

In the general case of a finite-element discretisation of an elliptic PDE the cheap variants are not always applicable, so, except in the comparison with the CHEBYSHEV

TABLE 1
Computational work and storage.

DISCR	PREC	W_{PREC}	$W_{\mathbf{P}_L}$	$W_{\mathbf{A}}$	$W_{\mathbf{P}_R}$	$W_{\mathbf{P}_L\mathbf{AP}_R}$	STORAGE
5-pts	ILU	10	4	9 (13)	5	18 (13)	5 (3)
5-pts	ILLU	19	7	9 (9)	12	28 (26)	3 (3)
7-pts	ILU	21	6	13 (17)	7	26 (17)	7 (3)

algorithm, we use the straightforward amounts of work throughout. This acts slightly in favour of the operation count for ORTHOMIN, as can be understood from (5.1).

In the tables of results the total work per degree of freedom is based on (5.1) and (5.2), using Table 1.

5.2. Test examples.

PROBLEM 1. Convection Diffusion equation on the unit square $\Omega = (0, 1) \times (0, 1)$.

$$-\varepsilon\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+\cos(\alpha)\frac{\partial u}{\partial x}+\sin(\alpha)\frac{\partial u}{\partial y}=0, \qquad u(x, y)=x^2+y^2 \quad \text{and } \partial\Omega.$$

This equation is used for comparison of the Bi-CG, ORTHOMIN, and CGS algorithms. A 5-point hybrid difference scheme is used on several square grids. That is, upwind differencing is done only if necessary for maintaining diagonal dominance; otherwise central differencing is applied. The dimension of the algebraic system of equations N corresponds to the number of interior points of the grid

$$N=\left[\frac{1}{\Delta x}-1\right]^2$$

with $\Delta x = \Delta y$.

PROBLEM 2. Berkeley problem on $\Omega = (-1, +1) \times (0, 1)$.

$$-\varepsilon\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+v_x\frac{\partial u}{\partial x}+v_y\frac{\partial u}{\partial y}=0$$

with

$$v_x = 2y(1-x^2), \qquad v_y = -2x(1-y^2),$$

$$u(x, y)=0, \quad \begin{array}{ll} x=-1, & 0 \leqq y \leqq 1, \\ x=+1, & 0 \leqq y \leqq 1, \\ y=+1, & -1 \leqq x \leqq 1, \end{array}$$

$$u(x, 0) = 1+\tanh(10(2x+1)), \qquad y=0, \quad -1 \leqq x \leqq 0,$$

$$\frac{\partial u}{\partial n}=0, \qquad y=0, \quad 0 \leqq x \leqq 1.$$

This problem, described in [10] and [20], is discretised on a rectangular grid, using 5-point finite differences, the boundary $\partial\Omega$ chosen between the appropriate exterior two gridlines (virtual boundary). Central as well as hybrid differencing is applied. Since the tests on Problem 1 show that Bi-CG is far less efficient if compared with ORTHOMIN and CGS, we have compared only the latter two methods.

In both Problem 1 and Problem 2 the starting vector is chosen uniformly quasi-randomly distributed between -2 and 2.

PROBLEM 3. Van der Vorst problem (5.1) on $\Omega = (0, 30) \times (0, 30)$.

$$-\frac{\partial^2 u}{\partial x^2}-\frac{\partial^2 u}{\partial y^2}+\beta\left(\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}\right)+u=1$$

with

$$u(x, y) \equiv 1 \quad \text{on } \partial\Omega.$$

This equation, taken from [21], is discretised on a square grid with $\Delta x = \Delta y = 1$. As starting vector the null-vector is chosen.

PROBLEM 4. Van der Vorst problem (5.4) on $\Omega = (0, 1) \times (0, 1)$.

$$-\frac{\partial^2 u}{\partial x^2}+\frac{\partial u}{\partial x}+(1+y^2)\left(-\frac{\partial^2 u}{\partial y^2}+\frac{\partial u}{\partial y}\right)=f(x, y)$$

with Dirichlet type boundary condition and right-hand side such that

$$u(x, y) = e^{x+y}+x^2(1-x)^2 \ln (1+y^2).$$

This equation, also taken from [21], is discretised on square grids with $\Delta x = \Delta y = 1/16$ and $\Delta x = \Delta y = 1/30$, respectively. Again the null vector is chosen as the starting vector.

5.3. Numerical results. Table 2 shows the behaviour of Bi-CG, ORTHOMIN, and CGS on Problem 1, for values of ε, α, and N as listed. Five orthogonal vectors are used in the ORTHOMIN procedure. The two versions of the ILU preconditioning are different in the choice of the nonzero pattern. In the case of ILU (7) the nonzero

TABLE 2
Problem 1 with $\alpha = -30$ deg.

	Average work per digit, required to obtain a reduction $< 10^{-12}$.							
			Reduction, iterations			Work per digit		
ε	N	Prec.	B-CG	ORTH	CGS	Bi-CG	ORTH	CGS
10^{-1}	100	ILU(5)	13 (18)	11 (20)	13 (11)	75	100	48
		ILU(7)	13 (12)	13 (13)	15 (8)	66	62	40
		ILLU	12 (9)	13 (10)	13 (5)	55	47	32
	400	ILU(5)	6 (20)	6 (20)	14 (20)	172	192	81
		ILU(7)	12 (20)	10 (20)	12 (12)	114	125	71
		ILLU	12 (15)	13 (17)	13 (10)	91	86	58
	1600	ILU(5)	3 (20)	4 (20)	5 (20)	336	290	203
		ILU(7)	5 (20)	5 (20)	9 (20)	289	239	153
		ILLU	9 (20)	9 (20)	14 (17)	171	152	95
	6400	ILLU	9 (40)	9 (40)	13 (32)	304	305	189
10^{-4}	100	ILU(5)	12 (11)	12 (20)	15 (8)	48	91	31
		ILU(7)	13 (8)	12 (15)	23 (7)	44	77	24
		ILLU	15 (4)	15 (4)	22 (3)	21	13	12
	400	ILU(5)	11 (20)	5 (20)	12 (13)	92	206	60
		ILU(7)	13 (12)	11 (20)	12 (8)	63	115	45
		ILLU	15 (5)	15 (5)	16 (3)	26	18	16
	1600	ILU(5)	4 (20)	3 (20)	8 (20)	268	380	131
		ILU(7)	7 (20)	2 (20)	15 (18)	189	561	89
		ILLU	14 (6)	15 (6)	12 (3)	34	24	21
	6400	ILLU	14 (8)	13 (7)	12 (4)	45	33	27

patterns of **L** and **U** correspond to a 7-point finite-difference star, whereas in the case of ILU (5), these patterns correspond to the ordinary 5-point star. In all cases the processes are terminated either at a residual reduction by a factor of 10^{-12}, or after 20 steps, except for the tests on an 80×80 grid, where a maximum of 40 iterations is admitted. Although the convergence behaviour of both Bi-CG and CGS is not linear, the efficiency of the methods can be measured very well by the mean work per degree of freedom necessary for gaining one decimal digit (corresponding to a reduction by a factor 10^{-1}). This quantity, denoted by *work per digit*, is defined as total work/$-\log_{10}$(reduction) and measured in flops; the quantity *reduction* is measured in decimal digits; the numbers between parentheses denote the numbers of iteration steps actually used.

Comparing the efficiency of Bi-CG and CGS, we see that the latter method requires about 60 percent of the work required by the former method. This is in reasonable agreement with the theoretically expected 50 percent. A reason for the deviation might be the superlinear convergence of these algorithms. Comparing the three methods we see that, especially in the "slower" cases, CGS is significantly more efficient than the other two methods.

Furthermore the superiority of ILLU preconditioning is shown (at least for this problem). The chosen value of α (-30 degrees) belongs to the class of medium difficult angles for this problem. In [19], an extensive comparison of ILU and ILLU preconditionings is made for rather many different values of α. In the case of the 80×80 grid, only the ILLU preconditioning is used, since the other preconditioners yield rather poor results for the three methods.

Finally, it is interesting to observe that ORTHOMIN suffers more from poor preconditioning (as ILU sometimes is) than does CGS.

In Fig. 1, the convergence characteristics of the methods are plotted for a few cases. The ORTHOMIN curves show the most linear convergence. Since the first steps of ORTHOMIN are considerably cheaper than the later ones, ORTHOMIN may be regarded as an efficient choice for results of low accuracy.

In some cases the residuals produced by Bi-CG and CGS make quite large jumps in the "wrong" direction. This has to do with the nonpositive definiteness of the "inner products" used. Since this phenomenon may be responsible for a severe loss of significant digits, it is necessary, in the author's opinion, to develop an escape from this problem.

Table 3 shows the behaviour of ORTHOMIN and CGS, with ILLU preconditioning, in solving Problem 2. The equation is discretised on a 100×50 grid. As may be expected, when using central differences, for small ε the matrix loses too much diagonal dominance, and the iterations do not converge. In the case $\varepsilon = 10^{-3}$, the convergence of ORTHOMIN has to be considered as spurious: the residual norm did not change after the seventh step. Apparently CGS is less sensitive to loss of diagonal dominance. For $\varepsilon \leqq 10^{-3.5}$ both methods diverge, however, ORTHOMIN does not show this as clearly as CGS does.

In Table 4 CGS is compared with Manteuffel's version of the CHEBYSHEV algorithm, with adaptive parameter estimation. The results for the latter algorithm were presented by Van der Vorst in [21]. Since we do not know the exact operation count for the adaptive part of the algorithm, only estimates of the quantity *work* are given. We do not take initial work into account for both methods. As preconditioning we used the ILU factorisation in the so-called *Eistenstat* implementation [6], [21]. In this implementation, $W_{\mathbf{P}_L\mathbf{A}\mathbf{P}_R}$ is 11 flops. The work for the CHEBYSHEV algorithm, including the Euclidean norm calculation, is 7 flops. The total work per iteration step

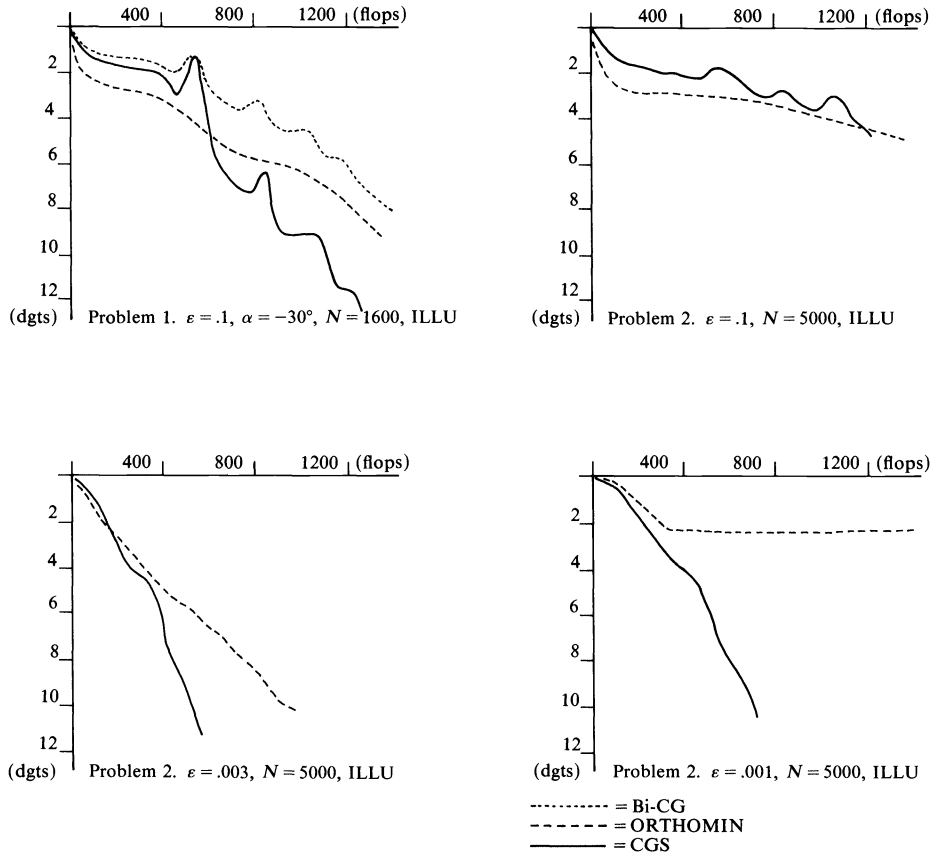


FIG. 1. *Convergence characteristics.*

TABLE 3
Problem 2 on a 100×50 *grid.*

Differences	ε	Reduction, iterations		Work per digit	
		ORTH	CGS	ORTH	CGS
Central	10^{-1}	4.8 (20)	4.3 (15)	273	270
	$10^{-1.5}$	4.5 (20)	4.9 (15)	267	236
	10^{-2}	10.4 (20)	11.7 (11)	125	73
	$10^{-2.5}$	10.2 (15)	11.2 (7)	94	49
	10^{-3}	2.4 (20)	10.7 (9)	538	66
Hybrid	10^{-2}	9.9 (20)	11.1 (11)	132	77
	$10^{-2.5}$	10.9 (13)	11.1 (7)	76	50
	10^{-3}	10.7 (8)	12.3 (5)	45	33
	$10^{-3.5}$	10.4 (6)	13.6 (4)	33	24
	10^{-4}	11.1 (5)	12.8 (3)	25	20

TABLE 4
Problems 3 and 4.

Problem	Reduction, iterations		Work per digit	
	CHEB	CGS	CHEB	CGS
3, $\beta = 4.0$	$9.0\,10^{-9}$ (25)	$4.5\,10^{-10}$ (9)	56	39
3, $\beta = 20.0$	$8.3\,10^{-9}$ (37)	$8.1\,10^{-9}$ (16)	82	79
4, $N = 15\,15$	$2.0\,10^{-8}$ (38)	$4.3\,10^{-9}$ (12)	89	57
4, $N = 29\,29$	$9.1\,10^{-9}$ (45)	$2.6\,10^{-9}$ (22)	100	102

is therefore 18 flops for the CHEBYSHEV algorithm, and 40 flops for the CGS algorithm.

5.4. Conclusions. For large, sparse, nonsymmetric linear systems, we may regard the combination of CGS with ILLU preconditioning as a competitive solver, at least for problems that are not too large, and when high accuracies are required. From Table 2 we may conclude that the average work for solving convection-diffusion equations with medium ε in two dimensions is roughly proportional to $N\sqrt{N}$. Although this is comparable with the SOR method, the proportionality coefficient is much smaller, and also the (critical) choice for a parameter is avoided. Nevertheless, for very large N we may expect that *multiple-grid* methods will be preferable to this combination of CGS with ILLU preconditioning (see, for instance, [19] and [24]). As has been shown many times, the quality of a method is strongly dependent on the quality of the preconditioning used. For that reason, it is important that codes be developed for ILLU factorisation of more complicated matrices, such as those arising from finite-element discretisations.

Acknowledgments. The author thanks Professors Piet Wesseling and Henk Van der Vorst and others at Delft for their stimulating discussions, and is grateful to Koos Meijerink for testing the CGS method under "real life" circumstances. Finally the author is indebted to the referees for some valuable remarks.

REFERENCES

[1] O. AXELSSON, *A Generalised Conjugate Direction Method and Its Application on a Singular Perturbation Problem*, Lecture Notes in Mathematics 773, Springer-Verlag, Berlin, Heidelberg, New York, 1980, pp. 1-11.

[2] ———, *Conjugate gradient type methods for unsymmetric and inconsistent systems of linear equations*, Linear Algebra Appl., 29 (1980), pp. 1-16.

[3] ———, *Solution of Linear Systems of Equations: Iterative Methods*, Lecture Notes in Mathematics 572, Springer-Verlag, Berlin, Heidelberg, New York, 1977, pp. 2-51.

[4] P. CONCUS AND G. H. GOLUB, *A Generalised Conjugate Gradient Method for Unsymmetric Systems of Linear Equations*, Lecture Notes in Economics and Mathematical Systems 139, Springer-Verlag, Berlin, Heidelberg, New York, 1976.

[5] P. CONCUS, G. H. GOLUB, AND G. MEURANT, *Block preconditioning for the conjugate gradient method*, Report LBL-14856, Lawrence Berkeley Laboratories, University of California, Berkeley, CA, 1982.

[6] S. C. EISENSTAT, *Efficient implementation of a class of preconditioned conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 1-4.

[7] H. C. ELMAN, Y. SAAD, AND P. E. SAYLOR, *A hybrid Chebyshev Krylov subspace algorithm for solving nonsymmetric systems of linear equations*, Tech. Report 301, Department of Computer Science, Yale University, New Haven, CT, 1984; SIAM J. Sci. Statist. Comput., 7 (1986), pp. 840-855.

[8] R. FLETCHER, *Conjugate Gradient Methods for Indefinite Systems*, Lecture Notes in Mathematics 506, Springer-Verlag, Berlin, Heidelberg, New York, 1976, pp. 73–89.
[9] I. GUSTAFSSON, *A class of first order factorisation methods*, BIT, 18 (1978), pp. 142–156.
[10] A. HUTTON AND R. SMITH, *J. Numer. Heat Transfer*, to appear.
[11] D. S. KERSHAW, *The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations*, J. Comp. Phys., 26 (1978), pp. 43–65.
[12] T. A. MANTEUFFEL, *Adaptive procedure for estimating parameters for the nonsymmetric Tchebychev iteration*, Numer. Math., 31 (1978), pp. 183–208.
[13] ———, *The Tchebyschev iteration for nonsymmetric linear systems*, Numer. Math. 28 (1977), pp. 307–327.
[14] J. A. MEIJERINK, *Iterative methods for the solution of linear equations based on incomplete block factorisation of the matrix*, paper SPE 12262, Proc. 7th SPE Symposium on Reservoir Simulation, San Francisco, CA, November 1983, pp. 297–304.
[15] 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*, Math. Comp., 31 (1977), pp. 148–162.
[16] ———, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, J. Comp. Phys., 44 (1981), pp. 134–155.
[17] C. C. PAIGE AND M. A. SAUNDERS, LSQR: *An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71.
[18] Y. SAAD AND H. SCHULTZ, *Conjugate gradient-like algorithms for solving nonsymmetric linear systems*, Res. Report 283, Department of Computer Science, Yale University, New Haven CT, 1983.
[19] P. SONNEVELD, P. WESSELING, AND P. M. DE ZEEUW, *Multigrid and conjugate gradient methods as convergence acceleration techniques*, in Proc. Multigrid Conference, H. Holstein and D. Paddon, eds., Bristol, 1983.
[20] C. THOMPSON AND N. WILKES, *Experiments with higher order finite difference formulae*, Report AERE R-10493, United Kingdom Atomic Energy Authority, Harwell, UK, 1983.
[21] H. A. VAN DER VORST, *Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from* PDE-*Problems*, J. Comp. Phys., 44 (1981), pp. 1–19.
[22] ———, *Preconditioning by incomplete decompositions*, Ph.D. thesis, ACCU-series 32, Academic Computer Centre Utrecht, University of Utrecht, the Netherlands, 1982.
[23] P. W. VINSOME, *Orthomin, an Iterative Method for Solving Sparse Sets of Simultaneous Linear Equations*, paper SPE 5729, Society of Petroleum Engineers of AIME, 1976.
[24] P. WESSELING AND P. SONNEVELD, *Numerical Experiments with a Multiple Grid- and a Preconditioned Lanczos Type Method*, Lecture Notes in Mathematics 771, Springer-Verlag, Berlin, Heidelberg, New York, 1980, pp. 543–562.
[25] O. WIDLUND, *A Lanczos method for a class of nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 15 (1978), pp. 801–812.
[26] D. M. YOUNG AND K. C. JEA, *Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl., 34 (1980), pp. 159–194.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 53-57, January 1989
© 1989 Society for Industrial and Applied Mathematics
005

A STORAGE-EFFICIENT *WY* REPRESENTATION FOR PRODUCTS OF HOUSEHOLDER TRANSFORMATIONS*

ROBERT SCHREIBER† AND CHARLES VAN LOAN‡

Abstract. A product $Q = P_1 \cdots P_r$ of $m \times m$ Householder matrices can be written in the form $Q = I + WY^T$, where W and Y are each $m \times r$. This is called the WY representation of Q. It is of interest when implementing Householder techniques in high-performance computing environments that are especially good at matrix-matrix multiplication. In this note a storage-efficient way to implement the WY representation is described. In particular, it is shown how the matrix Q can be expressed in the form $Q = I + YTY^T$, where $Y \in R^{m \times r}$ and $T \in R^{r \times r}$ with T upper triangular. Usually $r \ll m$ and so this "compact" WY representation requires less storage. When compared with the recent block-reflector strategy proposed by Schreiber and Parlett [*SIAM J. Numer. Anal.*, 25 (1988), pp. 189-205], the new technique still has a storage advantage and involves a comparable amount of work.

Key words. block methods, Householder transformations

AMS(MOS) subject classifications. 65F, 65W

Introduction. Many important eigenvalue and least squares methods rely on Householder matrices, i.e., matrices of the form $H = I - 2vv^T$, where v has unit 2-norm. (The normalization of the Householder vector v is unnecessary in practice, but convenient for exposition.) Householder matrices are orthogonal and can be used to zero selected portions of a matrix (see Golub and Van Loan [4]). Important implementations are discussed in the LINPACK and EISPACK manuals, (Dongarra et al. [3], Smith et al. [6]).

The QR factorization of an $m \times n$ matrix A is a good setting for describing Householder matrix use. In this application Householder matrices $H_1, \cdots, H_n$ are generated such that $H_n \cdots H_2 H_1 A = R$ is upper triangular.

ALGORITHM 1 (Householder QR).
For $k = 1:n$
  Determine Householder H_k such that if $z = H_k A(1:m, k:k)$
    then $z(k+1:m) = 0$.
  $A \leftarrow H_k A$
end k

Here, the notation $A(i:j, p:q)$ denotes the submatrix of A consisting of rows i through j and columns p through q. For vectors, $z(i:j)$ denotes the subvector consisting of components i through j.

The Householder update $A \leftarrow H_k A$ is "rich" in matrix-vector multiplication. Unfortunately, many of the new supercomputing architectures require code that is rich in matrix-matrix multiplication in order to attain near-peak performance. This is not surprising, because parallel-processing speeds have outstripped memory bandwidths.

* Received by the editors August 26, 1987; accepted for publication May 16, 1988.

† Research Institute for Advanced Computer Science, MS 230-5, National Aeronautics and Space Administration-Ames Research Center, Moffett Field, California 94035. The work of this author was partially supported by Office of Naval Research contract N00014-86-K-0610 and by Army Research Office contract DAAL-03086-K-0112.

‡ Department of Computer Science, Upson Hall, Cornell University, Ithaca, New York 14853. The work of this author was partially supported by Office of Naval Research contract N00014-83-K-640, National Science Foundation contract DCR 86-2310, and the Army Research Office through the Mathematical Sciences Institute of Cornell University.

Matrix-vector products suffer from the need for one memory reference per multiply-add. The corresponding ratio for square matrix-matrix multiplication is one to n. This accounts for the current interest in the level-three BLAS and the increasingly vigorous search for efficient block algorithms. (See Dongarra [2].)

Along these lines there has been some recent work on "block" Householder methods. Two such techniques are of interest to us. They are the WY representation due to Bischof and Van Loan [1] and the block reflector method of Schreiber and Parlett [5]. Both of these techniques can be used to solve what we shall refer to as problem (P):

(P) Given $B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$, find an orthogonal Q so $Q^T B = C = \begin{bmatrix} C_1 \\ 0 \end{bmatrix}$.

Here, $B, C \in R^{m\times r}$ and $B_1, C_1 \in R^{r\times r}$. In the WY approach the solution to problem (P) is represented in the form

$$Q = I + WY^T, \qquad W, Y \in R^{m\times r}, \tag{1}$$

where Y is lower trapezoidal, i.e., $y_{ij} = 0$ if $i < j$. The submatrix C_1 is upper triangular. Q is a rank-r correction to the identity, and so can be regarded as a generalization of a Householder matrix.

A different solution to problem (P) is obtained by the block reflector approach. It has the form

$$Q = I - GG^T, \qquad G \in R^{m\times r}, \tag{2}$$

and obviously can be regarded as a block Householder matrix. It requires mr storage locations to represent Q in this fashion, about half of those required by the WY representation. The block reflector solution to problem (P) does *not* in general yield an upper triangular C_1.

The point of this note is to show how to modify the WY representation so that only mr storage is necessary. In particular, we show how to write the matrix Q in (1) as

$$Q = I + YTY^T, \tag{3}$$

where $Y \in R^{m\times r}$ (lower trapezoidal) and $T \in R^{r\times r}$ (upper triangular). We refer to (3) as the *compact WY representation* of Q and we discuss its use in the design of a block Householder QR procedure.

Block QR procedures. Any of the above block Householder representations can be used to implement a block QR factorization procedure that is rich in matrix multiplication. Suppose $A \in R^{m\times n}$ and that $n = rN$.

ALGORITHM 2 (Block Householder QR).
For $k = 1:N$
  $s \leftarrow (k-1)r+1$
  Determine block Householder Q such that if
    $Q^T A(s:m, s:s+r-1) = C$ then C is zero below row r.
  $A(s:m, s:n) \leftarrow Q^T A(s:m, s:n)$
end

Note that the update of $B = A(s:m, s:n)$ is rich in matrix multiplication if Q is represented as $I + WY^T$ or $I - GG^T$ or $I + YTY^T$, i.e.,

$$B \leftarrow (I + WY^T)^T B = B + Y(W^T B).$$

In the case of the WY representation, the kth W and Y are generated from the Householder matrices $P_j = I - 2v_j v_j^T$, $j = 1:r$, that upper triangularize the submatrix $A(s:m, s:s+r-1)$. The procedure is simple.

ALGORITHM 3 (WY Generation).
For $j = 1:r$
  If $j = 1$
    $w \leftarrow [-2v_1]$; $Y \leftarrow [v_1]$
  else
    $z \leftarrow -2(I + WY^T)v_j$; $W \leftarrow [W \quad z]$; $Y \leftarrow [Y \quad v_j]$
  endif
end

It follows that $I + WY^T = P_1 \cdots P_r$ and so A is reduced to upper triangular form when Algorithm 2 is implemented with block Householders in WY form. Note that each Y is just the aggregation of the Householder vectors and so is lower trapezoidal. Thus, all the Y matrices fit into the zeroed portion of A as Algorithm 2 proceeds. If the W factors are saved, then a workspace of size

$$\sum_{k=1}^{N} (m-(k-1)r)r \approx mn - n^2/2$$

is required. This is a serious storage overhead although in many applications it is not necessary to store all the W matrices.

These storage concerns do not arise if the block reflector approach is taken. If we choose to write Q as $I - GG^T$ then the Schreiber-Parlett procedure can be used to generate the G matrices in Algorithm 2 as follows.

ALGORITHM 4 (G Generation).
—Compute Householders $P_1, \cdots, P_r$ such that $P_r \cdots P_1 A(s:m, s:s+r-1)$ is upper triangular.
—Let $\begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$ be the first r columns of $P_1 \cdots P_r$ with $U_1 \in R^{r \times r}$.
—Compute orthogonal $Z \in R^{r \times r}$ and symmetric nonnegative definite $M \in R^{r \times r}$ such that $U_1 = ZM$, the polar decomposition.
—Compute Cholesky factorization $LL^T = I + M$.
—Set

$$G = \begin{bmatrix} ZL \\ U_2 L^{-T} \end{bmatrix}.$$

A block reflector implementation of Algorithm 2 results in a block upper triangular reduction of A. Subsequent computations are necessary to obtain true triangular form. If all the G matrices are to be saved, then a workspace of size $Nr^2 = nr$ is required. (There is no room in the A array for the top $r \times r$ portions of each G.)

To sum up, the WY representation is simpler to compute but it requires a much larger workspace if all the block Householder factors are saved ($mn - n^2/2$ versus nr).

The compact *WY* representation. The idea behind the compact WY representation is to exploit a connection between the W and Y matrices in (1). Here is the main result that enables us to build up Y and T just as W and Y are constructed in Algorithm 3.

THEOREM 1 (Compact WY update). *Suppose $Q = I + YTY^T \in R^{m \times m}$ is orthogonal with $Y \in R^{m \times j}$ ($m > j$) and $T \in R^{j \times j}$ (upper triangular). Suppose $P = I - 2vv^T$ is a Householder matrix with $v \in R^m$ and $\|v\|_2 = 1$. If*

$$Q_+ = QP,$$

then

$$Q_+ = I + Y_+ T_+ Y_+^T,$$

where $Y_+ = [\,Y \quad v\,] \in R^{m\times(j+1)}$ *and*

$$T_+ = \begin{bmatrix} T & z \\ 0 & \rho \end{bmatrix}$$

with $\rho = -2$ *and* $z = -2TY^T v$.

Proof. Note that

$$\begin{aligned} I + Y_+ T_+ Y_+^T &= I + [\,Y \quad v\,]\begin{bmatrix} T & z \\ 0 & \rho \end{bmatrix}\begin{bmatrix} Y^T \\ v^T \end{bmatrix} \\ &= I + [\,Y \quad v\,]\begin{bmatrix} TY^T + zv^T \\ \rho v^T \end{bmatrix} \\ &= I + YTY^T + Yzv^T + \rho vv^T. \end{aligned}$$

This equals

$$Q_+ = QP = (I + YTY^T)(I - 2vv^T) = I + YTY^T - 2YTY^T vv^T - 2vv^T,$$

as long as we set $\rho = -2$ and $z = -2TY^T v$. □

Returning to Algorithm 2, if $P_j = I - 2v_j v_j^T$, $j = 1:r$, are the Householder matrices that upper triangularize $A(s:m, s:s+r-1)$ during the kth step, then here is how T and Y are determined so that $P_1 \cdots P_r = I + YTY^T$.

ALGORITHM 5 (YT Generation).

For $j = 1:r$
  if $j = 1$ then
    $Y \leftarrow [v_1]$; $T \leftarrow [-2]$
  else
    $z \leftarrow -2TY^T v_j$
    $Y \leftarrow [\,Y \quad v_j\,]$
    $T \leftarrow \begin{bmatrix} T & z \\ 0 & -2 \end{bmatrix}$
  endif
end j

Remarks. We conclude with a number of remarks concerning the compact WY representation and its use in block Householder schemes such as Algorithm 2. To begin with, if each triangular matrix T is saved, then a workspace of size $Nr^2/2 = nr/2$ is required. As with WY, the Y matrices fit in the zeroed portion of A and so no additional workspace is required for their storage. Thus, compact WY is the most storage efficient of the three representations discussed.

From the standpoint of actually generating the representative factors, $\{W, Y\}$, $\{G\}$, or $\{T, Y\}$, the compact WY representation is also the most efficient. This can be seen by comparing Algorithms 3, 4, and 5.

We next compare the cost of the update $B \leftarrow Q^T B$, where Q is in one of our three block Householder forms. If $B \in R^{n_1 \times n_2}$ and rank $(Q - I) = r$, then simple flop counts reveal that the updates

(a) $B \leftarrow (I + WY^T)^T B = B + Y \cdot (W^T \cdot B)$,

(b) $B \leftarrow (I - GG^T)^T B = B - G \cdot (G^T \cdot B)$,

(c) $B \leftarrow (I + YTY^T)^T B = B + (Y \cdot T^T) \cdot (Y^T \cdot B)$,
(d) $B \leftarrow (I + YTY^T)^T B = B + Y \cdot (T^T \cdot (Y^T \cdot B))$,

each require about the same amount of arithmetic: $2n_1 n_2 r$ flops. (If Y's trapezoidal structure is exploited, then the flops thus saved offset those needed for the T multiplications.) Of course, counting flops is a crude predictor of performance. There is some penalty associated with the compact *WY* form as it involves three matrix-matrix operations instead of just two. However, if $r \ll n_1$ or n_2, as is often the case, then the multiplications involving T are not significant. Note that there are two possible strategies for compact *WY* updates; see (c) and (d) above. The proper choice depends on n_1, n_2, r and the underlying architecture.

We mention that a *QR* factorization routine based on the compact *WY* representation is part of the new level-three BLAS LINPACK.

Last, but not least, the compact *WY* representation is stable. We omit the proof, as it closely follows the demonstration of stability given in Bischof and Van Loan [1] for the ordinary *WY* representation.

Acknowledgments. We thank Ralph Schmidt of Advent Corporation for participating in early formulations of this work. The referees were also very helpful. Jeremy de Croz has independently developed procedures based on the compact *WY* idea.

REFERENCES

[1] C. Bischof and C. Van Loan, *The WY representation for products of Householder matrices*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s2–s13.

[2] J. Dongarra, *Workshop on level three* BLAS, ANL-MCS Report TM-89, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, 1987.

[3] J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1978.

[4] G. H. Golub and C. Van Loan *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1983.

[5] R. Schreiber and B. N. Parlett, *Block reflectors: theory and computation*, SIAM J. Numer. Anal., 25 (1987), pp. 189–205.

[6] B. T. Smith, J. M. Boyle, J. Dongarra, B. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines*: EISPACK *Guide*, Second edition, Springer-Verlag, New York, 1976.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 58-71, January 1989

ANOVA MODEL FITTING VIA SPARSE MATRIX COMPUTATIONS: A FAST DIRECT METHOD*

GEORGE OSTROUCHOV†

Abstract. A fast and storage-efficient direct method for fitting analysis-of-variance models to unbalanced data is presented. This method exploits sparsity and rank deficiency of the model matrix and is based on orthogonal Givens factorization of a set of sparse columns of the model matrix. A class of matrices generated by index sets is defined and used to obtain results on linear dependencies between columns of a model matrix and fill during factorization. These results are used to develop an algorithm for the selection, ordering, and symbolic factorization of a set of sparse columns of the model matrix. This facilitates a fast and storage-efficient numerical factorization and solution. A comparison to both a standard direct algorithm and a general-purpose sparse least-squares algorithm shows that the new algorithm reduces time and storage by orders of magnitude for large models and is competitive for small models.

Key words. nonorthogonal analysis of variance, model matrix, sparse least squares, Givens factorization, design matrix

AMS(MOS) subject classifications. 62J10, 65F05, 65F50, 15A03

1. Introduction.

The computation involved in fitting an analysis-of-variance (ANOVA) model to unbalanced data is an explicit solution of a least-squares problem

$$\min_{\mathbf{b}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|_2, \tag{1}$$

where $\mathbf{y}$ is the response vector, $\mathbf{X}$ is the model matrix (often called the design matrix, [14]), $\mathbf{b}$ is a solution vector, and $\|\cdot\|_2$ is the Euclidean norm. Kennedy and Gentle [15] discuss some computational methods for this problem as well as the computation of associated sums of squares.

The available direct methods either employ an orthogonal factorization of the model matrix $\mathbf{X}$ or solve the normal equations by factoring or sweeping $\mathbf{X}^T\mathbf{X}$. Generally, methods utilizing the normal equations require less computation; however, they are less numerically stable and some precision is lost by the formation of $\mathbf{X}^T\mathbf{X}$. Although numerical stability considerations are usually not of concern in small ANOVA problems, they may become important in large problems.

For models involving several factors at several levels each, the model matrix $\mathbf{X}$ is large, so current direct methods require large amounts of time and storage. The use of iterative methods has been shown to reduce the time and storage requirements [8], [12], [16], [13]. The aim in this paper is to develop a fast and storage-efficient direct algorithm by exploiting the sparsity and rank deficiency of the model matrix. Sums of squares for model terms are obtained via the R()-notation [19] by fast fitting of appropriate models. Fast computation of sums of squares for special parameterizations of marginal effects in the presence of higher-order interactions or other estimable hypotheses of low rank is then facilitated by the availability of the sparse factored model matrix after fitting a model.

* Received by the editors July 2, 1986; accepted for publication (in revised form) August 25, 1988. This research was sponsored 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.

† Mathematical Sciences Section, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831.

The earliest references to the idea of exploiting sparsity of an ANOVA problem seem to have been made by Gentleman [4], [5], who discussed orthogonal Givens transformations for solving general sparse least-squares problems and noted their application to matrices arising in analysis of variance. Recently, [2] discusses the use of sparse matrix methodology in estimation of variance components by likelihood methods.

Givens rotations have been shown to be particularly suitable for the sparse least-squares problem [4], [5], [9]. Aside from the usual stability and precision advantages of operating directly on the $\mathbf{X}$ matrix, the matrix is more sparse than $\mathbf{X}^T\mathbf{X}$ in large models (§ 2). The model matrix is also highly rank deficient and this can be exploited to gain more sparsity (§ 3).

George and Heath [6] have developed an algorithm for large sparse least-squares problems that uses orthogonal Givens rotations to process the matrix one row at a time. The method was later extended in [10] to rank-deficient problems. A column ordering and a symbolic factorization are first performed on the symbolic structure of $\mathbf{X}^T\mathbf{X}$ and then the numerical factorization of $\mathbf{X}$ and solution takes place in a fixed data structure. The speed and storage requirements of the numerical factorization depend strongly on the performance of the symbolic phase. This algorithm thus requires two passes through the data; the first pass reads the nonzero structure of the problem and the second reads the numerical values. The use of this general sparse least-squares algorithm already results in substantial time and storage savings over full matrix methods, and this is shown in § 6. However, general sparse methods do not exploit the rank deficiency of a model matrix. Further substantial improvements can be made by recognizing the rank deficiency and special structure of an ANOVA model matrix. In this paper we develop a symbolic phase for the ANOVA problem that exploits the special structure and rank deficiency of the model matrix and is driven only by the model specification (thus the first pass through the data is eliminated). The new symbolic phase results in further substantial time and storage savings over the general sparse least-squares algorithm.

The development of the symbolic phase is based on index sets associated with model terms. Hemmerle [11] used model term index sets to define required sums and degrees of freedom in balanced ANOVA designs. Here, index sets are used to compare the sparsity of the model matrix to the cross-products matrix (§ 2), and in § 3, a class of matrices is defined by index sets. These matrices and index sets are then used to describe the model matrix and to derive linear relationships between its columns. In § 4, the index sets are further used to perform the symbolic ordering and factorization. The numerical phase and testing of estimable hypotheses are discussed in § 5. Finally, in § 6 the resulting sparse algorithm is compared to a standard full matrix algorithm and to a general-purpose sparse least-squares algorithm.

2. Model specification and notation. The usual model specification for a two-factor experiment with an interaction is

$$y_{ijk} \doteq \mu + a_i + b_j + ab_{ij}, \tag{2}$$

where $i = 1, 2, \cdots, I$, $j = 1, 2, \cdots, J$, and $k = 1, \cdots, n_{ij}$ with $n_{ij} \geq 0$. If we assume no interaction, we simply omit the ab_{ij} term; if b is nested within a, we omit the b_j term while keeping the ab_{ij} term. Note that one index (subscript) is associated with each factor; each term in the model has a unique set of associated indices (subscripts); and cells in the design are indexed by the total set of the factor indices. For example, a two-factor experiment has factors A and B, and their associated indices are i and j,

respectively; each model term in (2) has a unique subset of $\{i, j\}$; and each cell of the design is determined by an i-j combination. This forms the basis for notation used in this paper as well as for the computer representation of model terms and cells of the design.

A model with $t+1$ terms and f factors can be written as

$$y_{\phi k} \doteq \sum_{m=0}^{t} \tau_{\phi_m}, \tag{3}$$

where $\phi = \{i_1, \cdots, i_f\}$ is the total set of indices associated with cells of the experiment, $\phi_m \subset \phi$ is the set of indices associated with model term m, each index i_j has I_j levels, and $k = 1, \cdots, n_\phi$, where n_ϕ (≥ 0) gives the number of observations in cell ϕ. The two-factor model (2) becomes

$$y_{\phi k} \doteq \tau_{\phi_0} + \tau_{\phi_1} + \tau_{\phi_2} + \tau_{\phi_3},$$

where $\phi = \{i, j\}$, $\phi_0 = \emptyset$, $\phi_1 = \{i\}$, $\phi_2 = \{j\}$, $\phi_3 = \{i, j\}$, and $k = 1, \cdots, n_\phi$.

Define the $\langle \cdot \rangle$ operator by $\langle \{i_1, i_2, \cdots, i_l\} \rangle = I_1 I_2 \cdots I_l$ and $\langle \emptyset \rangle = 1$. This operator gives the number of levels in an index set. For example, in the two-factor model, take $I = 3$ and $J = 4$ to get $\langle \phi_3 \rangle = 12$. The model matrix $\mathbf{X}$ arising from a given model (3) is an $n \times p$ sparse matrix containing only zeros and ones, where $n = \sum_\phi n_\phi$ and $p = \sum_{m=0}^{t} \langle \phi_m \rangle$. This matrix can be written as a product $\mathbf{X} = \mathbf{TZ}$, where $\mathbf{T}$ is a replication matrix and $\mathbf{Z}$ is a model matrix with one observation per cell. The matrix $\mathbf{Z}$ has $\langle \phi \rangle$ rows, where each row represents a distinct cell of the design and is indexed by a unique realization of indices in ϕ. The matrix $\mathbf{T}$ has n rows and each row is a unit vector of dimension $\langle \phi \rangle$ that simply defines the cell sampled in that observation.

By fitting cell means, any least-squares ANOVA problem can be restated as a weighted least-squares ANOVA problem with one observation per cell. We illustrate this with the assumption of no missing cells.

There exists an orthogonal matrix $\mathbf{U}$ such that $\mathbf{U}^T\mathbf{T} = \begin{bmatrix} \mathbf{D} \\ \mathbf{0} \end{bmatrix}$ and $\mathbf{U}^T\mathbf{y} = \begin{bmatrix} \mathbf{d} \\ \mathbf{e} \end{bmatrix}$, where $\mathbf{D}$ is a diagonal matrix of order $\langle \phi \rangle$ with diagonal entries $\sqrt{n_\phi}$ giving the weights. If in addition we let $\mathbf{c} = \mathbf{D}^{-1}\mathbf{d}$, the square of (1) becomes

$$\min_{\mathbf{b}} \|\mathbf{D}[\mathbf{c} - \mathbf{Zb}]\|_2^2 + \|\mathbf{e}\|_2^2, \tag{4}$$

where the elements of $\mathbf{c}$ are the cell means, $c_\phi = n_\phi^{-1} \sum_{k=1}^{n_\phi} y_{\phi k}$. The quantity $\|\mathbf{e}\|_2^2$ is the residual sum of squares after fitting the cell means. Missing cells can be accommodated by zero weights.

In view of the above, the discussion of the nonzero structure of problem (1) can be in terms of the matrix $\mathbf{Z}$ and the weighted problem (4). Because missing cells reduce the rank of some models, this may incur the expense of overestimating the number of nonzeros in some problems with a large proportion of missing cells, but it produces gains in simplicity.

Expressions for the number of nonzeros in $\mathbf{Z}$ and $\mathbf{Z}^T\mathbf{Z}$ can be obtained with the $\langle \cdot \rangle$ operator. Partition $\mathbf{Z}$ into $[\mathbf{Z}_0 | \mathbf{Z}_1 | \cdots | \mathbf{Z}_t]$ according to the model terms. Each $\mathbf{Z}_m$ has exactly one nonzero per row, so the number of nonzeros in each block is $\langle \phi \rangle$ and the number of nonzeros in $\mathbf{Z}$ is given by

$$(t+1)\langle \phi \rangle. \tag{5}$$

The number of nonzeros in $\mathbf{Z}_i^T\mathbf{Z}_j$ is $\langle\phi_i \cup \phi_j\rangle$, so the total number of nonzeros in $\mathbf{Z}^T\mathbf{Z}$ is $\sum_{i=0}^{t}\sum_{j=0}^{t}\langle\phi_i \cup \phi_j\rangle$. Because $\mathbf{Z}^T\mathbf{Z}$ is symmetric, it is sufficient to store the nonzeros in the upper-triangular portion that contains

$$\sum_{i=0}^{t}\sum_{j=i}^{t}\langle\phi_i \cup \phi_j\rangle \tag{6}$$

nonzeros. Expressions (5) and (6) depend on many parameters. However, note that if there exist two model terms i and j such that $\phi = \phi_i \cup \phi_j$, then (6) is as large as or larger than (5). Thus, models with high-order interactions (involving at least half of the factors) have more nonzeros in the cross-products matrix than in the model matrix.

3. Model matrix column structure. Let $r(< p)$ be the rank of $\mathbf{Z}$ and let $\mathbf{P}$ be a permutation matrix so that the first r columns of $\mathbf{ZP}$ are linearly independent. The solution of (4) via orthogonal Givens rotations computes a factorization of $\mathbf{DZP}$ that can be partitioned as

$$\mathbf{DZP} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 & \mathbf{R}_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \tag{7}$$

where $\mathbf{R}_1$ is upper triangular of order r, $\mathbf{R}_2$ is $r \times (p - r)$, and $\mathbf{Q}$ is a product of orthogonal Givens rotation matrices that is partitioned to conform. A solution to (1) is a vector $\mathbf{b}$ given by

$$\mathbf{P}^T\mathbf{b} = \begin{bmatrix} \mathbf{R}_1^{-1}\mathbf{Q}_1^T\mathbf{d} \\ \mathbf{0} \end{bmatrix} \tag{8}$$

and the residual sum of squares is given by

$$\left\| \begin{bmatrix} \mathbf{Q}_2^T\mathbf{d} \\ \mathbf{e} \end{bmatrix} \right\|_2^2. \tag{9}$$

The solution chosen in (8) is that of a parameterization where all parameters associated with columns of $\mathbf{R}_2$ are set to zero. Expressions (8) and (9) show that the calculation of asolution and the residual sum of squares requires only a set of r linearly independent columns of $\mathbf{Z}$ rather than the entire matrix. A standard procedure for discarding columns to obtain a full rank problem is to drop any column associated with level one of any factor. This has the desired effect of leaving only a set of linearly independent columns; however, it is not the best set in terms of sparsity. Indeed, it is one of the worst sets because most columns are discarded from high-order interaction terms that are most sparse. Selecting a better set requires a good understanding of linear relationships among columns of the model matrix. These linear relationships are derived below and are based on, but go far beyond, the simple idea that summing columns of an interaction term ab over the index associated with a produces columns of the main-effect term b. To obtain these results, we define a class of matrices that are generated from index sets. First, we introduce some notions about index sets and then the matrices.

DEFINITION 1. Let A be an index set. Then $(A)_i$ is the i^{th} realization of A, where $1 \le i \le \langle A\rangle$.

The rule for ordering the realizations is immaterial as long as it is consistent for all index sets. For example, if $A = \{i, j\}$, $i = 1, 2, 3$, and $j = 1, 2$, then $\langle A\rangle = 6$; with a

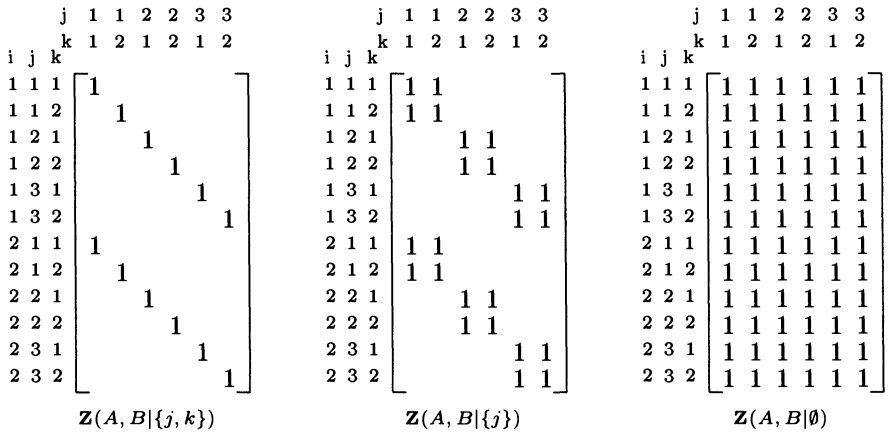


FIG. 1. *Three examples of a matrix generated by index sets* $A = \{i, j, k\}$ *and* $B = \{j, k\}$, *where* $i = 1, 2$, $j = 1, 2, 3$, *and* $k = 1, 2$, *with three different index sets* G.

rule that the rightmost index changes the fastest, we have $(A)_1 = (1,1)$, $(A)_2 = (1,2)$, $(A)_3 = (2,1)$, $(A)_4 = (2,2)$, $(A)_5 = (3,1)$, and $(A)_6 = (3,2)$.

DEFINITION 2. Let A, B, and G be three index sets such that $G \subset A \cap B$. Realization i of A *agrees* with realization j of B with respect to G, if the indices G have the same value in both realizations. This is denoted by $(A)_i \overset{G}{\sim} (B)_j$ or simply by $(A)_i \sim (B)_j$ when $G = A \cap B$.

For example, let $A = \{i, j, k\}$, $B = \{i, j, k, l\}$, and $G = \{j, k\}$. Then the realizations $(2, 5, 3)$ and $(4, 5, 3, 2)$ of A and B, respectively, agree with respect to G. That is, $(2, 5, 3) \overset{G}{\sim} (4, 5, 3, 2)$.

We now define a class of matrices that have some properties applicable to ANOVA models and are useful in describing linear dependencies between model terms in a model matrix. Each matrix is defined by three index sets. This definition can be thought of as an extension of the process of using a model statement to generate a model matrix that has one observation per cell.

DEFINITION 3. Let $G \subset A \cap B$ be any index sets, then $\mathbf{Z}(A, B|G)$ is an $\langle A \rangle \times \langle B \rangle$ matrix with rows indexed by A and columns indexed by B. The i-j entry of $\mathbf{Z}(A, B|G)$ is one when $(A)_i \overset{G}{\sim} (B)_j$ and zero otherwise.

In particular, $\mathbf{Z}(A, B|B)$ is the model matrix for the model $y_A \doteq \tau_B$ that has one observation per cell. This and other examples are given in Fig. 1.

These matrices have the property that their rank is the number of levels in the third defining index set. If we require that index sets conform for multiplication of two such matrices, then the class of matrices $c\,\mathbf{Z}(A, B|G)$, where c is a constant and $G \subset A \cap B$ are arbitrary index sets, is closed under matrix multiplication. Whenever two such matrices conform, their product is obtained by simple set operations, as is stated in the following lemma.

LEMMA 4. *For any index sets* A, B, C, G, *and* H, *such that* $G \subset A \cap B$ *and* $H \subset B \cap C$, *we have* $\mathbf{Z}(A, B|G)\,\mathbf{Z}(B, C|H) = \langle B - G \cup H \rangle \mathbf{Z}(A, C|G \cap H)$.

Proof. The i-j element of the matrix product is indexed by $(A)_i$ of A and $(C)_j$ of C. This element is given by the inner product of row $(A)_i$ of $\mathbf{Z}(A, B|G)$ and column $(C)_j$ of $\mathbf{Z}(B, C|H)$. The inner-product value is the number of k's for which $(A)_i \overset{G}{\sim} (B)_k \overset{H}{\sim} (C)_j$, since that is when a nonzero (a one) coincides in both vectors

of the inner product. Thus the element is nonzero only if $(A)_i \overset{G\cap H}{\sim} (C)_j$. When it is nonzero, its value is $\langle B - G \cup H \rangle$. □

It is interesting to note that the Moore–Penrose pseudoinverse of $\mathbf{Z}(A, B|G)$ is given by $(\langle A - G \rangle \langle B - G \rangle)^{-1} \mathbf{Z}(B, A|G)$, which again is a matrix in the same class. One can check this by verifying the four properties with the aid of Lemma 4.

In this section we will use $G = B$ and write $\mathbf{Z}(A, B|B)$ simply as $\mathbf{Z}(A, B)$. Since the rank now equals the number of columns, these matrices have full rank. Columns of such matrices are precisely the columns of a model term with indices B in an experiment whose cells are indexed by A. The more general matrices will be used in § 4. The partitioned model matrix for (3) with one observation per cell can be described in this notation as

$$\mathbf{Z} = [\mathbf{Z}(\phi, \phi_0) \mid \mathbf{Z}(\phi, \phi_1) \mid \cdots \mid \mathbf{Z}(\phi, \phi_t)].$$

Some special cases of Lemma 4 are well-known properties of the model matrix and are given in the following corollary.

COROLLARY 5. *If $\phi_n \subset \phi_m$, then $\mathbf{Z}(\phi, \phi_m)\mathbf{Z}(\phi_m, \phi_n) = \mathbf{Z}(\phi, \phi_n)$. In particular, if $\phi_n = \emptyset$, then $\mathbf{Z}(\phi, \phi_m)\mathbf{Z}(\phi_m, \emptyset) = \mathbf{Z}(\phi, \emptyset) = \mathbf{1}$, a column of ones.*

A more familiar way of interpreting this is to note that post-multiplication by $\mathbf{Z}(A, B)$ is equivalent to summation of columns over indices $A - B$. The corollary states that summing columns of a given model term over a subset of its indices gives the columns of a model term with the remaining indices. For example, summing columns of an interaction term ab_{ij} over i produces columns of the main-effect term b_j. Summing all columns of any model term produces a column of ones.

THEOREM 6. *The intersection of the column spaces of $\mathbf{Z}(\phi, \phi_m)$ and $\mathbf{Z}(\phi, \phi_n)$ is given by the column space of $\mathbf{Z}(\phi, \phi_m \cap \phi_n)$.*

Proof. Let $\phi_{mn} = \phi_m \cap \phi_n$. Lemma 4 gives $\mathbf{Z}(\phi, \phi_n)\mathbf{Z}(\phi_n, \phi_{mn}) = \mathbf{Z}(\phi, \phi_{mn})$ and $\mathbf{Z}(\phi, \phi_m)\mathbf{Z}(\phi_m, \phi_{mn}) = \mathbf{Z}(\phi, \phi_{mn})$ so that the column space of $\mathbf{Z}(\phi, \phi_{mn})$ belongs to the intersection. Let $\mathbf{a}$ belong to the intersection space, so there exist vectors $\mathbf{g}$ and $\mathbf{h}$ that satisfy $\mathbf{Z}(\phi, \phi_m)\mathbf{g} = \mathbf{Z}(\phi, \phi_n)\mathbf{h} = \mathbf{a}$. For every i and j such that $(\phi_m)_i \sim (\phi_n)_j$, there exists a k such that $(\phi_m)_i \sim (\phi)_k \sim (\phi_n)_j$. Therefore $g_i = h_j \ \forall i, j$ such that $(\phi_m)_i \sim (\phi_n)_j$. Thus $\mathbf{g}$ and $\mathbf{h}$ must have only $\langle \phi_{mn} \rangle$ distinct values, so there exists a vector $\mathbf{u}$ such that $\mathbf{g} = \mathbf{Z}(\phi_m, \phi_{mn})\mathbf{u}$ and $\mathbf{h} = \mathbf{Z}(\phi_m, \phi_{mn})\mathbf{u}$. This gives $\mathbf{a} = \mathbf{Z}(\phi, \phi_{mn})\mathbf{u}$ by Lemma 4, so that $\mathbf{a}$ also belongs to the column space of $\mathbf{Z}(\phi, \phi_{mn})$. □

COROLLARY 7. $\operatorname{rank}[\mathbf{Z}(\phi, \phi_m) \mid \mathbf{Z}(\phi, \phi_n)] = \langle \phi_m \rangle + \langle \phi_n \rangle - \langle \phi_m \cap \phi_n \rangle$.

Proof. By Theorem 6 and the inclusion-exclusion law $\operatorname{rank}[\mathbf{Z}(\phi, \phi_m) \mid \mathbf{Z}(\phi, \phi_n)] = \operatorname{rank}[\mathbf{Z}(\phi, \phi_m)] + \operatorname{rank}[\mathbf{Z}(\phi, \phi_n)] - \operatorname{rank}[\mathbf{Z}(\phi, \phi_m \cap \phi_n)]$. Each of the component matrices is of full rank. □

By induction, this corollary extends to an arbitrary number of model terms. The rank of a model matrix consisting of any combination of model terms (including non-hierarchical combinations) can thus be easily computed from the model specification.

COROLLARY 8. *The null space of $[\mathbf{Z}(\phi, \phi_m) \mid \mathbf{Z}(\phi, \phi_n)]$ is given by the column space of* $\begin{bmatrix} \mathbf{Z}(\phi_m, \phi_m \cap \phi_n) \\ -\mathbf{Z}(\phi_n, \phi_m \cap \phi_n) \end{bmatrix}$.

Proof. Use Lemma 4 and then Corollary 7. □

This last corollary tells us which columns can be discarded from a model term so that the remaining columns are linearly independent of the columns of another model term. Recall that post-multiplication by $\mathbf{Z}(\phi_m, \phi_m \cap \phi_n)$ is equivalent to summation over indices $\phi_m - \phi_m \cap \phi_n$ $(= \phi_m - \phi_n)$. Therefore, dropping columns of $\mathbf{Z}(\phi, \phi_m)$

that agree with a given level of $\phi_m - \phi_n$ leaves a set of columns linearly independent of columns in $\mathbf{Z}(\phi, \phi_n)$. When τ_{ϕ_m} is a marginal effect of τ_{ϕ_n} (that is, $\phi_m \subset \phi_n$), it is well known that all columns of $\mathbf{Z}(\phi, \phi_m)$ are linearly dependent on $\mathbf{Z}(\phi, \phi_n)$. In this special case $\phi_m - \phi_m \cap \phi_n = \emptyset$, which trivially agrees with any realization of any index set, we drop all columns of $\mathbf{Z}(\phi, \phi_m)$ as expected. Algorithm 1 is used to mark the columns to be dropped within a model term. It marks the columns of model term τ_{ϕ_m} that are linearly dependent on columns of model terms $\tau_{\phi_1}, \cdots, \tau_{\phi_n}$. It provides the flexibility to order model terms in any way, while maintaining full rank.

ALGORITHM 1 (Model Term Linear Dependency Analysis).
1. **for** $i = 1$ **to** $\langle \phi_m \rangle$
2. **if** $((\phi_m)_i \sim (\phi_m - \phi_j)_1$ for some $1 \leq j \leq n)$ mark column i.

4. Column order and nonzero structure of R. The factor $\mathbf{R}_1$ of (7) is produced by annihilating nonzeros below the main diagonal in **DZP** with Givens rotations. Zero entries that become nonzero during this reduction of **DZP** into $\mathbf{R}_1$ are said to be *filled.* It is well known (see, for example, [9]) that only the column order of **DZP** affects the sparsity of the triangular factor $\mathbf{R}_1$, but both column and row order affect the number of operations required to produce this factor.

A description of the nature of fill during Givens reduction of an arbitrary sparse matrix can be found in [18]. A single Givens rotation involves only two rows and their nonzero pattern becomes the union of their patterns before the rotation except for the annihilated element. This is in some cases conservative since numerical cancellation can result in additional zeros. Such cancellation occurs frequently during the reduction of a model matrix; however, its effect is limited when reduction is applied to a subset of linearly independent columns. In fact, this cancellation was the initial motivation for developing the index-set analysis of linear dependency.

Several column-ordering strategies were tested using a symbolic Givens reduction algorithm [18]. Variations of the minimum-degree strategy [20], [18] performed best. Rather than use a general-purpose ordering algorithm and then a general-purpose symbolic factorization algorithm, the structured nature of the model matrix suggests that the index-set notation may be useful here. In fact, both the ordering and the symbolic factorization can be performed using the index sets of a given model without explicitly forming the model matrix.

Suppose we wish to reduce the matrix $[\mathbf{Z}(\phi, \phi_1) \mid \cdots \mid \mathbf{Z}(\phi, \phi_s)]$, which contains s model terms. For simplicity, this description assumes that all columns of the s model terms are used in the reduction. The same ideas apply when only a linearly independent subset of the columns is used. Partition the resulting factor $\mathbf{R}$ according to the model terms as

$$\begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} & \mathbf{R}_{13} & \cdots & \mathbf{R}_{1s} \\ & \mathbf{R}_{22} & \mathbf{R}_{23} & \cdots & \mathbf{R}_{2s} \\ & & \mathbf{R}_{33} & \cdots & \mathbf{R}_{3s} \\ & & & \ddots & \vdots \\ & & & & \mathbf{R}_{ss} \end{bmatrix}, \tag{10}$$

where rows of $\mathbf{R}_{ij}$ are indexed by ϕ_i and columns are indexed by ϕ_j.

First we illustrate the nature of fill during reduction with a simple example. Suppose we have three factors A, B, and C at 4, 3, and 2 levels, respectively, and a model matrix with one observation per cell containing columns of terms ab_{ij}, ac_{ik}, and bc_{jk}. Thus $s = 3$, $\phi_1 = \{i, j\}$, $\phi_2 = \{i, k\}$, and $\phi_3 = \{j, k\}$. The initial matrix

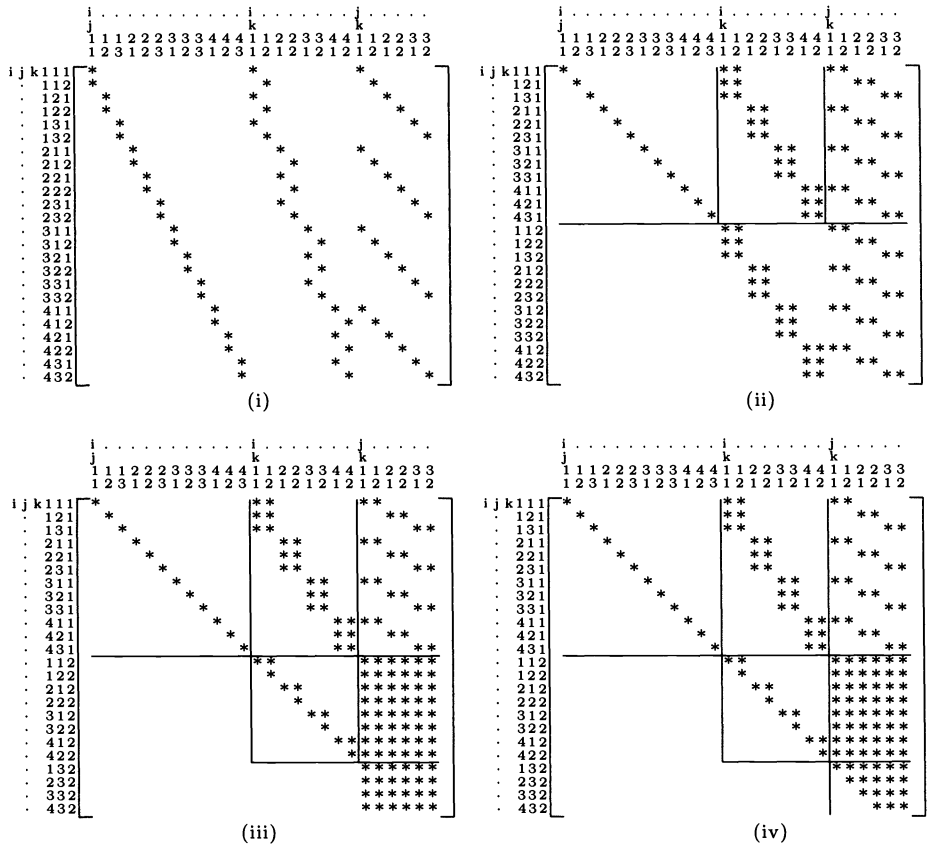


FIG. 2. (i) *Initial model matrix structure for terms* ab_{ij}, ac_{ik}, *and* bc_{jk}; (ii) *structure after reducing columns of* ab_{ij}; (iii) *structure after reducing columns of* ac_{ik}; (iv) *structure after reducing columns of* bc_{jk}.

structure is given in Fig. 2(i) and the structure after the columns of each of ab_{ij}, ac_{ik}, and bc_{jk} terms have been reduced is in parts (ii), (iii), and (iv), respectively, of the figure. Rows and columns are labeled with the appropriate index-set realizations. The rows are permuted so that completed rows of the triangular factor appear first. The partition corresponding to (10) is indicated in the completed rows. Reducing columns of the ab_{ij} term with Givens rotations produces fill in the columns of ac_{ik} and bc_{jk}. The fill occurs over any indices not in ϕ_1 (in this case k), so the structure of $\mathbf{R}_{11}$, $\mathbf{R}_{12}$, and $\mathbf{R}_{13}$ is given by the structure of $\mathbf{Z}(\phi_1, \phi_1|\phi_1)$, $\mathbf{Z}(\phi_1, \phi_2|\phi_1 \cap \phi_2)$ and $\mathbf{Z}(\phi_1, \phi_3|\phi_1 \cap \phi_3)$, respectively. Next, reducing columns of the ac_{ik} term which are already filled over k produces additional fill in the columns of bc_{jk} over indices not in ϕ_2 (in this case j). Except for zeros below the main diagonal in $\mathbf{R}_{22}$, the structure of $\mathbf{R}_{22}$ and $\mathbf{R}_{23}$ is given by $\mathbf{Z}(\phi_2, \phi_2|\phi_1 \cap \phi_2)$ and $\mathbf{Z}(\phi_2, \phi_3|\phi_1 \cap \phi_2 \cap \phi_3)$, respectively. Finally, we reduce columns of the bc_{jk} term. The structure of $\mathbf{R}_{33}$ is given by the upper-triangular portion of $\mathbf{Z}(\phi_3, \phi_3|\phi_1 \cap \phi_2 \cap \phi_3)$.

This idea generalizes to arbitrary model terms. The nonzero structure of $\mathbf{R}_{ij}$ in (10) is given by the nonzero structure of

$$\mathbf{Z}(\phi_i, \phi_j|\phi_{1\ldots i} \cap \phi_j), \tag{11}$$

where $\phi_{1\ldots i} = \phi_1 \cap \phi_2 \cap \cdots \cap \phi_i$, and when $i = j$ all entries below the main diagonal

are zero. So the number of nonzeros in $\mathbf{R}_{ij}$ is

$$\langle \phi_i \rangle \langle \phi_j - \phi_{1\cdots i} \rangle \quad \text{when } i < j, \text{ and} \tag{12}$$

$$\frac{\langle \phi_i \rangle (\langle \phi_j - \phi_{1\cdots i} \rangle + 1)}{2} \quad \text{when } i = j. \tag{13}$$

This is because $\mathbf{R}_{ij}$ has $\langle \phi_i \rangle$ rows and each row has $\langle \phi_j - \phi_{1\cdots i} \rangle$ nonzeros except that when $i = j$ the subdiagonal nonzeros are absent.

The above expressions are invariant to the order of columns within a model term. If we exclude the possibility of intermixing columns from different model terms, the column-ordering problem reduces to ordering model terms. Note that $\mathbf{R}_{11}$ has one nonzero per row. Thus, the first term ordered should be the one with most levels so that $\mathbf{R}_{11}$ is the largest block. The i^{th} term ordered determines the structures of blocks $\mathbf{R}_{ki}$, $k \leq i$. Block $\mathbf{R}_{ki}$ has $\langle \phi_i - \phi_{1\cdots k} \rangle$ nonzeros per row. Choosing the i^{th} term to maximize $\langle \phi_{1\cdots i} \rangle$ minimizes the number of nonzeros per row for $k = i$ and $k = i - 1$, and tends to make it small for other k. This is a simple heuristic for reducing the number of nonzeros in $\mathbf{R}$ that resembles minimum degree by its local nature. Thus, we begin by assigning the model term with most levels to ϕ_1. The next model term is assigned to make $\langle \phi_{12} \rangle$ as large as possible, and next to make $\langle \phi_{123} \rangle$ as large as possible, and so on. One could use expressions (12) and (13) to compute the optimal ordering of model terms when all columns are present. However, when we drop some columns of model terms due to linear dependence this order may no longer be optimal. Some experiments with a number of model terms indicate that the above heuristic for ordering is often optimal. Given a model (3), Algorithm 2 selects and orders model terms and uses Algorithm 1 to mark linearly dependent columns in the selected model terms.

ALGORITHM 2 (Model Term Selection and Ordering).

1. $\xi \leftarrow \emptyset$
2. $\psi \leftarrow \{1, 2, \cdots, t\}$
3. $\phi_\cap \leftarrow \phi$
4. **while** $\psi \neq \emptyset$ **do**:
5. Find $k \in \psi$ such that $\langle \phi_k \cap \phi_\cap \rangle$ is maximum.
6. Use Algorithm 1 on τ_{ϕ_k} with respect to terms ξ.
7. $\psi \leftarrow \psi - \{i \in \psi \mid \phi_i \subset \phi_k\}$
8. $\xi \leftarrow \xi \cup \{k\}$
9. $\phi_\cap \leftarrow \phi_\cap \cap \phi_k$

The result of this algorithm is the set ξ of selected and ordered model terms and a record of columns that were marked in step 6 to be dropped. The matrix of columns thus selected is of the same rank as the complete model matrix and is of full rank when there are no missing cells. As the number of missing cells increases, progressively smaller models still maintain full rank.

Conveniently, the majority of columns dropped due to linear dependency are from the later model terms that receive the largest amount of fill. The linear dependency analysis often reduces the number of nonzeros in the selected model terms by a factor larger than two. For example, in Fig. 2 (i), columns 11, 21, 31, and 41 of ac_{ik} and columns 11, 12, 21, and 31 of bc_{jk} are dropped. Applying Givens reduction to the remaining columns gives the structure in Fig. 3 as the structure of $\mathbf{R}_1$. The number of

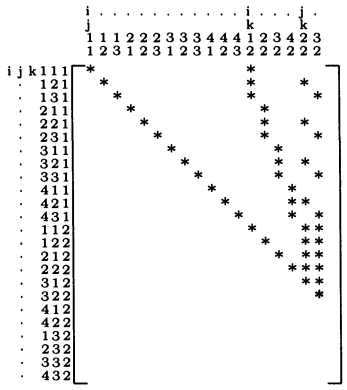


FIG. 3. *Reduced matrix of Fig.* 2 (i) *after dropping appropriate linearly dependent columns with Algorithm* 1.

nonzeros is thus reduced from 138 to 47 by dropping appropriate linearly dependent columns.

An algorithm that computes the nonzero structure of $\mathbf{R}_1$ is obtained by using expression (11) on model terms in ξ and dropping the appropriate columns.

5. Numerical phase and testing of estimable hypotheses. The structure of $\mathbf{R}_1$ determines the data structure required for sparse numerical factorization and solution. Details of the data structure can be found in [6]. The numerical phase is performed by routines adapted from SPARSPAK-B [7] that implement numerical sparse Givens factorization by rows and computation of the solution. The result of these computations is the sparse factor $\mathbf{R}_1$, the solution $\mathbf{b}$ of (8) and the residual sum of squares (9).

The sum of squares for a model term whose index set is not contained in that of any other model term is independent of the parameterization used, and thus independent of what columns were used in fitting the model. Thus, the sum of squares for such a model term is simply the difference in residual sum of squares between models with and without that model term. This amounts to using the R()-notation for computing sums of squares. (See [19] for a discussion of its use and limitations.) When searching for an appropriate model, this facility is sufficient. However, if we wish to compute the sum of squares for a marginal term in the presence of a higher-order interaction, the R()-notation cannot be used and the procedure outlined later in this section should be used. In comparison to a full matrix algorithm that obtains a number of sums of squares in a single fit, computing several fits may seem redundant. However, the freedom to build a column space basis from any available columns in any order reduces the storage requirement dramatically. Even after performing many fits, the total time for a large model is small compared to a full matrix algorithm, as will be seen from the times reported in the next section.

The error analysis of Gentleman [3] guarantees the accuracy of the Givens factorization. In fact, because of the special structure of the model matrix, a tighter error analysis may be possible. However, since the R()-notation requires computing the difference of two residual sums of squares from two independent Givens factorizations, there is concern for possible numerical cancellation. In an F-test, where the denominator is obtained from a single fit, cancellation in the numerator will lead to a nonsignificant result, as it should. However, in cases where the denominator is a difference of two fits, cancellation can produce serious roundoff errors. When cancel-

lation in the denominator occurs, we recommend the procedure below for computing the required sums of squares. This problem can occur in testing marginal variance components. In this case, the sums of squares produced by the R()-notation may not be the ones of interest anyway and the procedure below is needed to produce the required sums of squares.

To compute sums of squares for special estimable functions or specific parameterizations of marginal model terms, relatively little additional computation need be done since the sparse $\mathbf{R}_1$ factor is available after fitting a model. Suppose we wish to compute the sum of squares for testing $\mathbf{H}\beta = \mathbf{d}$, where $\mathbf{H}$ is a $q \times p$ matrix of rank q ($\ll p$) that is in the rowspace of $\mathbf{X}$, β is the vector of model parameters, and $\mathbf{d}$ is a $q \times 1$ vector. This is given in [15] as

$$(\mathbf{Hb} - \mathbf{d})^T[\mathbf{H}^T(\mathbf{R}_1^T\mathbf{R}_1)^{-1}\mathbf{H}]^{-1}(\mathbf{Hb} - \mathbf{d}). \tag{14}$$

Since the solution vector in (8) contains only r nonzeros, let $\mathbf{H}_1$ be the columns of $\mathbf{H}$ corresponding to these nonzeros and let $\mathbf{b}_1$ be these nonzeros. That is, partition $\mathbf{HP} = [\mathbf{H}_1 \mid \mathbf{H}_2]$ and $\mathbf{b}^T\mathbf{P} = [\mathbf{b}_1^T \mid \mathbf{0}]$, where $\mathbf{P}$ is an appropriate permutation matrix. Expression (14) can be computed with the following algorithm.

ALGORITHM 3 (Sum of Squares Due to Hypothesis).

1. Compute $\mathbf{h} = \mathbf{H}_1\mathbf{b}_1 - \mathbf{d}$.
2. Backsolve $\mathbf{R}_1^T\mathbf{M} = \mathbf{H}_1^T$.
3. Factor $\mathbf{M} = \mathbf{U}\begin{bmatrix} \mathbf{T} \\ \mathbf{0} \end{bmatrix}$.
4. Backsolve $\mathbf{T}^T\mathbf{v} = \mathbf{h}$.
5. Compute sum of squares $\mathbf{v}^T\mathbf{v}$.

Because all computations can be done in place, the only additional storage required is for the $q \times r$ matrix $\mathbf{H}_1$ and the $q \times 1$ vector $\mathbf{d}$. The most expensive step of this algorithm is the orthogonal factorization of step 3 of a $q \times r$ matrix. Generally $q \ll r$, so the amount of additional time and storage is small.

In general, it is not easy to write down an estimable $\mathbf{H}$, particularly for a large model. It is also not possible to anticipate all estimable hypotheses of interest and generate them automatically. However, especially for the case of no missing cells, the hypotheses that are normally tested in the balanced case could be generated automatically. More work is needed in this area, particularly on the use of index sets for the automatic generation of such hypotheses.

6. Testing and conclusions. The symbolic phase of the ANOVA model fitting algorithm, as discussed in § 4, as well as the generation of the data structure for $\mathbf{R}_1$, was programmed in FORTRAN and linked with the numerical phase adapted from SPARSPAK-B. The resulting new algorithm was run on a VAX 11/780 (with an FPA) under UNIX 4.2 BSD. The input to the symbolic portion is only the model to be fitted and the number of levels in each factor. It is only the numerical portion of the algorithm that actually reads and processes the observations, thus requiring only a single pass through the data.

The time and storage requirements of the new algorithm are compared to a full matrix algorithm that uses the sweep operator to process the cross-products matrix, and to SPARSPAK-B with its standard symbolic phase. The GLIM [1] implementation of the full-matrix algorithm was used. This is the same direct algorithm that is used in most other statistical packages that can fit unbalanced ANOVA models; thus the comparison is aimed at full-matrix direct methods. In order to be fair to

TABLE 1
Comparison of time and storage requirements.

Model	Columns used	Rank defic.	Time (min:sec) Full	SPK-B	New	Storage ('000s dbl prec.) Full	SPK-B	New
abcde	1296	55	-	4:48	:11	841.75	168.19	5.52
bcde	1056	18	637:58	5:05	:29	560.72	107.73	12.35
acde	976	18	522:02	5:03	:17	479.16	98.90	8.24
abde	928	13	460:55	7:09	:20	433.27	87.53	8.04
abce	688	2	221:56	11:54	1:50	238.66	66.31	19.85
abcd	568	0	139:53	12:25	3:03	162.90	59.98	27.98
cde	538	0	123:31	5:15	2:00	146.19	50.71	21.66
bde	522	0	114:29	5:06	1:47	137.66	46.81	19.57
bce	442	0	77:48	5:02	:20	98.89	39.20	19.37
bcd	402	0	62:20	5:04	0:39	81.89	34.18	9.61
ade	392	0	58:37	4:29	:26	77.88	31.23	6.85
ace	344	0	48:43	7:38	2:03	60.09	26.26	9.57
acd	320	0	39:29	3:37	:50	52.04	22.64	6.48
abe	314	0	38:53	3:33	:53	50.12	20.88	6.27
abd	194	0	13:22	3:59	2:13	19.34	14.63	9.40
abc	164	0	8:57	2:54	1:49	13.89	11.41	8.08
de	149	0	7:32	1:53	1:19	11.49	9.50	6.11
ce	133	0	5:53	1:55	1:17	9.19	7.73	5.09
cd	125	0	5:21	1:44	1:10	8.14	6.71	4.57
be	123	0	5:39	1:32	1:05	7.88	6.25	4.25
bd	83	0	2:29	1:17	:34	3.67	3.91	2.27
bc	73	0	1:45	:48	:19	2.86	2.89	1.26
ae	68	0	1:30	:32	:14	2.49	2.35	.80
ad	44	0	:42	:28	:20	1.09	1.33	.70
ac	38	0	:32	:24	:20	.83	1.03	.61
ab	35	0	:27	:20	:21	.70	.88	.55
e	20	0	:10	:17	:19	.25	.39	.28
d	12	0	:05	:10	:12	.11	.16	.12
c	10	0	:03	:08	:11	.08	.11	.09
b	9	0	:04	:07	:10	.06	.10	.07
a	4	0	:09	:05	:09	.02	.04	.02

SPARSPAK-B, we use the same model matrix as is used in GLIM. That is, the same linearly dependent columns are dropped. Using the entire model matrix without dropping columns increases both storage and execution time considerably.

A large data set from an epidemiological study is used to compare the two algorithms. The study contains five factors. The factors are labeled A, B, C, D, and E and contain 4, 6, 2, 3, and 9 levels, respectively. There are a total of 1,296 cells, of which 55 have no observations. Table 1 compares time and storage requirements of the three algorithms. The model fitted in a given row contains the term listed plus the mean and all terms in rows below; thus, the model size increases from the bottom. The second column of the table gives the number of columns in the matrix used in fitting each model. The full matrix algorithm and SPARSPAK-B use the columns obtained by the standard method of dropping all columns associated with level one of a factor. The third column gives the rank deficiency of the matrix used in fitting. The execution times are given in minutes:seconds and storage is in thousands of double precision locations used for working arrays. The model labeled "abcde" does not have a full time and its full storage is estimated. This is because GLIM required over 6 megabytes of memory, which was more than the available memory on the VAX. It should be noted that GLIM reads in and stores the data in memory, which takes approximately 40 seconds and 5,100 locations for the data set used. Thus, to aim the comparison at the full matrix algorithm rather than at its GLIM implementation, the full matrix time and storage reported in the tables have these figures already subtracted out. Finally, the reported CPU times should be considered as approximate since they are slightly affected by congestion due to other users of the system. Some

replicate runs typically showed about 10 to 15 percent variation.

Row ordering was not performed in this comparison and the observations were processed as they appeared in the original data (in increasing order by levels). A separate set of test runs of the new algorithm was done on data sets sorted in different ways on levels of factors. The difference between a good and a bad row order was about a factor of two in execution time. However, in most cases this was not enough to recover the cost of sorting the observations. More work is needed on the use of row ordering and possibly the use of general row merging [17] in ANOVA problems.

The models fitted do not exhaust all possible hierarchical models. They were simply selected to illustrate how the algorithm performs as the rank of the problem increases. The first observation is that both time and storage are dramatically reduced by the use of SPARSPAK-B over a full matrix algorithm, but a further dramatic improvement is obtained by the special symbolic phase of the new algorithm. The time and storage complexities of the full matrix algorithm are $O(r^3)$ and $O(r^2)$, respectively, where r is the number of columns in the model matrix. Similar simple complexity statements are not possible for the new algorithm because its requirements depend on the sparsity of the selected set of linearly independent columns for a given model. The time and storage requirement is not a monotone function of the number of columns. As the model size increases, columns with fewer nonzeros become available. On the set of test models, the new algorithm time complexity is less than $O(r)$ and storage complexity is less than $O(r \log r)$. The tables clearly show that for small models involving only main effects, the full matrix algorithm is slightly more efficient, but for larger models the new algorithm is both faster and more storage efficient. In large models, the reduction is several orders of magnitude.

A comparison of a direct algorithm to an iterative algorithm is difficult. A direct algorithm produces not only a solution and a sum of squares, but also a factored model matrix, which facilitates the computation of other sums of squares as well as variances of estimable functions. Iterative methods must start over for each computation of a different sum of squares or a variance of an estimable function [13]. For this reason, direct algorithms are considered more general purpose [13]. The convergence rate of gradient algorithms [8], [12], [16], [13] depends on the amount of imbalance in the data, while direct algorithms perform independently of the imbalance. Of the iterative methods for nonorthogonal ANOVA, conjugate gradient algorithms are reported to perform best [13]. The storage complexity for a conjugate gradient algorithm reported in [8] is $O(r)$. When compared over the models fitted in Table 1, the conjugate gradient storage requirement ranges from competitive to a factor of 10 less than the new direct algorithm.

The new algorithm shows that direct methods for large ANOVA problems can be made dramatically more efficient by the use of sparse matrix techniques while exploiting the rank deficiency of the model matrix. It shows that iterative methods are not the only choice for very large ANOVA problems. The new algorithm should also not be considered as the limit for direct method efficiency. Improvements seem possible through fine tuning, row ordering, and general row merging.

Acknowledgment. I am grateful to William J. Kennedy, Jr. for his advice and encouragement during the early stages of this work, to Edward L. Frome for providing the test data set, and to two referees for their helpful comments and suggestions on the original manuscript.

REFERENCES

[1] R. J. BAKER AND J. A. NELDER, *Generalized Linear Interactive Modeling* (GLIM), Release 3, Numerical Algorithms Group, Oxford, 1978.

[2] W. H. FELLNER, *Sparse matrices and the estimation of variance components by likelihood methods*, Commun. Statist. -Simula., 16 (1987), pp. 439–463.

[3] W. M. GENTLEMAN, *Error analysis of* QR *decompositions by Givens transformations*, Linear Algebra Appl., 10 (1975), pp. 189–197.

[4] ———, *Least squares computations by Givens transformations without square roots*, J. Inst. Math. Appl., 12 (1973), pp. 329–336.

[5] ———, *Row elimination for solving sparse linear systems and least squares problems*, in Proc. Dundee Conference on Numerical Analysis, G. A. Watson, ed., Springer-Verlag, 1975, pp. 122–133.

[6] J. A. GEORGE AND M. T. HEATH, *Solution of sparse least squares problems using Givens rotations*, Linear Algebra Appl., 34 (1980), pp. 69–83.

[7] J. A. GEORGE AND E. NG, SPARSPAK: *Waterloo sparse matrix package user's guide for* SPARSPAK-B, Tech. Report CS-84-37, University of Waterloo, Waterloo, Ontario, Canada, 1984.

[8] G. H. GOLUB AND S. G. NASH, *Nonorthogonal analysis of variance using a generalized conjugate-gradient algorithm*, J. Amer. Statist. Assoc., 77 (1982), pp. 109–116.

[9] M. T. HEATH, *Numerical methods for large sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 497–513.

[10] ———, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 223–237.

[11] W. J. HEMMERLE, *Algebraic specification of statistical models for analysis of variance computations*, J. ACM, 11 (1964), pp. 234–241.

[12] ———, *Nonorthogonal analysis of variance using iterative improvement and balanced residuals*, J. Amer. Statist. Assoc., 69 (1974), pp. 772–778.

[13] M. JAMSHIDIAN AND R. I. JENNRICH, *Nonorthogonal analysis of variance using gradient methods*, J. Amer. Statist. Assoc., 83 (1988), pp. 483–489.

[14] O. KEMPTHORNE, *The term design matrix, in: letters to the editor*, Amer. Statist., 34 (1980), p. 249.

[15] W. J. KENNEDY, JR. AND J. E. GENTLE, *Statistical Computing*, Marcel Dekker, New York, 1980.

[16] B. C. KIM, M. G. MARASINGHE, AND W. J. KENNEDY, JR., *A new conjugate gradient algorithm for analysis of variance computations*, in Proc. Statistical Computing Section, American Statistical Association, 1984, pp. 150–154.

[17] J. W. H. LIU, *On general row merging schemes for sparse Givens transformations*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1190–1211.

[18] G. OSTROUCHOV, *Symbolic Givens reduction and row-ordering in large sparse least squares problems*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 248–264.

[19] F. M. SPEED AND R. R. HOCKING, *The use of the* R()*-notation with unbalanced data*, Amer. Statist., 30 (1976), pp. 30–33.

[20] W. F. TINNEY, *Comments on using sparsity techniques for power system problems*, in Sparse Matrix Proceedings, IBM Res. Report RAI 3-12-69, 1969.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 72-88, January 1989
© 1989 Society for Industrial and Applied Mathematics
007

CONTINUATION AND COLLOCATION FOR PARAMETER-DEPENDENT BOUNDARY VALUE PROBLEMS*

G. BADER† AND P. KUNKEL†

Abstract. The numerical approximation of boundary value problems in the context of one-parameter continuation is considered. A collocation discretization of parameter-dependent boundary value problems is suggested. The convergence properties are analysed and a stable algorithm is presented. Tangent continuation and steplength control are combined with a Gauss-Newton type corrector, which is shown to be quadratically convergent. Based on this analysis, a general purpose continuation code, COLCON, has been implemented. Numerical examples demonstrating the efficiency of the approach are presented.

Key words. boundary value problems, parameter dependence, collocation, tangent continuation, steplength control, Gauss-Newton method

AMS(MOS) subject classification. 65L10

1. Introduction. The formulation of many computational problems includes physical parameters. Frequently, not only a single solution is of interest but also the dependence of the solution on selected parameters. The numerical solution of such problems is usually done by continuation methods.

The present paper deals with the tracing of the solution manifold of boundary value problems in ordinary differential equations depending on a single free parameter. There exists a number of codes for the numerical solution of such problems. Probably the best known codes in this area are AUTO of Doedel [16] in its recent version of Doedel and Kernevez [17] and BIFPACK of Seydel [27]. A more recent approach is taken in MULCON of Deuflhard, Fiedler, and Kunkel [14]. For further codes with continuation features see [1], [21], [22], [24], [26].

When tracing solution manifolds, usually a large number of boundary value problems must be solved. Therefore, the choice of an efficient and reliable boundary value solver is a most important aspect.

The multiple shooting-type codes [27], [14] inherit their efficiency from the high standard of initial value software available. However, applied to stiff boundary value problems, i.e., problems with large Lipschitz constants, this approach experiences severe limitations originating in the fact that the initial value problems to be solved during the solution process are unstable.

Global methods, on the other hand, are much less affected by stiffness. The crucial point for the efficiency and robustness of codes based on these methods seems to be the effectiveness of the implemented mesh-selection procedure (see [4]-[6]). A code of this type is the above-cited AUTO, which uses Gaussian collocation for discretizing the differential equations. It implements an adaptive mesh selection similar to COLSYS [4], [5], but keeps the total number of mesh intervals fixed. The discretization points are only equidistributed with respect to an estimation of the discretization error after a certain number of continuation steps. The overall accuracy of the computed collocation solution, however, is not checked. Therefore it is not clear how well the computed collocation solution approximates the exact solution. Especially in real life applications,

* Received by the editors March 23, 1987; accepted for publication (in revised form) March 8, 1988. This work was supported by the Deutsche Forschungsgemeinschaft.

† Institut für Angewandte Mathematik, Universität Heidelberg, Im Neuenheimer Feld 294, D-6900 Heidelberg, Federal Republic of Germany.

the solution profile of the discrete boundary value problem may differ considerably from the exact one. In fact, the whole structure of the solution diagram may be disturbed by the presence of so-called spurious solutions.

The purpose of the present paper is to report on the development of a continuation code aimed at an automatic tracing of solution branches for parameter-dependent boundary value problems in which the total error of the collocation approximation is kept below a user-prescribed error tolerance. To achieve this, an efficient steplength control for the continuation procedure is combined with a collocation discretization using a fully adaptive mesh-selection procedure. In § 2, the underlying continuation techniques are outlined. A tangent continuation with theoretically backed steplength control is combined with a quadratically convergent Gauss–Newton type iteration, extending the ideas of [13]. A collocation discretization for parameter-dependent boundary value problems based on the techniques used in the collocation code COLNEW of Bader and Ascher [10] is presented, and its convergence properties are analysed in § 3. Section 4 discusses the implementation of these ideas in a general purpose code, COLCON. Finally, in § 5, numerical examples demonstrating the performance of the code are given.

2. Tangent continuation with steplength control. For notational convenience we consider in this section and in § 3 a scalar mth-order differential equation depending on a single parameter only

$$\mathscr{L}(y, \tau) \equiv y^{(m)} - f(x, y, y', \cdots, y^{(m-1)}; \tau) = 0, \qquad a < x < b, \qquad \tau \in \mathbb{R} \tag{2.1a}$$

with m boundary conditions

$$\mathscr{B}_j(y, \tau) \equiv g_j(\mathbf{y}(\xi_j); \tau) = 0, \qquad a \leqq \xi_j \leqq b, \quad 1 \leqq j \leqq m \tag{2.1b}$$

where

$$\mathbf{y}(x) = (y(x), y'(x), \cdots, y^{(m-1)}(x))^T$$

denotes the state vector of the corresponding first-order system. Note, however, that all subsequent considerations directly carry over to systems of mixed-order differential equations depending on several parameters.

Remark. Most discretization schemes require (2.1a) to be rewritten as a system of first-order differential equations (cf. [14], [17], [27]). Collocation, however, allows for a direct treatment of higher-order differential equations in a more efficient way (cf. [3], [5]). In contrast to [14], [17], and [27], the boundary conditions (2.1b) are assumed to be separated. Nonseparated boundary conditions can be easily reformulated by introducing additional parameters or trivial differential equations (cf. [8]).

Assume that the (generally nonlinear) functions f and g_j are sufficiently smooth with respect to the arguments x, $\mathbf{y}$, and τ. In particular, require that any solution of (2.1) possess the (minimal) smoothness $y \in C^{m-1}[a, b] \cap C_\pi^m$, C_π^m requiring the mth derivative of any solution to be piecewise continuous on a partition π of $[a, b]$ (cf. (3.1)).

If we consider the parameter τ as an additional unknown, problem (2.1) becomes an underdetermined boundary value problem. Following the theoretical treatment in [18], we assume the solution set $\mathscr{M}$ of (2.1) to be a continuously differentiable one-dimensional manifold in $\mathscr{C} = (C^{(m-1)}[a, b] \cap C_\pi^m) \times \mathbb{R}$. This assures that the solution manifold $\mathscr{M}$ can be locally parametrized along its tangent direction.

To solve problem (2.1) we apply an iterative method based on linearization. For that purpose define the linearization of (2.1) by

$$\mathscr{L}[y, \tau](u, \delta) \equiv u^{(m)} - \sum_{l=1}^{m} c_l(x) u^{(l-1)} - d(x)\delta \tag{2.2a}$$

where

$$c_l(x) = \frac{\partial f(x, \mathbf{y}, \tau)}{\partial \mathbf{y}_l}, \quad l = 1, \cdots, m, \qquad d(x) = \frac{\partial f(x, \mathbf{y}, \tau)}{\partial \tau}$$

with the boundary conditions

$$\mathscr{B}_j[y, \tau](u, \delta) \equiv \sum_{l=1}^{m} b_{jl} u^{(l-1)}(\xi_j) + e_j \delta, \qquad 1 \leqq j \leqq m \tag{2.2b}$$

where

$$b_{jl} = \frac{\partial g_j(\mathbf{y}, \tau)}{\partial \mathbf{y}_l}, \quad l = 1, \cdots, m, \qquad e_j = \frac{\partial g_j(\mathbf{y}, \tau)}{\partial \tau}.$$

Before outlining the details of the continuation process, problem (2.1) is rewritten in the more compact form

$$F(z) = 0, \qquad z = (y, \tau), \tag{2.3}$$

where F contains both the nonlinear differential operator (2.1a) and the boundary conditions (2.1b). This operator equation is now considered on appropriate Hilbert spaces

$$F: \Omega \subset Z \to Y, \tag{2.4}$$

with F being a $C^r(\Omega)$-mapping, $r \geqq 1$, on some open subset $\Omega \subset \mathscr{C} \subset Z$ with $\mathscr{C}$ continuously embedded into Z. Let $W^\nu[a, b]$, $0 \leqq \nu \leqq m$ denote the Sobolev spaces

$$W^\nu[a, b] = \{u \mid u^{(\nu)} \in L_2[a, b]\} \tag{2.5a}$$

equipped with the scalar products

$$(u, v)_\nu = \sum_{l=0}^{\nu} \int_a^b u^{(l)}(s) v^{(l)}(s)\, ds, \tag{2.5b}$$

and the induced norms $\|u\|_\nu = (u, u)_\nu^{1/2}$. Then, with the canonical scalar products, the spaces Z and Y are naturally identified with the Hilbert spaces

$$\begin{aligned} Z &= W^m[a, b] \times \mathbb{R}, \qquad \|(u, \tau)\|_Z = (\|u\|_m^2 + \tau^2)^{1/2}, \\ Y &= W^0[a, b] \times \mathbb{R}^m, \qquad \|(u, g)\|_Y = \left(\|u\|_0^2 + \sum_{l=1}^{m} g_l^2 \right)^{1/2}. \end{aligned} \tag{2.5c}$$

Note that for the chosen spaces the Fréchet derivative $DF: \Omega \to Y$ of F is a bounded linear Fredholm operator of index 1 (cf. [18]). This ensures the existence of the Moore-Penrose pseudoinverse (see Chap. 8 of [11]). Thus, in order to find a solution of the underdetermined operator equation (2.3) we apply the Gauss-Newton iteration

$$\Delta z^k = -DF(z^k)^+ F(z^k), \tag{2.6a}$$

$$z^{k+1} = z^k + \Delta z^k, \qquad k = 0, 1, \cdots, \tag{2.6b}$$

where $DF(z^k)^+$ denotes the Moore-Penrose pseudoinverse of $DF(z^k)$. The choice of an appropriate starting value $z^0 \equiv z_p \in \Omega$ is described below. For the realization of this method in the case of finite-dimensional systems of equations see [13], [14]. The convergence properties of the iterative scheme are given by Theorem 1.

THEOREM 1. *Let $F: \Omega \to Y$, $F \in C^1(\Omega)$ for some convex subset $\Omega \subset Z$ such that the Fréchet derivative DF of F has the properties*

$$\begin{aligned} &\dim \ker DF(z) = 1 \\ &DF(z)Z = Y \end{aligned} \qquad \text{for } z \in \Omega.$$

Then, under the assumptions

(2.7a) $$\|\Delta z^0\|_Z \leqq \alpha_o, \qquad z^0 \in \Omega,$$

(2.7b) $$\|DF(z_1)^+[DF(z_2) - DF(z_3)]\|_Z \leqq \omega_o \|z_2 - z_3\|_Z$$
for any $z_1, z_2, z_3 \in \Omega$, $z_2 - z_3 \in \text{range } DF(z_1)^+$

(2.7c) $$h_o = \tfrac{1}{2}\alpha_o\omega_o < 1,$$

(2.7d) $$S(z^0, r) = \{z \in Z \mid \|z - z^0\|_Z \leqq r\} \subset \Omega \quad \text{for } r = \alpha_o/(1 - h_o)$$

iteration (2.6) *remains in $S(z^0, r)$ and converges to some solution $z^* \in \mathcal{M}$ with*

$$F(z^*) = 0$$

on the solution manifold. The rate of convergence is quadratic:

$$\|z^{k+2} - z^{k+1}\|_Z \leqq \frac{\omega_o}{2} \|z^{k+1} - z^k\|_Z^2. \qquad \square$$

The proof follows from Theorem 4 of [15], realizing that the $\kappa(z)$ defined in [15] vanishes identically under the above assumptions (cf. [13]).

Note that the conditions of the above theorem are affine invariant, i.e., they are invariant under bounded linear bijections of appropriate spaces (cf. [15]). Classical formulations of convergence theorems use the Lipschitz constant L of the Fréchet derivative $DF(z)$ in order to bound the radius of convergence for iterative schemes. Applied to problems with large Lipschitz constants, e.g., stiff boundary value problems, this results in a very small convergence radius proportional to $1/L$ being unnecessarily restrictive. In contrast to this, the above theorem gives a convergence radius that does not depend on $1/L$.

The Moore-Penrose pseudoinverse is known to find the *least squares solution* Δz^+ of $DF(z)\Delta z = -F(z)$, that is,

(2.8) $$\|\Delta z^+\|_Z = \inf\{\|\Delta z\|_Z \mid DF(z)\Delta z = -F(z)\}.$$

Furthermore, the solution Δz^+ is orthogonal to the local tangent vector

$$(\Delta z^+, t(z))_Z = 0$$

where $t(z)$ is defined in (2.9). In the above sense, iteration (2.6) searches a solution of $F(z) = 0$ closest to the starting value z^0. In practical applications this tends to minimize the number of iteration steps necessary to solve $F(z) = 0$.

From (2.8) we also conclude that the Moore-Penrose pseudoinverse has minimal norm K over all generalized inverses $DF(z)^-$ (see [11]). In terms of boundary value problems this leads to a minimal stability or conditioning constant for the underdetermined linear problem (2.2). See [3], [10], and [19] for the importance of a small stability constant when solving linear boundary value problems. In that respect the Moore-Penrose pseudoinverse leads to solutions of the linear problems with optimal stability bounds.

For the following define the tangent vector $t(z)$ as a solution of the linear system

$$DF(z)t(z)=0 \tag{2.9a}$$

normalized to unit norm

$$\|t(z)\|_Z=1. \tag{2.9b}$$

Let z_0 be a point on the solution manifold $\mathcal{M}$. We predict an estimation for an advanced point on the solution manifold along the tangent direction $t(z_0)$ by

$$z_p=z_0+st(z_0) \tag{2.10}$$

with an appropriately chosen steplength parameter s. Then, we apply the corrector iteration (2.6) with starting value $z^0=z_p$ in order to find a nearby solution z^* with $F(z^*)=0$.

The above approach is closely related to pseudo-arclength continuation (see [23]). Instead of employing the Moore–Penrose pseudoinverse, system (2.3) is enlarged by a normalization condition, e.g., of the form

$$\|z(s)-z_0\|_Z^2-s^2=0$$

where s is a prescribed steplength approximating the arclength of the solution branch. An expansion argument then shows that the two approaches are only identical in the limiting case $s\to 0$. For nonzero stepsizes, especially in the case of high curvature, they may differ considerably.

The approximation property of the tangent predictor is utilized by the following theorem.

THEOREM 2. *Consider the continuation process* (2.10) *with the iteration scheme* (2.6). *In addition to the assumptions of Theorem* 1, *let*

$$\|DF(z_5)^+[DF(z_4)-DF(z_0)]t(z_0)\|_Z\leqq\omega_t s_1\|t(z_0)\|_Z^2 \tag{2.11}$$

for any

$$\begin{aligned} z_4&=z_0+s_1t(z_0),\\ z_5&=z_0+s_2t(z_0), \end{aligned}\qquad 0\leqq s_1,\ s_2\leqq s$$

such that $z_{4,5}\in\Omega$ *and* z_0 *is a point on the solution manifold* $\mathcal{M}$. *Then the iteration* (2.6) *converges to a solution* $z^*\in\mathcal{M}$ *for any starting point* z_p *from* (2.10) *with*

$$s<s_{\max}=2\left(\frac{1}{\omega_o\omega_t}\right)^{1/2}. \tag{2.12}$$

□

For a proof see [13].

Relation (2.12) provides an upper bound for acceptable continuation stepsizes. On this basis a theoretical backed steplength control can be derived (see [13]). For this purpose the unknown global bounds ω_o, ω_t have to be replaced by accessible local estimates $[\omega_o]$ and $[\omega_t]$. To work with these estimates successfully, we must consider a predictor-corrector-type steplength strategy. Following [13], a value s_{pred} for the steplength is predicted after a successful continuation step with solution z^* and steplength s_{old} by

$$s_{\text{pred}}=\sqrt{\frac{\theta_{\max}}{\theta_0}\frac{\|\Delta z^0\|_Z}{\|z^0-z^*\|_Z}\frac{|c_*|^2}{|c_0|^3}}\,s_{\text{old}} \tag{2.13}$$

with the values $c_* := (t(z^*), t(z^0))_Z$, $c_0 := (t(z_0), t(z^0))_Z$ describing the curvature of the solution manifold and

$$\theta_k = \frac{\|\overline{\Delta z}^{k+1}\|_Z}{\|\Delta z^k\|_Z}, \qquad k = 0, 1, \cdots \tag{2.14}$$

where $\overline{\Delta z}^{k+1}$ solves the simplified correction

$$\overline{\Delta z}^{k+1} = -DF(z^k)^+ F(z^{k+1})$$

and $\theta_{\max} \leqq 1$ is a chosen safety factor. With this prediction a new value of z_p is found and the convergence of the iteration scheme (2.6) is monitored by

$$\theta_k < 1.$$

If the convergence of iteration (2.6) proves to be insufficient, the continuation steplength is reduced by the correction formula

$$s_{\text{corr}} = \sqrt{\frac{\theta_{\max}}{\theta_k}}\, s_{\text{pred}} \tag{2.15}$$

obtained from (2.12) using updated estimates for $[\omega_o]$ and $[\omega_t]$, and the iteration is repeated. The above steplength control is implicitly based on the assumption that $c_k := (t(z^k), t(z_0))_Z > 0$. To be on the safe side, the validity of $c_k \geqq c_{\min}$ for some chosen $c_{\min} > 0$ is monitored during the iteration process.

The above strategy, tangent continuation combined with the steplength control (2.13), (2.15) has proved to be highly efficient in the context of continuation for systems of nonlinear equations. A comparison with a number of different approaches including pseudo-arclength continuation is given in [13]. Solving continuation problems for boundary value problems, discretization yields large systems of nonlinear equations. Thus, it can be expected that the efficiency of this strategy carries over to boundary value problems.

Remark. The above considerations cover the treatment of limit or turning points. However, branch or bifurcation points are not covered because the assumptions of Theorem 1 no longer hold. Therefore, a special treatment is required for the calculation of such points. If only tracing a solution path is desired, practical experience has shown that we can expect that the above algorithm usually "jumps" over these critical points. For a justification see [23].

3. Discretization of parameter-dependent differential equations. Consider a k-stage collocation scheme with $k \geqq m$ for the boundary value problem (2.1). Thus, there is a partition (mesh) π of $[a, b]$

$$\pi\colon a = x_1 < x_2 < \cdots < x_N < x_{N+1} = b, \qquad h_i = x_{i+1} - x_i, \qquad h = \max_{1 \leqq i \leqq N} h_i \tag{3.1a}$$

and k points

$$0 \leqq \rho_1 < \rho_2 < \cdots < \rho_k \leqq 1, \tag{3.1b}$$

which we choose to be Gaussian points.

A piecewise polynomial function $y_\pi(x)$ is sought such that $y_\pi \in \mathcal{P}_{k+m,\pi}[a, b] \cap C^{m-1}[a, b]$[1] satisfies the m boundary conditions (2.1b) given on a subset

[1] We say that v is in $\mathcal{P}_{k+m}$ if $v(x)$ is a polynomial of order $k+m$ (degree $< k+m$) on an appropriate interval, and that v is in $\mathcal{P}_{k+m,\pi}$ if $v(x)$ is a piecewise polynomial in $\mathcal{P}_{k+m}$ on each subinterval of the mesh π.

of π, and $y_\pi(x)$ satisfies the differential equations (2.1a) at the collocation points

$$\mathscr{L}_\pi(y_\pi, \tau_\pi) = y_\pi^{(m)}(x_{ij}) - f(x_{ij}, y_\pi, y'_\pi, \cdots, y_\pi^{(m-1)}; \tau_\pi), \qquad x_{ij} = x_i + h_i\rho_j \tag{3.2}$$

for $1 \leqq j \leqq k$ and $1 \leqq i \leqq N$. Note that this collocation procedure directly carries over to mixed-order systems of differential equations depending on several parameters.

When we use local linearization techniques to solve problem (2.1) it is sufficient to apply the above collocation procedure iteratively to linear problems

$$\mathscr{L}[y, \tau](u, \delta) = q(x)$$

corresponding to (2.2). Recall the monomial Runge-Kutta basis representation [10], [7]. Let us focus on one subinterval $[x_i, x_{i+1}]$ and express the polynomial $u_\pi(x)$ by

$$u_\pi(x) = \sum_{l=1}^{m} \frac{(x-x_i)^{l-1}}{(l-1)!} u_\pi^{(l-1)}(x_i) + h_i^m \sum_{j=1}^{k} \psi_j\left(\frac{x-x_i}{h_i}\right) u_\pi^{(m)}(x_{ij}) \tag{3.3}$$

with polynomials ψ_j of order $k+m$ defined by

$$\begin{aligned} &\psi_j^{(l-1)}(0) = 0, && 1 \leqq l \leqq m, \\ &\psi_j^{(m)}(\rho_\nu) = \delta_{\nu j}, && 1 \leqq \nu \leqq k, \end{aligned} \qquad 1 \leqq j \leqq k. \tag{3.4}$$

Defining the vectors

$$\mathbf{u}_i = (u_\pi(x_i), u'_\pi(x_i), \cdots, u_\pi^{(m-1)}(x_i))^T,$$
$$\mathbf{v}_i = (u_\pi^{(m)}(x_{i1}), u_\pi^{(m)}(x_{i2}), \cdots, u_\pi^{(m)}(x_{ik}))^T$$

and applying the collocation equations, we obtain k relations

$$\mathbf{W}_i\mathbf{v}_i - \mathbf{V}_i\mathbf{u}_i - \mathbf{d}_i\delta_\pi = \mathbf{q}_i, \qquad 1 \leqq i \leqq N \tag{3.5}$$

where $\mathbf{W}_i$ and $\mathbf{V}_i$ are matrices of order $k \times k$ and $k \times m$, respectively, with entries

$$\mathbf{W}_{\nu j}^i = \delta_{\nu j} - \sum_{l=1}^{m} c_l(x_{i\nu}) h_i^{m+1-l} \psi_j^{(l-1)}(\rho_\nu), \qquad 1 \leqq \nu, j \leqq k, \tag{3.6}$$

$$\mathbf{V}_{\nu j}^i = \sum_{l=1}^{j} \frac{c_l(x_{i\nu})(h_i\rho_\nu)^{j-l}}{(j-l)!}, \qquad 1 \leqq \nu \leqq k, \quad 1 \leqq j \leqq m, \tag{3.7}$$

and

$$\mathbf{d}_i = (d(x_{i1}), d(x_{i2}), \cdots, d(x_{ik}))^T, \tag{3.8}$$

$$\mathbf{q}_i = (q(x_{i1}), q(x_{i2}), \cdots, q(x_{ik}))^T. \tag{3.9}$$

Next, we obtain m additional relations from the global continuity requirements

$$\mathbf{C}_i\mathbf{u}_i + \mathbf{D}_i\mathbf{v}_i = \mathbf{u}_{i+1}, \qquad 1 \leqq i \leqq N \tag{3.10}$$

where $\mathbf{C}_i$ is an $m \times m$ upper triangular matrix with entries

$$\mathbf{C}_{\nu j}^i = \frac{h_i^{j-\nu}}{(j-\nu)!}, \qquad 1 \leqq \nu \leqq j \leqq m \tag{3.11}$$

and $\mathbf{D}_i$ is an $m \times k$ matrix with entries

$$\mathbf{D}_{\nu j}^i = h_i^{m+1-\nu} \psi_j^{(\nu-1)}(1), \qquad 1 \leqq \nu \leqq m, \quad 1 \leqq j \leqq k. \tag{3.12}$$

From equations (3.5), (3.6) it follows immediately that for h_i sufficiently small $\mathbf{W}_i$ is nonsingular and

$$\mathbf{W}_i^{-1} = \mathbf{I} + \mathcal{O}(h_i).$$

Hence, using a standard linear equation solver with pivoting, the local unknowns $\mathbf{v}_i$ can be eliminated as long as $\mathbf{W}_i$ is nonsingular. The stability of this process with respect to roundoff error propagation has been shown in [3], [7], [10], where both a theoretical justification and numerical tests are presented. A different elimination process allowing for the direct treatment of nonseparated boundary conditions is employed in [17], [23].

Using (3.5) and (3.10) for this elimination leads to relations between the solution values at the endpoints of the ith subinterval

$$\mathbf{\Gamma}_i \mathbf{u}_i - \mathbf{u}_{i+1} + \mathbf{p}_i \delta_\pi = \mathbf{r}_i, \qquad 1 \leqq i \leqq N \tag{3.13}$$

where

$$\mathbf{\Gamma}_i = \mathbf{C}_i + \mathbf{D}_i \mathbf{W}_i^{-1} \mathbf{V}_i, \quad \mathbf{p}_i = \mathbf{D}_i \mathbf{W}_i^{-1} \mathbf{d}_i, \quad \mathbf{r}_i = -\mathbf{D}_i \mathbf{W}_i^{-1} \mathbf{q}_i. \tag{3.14}$$

Assembling these equations for $1 \leqq i \leqq N$ and the corresponding boundary conditions we obtain a global shooting-like system of equations

$$\mathbf{G} \begin{pmatrix} \mathbf{u} \\ \delta_\pi \end{pmatrix} = \mathbf{r} \tag{3.15a}$$

with an $n \times (n+1)$-matrix $\mathbf{G}$:

$$\mathbf{G} = \begin{pmatrix} \mathbf{B}_a & & & & & \mathbf{p}_a \\ \mathbf{\Gamma}_1 & -\mathbf{I} & & & & \mathbf{p}_1 \\ & \mathbf{\Gamma}_2 & -\mathbf{I} & & & \mathbf{p}_2 \\ & & \ddots & \ddots & & \vdots \\ & & & \mathbf{\Gamma}_N & -\mathbf{I} & \mathbf{p}_N \\ & & & & \mathbf{B}_b & \mathbf{p}_b \end{pmatrix} \tag{3.15b}$$

where $n = (N+1)m$ and $\mathbf{B}_a$, $\mathbf{B}_b$, $\mathbf{p}_a$, and $\mathbf{p}_b$ stand for the entries coming from the boundary conditions (2.2b). For ease of notation, we assume in the above formula to have boundary conditions only at the endpoints.

The linear system (3.15) is solved by computing its *linear least squares solution*

$$\begin{pmatrix} \mathbf{u} \\ \delta_\pi \end{pmatrix} = \mathbf{G}^+ \mathbf{r} \tag{3.16}$$

such that the norm $\|(u_\pi, \delta)\|_Z$, defined in (2.5), is minimized for the collocation solution (u_π, δ_π) of the linear problem (2.2). As the Sobolev norm $\|u_\pi\|_m$ is not directly accessible, it is approximated by Gaussian quadrature yielding a discrete norm

$$\|(u, \delta)\|_\pi = \left(\sum_{i=1}^{N} h_i \sum_{j=1}^{k} b_j \sum_{l=0}^{m} u^{(l)^2}(x_{ij}) + \delta^2 \right)^{1/2} \tag{3.17}$$

with b_j, $1 \leqq j \leqq k$, denoting the corresponding quadrature weights. This approximation possesses the convergence property

$$\|(u, \delta)\|_\pi - \|(u, \delta)\|_Z = \mathcal{O}(h^{2k}). \tag{3.18}$$

This completes the solution of linear collocation problems.

Returning to the nonlinear problem (2.1) and using the notation of § 2, we rewrite the collocation equations (3.2) in the form

$$F_\pi(z_\pi) = 0. \tag{3.19}$$

When we apply the Gauss-Newton iteration (2.6) to these nonlinear collocation equations, a discretized analogue for iteration (2.6) results. The solution of the linearized subproblems

$$DF_\pi(z_\pi)\Delta z_\pi = -F_\pi(z_\pi) \tag{3.20}$$

is achieved as specified in (3.5)-(3.16) above.

After these preparations the convergence properties of the discretization can be considered.

THEOREM 3. *Assume that problem* (2.1) *has a continuously differentiable solution manifold* $\mathcal{M}$ *in* $\mathcal{C}$ *and that its Fréchet derivative has a nullspace of dimension one. Consider the k-stage collocation at Gaussian points for* (2.1). *Then there exist positive constants* ρ *and* h_0 *such that for all meshes with* $h \leqq h_0$ *the following is valid.*

(a) *In a tube* S_ρ *of radius* ρ *around the solution manifold* $\mathcal{M}$ *there exists a unique one-dimensional solution manifold* $\mathcal{M}_\pi$ *in* $(\mathcal{P}_{k+m,\pi}[a, b] \cap C^{m-1}[a, b]) \times \mathbb{R}$ *of the collocation equations* (3.2).

(b) *The solution manifold* $\mathcal{M}_\pi$ *can be computed by the iteration scheme* (2.6) *applied to the collocation equations* (3.2). *The iteration converges quadratically to some point on* $\mathcal{M}_\pi$, *provided that the initial guess is sufficiently close to the solution manifold* $\mathcal{M}$.

(c) *For the linearized boundary value problem* (2.2), *there is a stable implementation with stability constant cN, where c depends on the stability constant of the linearized problem.*

(d) *Let* (y_π, τ_π) *be a solution of the discrete problem* (3.2). *Then there exists a solution* (y, τ) *of the exact problem with* $y \in C_\pi^{2k+1}$, *such that the following superconvergence estimates hold*:

$$\begin{aligned} &|y^{(l)}(x_i) - y_\pi^{(l)}(x_i)| = \mathcal{O}(h^{2k}), \qquad 0 \leqq l \leqq m-1, \quad 1 \leqq i \leqq N+1, \\ &|\tau - \tau_\pi| = \mathcal{O}(h^{2k}). \end{aligned} \tag{3.21a}$$

(e) *The global errors satisfy uniformly for* $x \in [a, b]$

$$y^{(l)}(x) - y_\pi^{(l)}(x) = h_i^{k+m-l} y^{(k+m)}(x_i) P^{(l)}\left(\frac{x - x_i}{h_i}\right) + \mathcal{O}(h_i^{k+m-l+1}) + \mathcal{O}(h^{2k}), \tag{3.21b}$$

$$x_i \leqq x \leqq x_{i+1}, \quad 1 \leqq i \leqq N, \quad 0 \leqq l \leqq k+m-1$$

where

$$P(\xi) = \frac{1}{k!(m-1)!} \int_0^\xi (t-\xi) \prod_{j=1}^{k} (t-\rho_j)\, dt. \qquad \square$$

The proof of the theorem follows along the lines of [2].

4. Implementation. The above considerations led us to implement a continuation code called COLCON. The code is based on the well-known general purpose code COLSYS (cf. [4], [5]) in its implementation COLNEW described in [10].

The class of problems treated by COLCON allows the following general form: a system of d nonlinear differential equations of orders m_ν, $1 \leqq m_\nu \leqq 4$, $1 \leqq \nu \leqq d$,

$$y_\nu^{(m_\nu)} = f_\nu(x, \mathbf{y}; p), \qquad a \leqq x \leqq b \tag{4.1}$$

depending on the unknown m^*-vector function $\mathbf{y}$

$$\mathbf{y} = (y_1, y_1', \cdots, y_1^{(m_1-1)}, y_2, \cdots, y_2^{(m_2-1)}, \cdots, y_d^{(m_d-1)})^T \tag{4.2}$$

and the parameter vector of length *npar* is subject to separated nonlinear side conditions

$$\begin{aligned} &g_j(\mathbf{y}(\xi_j); p) = 0, \qquad 1 \leqq j \leqq mside, \\ &a \leqq \xi_1 \leqq \xi_2 \leqq \cdots \leqq \xi_{mside} \leqq b. \end{aligned} \tag{4.3}$$

The total number of unknowns is $mstate = m^* + npar$ with $m^* = \sum_{\nu=1}^{d} m_\nu$. For continuation purposes we assume $mside = mstate - 1$ boundary conditions, i.e., one free parameter.

Remark. In the case of $mside = mstate$ boundary conditions, this implementation represents a generalization of COLNEW allowing a direct treatment of parameter-dependent problems without continuation referred to as COLPAR in the following. Thus, it is no longer necessary to write the parameters p as trivial differential equations $p' = 0$ resulting in an unnecessary enlargement of the size of the boundary value problem. According to this, COLPAR proves to be more efficient than COLNEW in terms of computing time and especially in terms of storage requirements (see Example 2 below).

When attempting to implement a general purpose collocation code, the following aspects arise:

(1) Solution representation;
(2) Linear system solution;
(3) Error estimations and mesh selection;
(4) Nonlinear problem solution.

Our aim in implementing the continuation code COLCON has been to write a modular piece of software that is compatible with the code COLNEW. As a result the two codes are almost identical with respect to solution representation, error estimation, and mesh selection. In fact, the latter may be used to compute an initial solution to start continuation (see Example 2). For continuation problems, however, the new code differs considerably with respect to nonlinear problem solving. While in COLNEW a damped Newton method is realized, COLCON realizes a Gauss–Newton iteration without damping but with a steplength control for the continuation procedure. The actual implementation is done in a predictor-corrector mode with a convergence monitor along the lines indicated in § 2. During the continuation process the accuracy of the computed solution approximations is checked in order to satisfy the user-prescribed tolerances. Further, a number of additional safeguards are implemented and checked before accepting a continuation step. In this respect the code provides a means to trace regular continuation paths automatically. This property seems to be important especially for the solution of real life problems.

One subject not yet covered is the numerical solution of the least squares equation (3.15) with (3.16). We do not intend to use QR-decomposition on the matrix $\mathbf{G}$ because this tends to destroy sparsity. Recall that the solution manifold of (3.15) is known to be of dimension one only. Hence, if we can compute, in a stable way, a tangent vector t

$$\mathbf{G}t = 0 \tag{4.4}$$

and a special solution z of

$$\mathbf{G}z = \mathbf{r} \tag{4.5}$$

with $z = (\mathbf{u}, \delta)$, then the least squares solution of (3.15) is easily computed by

$$z^+ = z - \frac{(z, t)_\pi}{(t, t)_\pi} t \tag{4.6}$$

where $(\cdot, \cdot)_\pi$ is the scalar product corresponding to (3.17).

In order to solve the linear system (3.15) we modify the code SOLVEBLOK [12] used in COLSYS and also in an updated form in COLNEW. This code solves regular systems of almost block diagonal linear systems, i.e., systems of the form (3.15) but without parameter columns. The extension of this code for parameter-dependent

problems has been previously considered in [9]. Note, however, that (3.15) is underdetermined. As a consequence, we must apply some kind of column permutation in order to obtain a stable decomposition of **G**. Unrestricted column permutation, however, would destroy the sparse structure of the matrix. Hence, we consider partial column permutation only. In order to retain numerical stability, additional row permutation is applied. Let

$$PGQ = LU \tag{4.7}$$

with $n \times n$ lower and $n \times (n+1)$ upper triangular matrices L and U and let P and Q be permutation matrices of corresponding dimensions. The decomposition by SOLVEBLOK is done in a blockwise manner with full row pivoting. Treating the parameter columns just as special right-hand sides, the same strategy is used for matrix **G** except for the last block. This block, however, is decomposed by LU-decomposition with column pivoting. In contrast to unrestricted column permutation, no additional fill-in compared with SOLVEBLOK is created in this way.

For general matrices the above algorithm is not necessarily stable. But, taking into account that **G** arises from the discretization of a boundary value problem, this procedure gives a decomposition of the form (4.7) with a nonsingular matrix L as **G** is known to be of full rank (cf. Theorem 3(b)). To show numerical stability of the decomposition in this context is an open question involving the stability of the underlying boundary value problem. So far, we have not yet experienced any difficulty with the implementation of this procedure.

The desired quantities t and z from (4.4) and (4.5), respectively, are now easily computed from decomposition (4.7). Observe that the underdetermined problem has a unique solution, if the last component of the solution permuted by Q^T is prescribed. In this way, we calculate a tangent vector t by

$$U\bar{t} = 0, \quad e_{n+1}^T \bar{t} = 1, \quad t = Q\bar{t} \tag{4.8}$$

and a special solution z of the inhomogeneous equation by

$$L\bar{\mathbf{r}} = P\mathbf{r}, \quad U\bar{z} = \bar{\mathbf{r}}, \quad e_{n+1}^T \bar{z} = 0, \quad z = Q\bar{z} \tag{4.9}$$

where e_{n+1} denotes the $(n+1)$st unit vector. Together with (4.6), this yields the desired least squares solution of the linear subproblems (3.16).

5. Numerical results. In this section, the performance of the new code COLCON is demonstrated on four problems experiencing various kinds of difficulty. All examples were run on an IBM 3090 using FORTRAN double precision (56-bit mantissa). For all examples the code was used with four collocation points per subinterval, requiring five-digit accuracy for the solution and an additional two digits' accuracy for the first derivative in case of second-order differential equations.

Example 1. Consider the boundary value problem

$$u'' + \tau \exp(u) = 0,$$
$$u(0) = 0, \quad u(1) = 0, \quad \tau \geqq 0.$$

This problem is frequently used as a standard test example (see [17], [14]). The solution shows a single turning point and an interior layer of increasing steepness. Starting from the origin, the code takes 1.1 sec CPU-time to complete 20 continuation steps. The corresponding diagram is given in Fig. 1.

Example 2. This problem is described in [28], [25] and was used in [10] as a test problem for COLNEW. The model describes mass and energy balance of the renal counterflow system. With F_{iv}, the axial volume flow in the ith tube, J_{iv}, the outward transmural volume flux, $F_{iv}C_{ik}$, the axial flow of the kth solute in the ith tube, and

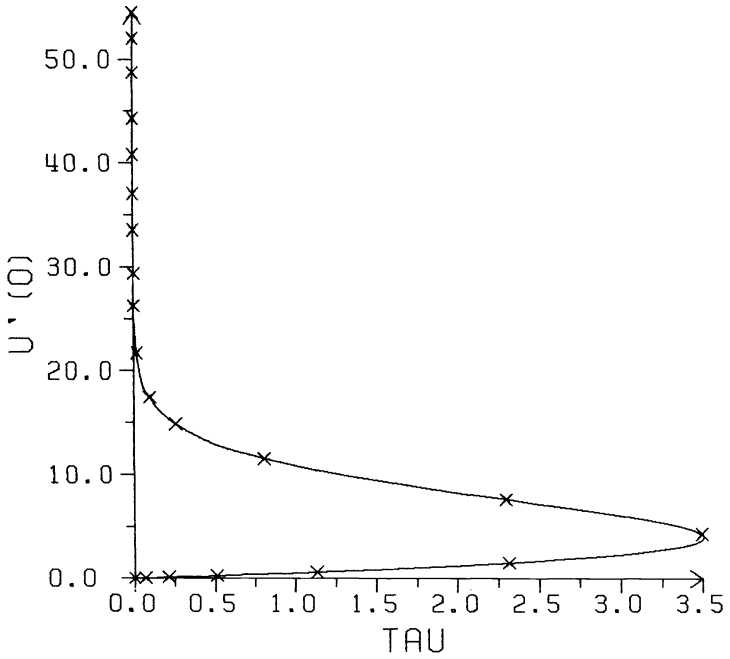


FIG. 1. *Continuation diagram for Example* 1. × *marks continuation steps.*

J_{ik}, the outward transmural flux per unit length of the kth solute from the ith tube, there are 18 differential equations with nonseparated boundary conditions for F_{iv}, C_{ik}, $1 \leqq i \leqq 6$, $1 \leqq k \leqq 2$, describing the steady state case (see [28]).

Now, it can be verified that this boundary value problem can be simplified to have only 11 differential equations depending on three parameters P_1, P_2, and $P_3 \equiv C_{22}$:

$$C'_{12} = 20h_{1v}C_{12}^2(C_{41} + C_{42} - 21C_{12}),$$

$$C_{11} = 20C_{12}, \quad F_{1v} = \frac{0.05}{C_{12}}, \quad F_{2v} = -\frac{0.05}{C_{22}},$$

$$C'_{21} = 20C_{22}J_{21}, \qquad C'_{22} = 0,$$

$$C'_{31} = \frac{h_{3v}}{P_1} C_{31}^2(C_{41} + C_{42} - C_{31} - C_{32}),$$

$$C'_{32} = \frac{C_{31}}{P_1}(J_{3v}C_{32} - J_{32}),$$

$$F'_{4v} = -J_{4v},$$

$$P_1 = \frac{C_{31}(0)}{2C_{32}(0)}, \quad F_{3v} = \frac{P_1}{C_{31}}, \quad F_{5v} = 5, \quad F_{6v} = \frac{0.05}{C_{62}},$$

$$C'_{41} = \frac{1}{F_{4v}}(J_{4v}C_{41} - J_{41}),$$

$$C'_{42} = \frac{1}{F_{4v}}(J_{4v}C_{42} - J_{42}),$$

$$C'_{51} = -200(C_{51} - C_{41}), \qquad C'_{52} = -200(C_{52} - C_{42}),$$

$$C'_{61} = 20C_{62}(J_{6v}C_{61} - J_{61}),$$

$$C'_{62} = 20C_{62}^2 J_{6v},$$

with separated boundary conditions

$$C_{12}(0)=0.05, \quad C_{51}(0)=1, \quad C_{52}(0)=0.05, \quad C_{31}(0)=P_2,$$

$$C_{31}(0)=20P_1C_{32}(0), \quad C_{62}(0)=C_{22}(0), \quad C_{61}(0)=C_{21}(0),$$

$$C_{12}(1)=C_{22}(1), \quad C_{21}(1)=20C_{12}(1), \quad C_{41}(1)=C_{51}(1), \quad C_{42}(1)=C_{52}(1),$$

$$C_{61}(1)=20P_1C_{62}(1), \quad F_{4v}(1)=-5, \quad C_{61}(1)=P_2.$$

Further, the functions J_{i1}, J_{i2}, and J_{iv}, $1 \leqq i \leqq 6$, are now specified. Transmural volume fluxes are defined as follows:

$$J_{iv}=\begin{cases} h_{iv}\sum_{k=1}^{2}(C_{4k}-C_{ik}), & i=1,2,3,5, \\ -\sum_{k\neq 4,6} J_{kv}, & i=4, \\ (1-C_{i1})+(0.05-C_{i2}), & i=6, \end{cases}$$

where $h_{1v}=h_{3v}=10$ and $h_{iv}=0$ for $i=2,5$. Transmural solute fluxes are:

$$J_{i1}=\begin{cases} 0, & i=1,3, \\ 1{,}000(C_{i1}-C_{41}), & i=5, \\ 0.75C_{i1}/(1+C_{i1}), & i=6, \end{cases}$$

$$J_{i2}=\begin{cases} 0, & i=1,2,6, \\ 1{,}000(C_{i2}-C_{42}), & i=5, \end{cases}$$

$$J_{21}(x)=\begin{cases} 1.8, & 0 \leqq x \leqq 0.4, \\ 1.8+[-18+100(C_{21}(x)-C_{41}(x))](x-0.4), & 0.4<x<0.5, \\ 10(C_{21}(x)-C_{41}(x)), & 0.5 \leqq x \leqq 1, \end{cases}$$

$$J_{32}(x)=\begin{cases} 0, & 0 \leqq x \leqq 0.4, \\ 0.1(C_{32}(x)-C_{42}(x))(x-0.4), & 0.4<x<0.5, \\ 0.01(C_{32}(x)-C_{42}(x)), & 0.5 \leqq x \leqq 1, \end{cases}$$

$$J_{4k}(x)=-\sum_{i\neq 4,6} J_{ik}(x), \qquad 0 \leqq x \leqq 1, \quad k=1,2.$$

This completes the specification of the problem.

We treat this example in order to compare the relative performances of COLCON and COLNEW. Using COLNEW, we begin with the simple initial guess that all solution components are set identically to 1 and apply a chain of classical continuation steps. Thus, the problem is solved with continuation parameter $\tau=h_{1v}=h_{3v}$ taking the values 0, 0.8, 1.6, 2.4, 4, 6, 8, 10, and starting with the solution obtained with each τ as initial guess for the problem with the next τ in the sequence. For running COLNEW as described in [10], trivial differential equations $P_i'=0$, $i=1,2,3$, have to be added for the parameters P_i. This results in a system of 14 first-order equations. To complete the entire sequence of continuation steps takes 17.1 sec CPU-time and 645 K storage.

Using COLPAR with direct treatment of parameters as mentioned in § 4, the above system of 11 equations needs no extension. The corresponding results are 12.3 sec CPU-time and 445 K storage. The savings, both in CPU-time and storage, are mainly due to the reduced size of the linear systems solved. Generally, we can expect that the direct treatment of parameters pays off, especially for problems with periodic boundary conditions, as the effective size of the computational problem is halved.

The continuation code COLCON is constructed to expect a reasonably good solution approximation from which to start. In this highly nonlinear example we use COLPAR to compute the solution for fixed parameter $\tau = 0$. Using this as the initial guess for COLCON, the complete problem is solved in 19 continuation steps taking 22.2 sec CPU-time and 490 K storage. The results indicate that it may be more expensive to trace a solution path by an automatic continuation procedure than to use a well-chosen sequence of classical continuation steps.

Example 3. Consider a forced nonlinear oscillator modeled by the differential equation

$$\left(\frac{\tau}{2\pi}\right)^2 u'' + \frac{1}{25}\left(\frac{\tau}{2\pi}\right) u' - \frac{1}{5} u + \frac{8}{15} u^3 = \frac{2}{5} \cos 2\pi x$$

together with the periodic boundary conditions

$$u(0) = u(1), \qquad u'(0) = u'(1).$$

Note that COLCON requires the problem to be formulated with separated boundary conditions. Thus, we introduce parameters p_1, p_2 and replace the above boundary conditions by

$$u(0) = p_1, \qquad u'(0) = p_2,$$
$$u(1) = p_1, \qquad u'(1) = p_2.$$

This problem is suggested as a test problem for continuation methods (see, e.g., [27]). Difficulties in the numerical solution of this example arise from the presence of 16 turning points and the same number of bifurcation points along the main branch of periodic solutions in the range $\tau \in [0.06, 3]$. This branch is shown in Fig. 2. Starting the computation from $\tau = 3$, COLCON takes 212 continuation steps and 93.7 sec CPU-time.

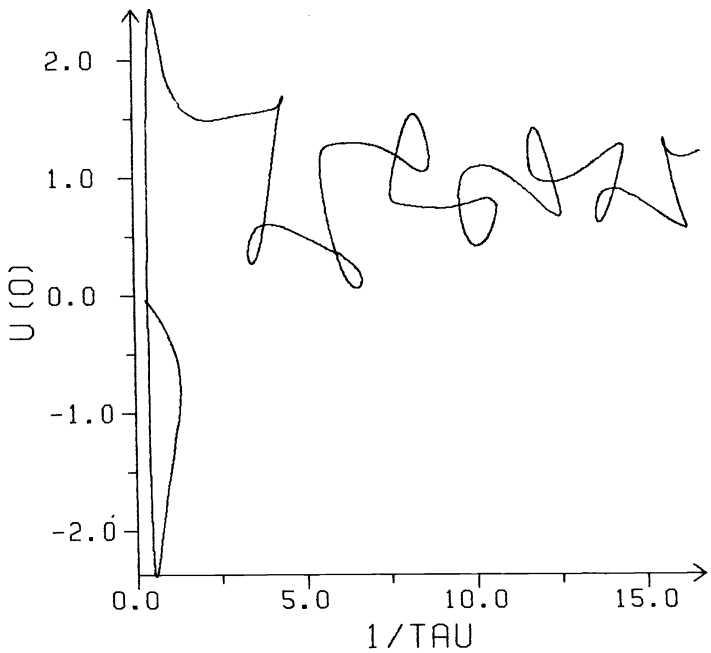


FIG. 2. *Main branch of periodic solutions for Example* 3.

Example 4. This problem describing the dynamic behavior of a tubular reactor is considered in [20]. The steady state equations result after transformation in two differential equations for relative reactant concentration and temperature given by

$$\frac{1}{\mathrm{Pe}_1} u_1'' = u_1' - \mathrm{Da}\, f(u), \qquad \frac{1}{\mathrm{Pe}_2} u_2'' = u_2' - \beta(u_2 - u_{2w}) - \mathrm{Da}\, Bf(u),$$

with the nonlinearity

$$f(u) = (1-u_1)\exp\left(\frac{u_2}{1+u_2/\gamma}\right)$$

describing an irreversible first-order reaction, and boundary conditions

$$u_i'(0) - \mathrm{Pe}_i u_i(0) = 0, \quad u_i'(1) = 0, \quad i = 1, 2.$$

This nonlinear boundary value problem is known to exhibit solution profiles with extremely steep layers for the following set of parameters:

$$\mathrm{Pe}_1 = 320, \qquad \mathrm{Pe}_2 = 100,$$
$$\beta = 0.72, \qquad u_{2w} = -9.6,$$
$$B = 16.8, \qquad \gamma = 16.9.$$

As continuation parameter, the Damköhler number $\mathrm{Da} \geqq 0$ is chosen. A characteristic of this example is the moving of a steep layer region indicating the reaction zone with varying Damköhler number. This property proves to be a difficult feature for any adaptive mesh-selection procedure involved in a global boundary value solver. Figure 3 shows a sample of typical solution profiles for u_1. Starting from $\mathrm{Da} = 0$, COLCON

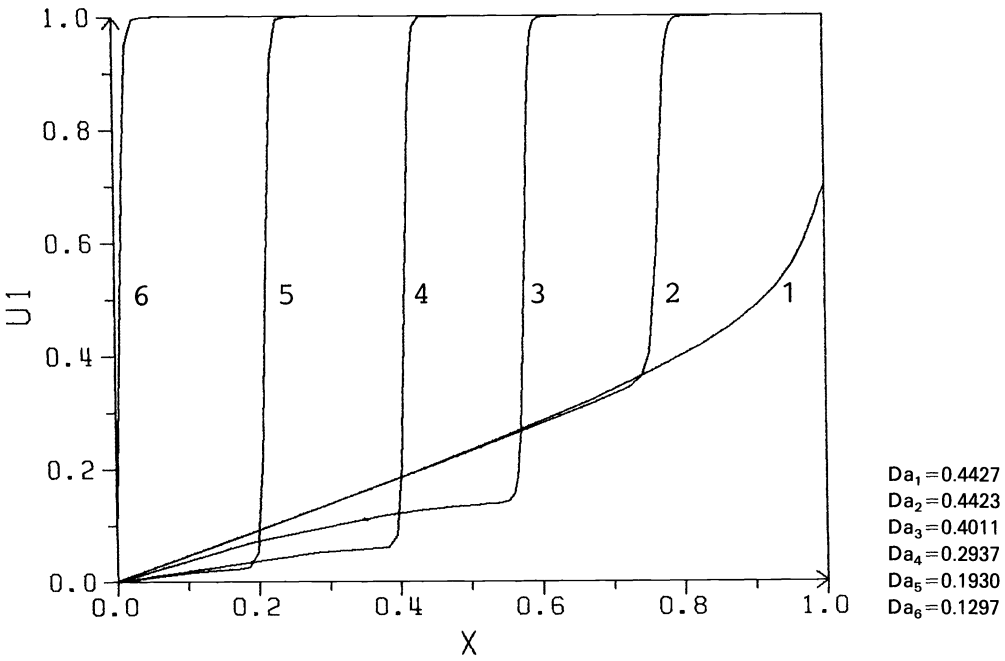


FIG. 3. *Typical solution profiles for Example* 4.

takes 253 continuation steps and 104.8 sec CPU-time to complete the run for the range $Da \in [0, 1]$.

REFERENCES

[1] M. ALUKO AND H.-C. CHANG, PEFLOQ: *An algorithm for the bifurcational analysis of periodic solutions of autonomous systems*, Comp. Chem. Engrg., 8 (1984), pp. 355-365.

[2] U. ASCHER, *Collocation for two-point boundary value problems revisited*, SIAM J. Numer. Anal., 23 (1986), pp. 596-609.

[3] U. ASCHER AND G. BADER, *A note on conditioning, stability and collocation matrices*, Tech. Report CMA-R16-86, Centre for Mathematical Analysis, Australian National University, Canberra, Australia, 1986.

[4] U. ASCHER, J. CHRISTIANSEN, AND R. D. RUSSELL, *A collocation solver for mixed order systems of boundary value problems*, Math. Comput., 33 (1979), pp. 659-679.

[5] ———, *Collocation software for boundary value* ODE's, ACM Trans. Math. Software, 7 (1981), pp. 209-222.

[6] U. ASCHER AND S. JACOBS, *On collocation implementation for singularly perturbed two-point problems*, Tech. Report 86-19, Department of Computer Science, University of British Columbia, British Columbia, Canada, 1986.

[7] 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.

[8] U. ASCHER AND R. D. RUSSELL, *Reformulation of boundary value problems into "standard form,"* SIAM Rev., 23 (1981), pp. 238-254.

[9] G. BADER, *Numerische Behandlung von Randwertproblemen für Funktionaldifferentialgleichungen*, Ph.D. thesis, University of Heidelberg, Heidelberg, Federal Republic of Germany, 1983.

[10] G. BADER AND U. ASCHER, *A new basis implementation for a mixed order boundary value solver*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 483-500.

[11] A. BEN-ISRAEL AND T. N. E. GREVILLE, *Generalized Inverses: Theory and Applications*, John Wiley, New York, 1974.

[12] C. DE BOOR AND R. WEISS, SOLVEBLOK: *A package for solving almost block diagonal linear systems*, ACM Trans. Math. Software, 6 (1980), pp. 80-87.

[13] P. DEUFLHARD, B. FIEDLER, AND P. KUNKEL, *Efficient numerical pathfollowing beyond critical points*, SIAM J. Numer. Anal., 24 (1987), pp. 912-927.

[14] ———, *Numerical pathfollowing beyond critical points in* ODE *models*, in Proc. Large Scale Scientific Computing, P. Deuflhard and B. Engquist, eds., Birkhäuser, Boston, 1987.

[15] P. DEUFLHARD AND G. HEINDL, *Affine invariant convergence theorems for Newton's method and extensions to related methods*, SIAM J. Numer. Anal., 16 (1979), pp. 1-10.

[16] E. J. DOEDEL, AUTO: *A program for the automatic bifurcation analysis of autonomous systems*, Congr. Numer., 30 (1981), pp. 265-284.

[17] E. J. DOEDEL AND J. P. KERNEVEZ, AUTO: *Software for continuation and bifurcation problems in ordinary differential equations*, Report, Department of Applied Mathematics, California Institute of Technology, Pasadena, CA, 1986.

[18] J. P. FINK AND W. C. RHEINBOLDT, *On the discretization error of parametrized nonlinear equations*, SIAM J. Numer. Anal., 20 (1983), pp. 732-746.

[19] F. DE HOOG AND R. M. M. MATTHEIJ, *The role of conditioning in shooting techniques*, in Proc. Workshop on Numerical Boundary Value ODE's, U. Ascher and R. D. Russell, eds., Birkhäuser, Boston, 1985.

[20] K. F. JENSEN AND W. H. RAY, *The bifurcation behavior of tubular reactors*, Tech. Report MRC#2172, Mathematics Research Center, University of Wisconsin, Madison, WI, 1981.

[21] C. KAAS-PETERSEN, *Technique to trace curves of bifurcation points of periodic solutions of dissipative systems*, Tech. Report, Centre for Nonlinear Studies, University of Leeds, Leeds, United Kingdom, 1987.

[22] ———, PATH—*user's guide*, Tech. Report, Centre for Nonlinear Studies, University of Leeds, Leeds, United Kingdom, 1987.

[23] H. B. KELLER, *Numerical solution of bifurcation and nonlinear eigenvalue problems*, in Proc. Applications of Bifurcation Theory, P. H. Rabinowitz, ed., Academic Press, New York, 1977.

[24] M. LENTINI AND V. PEREYRA, PASVA4: *An* ODE *boundary solver for problems with discontinuous interfaces and algebraic parameters*, Mat. Apl. Comput., 2 (1983), pp. 103-118.

[25] R. MEJIA AND J. L. STEPHENSON, *Numerical solution of multinephron kidney equations*, J. Comput. Phys., 32 (1979), pp. 235-246.

[26] V. PEREYRA, *Deferred corrections software and its application to seismic ray tracing*, Comput. Suppl., 5 (1984), pp. 211–226.

[27] R. SEYDEL, *A continuation algorithm with step control*, in Proc. Numerical Methods for Bifurcation Problems, T. Küpper, H. D. Mittelmann, and H. Weber, eds., International Series of Numerical Mathematics, 70, Birkhäuser, Basel, 1984.

[28] J. L. STEPHENSON, R. P. TEWARSON, AND R. MEJIA, *Quantitative analysis of mass and energy balance in non-ideal models of the renal counterflow system*, Proc. Nat. Acad. Sci. U.S.A., 71 (1974), pp. 1618–1622.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 89-103, January 1989
© 1989 Society for Industrial and Applied Mathematics
008

EFFICIENT DIRECT METHODS FOR SOLVING THE SPECTRAL COLLOCATION EQUATIONS FOR STOKES FLOW IN RECTANGULARLY DECOMPOSABLE DOMAINS*

TIMOTHY N. PHILLIPS† AND ANDREAS KARAGEORGHIS†‡

Abstract. A spectral element method is described for solving Stokes flow in rectangularly decomposable domains. The flow region is divided into a number of rectangular subregions, some of which may be semi-infinite. The solution to the governing biharmonic equation for the stream function is represented by an expansion of Chebyshev polynomials in each subregion. The coefficients in these expansions are determined by collocating the differential equation and boundary conditions, and imposing C^3 continuity across subregion interfaces. The spectral element method produces systems of equations that are block tridiagonal. Efficient direct methods based on capacitance matrix ideas are proposed that take advantage of the structure of the spectral element matrix. The use of these techniques produces savings in storage and time.

Key words. spectral methods, collocation, capacitance matrix method, rectangularly decomposable domains

AMS(MOS) subject classifications. 65N35, 76D07

1. Introduction.

In spectral methods we represent the solution to a differential equation by a truncated expansion of eigenfunctions of a singular Sturm-Liouville problem. This choice of trial functions is responsible for the superior approximation properties of spectral methods when compared with finite difference and finite element methods. This is, of course, most evident for problems with smooth solutions, in which case the series expansion converges faster than any finite power of $1/n$ as $n \to \infty$. This phenomenon is known as spectral convergence.

Spectral methods are most easily applied to problems defined in rectangular or circular regions, in which case Chebyshev or Fourier series expansions, respectively, are appropriate. However, the natural choices of expansion functions for a problem defined in an irregular geometry are unwieldy and inefficient to use and need to be computed for each new irregular region. There are two ways of circumventing this difficulty, namely, mapping and patching.

The mapping technique involves transforming the irregular region into a simpler one by using a coordinate transformation. Spectral methods can then be applied in the simpler region using standard expansion techniques. Orszag [19] illustrates this process in solving the heat equation in an annular region whose inner and outer boundaries are defined in polar coordinates, and has found that the rapid convergence of the spectral solution is maintained. Streett, Zang, and Hussaini [24] solve the problem of compressible potential flow past a two-dimensional airfoil by using a numerically generated conformal mapping of Jameson [12] to transform the airfoil onto the unit circle. The appearance of shock discontinuities in supercritical flows degrades the spectral accuracy of the solution; however, they find that fewer degrees of freedom are required than for finite difference methods to obtain equivalent solutions.

The second technique, patching, divides the region into a number of simpler subregions or elements. A spectral approximation to the solution of the differential equation is sought within each element. The representations are matched by imposing

* Received by the editors November 11, 1987; accepted for publication (in revised form) April 13, 1988.
† Department of Mathematics, University College of Wales, Aberystwyth, SY23 3BZ, United Kingdom.
‡ The work of this author was supported by the U.K. Science and Engineering Research Council.

continuity conditions across element interfaces. This procedure results in a coupling of the expansion coefficients in two contiguous elements. These methods combine the flexibility of finite element methods to treat problems in irregular regions with the superior approximation properties of spectral methods for problems possessing smooth solutions. The global element method of Delves and Hall [7] and the spectral element methods of Patera [20], Macaraeg and Streett [16], and Phillips and Davies [23] are methods that have been developed with this philosophy in mind. These methods differ in the choice of trial functions and the continuity conditions satisfied at element interfaces.

The study of the numerical simulation of non-Newtonian fluids has generated much interest in recent years. At the present time, finite element methods enjoy a high level of popularity and success in this field mainly due to their flexibility, which facilitates the numerical solution of problems defined in general, irregular regions. One of the remaining problems of major importance in the simulation of such flows is the so-called high Weissenberg number problem. This concerns the breakdown of convergence of a method or degradation of numerical solutions as the elasticity parameter, or Weissenberg number, is increased. Marchal and Crochet [17] have shown that in a mixed finite element formulation this problem can be avoided for larger values of the Weissenberg number if higher-order (Hermitian) trial functions are used. We plan to take this idea a step further by considering spectral methods which, by definition, use high-order global trial functions.

The governing equations in non-Newtonian fluid mechanics consist of field equations and constitutive equations. The field equations comprise the equation of continuity, the conservation of momentum, and the local expression of the principle of balance of angular momentum. For the plane, inertialess flow of an incompressible fluid, these statements assume the following mathematical form:

$$\nabla \cdot \underline{v} = 0, \tag{1.1}$$

$$\nabla \cdot \underline{P} = \underline{0}, \tag{1.2}$$

where $\underline{v}$ denotes the velocity field and $\underline{P}$ the Cauchy stress tensor. The principle of balance of angular momentum in the absence of body and surface couples requires that the Cauchy stress tensor $\underline{P}$ be symmetric.

The constitutive equation for a corotational Maxwell model is given by

$$\underline{T} + \lambda \mathring{\underline{T}} = 2\eta \underline{D}, \tag{1.3}$$

where $\underline{D}$ is the rate of deformation tensor, $\underline{T}$ is the extra-stress tensor, and λ and η are material constants. For an incompressible fluid, the motion of the continuum determines the stress tensor up to an arbitrary isotropic tensor, and thus $\underline{P}$ and $\underline{T}$ are related as follows:

$$\underline{P} = -p\underline{I} + \underline{T}, \tag{1.4}$$

where p is an arbitrary pressure and $\underline{I}$ the identity tensor. The corotational derivative $\mathring{\underline{T}}$ is an average of the upper- and lower-convected derivatives.

The introduction of a stream function ψ, defined by

$$u = \frac{\partial \psi}{\partial y}, \qquad v = -\frac{\partial \psi}{\partial x}, \tag{1.5}$$

means that (1.1) is automatically satisfied. Further, the stress field $\underline{P}$ may be represented in terms of an Airy stress function χ (scalar) as

$$P_{11} = \frac{\partial^2 \chi}{\partial y^2}, \quad P_{22} = \frac{\partial^2 \chi}{\partial x^2}, \quad P_{12} = -\frac{\partial^2 \chi}{\partial x \partial y}. \tag{1.6}$$

This representation ensures the satisfaction of (1.2). The only differential equation not already satisfied is therefore the constitutive equation (1.3). This equation may be written in terms of the scalar functions ψ, χ, and p, yielding three scalar differential equations in terms of these variables.

For plane creeping Newtonian flow, Coleman [4] eliminates the pressure, and by defining the complex potential ϕ to be $\chi + 2i\eta\psi$ shows that the constitutive equation may be written in the form

$$\frac{\partial^2 \phi}{\partial \bar{z}^2} = 0. \tag{1.7}$$

In a later paper Coleman [5] generalizes (1.7) to first include inertial effects and then the corotational Maxwell fluid. A further reduction in the number of scalar functions to be determined is possible for fluids with a traceless extra-stress tensor. The corotational Maxwell is an example of such a fluid. In this case the pressure may be eliminated to provide two equations in terms of ψ and χ. In terms of the complex potential defined above, the governing equation for the creeping flow of a corotational Maxwell fluid may be written in the following form:

$$\frac{\partial^2 \phi}{\partial \bar{z}^2} + \frac{i\lambda}{\eta}\left(\frac{\partial \operatorname{Im} \phi}{\partial \bar{z}}\frac{\partial}{\partial z} - \frac{\partial \operatorname{Im} \phi}{\partial z}\frac{\partial}{\partial \bar{z}} - 2\frac{\partial^2 \operatorname{Im} \phi}{\partial z \partial \bar{z}}\right)\frac{\partial^2 \operatorname{Re} \phi}{\partial \bar{z}^2} = 0. \tag{1.8}$$

This equation may be written in terms of the real variables ψ and χ and manipulated to obtain the following system:

$$\nabla^4 \psi = WA(\partial^2\psi, \partial^3\psi, \partial^4\psi, \partial^2\chi, \partial^3\chi, \partial^4\chi), \tag{1.9a}$$

$$(1 + W\underline{v} \cdot \underline{\nabla})\nabla^4 \chi = WB(\partial^2\psi, \partial^3\psi, \partial^4\psi, \partial^2\chi, \partial^3\chi, \partial^4\chi), \tag{1.9b}$$

where A and B are bilinear forms in their arguments, and ∂ denotes partial differentiation with respect to x or y. The Weissenberg number, W, is an elasticity parameter and is a measure of the fluid's memory.

The success of spectral methods in solving the *primary* problem, which is a one-dimensional model of (1.9b), has already been demonstrated [6], [14]. Plane-creeping Newtonian flow may be described using a stream function that satisfies a biharmonic equation. This type of flow is also known as Stokes flow. Efficient solution techniques need to be developed for solving the equations resulting from a spectral discretization if we are to have a realistic chance of solving (1.9). The present paper addresses this subject for Stokes flow, which is the simplest form of (1.9a) with $W = 0$.

We consider Stokes flow through an abruptly changing channel with contraction ratio $1 : \alpha$. The spectral element method is naturally applied to this problem by dividing the flow region into two semi-infinite domains. The semi-infinite elements can be handled in one of two ways: domain truncation, or algebraic-type mappings. The former technique is used extensively in computational fluid dynamics and involves truncating each of the semi-infinite elements at a large, but finite, distance from the contraction. Boundary conditions need to be imposed along the entry and exit sections. This procedure can cause problems if the entry length has not been chosen long enough so that a fully developed velocity profile can be specified at entry and exit. Algebraic-type mappings are used to transform each of the semi-infinite elements into finite elements through a transformation of one of the coordinate directions [1], [11]. A double Chebyshev expansion is used to represent the solution in each of the truncated or mapped elements. The coefficients in these expansions are determined by the spectral collocation method. Further details of this will be given later. The collocation approach is favoured over a variational formulation since, although a variational principle exists

for the biharmonic equation, no simple such principle exists for the Navier-Stokes equations [25] or the equations governing non-Newtonian flow.

Spectral methods produce algebraic systems for the expansion coefficients that are not sparse, unlike those associated with the finite difference and finite element methods. However, the spectral element method produces systems of equations that are block tridiagonal in which the blocks are full. This fact was not exploited by Karageorghis and Phillips [13], who solved the full system using a Crout factorization technique from the Numerical Algorithms Group (NAG) Library [18].

Efficient inversion of the matrices associated with spectral domain decomposition methods have been the subject of other papers. Patera [21] presents a direct fast solver for spectral element discretizations of separable elliptic equations on rectangularly decomposable domains. Macaraeg and Streett [16] have adapted an influence matrix technique to spectral patching methods.

In this paper we take advantage of the structure of the spectral element matrix in constructing direct solution techniques. The first, known as coefficient splitting, partitions the matrix in such a way that the size of the blocks correspond to the number of unknown expansion coefficients in each element. The second is an application of the capacitance matrix technique [2]. We should note that the interface conditions in the spectral element method are much more complex than in the finite difference method [3], [9]. Typically they involve all the expansion coefficients in adjoining elements. The rows of the spectral element matrix corresponding to these conditions are full.

The paper is structured as follows. In § 2 we describe the problem and the spectral collocation strategy. In § 3 we describe methods of treating unbounded domains either by domain truncation or by an algebraic-type mapping. Direct methods of solving the spectral collocation equations are given in § 4. In § 5 we present numerical results and compare the different direct methods. Our concluding remarks are made in § 6.

2. Spectral collocation equations.

2.1. Governing equations. We consider Stokes flow through a $1:\alpha$ contraction, shown in Fig. 1. The governing equations are expressed in terms of a biharmonic equation for the stream function. This formulation has the advantage that the continuity equation is automatically satisfied and no boundary condition for the vorticity needs to be derived.

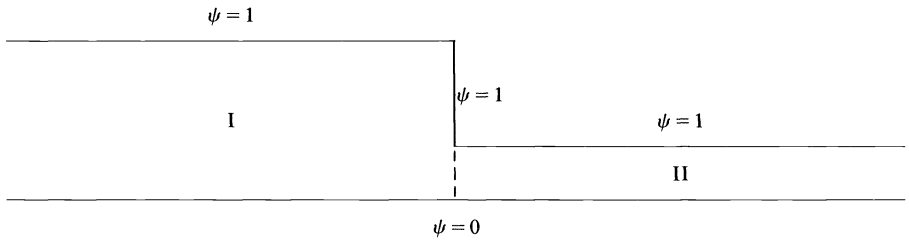


FIG. 1. *Contraction geometry.*

Some of the boundary conditions are also depicted in Fig. 1. We assume that the flow is symmetric about the line $y = 0$ so that only the upper half of the channel flow need be considered. We have no-slip constraints on the upper channel wall, and along

$y = 0$ we impose the symmetry conditions

$$\psi = 0, \qquad \frac{\partial^2 \psi}{\partial y^2} = 0.$$

At entry and exit we impose a parabolic velocity profile. In terms of the stream function this means that $\psi(x, y) \to G(y)$ as $x \to -\infty$ for $0 \leqq y \leqq 1$ and $\psi(x, y) \to G(y/\alpha)$ as $x \to \infty$ for $0 \leqq y \leqq \alpha$, where $G(y) = \frac{1}{2} y(3 - y^2)$.

2.2. Spectral collocation strategy. The flow region is divided into two semi-infinite rectangular elements, as shown in Fig. 1. In each of these elements we represent the solution in terms of a truncated series of trial functions. The coefficients in these series expansions are determined by collocating the differential equation and the solutions in the two elements are matched by imposing the correct order of weak continuity across the element interface.

In regions I and II, $\psi(x, y)$ is approximated by $\psi^{\mathrm{I}}(x, y)$ and $\psi^{\mathrm{II}}(x, y)$, respectively, where

$$\psi^{\mathrm{I}}(x, y) = G(y) + \sum_{m=0}^{M} \sum_{n=0}^{N} a_{mn} f_m^{\mathrm{I}}(x) g_n^{\mathrm{I}}(y), \tag{2.1}$$

$$\psi^{\mathrm{II}}(x, y) = G\left(\frac{y}{\alpha}\right) + \sum_{m=0}^{K} \sum_{n=0}^{N} b_{mn} f_m^{\mathrm{II}}(x) g_n^{\mathrm{II}}(y). \tag{2.2}$$

The undefined functions f and g appearing on the right-hand sides of (2.1) and (2.2) are appropriately chosen trial functions. Specific choices of these functions are given in § 3 when the treatment of the unbounded domain is discussed.

The spectral collocation strategy is performed as described in Karageorghis and Phillips [13]. Consider the following two sets of points defined, respectively, in regions I and II:

$$S_1 = \{(x_i^{\mathrm{I}}, y_j^{\mathrm{I}}) : 0 \leqq i \leqq M, 0 \leqq j \leqq N, x_M^{\mathrm{I}} = 0, y_0^{\mathrm{I}} = 0, y_N^{\mathrm{I}} = 1\},$$

$$S_2 = \{(x_i^{\mathrm{II}}, y_j^{\mathrm{II}}) : 0 \leqq i \leqq K, 0 \leqq j \leqq N, x_0^{\mathrm{II}} = 0, y_0^{\mathrm{II}} = 0, y_N^{\mathrm{II}} = \alpha\}.$$

The differential equation is collocated at sets $T_1 \subset S_1$ and $T_2 \subset S_2$ of collocation points in regions I and II, respectively. This means that the differential equation is exactly satisfied at points belonging to these sets, i.e.,

$$\nabla^4 \psi^{\mathrm{I}}(x, y) = 0 \quad \text{for } (x, y) \in T_1, \tag{2.3}$$

$$\nabla^4 \psi^{\mathrm{II}}(x, y) = 0 \quad \text{for } (x, y) \in T_2. \tag{2.4}$$

The correct order of weak continuity for this problem is C^3. Across the interface $x = 0$ between the elements the weak continuity is applied in a collocation sense. The precise way in which this is done is now described. The C^0 and C^1 continuity conditions are imposed at a set of collocation points $T_3 \subset S_1$ across the interface $x = 0$, $0 \leqq y \leqq 1$. The C^2 and C^3 continuity conditions, on the other hand, are applied at a set of collocation points $T_4 \subset S_2$. In mathematical terms these conditions may be written as follows:

$$\psi^{\mathrm{I}}(0, y) = \begin{cases} \psi^{\mathrm{II}}(0, y) & \text{if } (0, y) \in T_3 \text{ and } 0 \leqq y \leqq \alpha, \\ 1 & \text{if } (0, y) \in T_3 \text{ and } \alpha \leqq y \leqq 1, \end{cases}$$

$$\frac{\partial \psi^{\mathrm{I}}}{\partial x}(0, y)=\begin{cases}\dfrac{\partial \psi^{\mathrm{II}}}{\partial x}(0, y) & \text{if } (0, y) \in T_3 \text{ and } 0 \leqq y \leqq \alpha, \\ 0 & \text{if } (0, y) \in T_3 \text{ and } \alpha \leqq y \leqq 1,\end{cases}$$

$$\frac{\partial^2 \psi^{\mathrm{I}}}{\partial x^2}(0, y)=\frac{\partial^2 \psi^{\mathrm{II}}}{\partial x^2}(0, y) \quad \text{if } (0, y) \in T_4 \text{ and } 0 \leqq y \leqq \alpha,$$

$$\frac{\partial^3 \psi^{\mathrm{I}}}{\partial x^3}(0, y)=\frac{\partial^3 \psi^{\mathrm{II}}}{\partial x^3}(0, y) \quad \text{if } (0, y) \in T_4 \text{ and } 0 \leqq y \leqq \alpha.$$

When all the other boundary conditions are satisfied the sets T_1, T_2, T_3, and T_4 are chosen in the following way to obtain the same number of equations as unknowns:

$$T_1=\{(x_i^{\mathrm{I}}, y_j^{\mathrm{I}}): 0 \leqq i \leqq M-2, 2 \leqq j \leqq N-2\},$$

$$T_2=\{(x_i^{\mathrm{II}}, y_j^{\mathrm{II}}): 3 \leqq i \leqq K, 2 \leqq j \leqq N-2\},$$

$$T_3=\{(0, y_j^{\mathrm{I}}): 2 \leqq j \leqq N-2\},$$

$$T_4=\{(0, y_j^{\mathrm{II}}): 2 \leqq j \leqq N-2\}.$$

The omission of the extreme collocation points in S_1 and S_2 means that near the boundaries and the element interface the approximations ψ^{I} and ψ^{II} are influenced more by the boundary conditions and interface continuity conditions than by the differential equation. When some of the boundary conditions remain to be satisfied then the sets T_1 and T_2 are modified accordingly.

3. Treatment of unbounded domains.

3.1. Truncation of domain. In this approach the domain is truncated at a distance h_1 from the origin in element I and h_2 in element II. In the truncated domain we use a double Chebyshev representation to approximate the solution in each element. This choice of trial functions is known to have good approximating properties [8], [10] for both linear and nonlinear differential equations. The trial functions in the y-direction are modified so that they automatically satisfy the boundary conditions on the top and bottom of the channel. It is straightforward to show that these trial functions, $P_n(y)$, are defined as follows on the interval $[0, 1]$:

$$P_n(y)=T_n^*(y)+\alpha_n T_3^*(y)+\beta_n T_2^*(y)+\gamma_n T_1^*(y)+\delta_n T_0^*(y),$$

where

$$T_n^*(y)=T_n(2y-1)=\cos\left(n \cos^{-1}(2y-1)\right).$$

The constants α_n, β_n, γ_n, and δ_n are given by

$$\alpha_n=\{-n^2-\tfrac{1}{2}(-1+(-1)^n)+(-1)^n n^2(n^2-1)/3\}/32,$$

$$\beta_n=6\alpha_n-(-1)^n n^2(n^2-1)/12,$$

$$\gamma_n=\alpha_n-\tfrac{1}{2}(-1+(-1)^n),$$

$$\delta_n=-1-\alpha_n-\beta_n-\gamma_n.$$

The points at which the extreme values of the Chebyshev polynomials are attained are well known to give rise to optimal approximation properties of smooth functions. Therefore we choose the collocation points in the x- and y-directions to be related to the extrema of the Chebyshev polynomial of highest degree used in the solution representations.

Since the domain is truncated we need to impose boundary conditions along the entry and exit sections. In this respect we assume a fully developed velocity profile at the inflow $x=-h_1$ and at the outflow $x=h_2$. The choice of h_1 and h_2 is discussed in § 5.

3.2. Mapping of domain. Each of the semi-infinite elements can be transformed onto finite elements using an algebraic mapping in the semi-infinite direction. This extends the work of Grosch and Orszag [11] to differential equations defined in two space dimensions. Let R_I denote element I, i.e.,

$$R_I=\{(x, y): -\infty<x\leqq 0, 0\leqq y\leqq 1\};$$

then under the algebraic mapping

$$z=x/(x+L_I), \qquad L_I<0,$$

R_I is mapped onto V_I, where

$$V_I=\{(z, y): 0\leqq z\leqq 1, 0\leqq y\leqq 1\}.$$

The constant L_I is known as the mapping factor.

Similarly element II is mapped onto a finite element using a corresponding transformation. The differential equation is transformed to be valid in each of the two new finite rectangular elements by transforming derivatives with respect to x to derivatives with respect to z. The interface continuity conditions are also modified under this mapping.

In the mapped elements a double Chebyshev expansion of the kind described in the domain-truncation method is used together with a similar choice of collocation points.

4. Direct methods of solution. In both the domain-truncation and algebraic-mapping approaches, the resulting system of linear equations for the expansion coefficients takes the form

$$\left(\begin{array}{c|c} A & 0 \\ \hline C & \\ \hline D & \\ \hline 0 & B \end{array}\right)\left(\begin{array}{c} \mathbf{a} \\ \hline \mathbf{b} \end{array}\right)=\left(\begin{array}{c} \mathbf{0} \\ \hline \mathbf{r} \\ \hline \mathbf{0} \end{array}\right). \tag{4.1}$$

The dimension of the system appearing in (4.1) is $n_1+2n_2+n_3$, where

$$n_1=(M-1)(N-3), \quad n_2=2(N-3), \quad n_3=(K-1)(N-3).$$

The vectors **a** and **b** contain the unknown expansion coefficients in regions I and II, respectively.

The matrices A and B are of dimension $n_1\times(n_1+n_2)$ and $n_3\times(n_2+n_3)$, respectively. The first n_1 rows and last n_3 rows of (4.1) correspond to the satisfaction of the biharmonic equation at the collocation points in regions I and II, respectively. In the case of the domain-truncation method, the first n_2 rows of A and last n_2 rows of B correspond to the satisfaction of the fictitious boundary conditions along the lines of truncation $x=-h_1$ and $x=h_2$, respectively. The matrix C is of dimension $n_2\times(n_1+2n_2+n_3)$ and corresponds to the collocation of the C^0 and C^1 continuity conditions across $x=0$, $0\leqq y\leqq 1$. The matrix D has the same dimension as C and corresponds to the collocation of the C^2 and C^3 continuity conditions across the element interface $x=0$, $0\leqq y\leqq\alpha$. The vector **r** appearing in the right-hand side of (4.1) corresponds to the quantities independent of the expansion coefficients in the continuity conditions, and its last n_2 rows are null.

The system (4.1) is solved [13] without modification using a Crout factorization technique from the NAG Library [18]. Thus $k(n_1+2n_2+n_3)^3$ operations are needed to obtain the solution where k is some constant. In the following, whenever a linear system needs to be solved, we use this subroutine from the NAG Library. No account is taken of the large number of zero elements that appear in (4.1). The two large zero submatrices in (4.1) contain a total of $n_1n_2+2n_1n_3+n_2n_3$ zero elements. Failure to incorporate the underlying matrix structure into our solution technique means that the size of the system we are able to solve is restricted. We now consider two partitioning techniques that suggest alternative direct methods of solution that are more efficient in terms of storage locations and computational time.

4.1. Coefficient splitting. In this technique we partition the system (4.1) in such a way that the size of the blocks correspond to the number of unknown expansion coefficients in each element. Therefore we rewrite (4.1) in the form

$$\left(\begin{array}{c|c} S & \begin{matrix}0\\ V\end{matrix} \\ \hline \begin{matrix}U\\ 0\end{matrix} & T \end{array}\right)\left(\frac{\mathbf{a}}{\mathbf{b}}\right)=\left(\frac{\mathbf{s}}{\mathbf{0}}\right). \tag{4.2}$$

The square matrices S and T are of order (n_1+n_2) and (n_2+n_3), respectively, and are related to the submatrices appearing in (4.1) in the following way:

$$S=\left(\frac{A}{C^{(1)}}\right), \qquad T=\left(\frac{D^{(2)}}{B}\right),$$

where

$$C^{(1)}(i,j)=C(i,j), \qquad 1\leqq i\leqq n_2, \quad 1\leqq j\leqq n_1+n_2,$$
$$D^{(2)}(i,j)=D(i,j+n_1+n_2), \qquad 1\leqq i\leqq n_2, \quad 1\leqq j\leqq n_2+n_3.$$

The matrices U and V have dimensions $n_2\times(n_1+n_2)$ and $n_2\times(n_2+n_3)$, respectively, and are related to C and D in (4.1) by

$$U(i,j)=D(i,j), \qquad 1\leqq i\leqq n_2, \quad 1\leqq j\leqq n_1+n_2,$$
$$V(i,j)=C(i,j+n_1+n_2), \qquad 1\leqq i\leqq n_2, \quad 1\leqq j\leqq n_2+n_3.$$

Finally, the vector $\mathbf{s}$ is related to $\mathbf{r}$ by

$$\mathbf{s}(i)=0, \qquad 1\leqq i\leqq n_1,$$
$$\mathbf{s}(i+n_1)=\mathbf{r}(i), \qquad 1\leqq i\leqq n_2.$$

We may write the system (4.2) in component form as follows:

$$S\mathbf{a}+V^*\mathbf{b}=\mathbf{s}, \tag{4.3}$$

$$U^*\mathbf{a}+T\mathbf{b}=\mathbf{0}, \tag{4.4}$$

where

$$V^*=\left(\frac{0}{V}\right), \qquad U^*=\left(\frac{U}{0}\right).$$

Assuming that T is invertible, we may premultiply (4.4) by T^{-1} and substitute for $\mathbf{b}$ in (4.3) to obtain a system of order n_1+n_2 for $\mathbf{a}$:

$$(S-V^*T^{-1}U^*)\mathbf{a}=\mathbf{s}. \tag{4.5}$$

The system is efficiently solved using the following algorithm.

ALGORITHM 1.
(1) Invert T.
(2) Evaluate $S - V^* T^{-1} U^*$.
(3) Solve the system $(S - V^* T^{-1} U^*)\mathbf{a} = \mathbf{s}$ for $\mathbf{a}$.
(4) Evaluate $\mathbf{b} = -T^{-1} U^* \mathbf{a}$.

The computationally expensive part of this process occurs in Steps (1) and (3) of Algorithm 1. Matrix inversion and the solution of linear systems of equations are performed using the NAG Library [18]. Care is taken in forming $S - V^* T^{-1} U^*$ to ensure that advantage is taken of the large number of zeros appearing in U^* and V^* by not performing needless multiplications involving zero.

When the system (4.1) was solved originally [13], without taking into account its sparsity, the total number of operations required was $k(n_1+2n_2+n_3)^3+(n_1+2n_2+n_3)^2$. The major costs in the coefficient-splitting technique are the $k(n_2+n_3)^3+(n_1+n_2)\times(n_2+n_3)^2$ operations required to invert T (Step 1), and the $k(n_1+n_2)^3+(n_1+n_2)^2$ operations required to solve (4.5) (Step 3). Since $n_1, n_3 \gg n_2$ the coefficient-splitting method incurs savings in the number of operations of $O(n_1 n_3(n_1+n_3))$ to solve the system and in the number of storage locations required of $O(n_1 n_3)$.

4.2. Capacitance matrix method. In this section we show how the capacitance matrix method [2] may be generalized to solve the spectral element equations in rectangularly decomposable domains. To illustrate this we rewrite (4.1) in the partitioned form:

$$\left(\begin{array}{c|c|c} E & F & 0 \\ \hline G & H & R \\ \hline 0 & P & Q \end{array}\right)\left(\begin{array}{c} \mathbf{x} \\ \hline \mathbf{y} \\ \hline \mathbf{z} \end{array}\right)=\left(\begin{array}{c} \mathbf{0} \\ \hline \mathbf{t} \\ \hline \mathbf{0} \end{array}\right). \tag{4.6}$$

The square matrices E, H, and Q are of order n_1, $2n_2$, and n_3, respectively, and are related to those in (4.1) as follows:

$$E(i,j) = A(i,j), \qquad 1 \leqq i \leqq n_1, \quad 1 \leqq j \leqq n_1,$$
$$H(i,j) = C(i,j+n_1), \qquad 1 \leqq i \leqq n_2, \quad 1 \leqq j \leqq 2n_2,$$
$$H(i+n_2,j) = D(i,j+n_1), \qquad 1 \leqq i \leqq n_2, \quad 1 \leqq j \leqq 2n_2,$$
$$Q(i,j) = B(i,j+n_2), \qquad 1 \leqq i \leqq n_3, \quad 1 \leqq j \leqq n_3.$$

Similarly the matrices F, G, R, and P are related to those of (4.1) in the following way:

$$F(i,j) = A(i,j+n_1), \qquad 1 \leqq i \leqq n_1, \quad 1 \leqq j \leqq n_2,$$
$$F(i,j+n_1+n_2) = 0, \qquad 1 \leqq i \leqq n_1, \quad 1 \leqq j \leqq n_2,$$
$$G(i,j) = C(i,j), \qquad 1 \leqq i \leqq n_2, \quad 1 \leqq j \leqq n_1,$$
$$G(i+n_2,j) = D(i,j), \qquad 1 \leqq i \leqq n_2, \quad 1 \leqq j \leqq n_2,$$
$$R(i,j) = C(i,j+n_1+2n_2), \qquad 1 \leqq i \leqq n_2, \quad 1 \leqq j \leqq n_3,$$
$$R(i+n_2,j) = D(i,j+n_1+2n_2), \qquad 1 \leqq i \leqq n_2, \quad 1 \leqq j \leqq n_3,$$
$$P(i,j) = 0, \qquad 1 \leqq i \leqq n_3, \quad 1 \leqq j \leqq n_2,$$
$$P(i,j+n_2) = B(i,j), \qquad 1 \leqq i \leqq n_3, \quad 1 \leqq j \leqq n_2.$$

The vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ contain the unknown expansion coefficients and are of length n_1, $2n_2$, and n_3, respectively. The vector $\mathbf{t}$ forms part of the right-hand side vector and is related to $\mathbf{r}$ in (4.1) by

$$\mathbf{t}(i) = \mathbf{r}(i), \qquad 1 \leqq i \leqq n_2,$$

$$\mathbf{t}(i) = 0, \qquad n_2 + 1 \leqq i \leqq 2n_2.$$

To apply the capacitance matrix approach we write system (4.6) in the natural component form suggested by the partitioning

$$E\mathbf{x} + F\mathbf{y} = \mathbf{0}, \tag{4.7}$$

$$G\mathbf{x} + H\mathbf{y} + R\mathbf{z} = \mathbf{t}, \tag{4.8}$$

$$P\mathbf{y} + Q\mathbf{z} = \mathbf{0}. \tag{4.9}$$

We write $\mathbf{x}$ and $\mathbf{z}$ in terms of $\mathbf{y}$ by premultiplying (4.7) and (4.9) by the inverses of E and Q, respectively, i.e.,

$$\mathbf{x} = -E^{-1}F\mathbf{y}, \qquad \mathbf{z} = -Q^{-1}P\mathbf{y}. \tag{4.10}$$

Eliminating $\mathbf{x}$ and $\mathbf{z}$ from (4.8), we obtain the following system of order $2n_2$ for $\mathbf{y}$:

$$(H - GE^{-1}F - RQ^{-1}P)\mathbf{y} = \mathbf{t}, \tag{4.11}$$

where the coefficient matrix on the left-hand side of (4.11) is known as the capacitance matrix. This system can be solved efficiently using the following algorithm.

ALGORITHM 2.

(1) Calculate $E^{-1}F$ by solving a system of the form

$$EW_1 = F.$$

Exploit the fact that F has only n_2 nonzero columns, and therefore we only need to solve a system with n_2 right-hand sides. We use the same subroutine from the NAG Library as mentioned earlier to do this.

(2) Calculate $Q^{-1}P$ by solving the system

$$QW_2 = P,$$

for W_2. Again we exploit the fact that P has n_2 zero columns. The solution W_2 is again obtained using the NAG Library.

(3) Evaluate the capacitance matrix

$$H - GW_1 - RW_2,$$

exploiting the n_2 zero columns in both W_1 and W_2.

(4) Solve the following system of order $2n_2$ for $\mathbf{y}$:

$$(H - GW_1 - RW_2)\mathbf{y} = \mathbf{t}.$$

(5) Evaluate $\mathbf{x}$ and $\mathbf{z}$ by substitution of $\mathbf{y}$ in (4.10).

Most of the work in this algorithm is expended in Steps (1), (2), and (4). The solution of n_2 systems of order n_1 in Step (1) with the same coefficient matrix requires $k(n_1)^3 + n_2(n_1)^2$ operations, where k is a constant. Step (2) involves the solution of n_2 systems each of order n_3 requiring a total of $k(n_3)^3 + n_2(n_3)^2$ operations. The solution of the system of order $2n_2$ in Step (4) accounts for $8k(n_2)^3 + (n_2)^2$ operations. The

capacitance matrix method requires $O(n_2(n_1+n_3)^2)$ fewer operations to solve the system (4.6) than the coefficient-splitting method and requires a smaller number of storage locations.

5. Numerical results. We compare the performance and efficiency of the techniques described in § 4, i.e., the original method that does not take into account the sparsity, the coefficient-splitting method, and the capacitance matrix method. To do this we examine the computational cost of each method in solving the Stokes flow through a 4:1 contraction, i.e., $\alpha = \frac{1}{4}$ in the notation of § 2. We consider two cases involving different numbers of degrees of freedom. In the first test case, relatively low-order polynomials are used in the x-direction in each element. We choose $N = 16$, $M = 10$, and $K = 6$, setting the total number of degrees of freedom at 234. In the second test case higher-order polynomials are used to approximate the solution. For this problem there are a total of 338 degrees of freedom with $N = 16$, $M = 16$, and $K = 8$. Table 1 contains the CPU (CDC 7600) time in seconds for each case and for each method. Table 2 contains the storage required in kbytes for each case and each of the three methods. The capacitance matrix method is always more efficient than the other two, and becomes more efficient as the size of the problem grows. The capacitance matrix method requires slightly fewer storage locations than the coefficient-splitting method and less CPU time. In the first case the original method performs better than the coefficient-splitting method. This is because for a relatively small system the work needed in the manipulation required to obtain a system of reduced size is not negligible. The capacitance matrix technique enables us to solve problems with the order of 1200 degrees of freedom, which is far more than is possible using the original method. The savings in CPU time become greater when solving a nonlinear problem such as the Navier–Stokes equations, where matrix systems possessing the structure of those considered here need to be solved at each Newton step.

The eigenvalues of all the systems were investigated. Recalling that M and N are the degrees of the polynomial of highest order used in the x- and y-directions, respectively, then it was observed that the largest eigenvalue behaves like $O(M^4N^4)$. This ill-conditioning of Chebyshev systems is well known [10], [26].

TABLE 1

CPU (CDC 7600) times required to solve the 4:1 contraction problem for different numbers of degrees of freedom.

Degrees of freedom	Original method	Coefficient splitting	Capacitance matrix
234	27.0	31.0	25.8
338	68.0	62.0	51.0

TABLE 2

Storage in kbytes for solving the 4:1 contraction problem for different numbers of degrees of freedom.

Degrees of freedom	Original method	Coefficient splitting	Capacitance matrix
234	99	93	89
338	175	138	133

Finally, we give contour plots of the solution for different values of contraction ratio α. In Fig. 2 we present contours of the stream function for $\alpha = \frac{1}{4}$. The system of equations in this case was constructed using the algebraic mapping technique with the particular choices of mapping factor being -2.4 in region I and 0.7 in region II. There are a total of 390 degrees of freedom with $N = 16$, $M = 19$, and $K = 9$. In Figs. 3 and 4 we give the contours of the stream function when $\alpha = \frac{1}{2}$ for 336 and 392 degrees of freedom, respectively. Here the domain-truncation method is used with $h_1 = 1.5$ and $h_2 = 0.5$. This demonstrates how the smoothness of the contours improve as the number of degrees of freedom is increased. The use of the capacitance matrix method has enabled us to solve a problem with a larger number of degrees of freedom than was possible previously.

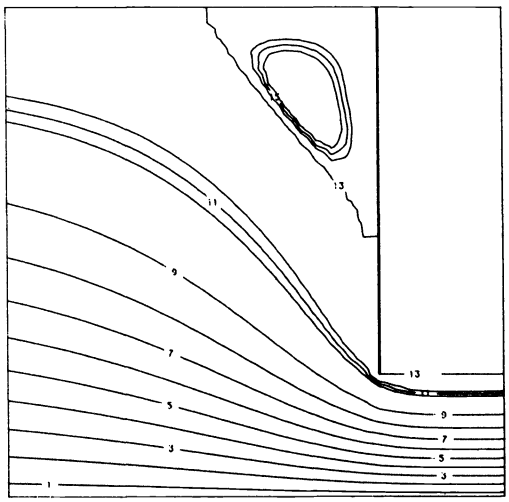


FIG. 2. *Contours of the solution of the* 4:1 *contraction problem with* 390 *degrees of freedom.*

TABLE 3
Contour key for the stream function figures for Figs. 2-4.

	$\alpha = \frac{1}{4}$	$\alpha = \frac{1}{2}$
1	0.0500	0.0500
2	0.1500	0.1500
3	0.2500	0.2500
4	0.3500	0.3500
5	0.4500	0.4500
6	0.5500	0.5500
7	0.6500	0.6500
8	0.7500	0.7500
9	0.8500	0.8500
10	0.9500	0.9500
11	0.9600	0.9600
12	0.9700	0.9700
13	1.0000	1.0000
14	1.0001	1.0004
15	1.0003	1.0005
16	1.0005	1.0006

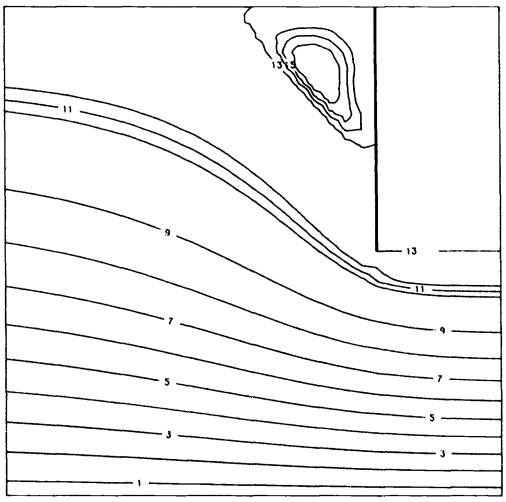


FIG. 3. *Contours of the solution of the* 2:1 *contraction problem with* 336 *degrees of freedom.*

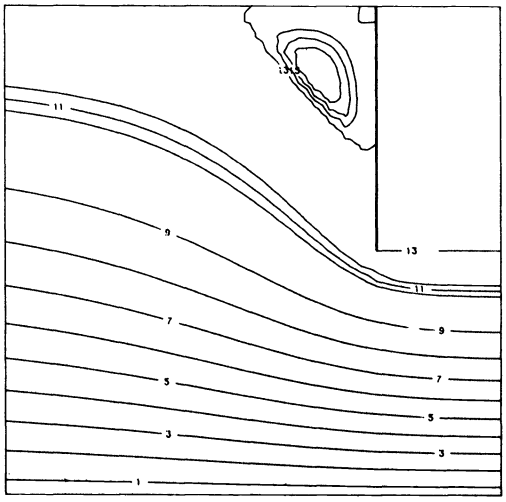


FIG. 4. *Contours of the solution of the* 2:1 *contraction problem with* 392 *degrees of freedom.*

The contours appear a little ragged due to the contouring routine. Although we have a global solution, the NAG routine for contouring a continuous function failed because of sharp changes in the gradient of the solution in the recirculation region. To overcome this a uniform mesh was placed over part of the domain and a NAG routine interpolating the values of the solution at these points was used. The Stokes problem possesses a singularity at the reentrant corner that may be treated using a postprocessing technique [22], but this is not the subject of the present paper.

6. Conclusions. The spectral element collocation method for the solution of the biharmonic equation results in a system of linear equations with a block tridiagonal structure. Each of these nonzero blocks is full. The same structure is obtained for both

a domain truncation and algebraic-type mapping treatment of the semi-infinite elements. The structure of the spectral element matrix is exploited using a coefficient-splitting technique and also a capacitance matrix method. These methods have been used previously in the solution of elliptic differential equations in conjunction with other numerical techniques like finite difference methods, for example. In the present paper these methods have been used for the first time in the spectral element context.

The capacitance matrix method proved to be the most efficient technique leading to reductions in the time, cost, and storage locations required for the solution of the problem. For the examples considered here an improvement of around 25% was achieved in CPU time. However, we can expect the savings to be greater for problems with larger numbers of degrees of freedom. The coefficient-splitting method was more economical than the original method for large problems. The capacitance matrix method enabled us to solve larger systems than when the original method was used. This means that more accurate solutions can be obtained. This is particularly important as we set out to solve nonlinear problems such as the Navier–Stokes equations and the equations governing non-Newtonian flow. When either of these systems of nonlinear equations are solved using a Newton linearization process, the spectral element method produces a coefficient matrix of the same structure as that investigated in this paper. Although we have restricted our attention to the contraction geometry the capacitance matrix idea can be extended to other rectangularly decomposable domains. The application of the spectral element method leads to a block tridiagonal system in each case.

REFERENCES

[1] J. P. BOYD, *The optimization of convergence for Chebyshev polynomial methods in an unbounded domain*, J. Comput. Phys., 45 (1982), pp. 43–79.

[2] B. L. BUZBEE, F. W. DORR, J. A. GEORGE, AND G. H. GOLUB, *The direct solution of the discrete Poisson equation on irregular regions*, SIAM J. Numer. Anal., 8 (1971), pp. 722–736.

[3] T. F. CHAN AND D. C. RESASCO, *A domain-decomposed fast Poisson solver on a rectangle*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s14–s26.

[4] C. J. COLEMAN, *A contour integral formulation of plane creeping Newtonian flow*, Quart. J. Mech. Appl. Math., 34 (1981), pp. 453–464.

[5] ———, *On the use of complex variables in the analysis of flows of an elastic fluid*, J. Non-Newtonian Fluid Mech., 15 (1984), pp. 227–238.

[6] A. R. DAVIES, A. KARAGEORGHIS, AND T. N. PHILLIPS, *Spectral Galerkin methods for the primary two-point boundary value problem in modelling viscoelastic flows*, Internat. J. Numer. Methods Engrg., 26 (1988), pp. 647–662.

[7] L. M. DELVES AND C. A. HALL, *An implicit matching principle for global element calculations*, J. Inst. Maths. Applics., 23 (1979), pp. 223–234.

[8] L. FOX AND I. B. PARKER, *Chebyshev Polynomials in Numerical Analysis*, Clarendon Press, Oxford, 1968.

[9] G. H. GOLUB AND D. F. MAYERS, *The use of preconditioning over irregular regions*, 1983 Lecture at Sixth International Conference on Computing Methods in Applied Sciences and Engineering, Versailles, France, December 1983.

[10] 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.

[11] C. E. GROSCH AND S. A. ORSZAG, *Numerical solution of problems in unbounded regions: coordinate transforms*, J. Comput. Phys., 25 (1977), pp. 273–296.

[12] A. JAMESON, *Acceleration of transonic flow calculations on arbitrary meshes by the multiple grid method*, AIAA Paper 79-1458, American Institute of Aeronautics and Astronautics, New York, 1979.

[13] A. KARAGEORGHIS AND T. N. PHILLIPS, *Spectral collocation methods for Stokes flow in contraction geometries and unbounded domains*, Tech. Report ABERNA 13, Department of Mathematics, University College of Wales, Aberystwyth, U.K., 1987; J. Comput. Phys., to appear.

[14] A. KARAGEORGHIS, T. N. PHILLIPS, AND A. R. DAVIES, *Spectral collocation methods for the primary two-point boundary value problem in modelling viscoelastic flows*, Internat. J. Numer. Methods Engrg., 26 (1988), pp. 805–813.

[15] 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., 8 (1987), pp. s166–s202.

[16] M. MACARAEG AND C. L. STREETT, *Improvements in spectral collocation through a multiple domain technique*, Appl. Numer. Math., 2 (1986), pp. 95–108.

[17] J. M. MARCHAL AND M. J. CROCHET, *Hermitian finite elements for calculating viscoelastic flow*, J. Non-Newtonian Fluid Mech., 20 (1986), pp. 187–208.

[18] Numerical Algorithms Group Library Mark 11, NAG (UK) Ltd., Banbury Road, Oxford, U.K.

[19] S. A. ORSZAG, *Spectral methods in complex geometries*, J. Comput. Phys., 37 (1980), pp. 70–92.

[20] A. T. PATERA, *A spectral element method for fluid dynamics: laminar flow in a channel expansion*, J. Comput. Phys., 54 (1984), pp. 468–488.

[21] ———, *Fast direct Poisson solvers for high-order finite element discretizations in rectangularly-decomposable domains*, J. Comput. Phys., 65 (1986), pp. 474–480.

[22] T. N. PHILLIPS, *Singular matched eigenfunction expansions for Stokes flow around a corner*, Tech. Report ABERNA 18, Department of Mathematics, University College of Wales, Aberystwyth, U.K., 1988; IMA J. Appl. Math., submitted.

[23] T. N. PHILLIPS AND A. R. DAVIES, *On semi-infinite spectral elements for Poisson problems with reentrant boundary singularities*, J. Comput. Appl. Math., 21 (1988), pp. 173–188.

[24] C. L. STREETT, T. A. ZANG, AND M. Y. HUSSAINI, *Spectral multigrid methods with applications to transonic potential flow*, J. Comput. Phys., 57 (1985), pp. 43–76.

[25] F. THOMASSET, *Implementation of Finite Element Methods for Navier–Stokes Equations*, Springer-Verlag, New York, 1981.

[26] T. A. ZANG, Y. S. WONG, AND M. Y. HUSSAINI, *Spectral multigrid methods for elliptic equations*, J. Comput., 48 (1982), pp. 485–501.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 104-119, January 1989
© 1989 Society for Industrial and Applied Mathematics
009

TOEPLITZ EQUATIONS BY CONJUGATE GRADIENTS WITH CIRCULANT PRECONDITIONER*

RAYMOND H. CHAN† AND GILBERT STRANG‡

Abstract. This paper studies the solution of symmetric positive definite Toeplitz systems $Ax = b$ by the preconditioned conjugate gradient method. The preconditioner is a circulant matrix C that copies the middle diagonals of A, and each iteration uses the Fast Fourier Transform. Convergence is governed by the eigenvalues of $C^{-1}A$—a Toeplitz-circulant eigenvalue problem—and it is fast if those eigenvalues are clustered. The limiting behavior of the eigenvalues is found as the dimension increases, and it is proved that they cluster around $\lambda = 1$. For a wide class of problems the error after q conjugate gradient steps decreases as r^{q^2}.

Key words. Toeplitz, circulant, conjugate gradient, Hankel

AMS(MOS) subject classification. 65F

1. Introduction. In this paper we discuss a class of linear systems $Ax = b$. The matrix A has the ***Toeplitz Property***: Down each diagonal its entries are constant. The i, j entry is a_{i-j}, and we assume symmetry and positive definiteness. Such systems are fundamental in signal processing and time series, where the convolution form reflects invariance in time or in space (stationarity or homogeneity). Toeplitz matrices also arise directly from constant-coefficient partial differential equations, and from integral equations with a convolution kernel, when those equations are made discrete.

With periodicity, these problems can be solved quickly by Fourier transform. The convolution becomes a multiplication and deconvolution is straightforward. In the nonperiodic case, which is analogous to a problem on a finite interval (or on a bounded region in the multidimensional case), this direct solution is lost. The inverse of a Toeplitz matrix is not Toeplitz, because of the presence of a boundary and the absence of periodicity. Nevertheless the matrix A is determined by only n coefficients $a_0, \cdots, a_{n-1}$, rather than by n^2. Algorithms that exploit the Toeplitz property are much faster than the $n^3/6$ operations of symmetric elimination, and *direct methods based on the Levinson recursion formula* [1] are in constant use. A number of ***superfast methods*** have been created in the last ten years, and an implementation by Ammar and Gragg [2] is giving excellent results. (See [2]-[5] for references, and note also the algorithms developed for systolic arrays [6].) More recently, the second author proposed an ***iterative method*** [7] that, it is hoped, will be fast and flexible. We report at the end on recent experiments, after an analysis of this iterative method.

Note. The Levinson algorithm conventionally uses $2n^2$ multiplications, but the "split-Levinson" form [8] reduces the count to nearly n^2. The superfast methods are $O(n \log^2 n)$, and the constant in the new implementation based on Schur's algorithm is about 8. It has become competitive with Levinson at reasonable n—a remarkable achievement. Our iterative algorithm needs $O(n \log n)$ operations per step from the ***Fast Fourier Transform*** (FFT), and the number of steps depends (inevitably) on the

* Received by the editors May 29, 1987; accepted for publication (in revised form) March 16, 1988.

† Department of Mathematics, University of Hong Kong, Hong Kong. The research of this author was supported by National Science Foundation grants DCR84-05506 and DCR86-02563.

‡ Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The research of this author was supported by Army Research Office grants DAAG29-83-K0025 and DAAL03-86-K0171, and National Science Foundation grant DMS87-03313.

Toeplitz matrix and the accuracy required. The goal of this paper is to analyze the rate of convergence in terms of the function $f = \sum a_k z^k$ constructed from the matrix, and to show that *for analytic f* (and many other functions) *the number of iterations is independent of n.* Tests on serious applications (to time series, scattering, or exploration data) are still in the future.

The iterative method uses a preconditioner. The Toeplitz matrix is replaced by a ***circulant matrix.*** It retains the Toeplitz property and adds periodicity. Each diagonal in the lower triangular part wraps around into a diagonal in the upper triangular part, and the entries satisfy $c_{ij} = c_{i-j} = c_{i-j+n}$. The distinction between Toeplitz and circulant matrices is seen (in the symmetric case) in

$$A = \begin{bmatrix} a_0 & a_1 & \cdot & a_{n-2} & a_{n-1} \\ a_1 & a_0 & a_1 & \cdot & a_{n-2} \\ a_2 & a_1 & a_0 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & a_1 \\ a_{n-1} & \cdot & a_2 & a_1 & a_0 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} c_0 & c_1 & \cdot & c_2 & c_1 \\ c_1 & c_0 & c_1 & \cdot & c_2 \\ c_2 & c_1 & c_0 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & c_1 \\ c_1 & \cdot & c_2 & c_1 & c_0 \end{bmatrix}.$$

The diagonals containing c_1 reappear in the corners, where the matrix A has a new (and probably smaller) entry a_{n-1}. To go from A to C will require changing about $n^2/4$ entries, and the key question in analyzing convergence will be the eigenvalues of $C^{-1}A$.

Multiplication by a circulant C is identical to discrete convolution. The linear system $Cz = b$ is the convolution equation $c * z = b$, where c is the first column of C. After a discrete Fourier transform it becomes $\hat{c}\hat{z} = \hat{b}$. Therefore $\hat{z}$ is given by a component-by-component division, and z is recovered from the inverse Fourier transform. The components of $\hat{c}$ are proportional to the eigenvalues of C, and *this convolution rule is the diagonalization of the circulant matrix* [9]: $z = F^{-1}\Lambda^{-1}Fb$. This is one of the rare instances in which a linear system is solved by diagonalizing the coefficient matrix! Normally elimination is much faster, but the Fourier matrix F (its entries are the complex roots of unity $F_{jk} = w^{jk} = \exp 2\pi ijk/n$, and its columns are the eigenvectors of every circulant matrix) is very special.

The speed of the iterative method depends on the fact that multiplication by F and F^{-1}—the discrete Fourier transform and its inverse—can be done so quickly. Those multiplications are computed by the FFT, which dominates each step of the iteration. It requires only $n \log n$ multiplications, and the calculations can be done in parallel. It applies directly to C and our goal is to apply it also to $Ax = b$—reaching the required tolerance in a number of steps which in the best case is independent of n.

We mention that *multiplication* by a Toeplitz matrix A (but not inversion) is also quick by the FFT. The matrix is extended to a circulant A^* of order $2n$, the vector d is completed to d^* by n zeros, and Ad appears in the first n components of A^*d^*, which is another discrete convolution. The goal is to replace A by C in any linear system to be solved, and to use A itself only in matrix multiplications.

This is exactly what is achieved by the ordinary iterative method $Cx_{n+1} = (C - A)x_n + b$, and also by the ***preconditioned conjugate gradient method.*** There is a Toeplitz multiplication on the right side and a circulant inversion on the left. We will see that the ordinary iterations can diverge; they depend on the extreme eigenvalues of $C^{-1}A$, which are not in close control. However the conjugate gradient method can be very effective. Its convergence rate also depends on the eigenvalues λ_i of $C^{-1}A$, but not exclusively on λ_1 and λ_n. *Conjugate gradient convergence is fast when the eigenvalues are* ***clustered,*** and that is the property established in this paper. Thus we

want to show that the circulant matrix satisfies, for large n, the following two essential requirements for a good preconditioner:

(1) $Cz = d$ can be solved quickly and stably (Theorem 1 will estimate $\|C^{-1}\|$).

(2) C is close to A (the eigenvalues of $C^{-1}A$ are clustered near 1).

For completeness we list the steps of the preconditioned conjugate gradient method, which gives the exact solution at step n; however, it is treated as an iterative method and stops earlier. Each iteration contains the periodic linear system with coefficient matrix C, the multiplication by A, and the two inner products that appropriately orthogonalize the directions d_j. Starting from $x_0 = 0$ and $r_0 = b$,

Solve $Cz_{j-1} = r_{j-1}$,

$\beta_j = z_{j-1}^T r_{j-1} / z_{j-2}^T r_{j-2}$ (except $\beta_1 = 0$),

$d_j = z_{j-1} + \beta_j d_{j-1}$ (except $d_1 = z_0$),

$\alpha_j = z_{j-1}^T r_{j-1} / d_j^T A d_j$,

$x_j = x_{j-1} + \alpha_j d_j$,

$r_j = r_{j-1} - \alpha_j A d_j$.

2. The eigenvalues of $C^{-1}A$. We want to choose C close to A. The simplest construction is to ***copy the central diagonals of*** A and bring them around to complete the circulant. Starting from the first column $a_0, \cdots, a_{n-1}$ of A, with $n = 2m$, the first column of C is $a_0, \cdots, a_m, \cdots, a_1$. If A decays quickly away from the main diagonal, then C starts to do the same but increases again as we approach the corner. By substituting the vector $(1, 0, \cdots, 0, -1)$ into the Rayleigh quotient $x^T Ax / x^T Cx$, we see that the largest eigenvalue of $C^{-1}A$ is at least

$$\frac{a_0 - a_{n-1}}{a_0 - a_1} \leqq \lambda_{\max}(C^{-1}A). \tag{1}$$

This can easily exceed 2, in which case *the ordinary iteration* $Cx_{n+1} = (C - A)x_n + b$ *will fail.* The iterating matrix $I - C^{-1}A$ has $1 - \lambda_{\max}$ outside the unit circle. However, the conjugate gradient method can compensate for any single outlying eigenvalue in a single iteration. The question is whether many other eigenvalues are far from unity, when corners of order $m = n/2$ are different in C and A.

Our first results are experimental [7]. With diagonal entries $a_k = 1/(1+k)$ the eigenvalues for $n = 12$ are .707, .957, $\cdots$, 1.047, 1.880. The largest and smallest make ordinary iteration too slow, but the other eigenvalues are clustered around 1. As the order n is increased, they approach limiting values. It is not always clear numerically, say for λ_4, whether the limit is 1. In these experiments, and in others with different diagonals a_k, the x_j converge quickly to $x = A^{-1}b$.

The next results are theoretical [10]. The matrices with geometrically decreasing diagonals $a_k = t^k$ exhibit a remarkable pattern in the computations, and the eigenvalues (and eigenvectors) of $C^{-1}A$ can be verified analytically. The extremes are $1/(1+t)$ and $1/(1-t)$, and $\lambda = 1$ is a double eigenvalue. What is striking is that *there are only two other eigenvalues of* $C^{-1}A$. For a matrix of order 1024, each is repeated 510 times! Those eigenvalues are exponentially close to 1: $\lambda = 1/(1+t^{n/2})$ and $\lambda = 1/(1-t^{n/2})$. Convergence of conjugate gradients is extremely fast, and our goal is to see when this exponential clustering can be predicted.

These matrices A were studied by Kac, Murdock, and Szegö [11], who observed that they have a special property: A^{-1} is tridiagonal. The generating function of A is

$$f = \sum_{-\infty}^{\infty} a_k e^{ik\theta} = \sum_{-\infty}^{\infty} t^{|k|} e^{ik\theta} = \frac{1 - t^2}{(1 - t e^{i\theta})(1 - t e^{-i\theta})}. \tag{2}$$

It is real and positive, so that A is symmetric positive definite (for $|t|<1$). It is the reciprocal of a three-term polynomial, which underlies the fact that A^{-1} is banded and that only two limiting eigenvalues are different from 1.

The ideal approach is to learn about the spectrum of $C^{-1}A$ from this function $f(\theta)$. We recognize that in practice the matrices are finite and the very distant diagonals a_k will not be used. But the asymptotic properties appear to be decisive, and all the information about C and A is in f. The goal is to turn a problem in operator theory into a problem in function theory.

Remark. The eigenvalues of the Toeplitz matrix alone have been studied in detail [12]-[14] and their behavior is entirely different. Instead of clustering around 1, they are "equidistributed" with the values of f itself [15]. The same is true for the circulants alone [16]. The new problem is to study the product $C^{-1}A$, and this paper computes the limits of the eigenvalues as $n\to\infty$. We believe it will also be a key to the finite case—where we look first at examples.

3. Uniform invertibility of C. Suppose that the Toeplitz matrices A_n, of order n, are finite sections of a fixed singly infinite positive definite matrix A_∞. The i,j entries of A_n and A_∞ are a_{i-j}, and the associated function

$$f(\theta)=\sum_{-\infty}^{\infty} a_k e^{ik\theta}$$

is real and positive. We will assume that the sequence a_k is in l^1, so that f belongs to the *Wiener class*: $\sum |a_k|<\infty$. Then the function $1/f$ associated with A_∞^{-1} belongs to the same class, and a more precise analysis becomes possible.

The first step is to consider C. The construction that copies the middle diagonals of A does not guarantee the invertibility of C.

Example 1.

$$A=\begin{bmatrix} 2 & -1 & 0 & 0\\ -1 & 2 & -1 & 0\\ 0 & -1 & 2 & -1\\ 0 & 0 & -1 & 2\end{bmatrix} \quad\text{and}\quad C=\begin{bmatrix} 2 & -1 & 0 & -1\\ -1 & 2 & -1 & 0\\ 0 & -1 & 2 & -1\\ -1 & 0 & -1 & 2\end{bmatrix}.$$

For this "second-difference matrix" the construction produces a singular C. That occurs whenever A comes from discretizing an operator with no zero-order term. Dirichlet boundary conditions leave A invertible, while periodic boundary conditions make C singular. It is a case when *sines should replace exponentials.* The matrix S with entries $\sin jk\pi/(n+1)$ has the eigenvectors of this A as its columns. The Fast Sine Transform carries out multiplication by S and S^{-1} in $n\log n$ steps. Therefore the new preconditioner can be SDS^{-1}, where the diagonal matrix D has entries $d_j=\sum a_k e^{ijk\theta}$, $\theta=\pi/(n+1)$.

In this example A_∞ is only semidefinite, and $f(\theta)=2-2\cos\theta$ is only nonnegative. Iteration is not needed because the matrix is banded, but for a full matrix with $\sum a_k=0$ the idea may be useful—conjugate gradients preconditioned by a sine transform. In this paper we stay with the positive definite case $f>0$.

Example 2.

$$A=\begin{bmatrix} .7 & \frac12 & \frac14 & \frac18\\ \frac12 & .7 & \frac12 & \frac14\\ \frac14 & \frac12 & .7 & \frac12\\ \frac18 & \frac14 & \frac12 & .7\end{bmatrix} \quad\text{and}\quad C=\begin{bmatrix} .7 & \frac12 & \frac14 & \frac12\\ \frac12 & .7 & \frac12 & \frac14\\ \frac14 & \frac12 & .7 & \frac12\\ \frac12 & \frac14 & \frac12 & .7\end{bmatrix}.$$

The smallest eigenvalue of A is $3/40$, whereas C has the eigenvalue $-1/20$. The new feature is that this A extends to a positive definite A_∞—it is the Kac-Murdock-Szegö matrix with diagonals $a_k = (\frac{1}{2})^k$ and with a_0 changed to .7. Note that our algorithm will recognize the indefiniteness of C at the first step, when $Cz_0 = r_0$ is solved by the FFT. C is diagonalized so its eigenvalues are made explicit, and the algorithm can adapt by making a different choice of the circulant.

We now prove that when A_∞ is positive definite and n is sufficiently large, *the circulants C_n are uniformly positive definite.* Of course the finite sections A_n are also positive definite. The point is that an indefinite C_n—the possibility illustrated in Example 2—cannot continue as n increases.

THEOREM 1. *Suppose $f(\theta) = \sum_{-\infty}^{\infty} a_k e^{ik\theta}$ is real and positive and in the Wiener class ($\sum |a_k| < \infty$). Then the circulants C_n and C_n^{-1} are uniformly bounded and positive definite for large n.*

Proof. The first column of C_n contains by construction the numbers $a_0, \cdots, a_m, \cdots, a_1$. (For simplicity we take n even and $m = n/2$.) Because the matrix is a circulant, its jth eigenvalue is

$$\lambda_j = a_0 + a_1 w^j + \cdots + a_m w^{jm} + \cdots + a_1 w^{j(n-1)}. \tag{3}$$

Here $w = e^{2\pi i/n}$ is the primitive nth root of unity. The corresponding eigenvector of C (and of every circulant) is $x_j = (1, w^j, \cdots, w^{j(n-1)})$; a direct multiplication gives $Cx_j = \lambda_j x_j$. Simplifying (3) by $w^n = 1$ yields

$$\lambda_j = a_0 + a_1(w^j + w^{-j}) + \cdots + a_{m-1}(w^{j(m-1)} + w^{j(1-m)}) + a_m w^{jm}. \tag{3'}$$

Thus *the eigenvalue equals the partial sum from $k = 1-m$ to m* of the series $\sum a_k e^{ik\theta}$, evaluated at the point $\theta = 2\pi j/n$ (where $e^{i\theta} = w^j$). Since the infinite series is absolutely convergent and its sum satisfies $f(\theta) \geqq \delta > 0$, the partial sums are uniformly positive for large n and the proof is complete.

Example 3. Diagonally dominant matrices, with $a_0 > 2\sum_{k\neq 0} |a_k|$, are positive definite and so are the circulants C_n.

Example 4. The function $f = \cosh\theta$ is even and positive. Therefore the matrices with $a_k = (1+k^2)^{-1} \cos k\pi$ lead to uniformly bounded C_n and C_n^{-1}, although the original matrix A is not diagonally dominant.

4. The limits of the eigenvalues. We come now to the central problem, to study the eigenvalues of $C_n^{-1}A_n$ for large n. In the next sections we transform that problem in order to carry out the analysis, and a Hankel matrix appears. At the end, when the limit is found, we transform back. The result was anticipated in [10], and it may be useful to separate its statement from the details of its proof.

THEOREM 2. *As $n \to \infty$, the eigenvalues of $C_n^{-1}A_n$ approach the eigenvalues of the following doubly infinite problem:*

$$\begin{bmatrix} \ddots & \vdots & \\ & a_1 & \\ \cdots & a_0 & a_1 \\ & a_1 & a_0 \\ & \vdots & \ddots \end{bmatrix} x = \lambda \begin{bmatrix} \ddots & \vdots & & \\ & a_1 & a_0 & a_1 \\ \cdots & a_0 & a_1 & a_0 \\ & a_1 & a_0 & a_1 \\ & \vdots & a_1 & \ddots \end{bmatrix} x. \tag{4}$$

The matrix on the right is an infinite circulant. The matrix on the left contains two back-to-back copies of the singly infinite Toeplitz matrix A_∞. Somehow the source of all the difficulty—the two boundaries that prevented the finite matrices A_n from being directly invertible by Fourier analysis—has reappeared in a new form.

Later in this paper we give an expression for the limiting eigenvalues $\bar{\lambda}$, by connecting them to a Hankel matrix and thus to a problem in rational approximation. That problem is approximation on the unit circle of a function $\tilde{v}(\theta)$ derived from $f(\theta)$, and it achieves our goal. The limits of the eigenvalues are determined from f. (The function has $|\tilde{v}(\theta)| \equiv 1$ and it appears as a "phase function" in systems theory [17]-[18] and apparently also in methods for numerical conformal mapping.)

At the end we return to the preconditioned conjugate gradient method, to prove superlinear convergence.

5. Orthogonal similarity and Hankel matrices. The key problem is $Ax = \lambda Cx$. There is a preliminary transformation which cuts this problem in half (for $n = 2m$), since all eigenvectors are odd or even:

$$x_- = \begin{bmatrix} y \\ -Jy \end{bmatrix} \quad \text{and} \quad x_+ = \begin{bmatrix} z \\ Jz \end{bmatrix} \quad \text{with } J = \begin{bmatrix} & & 1 \\ & 1 & \\ \cdot & & \\ 1 & & \end{bmatrix}. \tag{5}$$

This property comes from the "centrosymmetry" of C and A, and it leads to the orthogonal transformation suggested by Cantoni and Butler [19]:

$$Q = \frac{1}{\sqrt{2}} \begin{bmatrix} I & I \\ -J & J \end{bmatrix}.$$

This combination of the identity I and the counteridentity J will produce two diagonal blocks in $Q^{-1}AQ$ and $Q^{-1}CQ$. Suppose we write

$$A = \begin{bmatrix} T & R \\ R^T & T \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} T & S \\ S^T & T \end{bmatrix}, \tag{6}$$

in which T, R, S are Toeplitz of order m. T is symmetric around its main diagonal a_0, and S is symmetric around a_m, while R has diagonals $a_1, \cdots, a_{n-1}$ and is displayed below. From $JTJ = T$ and $JRJ = R^T$ we reach

$$\begin{aligned} Q^{-1}AQ = Q^TAQ &= \begin{bmatrix} T - RJ & 0 \\ 0 & T + RJ \end{bmatrix}, \\ Q^{-1}CQ = Q^TCQ &= \begin{bmatrix} T - SJ & 0 \\ 0 & T + SJ \end{bmatrix}. \end{aligned} \tag{7}$$

Thus ***the eigenvalue problem*** $Ax = \lambda Cx$ ***splits into***

$$(T - RJ)y = \lambda_-(T - SJ)y \quad \text{and} \quad (T + RJ)z = \lambda_+(T + SJ)z. \tag{8}$$

(*Note.* Those equations also appear directly when (5) and (6) are substituted into $Ax = \lambda Cx$.) There are m eigenvalues λ_+ and m eigenvalues λ_- which together represent the $n = 2m$ eigenvalues λ. The eigenvectors x_+ and x_- are C-orthogonal as required ($x_+^TCx_- = 0$), and they are also orthogonal.

We emphasize the effect of the counteridentity J. The matrices RJ and SJ are no longer Toeplitz. Instead ***they are Hankel matrices.*** Like J itself, they are constant down each counterdiagonal. The i, j entry depends on the ***sum*** $i + j$ instead of the difference $i - j$:

$$R = \begin{bmatrix} a_m & & a_{n-1} \\ & \ddots & \\ a_1 & & a_m \end{bmatrix} \quad \text{and} \quad RJ = \begin{bmatrix} a_{n-1} & & a_m \\ & \cdot^{\cdot} & \\ a_m & & a_1 \end{bmatrix}.$$

Thus equation (8) becomes a Toeplitz-Hankel eigenvalue problem. The northeast and southwest quarters of the original Toeplitz matrices have swung into blocks on the diagonal, and in the process they have become Hankel matrices.

A Hankel matrix is determined from its first column entries $v_1, v_2, \cdots$ by $V_{ij} = v_{i+j-1}$. In the singly infinite case its operator norm comes from the associated function $v(\theta) = \sum_1^\infty v_j e^{ij\theta}$, by Nehari's theorem [20]-[21], but not quite in the same way that the norm of a Toeplitz matrix comes from $f(\theta)$: *The anti-analytic terms are at our disposal*:

$$\|A\| = \sup |f(\theta)| \quad \text{and} \quad \|V\| = \sup |\tilde{v}(\theta)| = \inf_{v_0, v_{-1}, \cdots} \sup_\theta \left| \sum_{-\infty}^{\infty} v_j e^{ij\theta} \right|. \tag{9}$$

We emphasize that the eigenvalues in the two cases are very different. The spectrum of A is the interval $[f_{\min}, f_{\max}]$, while V is a *compact operator* for v in l^1—and its eigenvalues are the errors in a rational approximation problem.

6. The limiting equation $\bar{H}\bar{z} = \bar{\nu}\bar{T}\bar{z}$. This section begins our first approach to the eigenvalues of $C_n^{-1}A_n$ as $n \to \infty$. We return to (8), where the problem was split in half, and look at either of the two problems of order m:

$$(T + RJ)z = \lambda_+(T + SJ)z. \tag{10}$$

Our intention is to find the limits of the eigenvalues λ_+ as $n \to \infty$; numerical experiments indicate that limits exist.

First a simplification. Recall that the Hankel matrices RJ and SJ are identical on and below the main counterdiagonal. Their difference is the Hankel matrix:

$$H = SJ - RJ = \begin{bmatrix} h_1 & h_2 & \cdot & 0 \\ h_2 & & & 0 \\ \cdot & & & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{with } h_j = a_j - a_{n-j} \quad \text{for } j \leqq m.$$

Subtracting $\lambda_+(T + RJ)z$ from both sides of (10) and dividing through by λ_+, we obtain the result

$$\frac{(1 - \lambda_+)}{\lambda_+}(T + RJ)z = Hz. \tag{11}$$

We study that form of the problem, writing ν for the eigenvalue: $Hz = \nu(T + RJ)z$ with

$$\nu = \frac{1 - \lambda_+}{\lambda_+} \quad \text{and} \quad \lambda_+ = \frac{1}{1 + \nu}.$$

The clustering of λ_+ around 1 corresponds to the clustering of ν around 0.

We go directly to a statement of the limiting problem, and then consider its justification. As the order n increases, T and H approach singly infinite Toeplitz and Hankel matrices:

$$\bar{T} = \begin{bmatrix} a_0 & a_1 & a_2 & \cdot \\ a_1 & a_0 & a_1 & \cdot \\ a_2 & a_1 & a_0 & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad \bar{H} = \begin{bmatrix} a_1 & a_2 & a_3 & \cdot \\ a_2 & a_3 & \cdot & \\ a_3 & \cdot & & \\ \cdot & & & \end{bmatrix}.$$

This is strong convergence of operators, if we think of all operators as acting on l^1 or l^2. The matrices T and H represent operators T_m and H_m, which act as the zero operator after the first m components:

$$T_m = \begin{bmatrix} T & 0 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad H_m = \begin{bmatrix} H & 0 \\ 0 & 0 \end{bmatrix}.$$

LEMMA 1. *The Hankel sequence H_m converges uniformly to $\bar{H}$: $\|\bar{H} - H_m\| \to 0$. The Toeplitz sequence T_m converges strongly to $\bar{T}$, and the sequence $(RJ)_m$ converges strongly to the zero operator*:

$$\|\bar{T}x - T_m x\| \to 0 \quad \textit{and} \quad \left\| \begin{bmatrix} RJ & 0 \\ 0 & 0 \end{bmatrix} x \right\| \to 0 \quad \textit{for each vector } x.$$

Proof. For the Hankel matrices $\bar{H} = H_m$ the norm is given by (9):

$$\|\bar{H} - H_m\| \leqq \sup \left| \sum_{1}^{m-1} a_{n-j} e^{ij\theta} + \sum_{m}^{\infty} a_j e^{ij\theta} \right| \tag{12}$$

$$\leqq 2 \sum_{m}^{\infty} |a_j| \to 0 \quad \text{as } m \to \infty.$$

For the others, the estimate for $(RJ)_m$ is typical. With N fixed, the N-by-N submatrix in the upper left corner has *operator* norm going to zero. (Its entries are a_{n-j}, displayed earlier.) The larger entry a_1 is in the (m, m) position and it moves out as $m \to \infty$. There is convergence to zero for each fixed x but not uniform convergence—pointwise convergence but not norm convergence. Here we give only the simplest consequence for the eigenvalues.

THEOREM 3. *Each eigenvalue $\bar{\nu}$ of the infinite Hankel-Toeplitz problem*

$$\bar{H}\bar{z} = \bar{\nu}\bar{T}\bar{z} \tag{13}$$

is a limit of eigenvalues of the finite problems $Hz = \nu(T + RJ)z$.

Proof. The eigenvector $\bar{z}$ will be our fixed vector x. Then the strong convergence noted above gives

$$\|H_m \bar{z} - \bar{\nu}(T + RJ)_m \bar{z}\| \to 0.$$

Looking only at the m-by-m submatrices, where the operators are nonzero, and at the vector z_m taken from the first m components of $\bar{z}$, this is

$$\|Hz_m - \bar{\nu}(T + RJ)z_m\| \to 0. \tag{14}$$

By writing B for $T + RJ$, this means that $\|(H - \bar{\nu}B)^{-1}\| \to \infty$ as $m \to \infty$. In case $H - \bar{\nu}B$ is singular it means that $\bar{\nu}$ is an exact eigenvalue of the finite problem.

Because this is the generalized eigenvalue problem, with a matrix B on the right side instead of the identity, we need one extra step. Notice that these matrices $B = T + RJ$ are *uniformly invertible.* They are diagonal blocks in the original $Q^T A Q$ of (7). The matrices A are uniformly invertible because they are finite sections of a positive definite singly infinite Toeplitz matrix (it is $\bar{T}$!). Therefore we can convert to the ordinary eigenvalue problem for $B^{-1/2} H B^{-1/2}$, maintaining symmetry with the symmetric positive definite square root of B:

$$H - \bar{\nu}B = B^{1/2}(B^{-1/2} H B^{-1/2} - \bar{\nu} I) B^{1/2}.$$

In the l^2 matrix norm this yields

$$\|(H - \bar{\nu}B)^{-1}\| \leqq \|B^{-1}\| \max \frac{1}{|\nu_i - \bar{\nu}|}. \tag{15}$$

Since $\|B^{-1}\|$ is bounded and the left side approaches infinity, we conclude that some eigenvalue ν_i of the finite problem converges to $\bar{\nu}$ as $m \to \infty$.

This completes an argument that could be made more precise. The fact that the eigenvectors belong to l^1 was established by Adamjan, Arov, and Krein [22] and pointed out to us by Nick Trefethen, who also led us into the eigenvalue theory of Hankel operators. In the Kac-Murdock-Szegö example $a_k = t^k$, where nearly half the eigenvalues are $\lambda = 1/(1+t^m)$, there is geometric convergence to $\bar{\lambda} = 1$. In terms of $\nu = (1-\lambda)/\lambda = t^m$, it is *geometric convergence to the limit* $\bar{\nu} = 0$. In that special example we will show that the only nonzero $\bar{\nu}$ is the number t, corresponding to the eigenvalue $\lambda = 1/(1+t)$ that stays away from 1.

We now check on the number of eigenvalues of the finite problem that are outside an interval around 1, after a remark on the other half of our original problem—the λ_- eigenvalues of $C^{-1}A$.

7. The twin problem $(T-RJ)y = \lambda_-(T-SJ)y$. The same simplification as in (11), but now subtracting $\lambda_-(T-RJ)y$ from both sides of the twin problem and dividing by λ_-, yields

$$\frac{(1-\lambda_-)}{\lambda_-}(T-RJ)y = -Hy. \tag{16}$$

In this case we set

$$\mu = \frac{1-\lambda_-}{\lambda_-} \quad \text{and, thus,} \quad \lambda_- = \frac{1}{1+\mu}.$$

The finite problem is $Hy = -\mu(T-RJ)y$. Exactly as before, the strong convergence of H to $\bar{H}$, T to $\bar{T}$, and RJ to 0 leads to the limiting problem

$$\bar{H}\bar{y} = -\bar{\mu}\bar{T}\bar{y}.$$

This is identical to $\bar{H}\bar{z} = \bar{\nu}\bar{T}\bar{z}$ except for the sign change: $\bar{\mu} = -\bar{\nu}$. As before, each $\bar{\mu}$ is the limit of eigenvalues μ of the finite problem. There is an interesting corollary for the original eigenvalues λ_- and λ_+: *In the limit there are pairs* λ_- *and* λ_+ *which satisfy*

$$\frac{1}{\lambda_-} + \frac{1}{\lambda_+} = 2. \tag{17}$$

The left side approaches $(1+\bar{\mu})+(1+\bar{\nu}) = 2$. This was first noticed by Alan Edelman in MATLAB experiments.

With this pair of limit problems, *we have completed the proof of Theorem* 2. The splitting into odd and even eigenvectors of that doubly infinite eigenvalue problem gives exactly (13) and its twin, with the same similarity Q and change from λ to ν and μ as in the finite case. A corresponding limit could be found for other constructions of the circulant C—and for multidimensional Toeplitz equations, in which our algorithm may be particularly useful. We concentrate here on understanding more clearly the asymptotic behavior for this choice of $C^{-1}A$.

8. The clustering of the spectrum of $C^{-1}A$. The limit problem gives us precise information about the asymptotic behavior of the algorithm. Even without that knowledge we can show that the eigenvalues of $C^{-1}A$ cluster at 1, by using the theory of *collectively compact operators*:

> A family S of bounded operators on l^2 is collectively compact if the set $\{Kx\colon K \in S, \|x\| \leqq 1\}$ is relatively compact in l^2.

This applies to $\{H_m\}$ and $\{\bar{H} - H_m\}$ when $\sum |h_j| < \infty$. For every ε we can choose $n(\varepsilon)$ so that outside the leading submatrix of that order, all these matrices have norms less than ε. Anselone [23] established the following consequences for approximation of the spectrum.

LEMMA 2 [23, Thm. 4.8]. *If an open set contains the spectrum of $\bar{H}$, it also contains the spectrum of H_m for all sufficiently large m.*

LEMMA 3 [23, Thm. 4.14]. *The number of eigenvalues of H_m in a small ball around a nonzero eigenvalue μ of $\bar{H}$ is, for large m, no greater than the multiplicity of μ.*

Since $\bar{H}$ is compact, those multiplicities are finite. (Our $\bar{H}$ is the norm limit of the finite-dimensional H_m, by (12). From Hartman's theorem [21] it remains compact if its associated function is only continuous, and not necessarily in the Wiener class.) It follows easily that the eigenvalues of H_m are clustered around zero.

THEOREM 4. *There exist $M(\varepsilon)$ and $N(\varepsilon)$ such that for $m > M$, at most N eigenvalues of H_m and of $(T \pm RJ)^{-1}H$ have absolute value exceeding ε.*

Only a finite number of eigenvalues of $\bar{H}$ have $|\mu| > \varepsilon$. Therefore Lemma 2 allows us to locate the eigenvalues of H_m for large m, and Lemma 3 allows us to count them. The total cannot exceed the total for $\bar{H}$.

It is this count that was not included in Theorem 3 on the limiting values. As there, we have to handle the matrices $B = T \pm RJ$ on the right side of the eigenvalue problem. They are diagonal blocks in $Q^T A Q$, and their eigenvalues are between $f_{\min}$ and $f_{\max}$ (where $f = \sum a_k e^{ik\theta}$). Therefore the eigenvalues of $(T \pm RJ)^{-1}H$ are also counted by Theorem 4.

Thus the eigenvalues λ of $C^{-1}A$ cluster around 1. Only a fixed number, independent of n, can be outside $(1-\varepsilon, 1+\varepsilon)$. Now we go back to the equation $\bar{H}\bar{z} = \bar{\nu}\bar{T}\bar{z}$, which identifies the asymptotic limits of the eigenvalues.

9. The eigenvalues of $\bar{T}^{-1}\bar{H}$. It is this singly infinite problem that is attractive to work with, because ***all the information is in the function*** $f(\theta) = \sum a_k e^{ik\theta}$. The difficulty is to extract it. One preliminary difficulty is that we still have a generalized eigenvalue problem, with $\bar{T}$ on the right-hand side: $\bar{H}\bar{z} = \bar{\nu}\bar{T}\bar{z}$. The inverse of $\bar{T}$ is not Toeplitz, and $\bar{T}^{-1}\bar{H}$ is not Hankel, but there is a way to preserve those properties, by factoring $\bar{T}$ and putting part of $\bar{T}^{-1}$ on each side of $\bar{H}$:

$$\bar{T} = WW^T \text{ with } W = \text{upper triangular Toeplitz operator.}$$

This is equivalent to representing the positive function f as a square:

$$f = \sum_{-\infty}^{\infty} a_k e^{ik\theta} = \left| \sum_{-\infty}^{0} w_k e^{ik\theta} \right|^2 = |w|^2.$$

From the Wiener theory the sequence $\cdots, w_{-2}, w_{-1}, w_0$ is in l^1. The function w, with no zeros on or outside the unit circle, corresponds to W in the same way that f corresponds to $\bar{T}$:

$$w = \sum_{-\infty}^{0} w_k e^{ik\theta} \leftrightarrow W = \begin{bmatrix} w_0 & w_{-1} & w_{-2} & \cdot \\ & w_0 & w_{-1} & \cdot \\ & & w_0 & \cdot \\ & & & \cdot \end{bmatrix}.$$

Note that w is *anti-analytic*, with negative k. The same properties hold for the matrix $U = W^{-1}$, associated with the reciprocal function $u = w^{-1}$:

$$u(\theta) = \sum_{-\infty}^{0} u_k e^{ik\theta} = \left(\sum_{-\infty}^{0} w_k e^{ik\theta} \right)^{-1}.$$

These functions will take us from the Toeplitz-Hankel product $\bar{T}^{-1}\bar{H}$ (the eigenvalue problem with $\bar{T}$ on the right side) to a single Hankel matrix V. It has the same eigenvalues $\bar{\nu}$, and to study them we need to know its associated function.

THEOREM 5. *The matrix* $\bar{T}^{-1}\bar{H} = W^{-T}W^{-1}\bar{H} = U^TUH$ *is similar to the Hankel matrix* $V = UHU^T$. *The associated function is*

$$v(z) = \sum_1^\infty v_k z^k = \text{analytic part of } \frac{\sum_1^\infty a_k z^k}{(\sum_{-\infty}^0 w_k z^k)^2} \tag{18}$$
$$= \text{analytic part of } \bar{w}/w.$$

Proof. Certainly U^TUH is similar to $V = UHU^T$. To verify that this is a Hankel matrix, and to identify its function $v(z)$, we carry out an example with two nonzero coefficients h_1, h_2 and u_0, u_1. (The general rule is that upper triangular Toeplitz times Hankel is Hankel, and Hankel times lower triangular Toeplitz is Hankel; we hope the example will be convincing.) The infinite matrices can be cut off after three rows and columns:

$$\begin{bmatrix} u_0 & u_1 & 0 \\ 0 & u_0 & u_1 \\ 0 & 0 & u_0 \end{bmatrix}\begin{bmatrix} h_1 & h_2 & 0 \\ h_2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} u_0 & 0 & 0 \\ u_1 & u_0 & 0 \\ 0 & u_1 & u_0 \end{bmatrix} = \begin{bmatrix} h_1u_0^2+2h_2u_1u_0 & h_2u_0^2 & 0 \\ h_2u_0^2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The corresponding multiplication of functions correctly gives the analytic part of $(h_1z+h_2z^2)(u_0+u_1/z)^2 = h_1u_0^2z+2h_2u_0u_1z+h_2u_0^2z^2 = v(z)$. This verifies the first line of (18), since $u = 1/w$, and we only mention a surprise in comparison with the Toeplitz case. If the matrix H in the middle were Toeplitz, with associated function h, then the product UHU^T would again be Toeplitz, but its associated function would be $h|u|^2$— where in the Hankel case it is the analytic part of hu^2. Note that the analytic part is taken to start at the linear term in z— because the Hankel matrix starts at v_1.

Now we look at the last step in (18), the elegant form for v. It was noticed by Alan Edelman, and the second author observed that it follows immediately from the line above: We can add the anti-analytic terms $\sum_{-\infty}^0 a_k z^k$ to the numerator to obtain f, *without changing the analytic part.* Therefore

$$v \text{ is the analytic part of } \frac{f}{w^2} = \frac{w\bar{w}}{w^2} = \frac{\bar{w}}{w}.$$

Now the asymptotic problem is the spectrum of V. We recall that it is a compact operator, so its eigenvalues $\bar{\nu}$ cluster at zero. It is also a Hankel operator, and the eigenvalues are connected to a part of mathematics that looks entirely separate: ***approximation by rational functions on the unit circle.***

10. Hankel eigenvalues and rational approximation. We recall the main facts from [22] and [24]. The singular values of a Hankel matrix V—which in our symmetric case means the absolute values $|\bar{\nu}_0| \geqq |\bar{\nu}_1| \geqq \cdots$—are the errors in the best approximation of the function $v(z)$ from the class $\tilde{R}_n$. The error is measured in the sup norm on the unit circle:

$$|\bar{\nu}_n| = \inf_{r \in \tilde{R}_n} \|v(z) - \tilde{r}(z)\|_\infty. \tag{19}$$

It is important that $\tilde{R}_n$ is larger than the class R_n of rational functions (ratios of polynomials of degree n). *The anti-analytic part of* $\tilde{r}$ *is arbitrary*: $\tilde{R}_n = R_n +$ *anti-analytic.*

Thus the approximation problem for v is the same as for $\bar{w}/w$, because the analytic parts are the same. The approximants have the form

$$\tilde{r} = \sum_{-\infty}^{n} d_k z^k \Big/ \sum_{0}^{n} e_k z^k,$$

where the numerator is bounded in any bounded subset of $\{|z| \geqq 1\}$. The optimal error curve $v - \tilde{r}^*$ is a circle of radius $|\bar{\nu}_n|$ around the origin. Except in degenerate cases its winding number is $2n+1$ [25].

Thus the estimation of the asymptotic eigenvalues $\bar{\nu}$ is equivalent to a problem in rational approximation. That may not be a simplification; most applications go the other way. Of course there is a special (but important) case when f itself is rational; this corresponds to banded Toeplitz matrices times banded inverses, and *the Hankel matrix V has only finitely many nonzero eigenvalues.* That was the case for the Kac-Murdock-Szegö example in which

$$f = \frac{1-t^2}{|1-tz|^2}, \quad u = \frac{1-t/z}{(1-t^2)^{1/2}}, \quad v = \frac{(1-t^2)tz}{1-tz}. \tag{20}$$

Here v maps the unit circle to a circle with center at t^2 and radius t. Therefore the best constant approximation to v is $r = t^2$, and the error is $|\bar{\nu}| = t$. The corresponding λ's are $1/(1 \pm t)$. That is the correct limit, from our explicit calculation, of the original eigenvalue problem $Ax = \lambda Cx$, in which all other eigenvalues approached 1.

A second example will bring out the important point, which is the very rapid decrease of the errors $|\bar{\nu}|$ in rational approximation. Suppose the matrix is not tridiagonal but pentadiagonal. The function f will be

$$f = |(1 - t\,e^{i\theta})(1 - s\,e^{i\theta})|^2.$$

As s approaches zero this goes back to the previous example (or more precisely to its inverse—it is a corollary of (18) that f and f^{-1} lead to the same results, and now it is A instead of A^{-1} that is banded). There are *two* nonzero errors $|\bar{\nu}_0|$ and $|\bar{\nu}_1|$, after which the approximation is exact and all eigenvalues approach $\bar{\nu} = 0$ (which is $\lambda = 1$). Those two errors are given by the quadratic equation

$$\nu^2 - \nu(t+s)(ts-1) - t^2 s^2 = 0. \tag{21}$$

For $s = t = \frac{1}{2}$ the limiting eigenvalues are $|\bar{\nu}_0| = .825$ and $|\bar{\nu}_1| = .076$. Thus it is not the case that the two limits $|\bar{\nu}|$ are near t and s. For $t = s$ the leading terms in $|\bar{\nu}|$ are $2t$ and $t^3/2$, and it is this *cube* of t that indicates rapid decrease. A similar phenomenon was noticed by Trefethen [24, Thm. 6.3].

We turn to the consequences for the iterative algorithm when there is rapid decrease of the $|\bar{\nu}|$.

11. The rate of convergence to $x = A^{-1}b$. The conjugate gradient method is a recursive calculation of a sequence of projections. After q cycles, x_q is as close as possible (in an appropriate norm) to the solution $x = A^{-1}b$, among all vectors in the Krylov subspace spanned by $C^{-1}b$, $C^{-1}AC^{-1}b$, $C^{-1}AC^{-1}AC^{-1}b, \cdots$. This makes possible an estimate of the error $e_q = x - x_q$:

$$\|e_q\| \leqq [\min_{P_q} \max_{\lambda} |P_q(\lambda)|]\|e_0\|. \tag{22}$$

The maximum is taken over the eigenvalues of $C^{-1}A$. The minimum is over polynomials of degree q with constant term 1. The problem is to estimate that minimum.

One choice of P_q, if the eigenvalues are known to lie in the interval $[\alpha, \beta]$, is the Chebyshev choice: the best polynomial when the maximization is taken over all $\alpha \leqq \lambda \leqq \beta$. That has the drawback of using only α and β; it cannot take advantage of clustering of the eigenvalues. By scale invariance, the estimate depends only on the condition number β/α.

At the other extreme, we can choose P_q *to annihilate the q extreme eigenvalues.* In our problem those eigenvalues come in pairs λ_+ and λ_-, on opposite sides of 1, and such a pair is annihilated by the factor

$$p(x) = \left(1 - \frac{x}{\lambda_+}\right)\left(1 - \frac{x}{\lambda_-}\right). \tag{23}$$

Between those roots the maximum of $|p|$ is attained at the average $x = \frac{1}{2}(\lambda_+ + \lambda_-)$, where $|p| = (\lambda_+ - \lambda_-)^2/4\lambda_+\lambda_-$. It is easy to find the asymptotic convergence rate of the conjugate gradient method in the important case when the rational approximation errors decrease geometrically to zero:

$$|\bar{\nu}_j| = O(r^j) \quad \text{with } r < 1. \tag{24}$$

THEOREM 6. *Suppose* (24) *holds, which depends on the original f.* (*It is certainly true if f is analytic in a neighborhood of* $|z| = 1$.) *Then the errors in the circulant-preconditioned conjugate gradient method decrease, asymptotically as* $n \to \infty$, *at the superlinear rate*

$$\|e_q\| \leqq c^q r^{q^2/4 + q/2} \|e_0\|. \tag{25}$$

The decisive factor is $r^{q^2/4}$. To find it we note that $\lambda_\pm = 1/(1 \mp \bar{\nu})$:

$$|p_j| \leqq \frac{(\lambda_{+j} - \lambda_{-j})^2}{4\lambda_{+j}\lambda_{-j}} = \frac{\bar{\nu}_j^2}{1 - \bar{\nu}_j^2} \leqq c^2 r^{2j} \quad \text{by (24).}$$

The polynomial $P_{2q} = p_1 p_2 \cdots p_q$ annihilates the q extreme pairs of eigenvalues and satisfies

$$|P_{2q}(\lambda)| \leqq c^{2q} r^2 r^4 \cdots r^{2q} = c^{2q} r^{q^2+q}.$$

This holds for all λ in the inner interval between λ_{+q} and λ_{-q}, where the remaining eigenvalues are. Therefore (25) comes directly from (22), after a change from $2q$ to q.

12. Superlinear convergence. In a sense the convergence rate $r^{q^2/4}$ was the object of this paper. By connecting the eigenvalues of $C^{-1}A$ to the spectrum of the Hankel matrix V, and by assuming (24), we were led to that unusual rate. Trefethen observed that (24) will hold *if the original* $f(z) = \sum a_j z^j$ *is analytic in a neighborhood of* $|z| = 1$, and, further, that this condition is *far from necessary.* We expect that rate for a wide class of applications, but we can also prove superlinear convergence in its absence.

For this we modify the polynomial P_q as in [26]-[27]. It will annihilate N pairs of extreme eigenvalues, by including N of the quadratic factors (23). The remaining factor of degree $q - 2N$ will be the Chebyshev choice on the interval between λ_{-N} and λ_{+N}, which contains the remaining eigenvalues. We know from Theorem 4 (which applied to all f in the Wiener class) that for any fixed ε, all but $N(\varepsilon)$ eigenvalues are within ε of 1. The N extreme eigenvalues are in the fixed interval

$$\frac{f_{\min}}{f_{\max}} - \varepsilon < \lambda(C^{-1}A) < \frac{f_{\max}}{f_{\min}} + \varepsilon$$

for large n, applying Theorem 1 to C^{-1} and the elementary bounds $f_{\min} \leqq \lambda(A) \leqq f_{\max}$ to A. The quadratics p that annihilate those extreme pairs are bounded by a fixed constant K on $(1-\varepsilon, 1+\varepsilon)$. The other (Chebyshev) factor of degree $q-2N$ is of order ε^{q-2N} on that interval. Therefore the polynomial P_q satisfies a crude bound

$$|P_q(\lambda)| \leqq c\varepsilon^{q-2N} K^N \leqq C(\varepsilon)\varepsilon^q \tag{26}$$

for all eigenvalues λ of $C^{-1}A$, when the order n is sufficiently large.

It follows from (22) that $\|e_q\| \leqq C(\varepsilon)\varepsilon^q \|e_0\|$. The number of iterations to achieve a fixed accuracy remains bounded as the matrix order n is increased. Each iteration requires $O(n \log n)$ operations using the FFT. Therefore the work to obtain the solution $x = A^{-1}b$ to given accuracy δ is $c(f, \delta)n \log n$. The real question is the efficiency in practice, and we will be glad to see this straightforward algorithm tried on genuine applications.

13. Tentative experiments. One family of Toeplitz matrices is particularly convenient for testing, and we report here on the results. The entries down the kth diagonal (with $k = 1$ for the main diagonal) are $a_k = k^{-p}$. For $p = 2$ the entries $1, \frac{1}{4}, \frac{1}{9}, \cdots$ decrease quickly away from the center. At $p = 1$ the sum $1 + \frac{1}{2} + \frac{1}{3} + \cdots$ is divergent, and we leave the Wiener class. At $p = \frac{1}{2}$ the eigenvalues are less clustered (and we had not known that A and C were positive definite). But even at $p = .01$ the conjugate gradient convergence is remarkable. Nine or ten steps reduce the residual by 10^{-8}, independent of the order n. That last statement is entirely experimental (this is MATLAB mathematics), because the theory for the Wiener class has been left behind.

Table 1 shows the four largest eigenvalues of $C^{-1}A$ when $n = 40$. We see how the largest grows as p decreases (and the smallest is related to it by $\lambda_{\min}^{-1} + \lambda_{\max}^{-1} \approx 2$). However, there is still strong clustering around $\lambda = 1$.

Table 2 shows the norms of the residuals $r_q = b - Ax_q$ in the conjugate gradient method. It is preconditioned by the circulant C, and the right-hand side b has randomly chosen entries from a uniform distribution over $(0, 1)$. N is the number of iterations to reach a residual norm below 10^{-8}. In these four cases, the smallest eigenvalues of C were, respectively, .645, .385, .207, and .004. Alan Edelman convinced us that the asymptotic behavior of this eigenvalue (small p and large n) is $(\log \pi/2)p$. The positive definiteness is significant; we give an indefinite example next.

TABLE 1

p	λ_1	λ_2	λ_3	λ_4
2	1.360	1.029	1.003	1.002
1	2.072	1.079	1.018	1.013
$\frac{1}{2}$	3.100	1.111	1.049	1.035
$\frac{1}{100}$	5.596	1.190	1.136	1.102

TABLE 2

p	$\|r_1\|$	$\|r_2\|$	$\|r_3\|$	$\|r_4\|$	N
2	3.441	.094	.007	.000	6
1	4.131	.338	.031	.014	7
$\frac{1}{2}$	4.136	.634	.290	.028	8
$\frac{1}{100}$	3.898	2.560	.219	.027	10

Suppose that the Toeplitz entries are $a_k = 1/k!$. The underlying function is

$$f(z) = e^z + e^{1/z} - 1$$

and there will be extremely good approximation by rational functions, which implies quick convergence of the Hankel eigenvalues $\nu \to 0$, and strong clustering of the Toeplitz-circulant eigenvalues $\lambda \to 1$. Analyticity is clear, but positive definiteness is not. The function f is negative at $z = -1$, and $\lambda_{\min}(C) = -.264$ with $n = 40$. Nevertheless, the eigenvalues of $C^{-1}A$ cluster near 1 (amazingly so, since some are complex). What may be of interest is the sequence of norms of residuals, which show fast convergence after a kink to digest the indefiniteness of the problem:

$$\|r\| = 3.59 \quad .51 \quad 1.38 \quad 2.29 \quad .22 \quad .01 \quad .00.$$

By adding 1 to the main diagonal and changing f to $e^z + e^{1/z}$, the matrices A and C become positive definite (though still not diagonally dominant). $C^{-1}A$ has only three eigenvalues that are further than 10^{-5} above $\lambda = 1$, and three more at that distance below:

$$\lambda(C^{-1}A) = 2.02 \quad 1.06 \quad 1.0009 \quad 1.000007 \quad 1.00000003.$$

The result is convergence in six steps of the preconditioned conjugate gradient method.

In all these cases the operation count is $O(n \log n)$.

REFERENCES

[1] N. LEVINSON, *The Wiener* RMS *(Root-Mean Square) error criterion in filter design and prediction*, J. Math. Phys., 25 (1947), pp. 261–278.

[2] G. AMMAR AND W. GRAGG, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 61–76.

[3] F. DE HOOG, *A new algorithm for solving Toeplitz systems of equations*, Linear Algebra Appl., 88/89 (1987), pp. 122–138.

[4] J. BUNCH, *Stability of methods for solving Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 349–364.

[5] T. KAILATH, *A theorem of I. Schur and its impact on modern signal processing*, in I. Schur: Methods in Operator Theory and Signal Processing, I. Gohberg, ed., Birkhäuser, Basel, 1986.

[6] R. BRENT AND F. LUK, *A systolic array for the linear-time solution of Toeplitz systems of equations*, J. VLSI Comput. Syst., 1 (1983), pp. 1–22.

[7] G. STRANG, *A proposal for Toeplitz matrix calculations*, Stud. Appl. Math., 74 (1986), pp. 171–176.

[8] P. DELSARTE AND Y. GENIN, *The split Levinson algorithm*, IEEE Trans. Acoust. Speech Signal Process., 34 (1986), pp. 470–478.

[9] G. STRANG, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, MA, 1986.

[10] G. STRANG AND A. EDELMAN, *The Toeplitz-circulant eigenvalue problem* $Ax = \lambda Cx$, Oakland Conference on PDE's, L. Bragg and J. Dettman, eds., Longmans, London, 1987.

[11] M. KAC, W. MURDOCK, AND G. SZEGÖ, *On the eigenvalues of certain Hermitian forms*, Arch. Rational Mech. Anal., 2 (1953), pp. 767–800.

[12] P. DELSARTE AND Y. GENIN, *Spectral properties of finite Toeplitz matrices*, Philips Research Lab. Manuscript M50, Proc. Internat. Symposium Math. Theory of Networks and Systems, Beer-Sheva, Israel, 1983.

[13] G. CYBENKO, *On the eigenstructure of Toeplitz matrices*, IEEE Trans. Acoust. Speech Signal Process., 32 (1984), pp. 918–920.

[14] W. F. TRENCH, *On the eigenvalue problem for Toeplitz matrices generated by rational functions*, Linear and Multilinear Algebra, 17 (1985), pp. 337–353.

[15] U. GRENANDER AND G. SZEGÖ, *Toeplitz Forms and Their Applications*, University of California Press, Berkeley, CA, 1958.

[16] R. M. GRAY, *On the asymptotic eigenvalue distribution of Toeplitz matrices*, IEEE Trans. Inform. Theory, 18 (1972), pp. 725–729.

[17] M. GREEN AND B. D. O. ANDERSON, *The approximation of power spectra by phase matching*, Proc. IEEE Conference on Decision and Control, Athens, GA, 1986.

[18] K. GLOVER, R. CURTAIN, AND J. PARTINGTON, *Realisation and approximation of linear infinite dimensional systems with error bounds*, # 258, Engineering Department, Cambridge University, Cambridge, UK.

[19] A. CANTONI AND P. BUTLER, *Eigenvalues and eigenvectors of symmetric centrosymmetric matrices*, Linear Algebra Appl., 13 (1976), pp. 275–288.

[20] Z. NEHARI, *On bounded bilinear forms*, Ann. Math., 65 (1957), pp. 153–162.

[21] S. C. POWER, *Hankel Operators on Hilbert Space*, Pitman, Boston, 1982.

[22a] V. ADAMJAN, D. AROV, AND M. KREIN, *Analytic properties of Schmidt pairs for a Hankel operator and the generalized Schur-Takagi problem*, Math. USSR-Sb., 15 (1971), pp. 31–73.

[22b] ———, *Infinite Hankel matrices and generalized Carathéodory-Féjér and Riesz problems*, Functional Anal. Appl., 2 (1968), pp. 1–18.

[23] P. M. ANSELONE, *Collectively Compact Operator Approximation Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[24] L. N. TREFETHEN, *Rational Chebyshev approximation on the unit disk*, Numer. Math., 37 (1981), pp. 297–320.

[25] E. HAYASHI, L. N. TREFETHEN, AND M. GUTKNECHT, *The* CF *table*, Constructive Approximation, submitted.

[26] H. VAN DER VORST AND A. VAN DER SLUIS, *The rate of convergence of conjugate gradients*, Preprint 354, University of Utrecht, Utrecht, the Netherlands, 1984.

[27] O. AXELSSON AND G. LINDSKOG, *On the rate of convergence of the preconditioned conjugate gradient method*, Numer. Math., 48 (1986), pp. 499–523.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 120-132, January 1989
© 1989 Society for Industrial and Applied Mathematics
010

COMPUTATION OF HYPERBOLIC EQUATIONS ON COMPLICATED DOMAINS WITH PATCHED AND OVERSET CHEBYSHEV GRIDS*

DAVID A. KOPRIVA†

Abstract. A patched/overset grid strategy that allows hyperbolic partial differential equations to be solved in complicated geometries with Chebyshev spectral collocation is presented. Unlike similar approaches used with finite difference methods, the spectral method defines how interpolations should be carried out. The result is an approach that retains spectral accuracy. Numerical experiments are performed on both one- and two-dimensional examples.

Key words. spectral methods, hyperbolic equations

AMS(MOS) subject classifications. 65M99, 65M50, 65N35.

1. Introduction. A basic difficulty with the application of Chebyshev spectral methods to the solution of boundary value problems for hyperbolic equations comes from their inability to handle complex geometries. Spectral methods are global methods that require square domains. Hyperbolic equations, however, must often be solved in the presence of multiple boundaries that do not correspond to coordinate curves in any single coordinate system. While it is often possible to map an entire physical domain onto a square, such mappings can be difficult to find if the geometry is very complicated.

A convenient solution to the geometry problem is to use domain decomposition. Instead of trying to map an entire geometry onto a single square, the mapping is done in piecewise fashion. A computational domain is divided into a number of subdomains, each of which can be conveniently mapped onto a square. The subdomains, which collectively cover the entire computational domain, may overlap if this leads to simpler mappings. Interface conditions are applied to couple the solutions on the multiple subdomains.

Domain decomposition techniques are widespread in the solution of hyperbolic equations by finite difference methods and are often referred to as "zonal," "multiple-grid," or "composite-grid" methods. Representative examples of papers describing such techniques include those of Kreiss [15], Berger and Oliger [2], Benek, Steger, and Dougherty [1], and Rai [16]. Although the terminology is not completely standard, if subdomains share a common boundary they are said to be "patched" together. If they overlap, the grids are said to be "overlaid" or "overset."

Spectral multidomain methods for hyperbolic equations have been presented by Kopriva [11]. The two-dimensional calculations of Kopriva [11], [12] featured patched grids which required continuity of grid lines across subdomain boundaries. In this paper we extend the multidomain strategy and allow subdomains to overlap in order to compute Chebyshev spectral approximations of hyperbolic systems in complex geometries. Patched grid interfaces are made more general by the use of spectral interpolants at subdomain boundaries. In this way, different numbers of collocation points can be used in different subdomains so that local grid refinement in more than one space dimension becomes more efficient.

* Received by the editors August 3, 1987; accepted for publication (in revised form) April 13, 1988. This research was supported in part by the Florida State University through time granted on its Cyber 205 supercomputer.

† Mathematics Department and Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida 32306.

2. Chebyshev collocation for hyperbolic equations. We are interested in the solution of hyperbolic systems of the form

$$Q_t + A(Q)Q_x + B(Q)Q_y = F \qquad x, y \in \Omega \subset \mathbb{R}^2, \quad t > 0 \tag{1}$$

where $Q \in \mathbb{R}^m$ and A and B are $m \times m$ matrices. Along with (1), suitable initial and boundary conditions must be specified.

Spectral collocation methods are powerful techniques for the solution of hyperbolic partial differential equations such as (1). Their main attraction is the high accuracy, usually known as "spectral accuracy." In particular, errors decay at a rate that is determined by the smoothness of the solution. If the solution and coefficients are C^∞ on the computational domain, then the error decays faster than any polynomial of the reciprocal of the number of grid points. In contrast, the errors of finite difference methods decay with constant order. Furthermore, spectral methods are characterized by low dissipation and dispersion errors when compared to typical finite difference methods. For wave-propagation problems, this means that waves are not seriously damped for long-time integrations and that wave components travel at their correct speeds. For steady problems in inviscid aerodynamics, this would mean that errors due to implicit artificial viscosity could be less important.

In the usual Chebyshev collocation (often called "pseudospectral") approach, the region Ω is mapped by some global mapping onto a rectangle $R = [-1, 1] \times [-1, 1]$. The solution Q is approximated by a grid function Q_{ij} at a finite number of collocation (grid) points, (x_i, y_j). The collocation points are specified as the nodes of the Gauss-Lobatto-Chebyshev quadrature rule. For example,

$$x_i = \cos(i\pi/N), \qquad i = 0, 1, \cdots, N. \tag{2}$$

A global Chebyshev interpolant, $I_{N,M}Q(x, y)$, is then passed through the grid values:

$$I_{N,M}Q(x, y) = \sum_{n,m=0}^{N,M} a_{nm} T_n(x) T_m(y). \tag{3a}$$

The coefficients are defined by the inverse Chebyshev transform

$$a_{nm} = \frac{4}{NM\bar{c}_n\bar{c}_m} \sum_{n,m=0}^{N,M} \frac{Q_{jk} T_n(x_j) T_m(y_k)}{\bar{c}_j\bar{c}_k} \tag{3b}$$

where

$$\bar{c}_n = \begin{cases} 2, & n = 0 \text{ or } N, \\ 1, & 1 \leqq n \leqq N-1. \end{cases}$$

To approximate (1), we replace Q and its continuous space derivatives by approximate values at the collocation points. The grid-point values of the derivatives are obtained from the derivative of (3a) evaluated at (x_i, y_j). In practice, this is accomplished by a matrix-multiplication technique. For the small number of grid points used within each subdomain (less than 32 points in each direction), this is more efficient than using fast Fourier transforms on the Cyber 205 used in the calculations presented here [5].

The spatial discretization of (1) leaves an ordinary differential equation in time to be solved for each grid point. Typically, an explicit time integration method such as second- or fourth-order Runge-Kutta is used. See Canuto et al. [5] for a complete description of the implementation of the Chebyshev collocation method.

Chebyshev interpolation possesses "spectral accuracy" and becomes very accurate as the number of collocation points becomes large. Let $H_w^m(-1, 1)$ denote a weighted Sobolev space with Chebyshev weight $w=(1-x^2)^{-1}$ and norm $\|u\|_{H_w^m(-1,1)}=(\sum_{k=1}^m \|d^k u/dx^k\|_w^2)^{1/2}$. Here, $\|\cdot\|_w$ denotes the usual weighted L_2 norm. If $U(x)$ is the Nth-order interpolant

$$U(x)=\sum_{n=0}^{N} a_n T_n(x), \qquad -1\leqq x\leqq 1 \tag{4}$$

of $u(x)\in H_w^m(-1, 1)$ for $m>1$, the interpolation error in the sup norm is given by [5]

$$\|u(x)-U(x)\|_{L_\infty(-1,1)}\leqq CN^{1/2-m}\|u\|_{H_w^m(-1,1)}. \tag{5}$$

Thus, the rate of decay of the error depends only on the smoothness of u, and that error decay is uniform over the entire interval. Furthermore, if the function is analytic in an ellipse about the real axis, the error decay is exponential [17].

A consequence of the spectral interpolation error is that the derivative approximations are also spectrally accurate. For instance, the error of the Chebyshev difference approximation of a function analytic in an ellipse about the origin decays exponentially as [17]

$$\left\|\frac{\partial u(x)}{\partial x}-\frac{\partial U(x)}{\partial x}\right\|_{L_2(-1,1)}\leqq C(\eta)N^2 e^{-\eta N} \tag{6}$$

where η is determined by the size of the ellipse.

Convergence results for scalar hyperbolic equations have been presented by Canuto and Quarteroni [3], and for systems by Gottlieb, Lustman, and Tadmor [7]. Comparisons with finite difference methods for the solutions of some nonlinear gas-dynamics problems can be found in Hussaini et al. [9], [10], and Kopriva et al. [13]. They show that the Chebyshev collocation method can give higher accuracy with fewer grid points than can be obtained with typical second-order finite difference methods, provided the solution is smooth.

3. Domain decomposition. To avoid the need for global mappings and to allow for local grid refinement, multidomain spectral methods for hyperbolic equations have been presented by Kopriva [11]. The essential feature of the approach is that the computational domain is divided into nonoverlapping subdomains. Within each subdomain the standard collocation procedure is applied. At interfaces between subdomains, a characteristic approach which allowed information to be propagated from either side of the interface was used. Two-dimensional computations were performed by Kopriva [11], [12]. In all cases, grid lines are aligned across interfaces so that the number of grid points in each subdomain cannot be specified independently.

3.1. Grid description. In the current approach, a general domain Ω is divided into a number of logically rectangular subdomains. The subdomains need not be represented in a single coordinate system and they are allowed to overlap. Figure 1 presents a schematic of the situation. Each subdomain ("grid") is assigned a grid number, k, and level, l, and is represented by $G_{k,l}$. Grids on the same level have boundaries which share a common line called "patch" boundaries. Subdomains on different levels have "overlap" boundaries between them.

We do not consider full overlap or hierarchical grid embedding in the manner described by Berger and Oliger [2]. Instead, overlap is used only as is necessary for the convenient mapping of subdomains, a technique similar to the approach of Benek et al. [1]. The reason is because the Chebyshev approximation is global within each

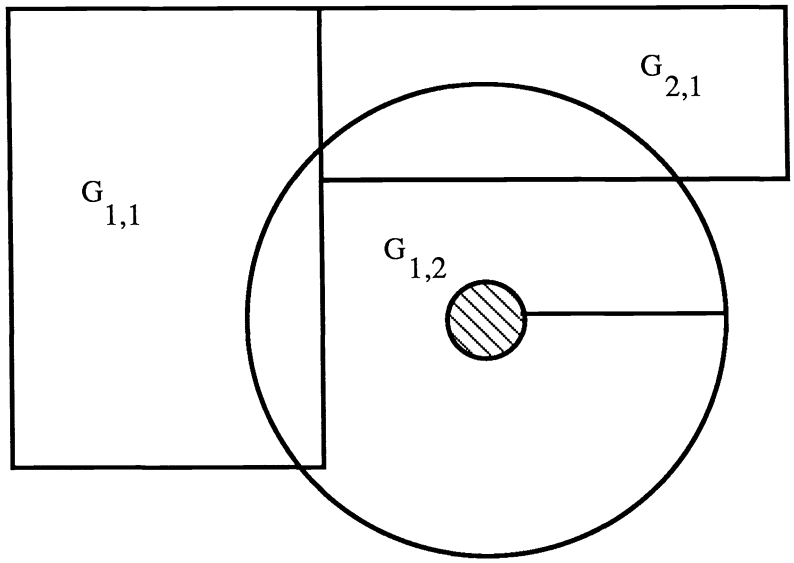


FIG. 1. *Grid diagram showing patched grids $G_{1,1}$ and $G_{2,1}$ with overset grid $G_{1,2}$.*

subdomain and communication occurs only through subdomain boundaries. Thus, if a fine grid were to overlay a portion of a coarse grid in such a way that the fine grid lay completely within the coarse grid boundaries, information would be propagated only from the coarse grid to the fine grid through the fine grid's boundaries. The fine-grid solution would not be used to replace locally the coarse-grid solution, and the accuracy of the fine-grid solution would be limited by that of the coarse grid.

Each subdomain is specified by the locations of its four corner points, (x_ν, y_ν) for $\nu = 1, 2, 3, 4$. The corners are numbered counterclockwise, as shown in Fig. 2, starting with the lower left. Subdomain boundaries are also numbered counterclockwise, as shown in Fig. 2.

A bilinear mapping is used to map the subdomain $G_{k,l}$ onto the square $R_{k,l}(X, Y) = [0, 1] \times [0, 1]$. Figure 2 shows the result of this mapping. If z represents either x or y,

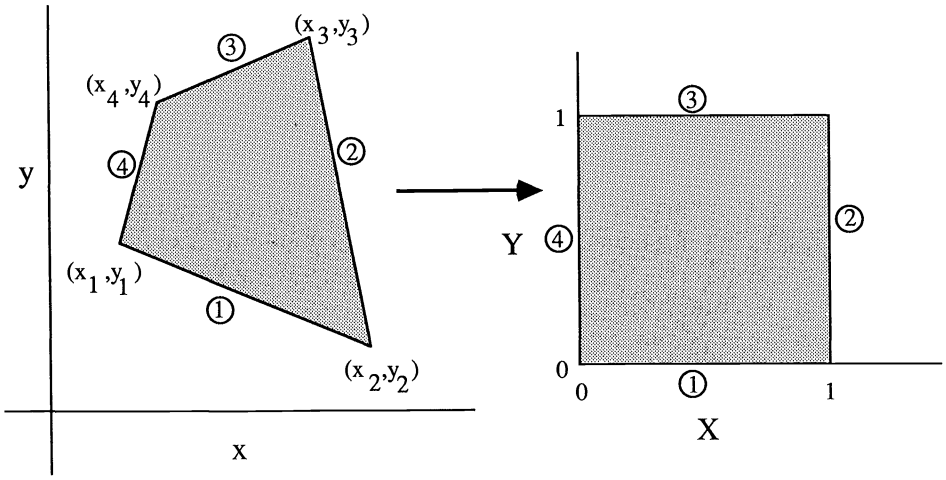


FIG. 2. *Mapping of a quadrilateral onto a square.*

then

$$z = z_1(1-X)(1-Y) + z_2X(1-Y) + z_3XY + z_4(1-X)Y. \tag{7}$$

It is not necessary to be restricted to this mapping. We can allow for curved boundaries in x and y. If the boundaries are exactly parametrizable, Gordon and Hall [6] show how to construct curvilinear coordinate systems. An isoparametric description of the subdomains, such as that used in the spectral element method of Korczak and Patera [14], is also possible. Grid-generation procedures have been detailed by Thompson, Warsi, and Mastin [18].

Each subdomain carries with it information about its neighbors. In particular, each grid $G_{k,l}$ has a list containing the

Grid number;
Level;
Boundary number;
Grid points;

of all grids needing boundary data from it.

3.2. Spectral discretization. Within each subdomain, (1) is rewritten in the local coordinate system

$$Q_t + A'Q_X + B'Q_Y = F, \qquad X, Y \in R_{k,l} \tag{8}$$

where

$$A' = AX_x + BX_y, \qquad B' = AY_x + BY_y.$$

The usual Chebyshev collocation method is applied locally. The grid is discretized at collocation points $X_i = \frac{1}{2}(1+\cos(i\pi/N))$, $Y_j = \frac{1}{2}(1+\cos(j\pi/M))$. Derivatives are approximated locally within the subdomain and are computed using matrix multiplication (Canuto et al. [5]). Since the interpolation, (3), does not require points outside of the grid, the derivatives are defined everywhere, including at the boundaries.

To allow for nonaligned grids between subdomains, data must be interpolated. To retain spectral accuracy, it is natural to use Chebyshev interpolation (3). As written, however, (3) is very expensive to compute. For high efficiency, the interpolations at a point (X, Y) are not computed in the form (3) but rather as

$$I_{NM}Q(X, Y) = \sum_{j,k=0}^{N,M} l_{jk}(X, Y)Q_{jk} \tag{9a}$$

where

$$l_{jk}(X, Y) = \frac{4}{NM\bar{c}_j\bar{c}_k} \sum_{n,m=0}^{N,M} \frac{T_n(2X_j-1)T_m(2Y_k-1)T_n(2X-1)T_m(2Y-1)}{\bar{c}_n\bar{c}_m}. \tag{9b}$$

The coefficients l_{jk} are computed once at the beginning of the calculation for each point (X, Y) that needs an interpolated value. At patch boundaries, the dimension of the interpolation is reduced by one, making those boundaries less expensive to compute.

Chebyshev interpolation is an $O(NM)$ operation in two dimensions and an $O(N)$ operation in one dimension and must be done at each boundary point. This may seem excessively expensive except for the fact that the computation of the derivatives is $O(N^2)$ for x-derivatives and $O(M^2)$ for y-derivatives. These derivatives must be computed at every grid point, not just those on the boundary. For the example pictured in Figs. 6 and 7, derivatives accounted for 49.4 percent of the calculation time.

Interpolations, written in the form of the double loop implied by (9a), accounted for only 23.7 percent of the calculation time. Unlike the matrix multiplication used for computing the derivatives, however, (9a) can be written as a single loop with index $p = j \cdot k$. This single loop can be explicitly vectorized on the Cyber 205 as a vector dot product with a Q8SDOT call. With explicit vectorization of (9a), the time for all interpolations used in the two-dimensional example pictured in Figs. 6 and 7 drops to only 8.7 percent of the total computation time.

The subdomains must be coupled in a way that takes into account the hyperbolic nature of the equations. This must be done carefully because spectral methods are very sensitive to boundary conditions. For instance, Hussaini et al. [10] have shown the disastrous effects of overspecifying boundary conditions in a situation where finite differences give acceptable results. See, also, Kopriva et al. [13]. Direct interpolation of solution unknowns from one grid to another (e.g., Benek et al. [1], Berger and Oliger [2]) overspecifies the boundary conditions, and hence is not suitable for use with spectral methods. Instead, characteristic data must be interpolated.

We classify characteristic interface treatments as either differential or correction methods. The differential methods use derivatives (interpolated if necessary) from neighbor subdomains to provide characteristic information. Procedures proposed by Kopriva [11] fall into this category. Correction schemes compute the solutions at subdomain boundaries with an unmodified interior scheme and correct the solutions to account for the characteristics. The boundary treatment of Gottlieb, Gunzburger, and Turkel [8] falls into this category.

To illustrate the interface treatments, we consider a one-dimensional hyperbolic problem on two overlapping grids as shown in Fig. 3. In the interior of each subdomain, the solution satisfies

$$Q_t + A(Q)Q_x = F. \tag{10}$$

If Z is the matrix that diagonalizes A, i.e., $A = Z\Lambda Z^{-1}$, an equation for the right-hand boundary of grid $G_{1,2}$ can be written as follows:

$$Q_t + \tfrac{1}{2}(A + |A|)Q_x^L + \tfrac{1}{2}(A - |A|)Q_x^R = F \tag{11}$$

where $|A| = Z|\Lambda|Z^{-1}$.

Equation (11) is just an extension of that found in Kopriva [11]. The difference is that the derivative Q_x^R is computed by interpolating the derivatives on grid $G_{1,1}$. For linear problems, this boundary treatment is nothing more than the method of

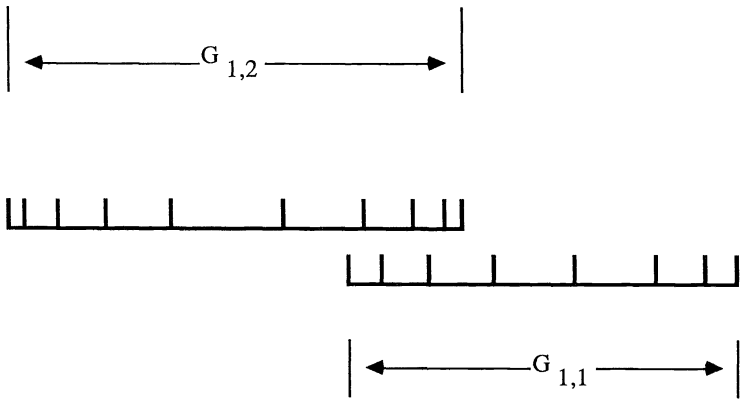


FIG. 3. *Two overlapping Chebyshev grids in one dimension.*

characteristics. The disadvantage of this straightforward extension arises in two dimensions, where both x- and y-derivatives must be interpolated.

The interface might also be handled by using a characteristic combination of the differential equations at the interface (cf. Canuto and Quarteroni [4]). In this way, (10) is written as follows:

$$Z^{-1}Q_t + \Lambda Z^{-1}Q_x = Z^{-1}F. \tag{12}$$

All quantities corresponding to positive elements of Λ are computed from $G_{1,2}$, while all quantities associated with negative elements of Λ are computed from $G_{1,1}$. In one dimension, this requires more interpolations than (11) since Q must also be interpolated. In two dimensions, Q and the derivatives in both directions must be interpolated.

The simplest and most efficient boundary procedure is a correction scheme. What we describe here is actually what was found in Gottlieb et al. [8], recast in a different form. It has the advantage of being the simplest to program and it requires the least number of interpolations.

If Z is linearized about some reference state, (12) indicates that the entries of the $Z^{-1}Q$ correspond to characteristic variables associated with the positive and negative values of A. Suppose that A has r positive and $s = m - r$ negative eigenvalues. Then we partition

$$\begin{bmatrix} w^+ \\ w^- \end{bmatrix} = Z^{-1}Q \tag{13}$$

where $w^+ \in \mathbb{R}^r$, $w^- \in \mathbb{R}^s$. Suppose Q in (13) is the result of using the interior approximation to (10) at the right boundary of $G_{1,2}$. Then the w^- vector should be replaced by the w^- vector from $G_{1,1}$. Specifically, if we call

$$\begin{bmatrix} w^+_{1,2} \\ w^-_{1,2} \end{bmatrix} = Z^{-1}Q_{1,2}, \qquad \begin{bmatrix} w^+_{1,1} \\ w^-_{1,1} \end{bmatrix} = Z^{-1}Q_{1,1} \tag{14}$$

the corrected vector is

$$Q^c_{1,2} = Z\begin{bmatrix} w^+_{1,2} \\ w^-_{1,1} \end{bmatrix}. \tag{15}$$

In two dimensions, this characteristic combination is computed in the direction normal to the boundary. At corners, only one direction is used. This has worked effectively in the two-dimensional test problem of § 4.

Using the correction method, only the dependent variables must be interpolated. Thus, in two dimensions it requires $\frac{1}{2}$ the number of interpolations needed by (11) and $\frac{1}{3}$ the number of interpolations needed by (12). For this reason, we prefer the correction method in more than one space dimension.

4. Numerical examples. In this section, we present one- and two-dimensional examples of overset and patched domain calculations. In one dimension we solve a linear time-dependent example using a differential interface condition. In two dimensions a complicated geometry is considered when we find the steady-state solution of a nonlinear system. In both cases, a grid-refinement study is done to show that spectral accuracy is obtained with either interface condition.

4.1. The one-dimensional time-dependent example. The first problem is the time-dependent 2×2 system

$$\begin{bmatrix} u \\ v \end{bmatrix}_t + \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}_x = 0 \tag{16}$$

solved on two domains $G_{1,2}=[-1, 0.4]$, $G_{1,1}=[0.13, 2.0]$. This system has two characteristic speeds, +3 and −1. The initial conditions are the Gaussian pulses

$$u(x, 0) = e^{-(x+0.5)^2/0.3},$$
$$v(x, 0) = e^{-(x-0.5)^2/0.3}. \tag{17}$$

One boundary condition is specified at $x=-1$ and one at $x=2$. On the left, the characteristic quantity associated with the right-going wave is specified:

$$\tfrac{1}{2}(u(-1, t)+v(-1, t)) = e^{-(0.5+3t)^2/0.3}+e^{-(t-1.5)^2/0.3}. \tag{18}$$

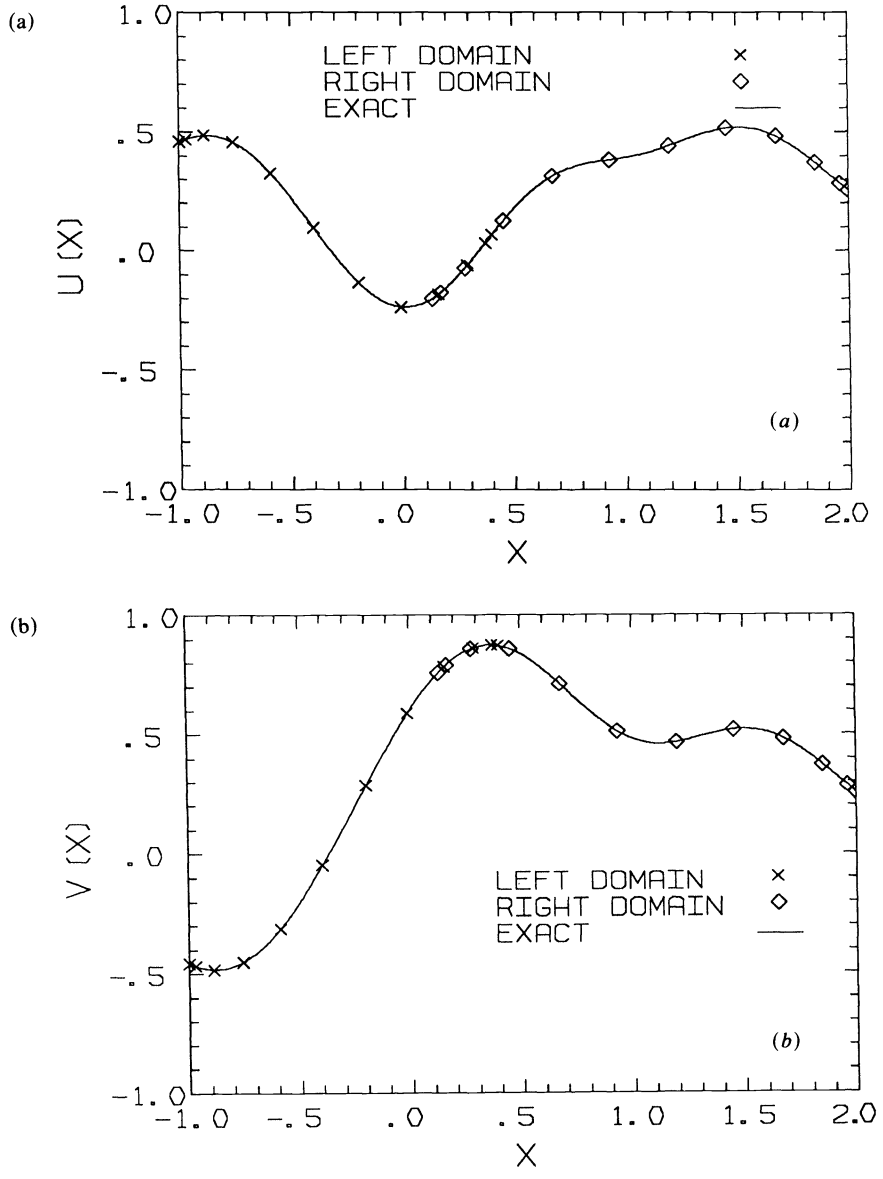


FIG. 4. *Solutions of the linear system* (16) *using overset Chebyshev grids.* (a) *u solution.* (b) *v solution.*

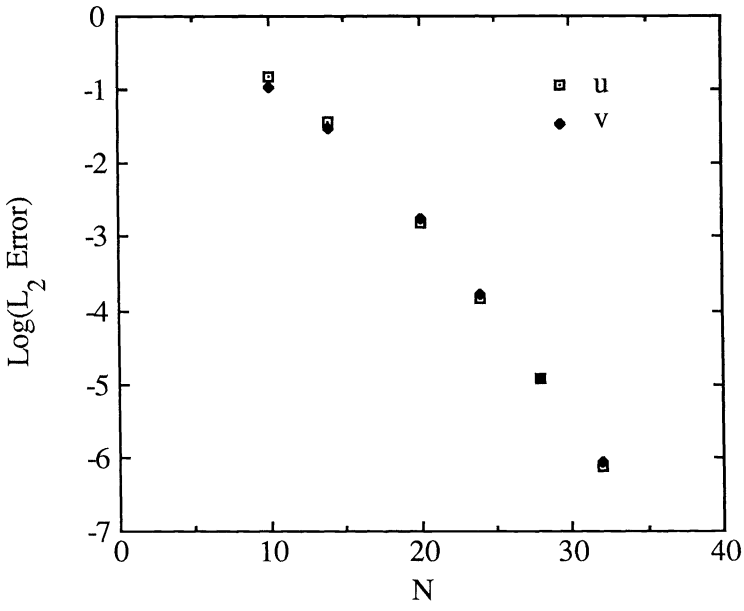


FIG. 5. *Error decay of the solutions to* (16) *using the overset Chebyshev grids.*

On the right, we specify the other characteristic quantity:

$$\tfrac{1}{2}(u(2,t)-v(2,t)) = e^{-(2.5-3t)^2/0.3} - e^{-(t+1.5)^2/0.3}. \tag{19}$$

The interface was solved using (11). Time integration was performed with a second-order Runge-Kutta, and the timestep was chosen small so that the temporal errors could be neglected. Figure 4 shows the solutions to this system at time $t = 0.35$ with 12 collocation points in each subdomain.

Spectral error decay is observed for this problem. The error behavior as the number of collocation points is increased is shown in Fig. 5. As (6) indicates, the discretization error should decay exponentially fast. Figure 5, which plots the natural logarithm of the discrete weighted L_2 error against the total number of points, shows that this is the case.

4.2. The two-dimensional steady-state example. As a two-dimensional example, we seek the time asymptotic steady-state solution to the nonlinear system

$$\begin{bmatrix} u \\ v \end{bmatrix}_t + \begin{bmatrix} u & 0 \\ 0 & u \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}_x - \begin{bmatrix} 0 & v \\ v & 0 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}_y = F(x,y),$$

$$u(x,y,0) = u_0(x,y), \qquad v(x,y,0) = v_0(x,y), \tag{20}$$

$$F(x,y) = \begin{bmatrix} y\,e^{2xy} + \sin(y)\cos(x)\,e^{2\cos(x)\cos(y)} \\ -(\sin(x)\cos(y)+x)\,e^{xy+\cos(x)\cos(y)} \end{bmatrix}.$$

The source term $F(x,y)$ and the characteristic boundary conditions were chosen so that the exact steady solution is

$$u = e^{(xy)}, \qquad v = e^{\cos(x)\cos(y)}.$$

The initial conditions u_0 and v_0 were chosen as 125 percent of the exact steady solution. A four-stage Runge–Kutta scheme was used to integrate in time at a Courant number of 2.5 until the maximum residual on every grid dropped below 10^{-10}.

The system was solved on the "Whiskey Barrel" grid shown in Fig. 6. The four patched subdomains on level one use Cartesian coordinates. The overset level is specified in polar coordinates with the center of the cut-out circle at the origin. Grid lines do not match at subdomain boundaries. The subdomain boundaries were computed with the correction scheme described in the previous section.

Contour plots of the computed steady solutions are shown in Fig. 7. The largest maximum errors of both solutions on any subdomain is roughly 1.2×10^{-3}. To produce this figure, the solutions on each subdomain were interpolated to a fine grid with 40 points in each direction. The contours were then drawn independently with a simple straight-line contour routine without smoothing. The results are quite smooth. In fact, it is almost impossible to tell where the subdomain boundaries are located.

Spectral accuracy is also observed for this problem. Figure 8 shows the error decay as the total number of points is increased. Plotted is the logarithm of the maximum error over all subdomains and both solutions versus the square root of the total number

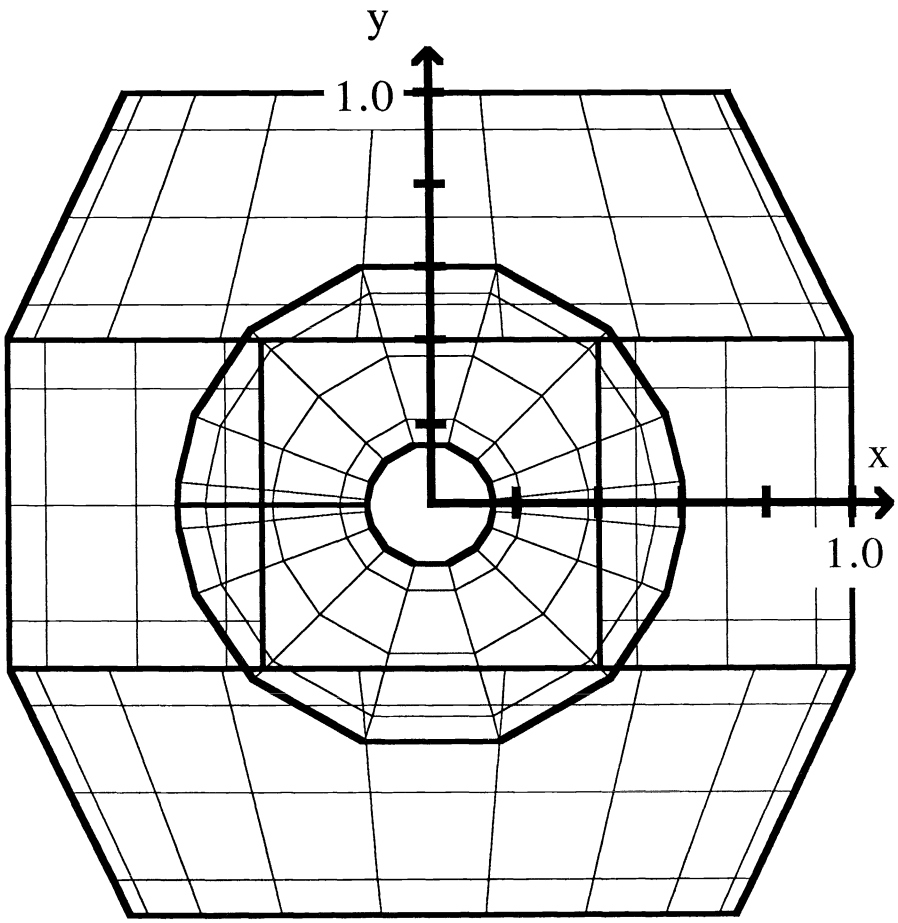


FIG. 6. *Grid geometry for the solution of the two-dimensional system,* (20). *Heavy lines denote subdomain boundaries, light lines show the Chebyshev grid.*

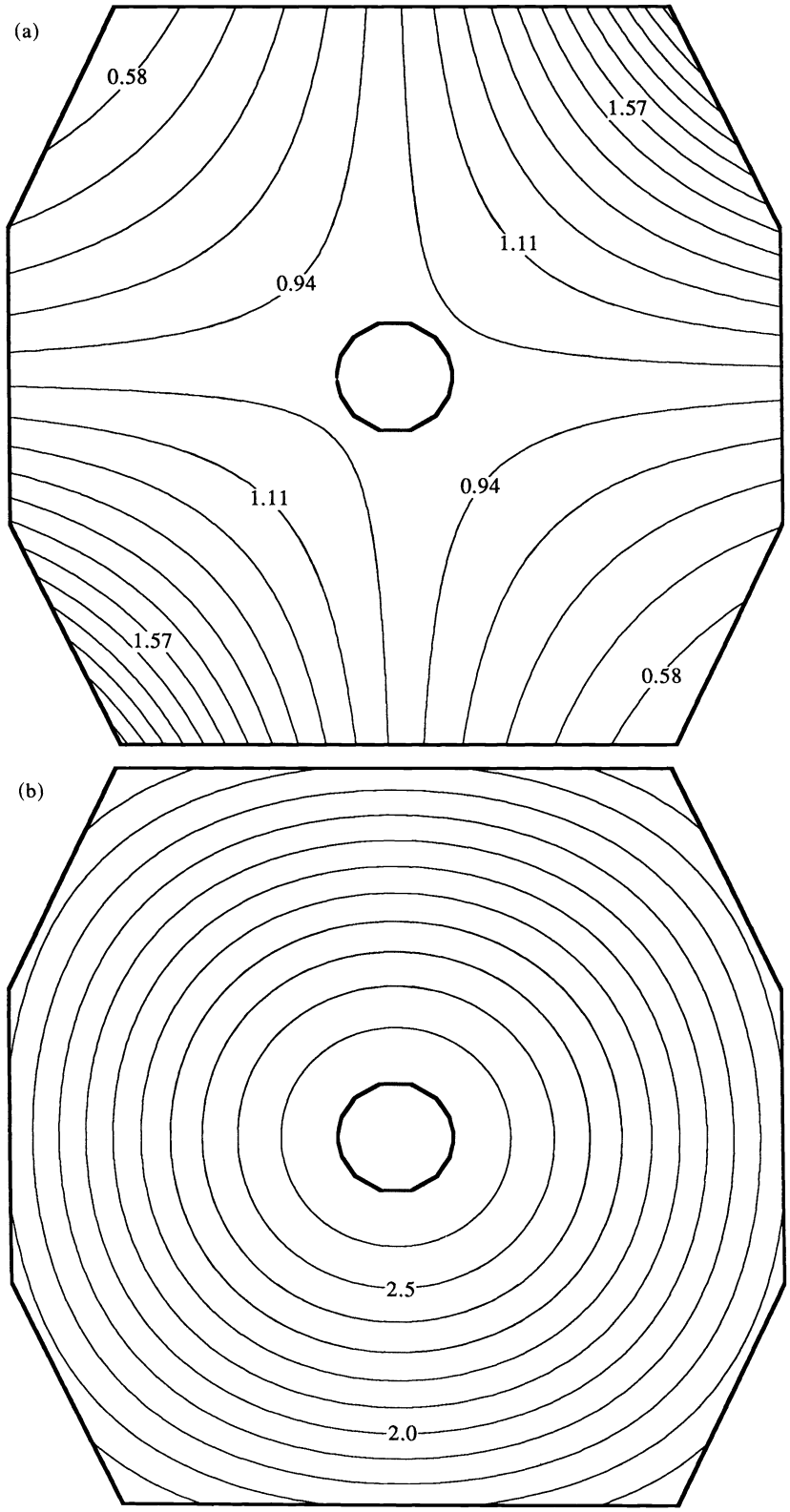


FIG. 7. *Chebyshev solutions to the system* (20) *on the grid of Fig.* 6. (a) *u solution.* (b) *v solution.*

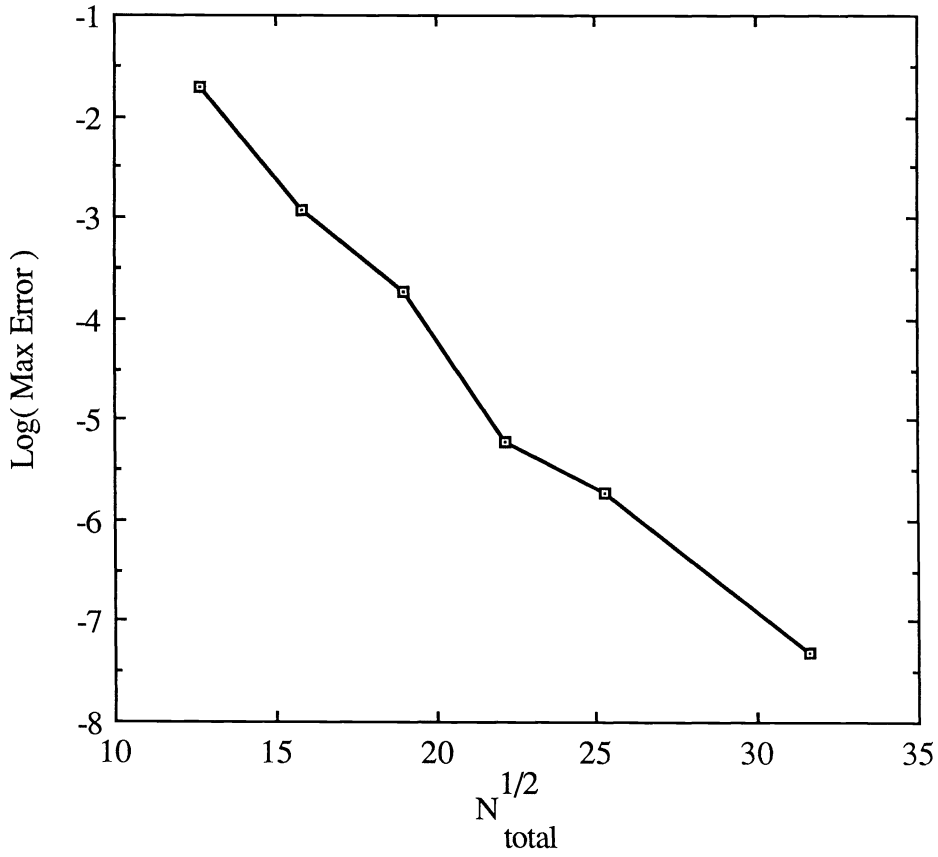


FIG. 8. *Error decay of the solutions to* (20).

of grid points. Since both of the exact solutions are analytic, we observe exponential error decay.

5. Summary. Solutions to hyperbolic systems in complex geometries with spectral methods can be obtained using a combination of patched and overset grids. The essential feature of the approach is to use the spectrally accurate interpolation, which is the basis of the Chebyshev collocation method. This interpolation is fairly expensive so it should be done as rarely as possible and in the form of (9) rather than in terms of the Chebyshev coefficients. With vectorization, the cost of the interpolation can be made even less. For the two-dimensional example with a total of 250 grid points, interpolation accounted for only 8.7 percent of the total computation time.

Interfaces can be treated by either a differential method or a correction scheme. Both give spectrally accurate results. However, the correction method is preferred in more than one dimension because it is the simplest to program and requires the fewest number of interpolations.

REFERENCES

[1] J. A. BENEK, J. L. STEGER, AND F. C. DOUGHERTY, *A flexible grid embedding technique with application to the Euler equations*, AIAA Paper 83-1944, American Institute of Aeronautics and Astronautics, New York, 1984.

[2] M. BERGER AND J. OLIGER, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comp. Phys., 53 (1984), pp. 484-512.

[3] C. CANUTO AND A. QUARTERONI, *Error estimates for spectral and pseudo-spectral approximations of hyperbolic equations*, SIAM J. Numer. Anal., 19 (1982), pp. 629-642.

[4] ———, *On the boundary treatment in spectral methods for hyperbolic systems*, J. Comput. Phys., 71 (1987), pp. 100-110.

[5] C. CANUTO, M. Y. HUSSAINI, A. QUARTERONI, AND T. A. ZANG, *Spectral Methods in Fluid Mechanics*, Springer-Verlag, New York, Berlin, 1987.

[6] W. J. GORDON AND C. A. HALL, *Construction of curvilinear co-ordinate systems and applications to mesh generation*, Internat. J. Numer. Methods Engrg., 7 (1973), pp. 461-477.

[7] D. GOTTLIEB, L. LUSTMAN, AND E. TADMOR, *Convergence of spectral methods for hyperbolic initial-boundary value problems*, ICASE Report 86-8, ICASE-NASA Langley Research Center, Hampton, VA, 1986.

[8] D. GOTTLIEB, M. GUNZBURGER, AND E. TURKEL, *On numerical boundary treatment for hyperbolic systems*, SIAM J. Numer. Anal., 19 (1982), pp. 671-697.

[9] M. Y. HUSSAINI, D. A. KOPRIVA, M. D. SALAS, AND T. A. ZANG, *Spectral methods for the Euler equations: Part* I*—Fourier methods and shock-capturing*, AIAA J., 23 (1985), pp. 64-70.

[10] ———, *Spectral methods for the Euler equations: Part* II*—Chebyshev methods and shock-fitting*, AIAA J., 23 (1985), pp. 234-240.

[11] D. A. KOPRIVA, *A spectral multidomain method for the solution of hyperbolic systems*, Appl. Numer. Math., 2 (1986), pp. 221-241.

[12] ———, *A multidomain spectral collocation computation of the sound generated by a shock-vortex interaction* in Computational Acoustics, D. Lee, R. Sternberg, and M. Schultz, eds., North-Holland, Amsterdam, 1988.

[13] D. A. KOPRIVA, T. A. ZANG, M. D. SALAS, AND M. Y. HUSSAINI, *Pseudospectral solution of two-dimensional gas-dynamics problems*, in Proc. 5th GAMM Conference on Numerical Methods in Fluid Mechanics, M. Pandolfi and R. Piva, eds., Vieweg & Sohn, Braunschweig, 1984, pp. 185-192.

[14] K. Z. KORCZAK AND A. T. PATERA, *An isoparametric spectral element method for solution of the Navier-Stokes equations in complex geometry*, J. Comput. Phys., 62 (1986), pp. 361-382.

[15] B. KREISS, *Construction of a curvilinear grid*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 270-279.

[16] M. M. RAI, *Navier-Stokes simulations of rotor-stator interaction using patched and overlaid grids*, AIAA Paper 85-1519, American Institute of Aeronautics and Astronautics, New York, 1985.

[17] E. TADMOR, *The exponential accuracy of Fourier and Tchebyshev differencing methods*, SIAM J. Numer. Anal., 23 (1986), pp. 1-10.

[18] J. F. THOMPSON, Z. U. A. WARSI, AND C. W. MASTIN, *Numerical Grid Generation*, North-Holland, New York, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 133-153, January 1989
© 1989 Society for Industrial and Applied Mathematics
011

GLIMM'S SCHEME AND CONSERVATION LAWS OF MIXED TYPE*

HERVÉ GILQUIN†

Abstract. The focus of this paper is on the numerical solution of problems modeled by conservation laws of mixed type. The numerical approximation of a conservation law from nonlinear elasticity (the equation of the motion of an elastic string in the plane), which changes type from hyperbolic to mixed, is considered. An exact solution that will be the test problem is exhibited.

Two versions of Glimm's scheme are investigated. According to the results of Pego and Serre, the first version of Glimm's scheme does not allow the computation of the solution when it leaves the hyperbolic domain to enter the mixed one because the computed solution is not kinematically admissible. The second version of Glimm's scheme (the original one) allows the reconstruction of a numerical solution which is kinematically admissible and dynamically admissible on an average but not locally.

Key words. Glimm's scheme, conservation laws, nonlinear elastic string, change of type

AMS(MOS) subject classifications. 65M30, 73C50, 73G05

1. Introduction. We investigate the numerical resolution of problems modeled by conservation laws of mixed type. Many phenomena in elasticity are governed by such equations, for example, the following:

(A) Longitudinal (or pure shearing) motions of a material admitting phase changes:

$$\begin{aligned} &u_{tt} = \sigma(u_x)_x, \\ &u(x,t): \mathbb{R}\times\mathbb{R}^+ \to \mathbb{R}, \\ &\text{with } \sigma'(u) > 0 \quad \text{if } u \notin [\alpha, \beta], \\ &\text{and } \sigma'(u) \leqq 0 \quad \text{if } u \in [\alpha, \beta]. \end{aligned} \tag{1.1}$$

One phase of the material can be distinguished from another using intervals where $\sigma'(u_x)$ stays of constant sign.

System (1.1) is hyperbolic if $\sigma' < 0$ and elliptic if $\sigma' > 0$. The same basic system is obtained when we study the motion of an isentropic Van der Waals gas [HS].

(B) The motion of an elastic string

$$y_{tt} = \left(T(|y_s|) \frac{y_s}{|y_s|} \right)_s \tag{1.2}$$

$y(s,t): \mathbb{R}\times\mathbb{R}^+ \to \mathbb{R}^2$ or $\mathbb{R}^3$ is the location at time t of the point of the string that is point of reference s when the string is at rest. The stress T satisfies

$$\begin{aligned} &T(1) = 0, \\ &T'(r) > 0 \quad \text{for } r > 0, \end{aligned} \tag{1.3a}$$

and for the sake of simplicity, we will assume

$$\begin{aligned} &\text{either } T''(r) > 0, \quad \text{or} \\ &T''(r) < 0 \quad \text{and} \quad T'(r) > rT(r) \quad \forall r > 0. \end{aligned} \tag{1.3b}$$

* Received by the editors November 18, 1987; accepted for publication (in revised form) April 6, 1988.

† Equipe d'Analyse Numérique, Unité de Formation et de Recherche Sciences, 23, rue du Dr. Paul Michelon, 42023 Saint-Etienne Cedex 2, France. Present address, Ecole Normale Supérieure de Lyon, Laboratoire de Mathématiques, 46 Allée d'Italie, 69364 Lyon Cedex 07, France.

The system (1.2) is strictly hyperbolic if $y \in H = \{y | r = |y_s| > 1\}$ and of mixed type if $r < 1$. It has been studied in [Sh], [KK], and [CRS].

The difficulty of the numerical approximation of such systems arises from the following:

(I) Classical schemes (Godunov, Glimm, van Leer, etc.) are mainly based on solving Riemann problems, resolution of which is not always possible if the solution leaves the hyperbolic domain.

(II) These systems are ill posed when the solution leaves the hyperbolic domain and numerical approximations are very unstable (Hadamard instabilities).

Our goal is not to predict unstable states, but to find stable states, particularly when the solution comes back in the hyperbolic domain after having left it.

For systems (1.1) and (1.2) under assumptions (1.3), it has been proved [Sh], [CRS] that the hyperbolic domain is stable by Riemann problem solution and therefore by Glimm's scheme. A theoretical study of approximate solutions of (1.1) and (1.2) by Glimm's scheme [Gl] has been done in [PS] and it is proved that weak convergence and hence strong convergence cannot occur in some cases where the solution leaves the hyperbolic domain.

We restrict our investigation to system (1.2) because it is too expensive to solve the Riemann problem for system (1.1). In this article we show that the original version of Glimm's scheme [Gl] is sufficiently robust to allow computations even when the solution leaves the hyperbolic domain.

An elastic string, which is a circle and for which the exact solution is known, is analyzed as a two-dimensional (2D) test case and we investigate numerical approximations of system (1.2) computed using two versions of Glimm's scheme. According to the results of [PS], the first version of Glimm's scheme does not allow the computation of the solution when it leaves the hyperbolic domain to enter the mixed one because the numerical solution is not kinematically admissible. The second version of Glimm's scheme (the original one) allows the reconstruction of a numerical solution that is kinematically admissible and dynamically admissible on an average but not locally.

The paper is organized as follows. In § 2, we present the system (1.2) and its Riemann problem solution. The test problem is analyzed in § 3 and we define the numerical cases studied. In § 4, we summarize the two versions of Glimm's scheme and we recall a result given in [PS]. The numerical solutions obtained with the first version of Glimm's scheme are presented in § 5 and those obtained with the original version in § 6; § 7 is devoted to a discussion about the results.

2. The equation of the elastic string motion in the plane. The system (1.2) models the motion of an elastic string in the plane when curvature effects are neglected.

Let

$$M(s, t) = \begin{pmatrix} w_1(s, t) \\ w_2(s, t) \end{pmatrix}$$

be a point of the string and take

$$u_1 = \frac{\partial w_1}{\partial s}, \quad u_2 = \frac{\partial w_2}{\partial s}, \quad u_3 = \frac{\partial w_1}{\partial t}, \quad u_4 = \frac{\partial w_2}{\partial t}, \quad S(r) = \frac{T(r)}{r}, \quad r e^{i\theta} = u_1 + i u_2.$$

Then system (1.2) becomes

$$\begin{aligned} &\frac{\partial u_1}{\partial t} - \frac{\partial u_3}{\partial s} = 0, \qquad \frac{\partial u_2}{\partial t} - \frac{\partial u_4}{\partial s} = 0, \\ &\frac{\partial u_3}{\partial t} - \frac{\partial}{\partial s}(S(r)u_1) = 0, \qquad \frac{\partial u_4}{\partial t} - \frac{\partial}{\partial s}(S(r)u_2) = 0. \end{aligned} \tag{2.1}$$

The eigenvalues of (2.1) are

$$\lambda_1 = -\lambda_4 = -\sqrt{T'(r)}, \qquad \lambda_2 = -\lambda_3 = -\sqrt{T(r)/r}.$$

From [CRS], [KK], and [Sh], the system (2.1) is strictly hyperbolic in the domain

$$H = \{(u_1, u_2, u_3, u_4) | r = \sqrt{(u_1^2 + u_2^2)} > 1\}$$

if T satisfies hypothesis (1.3a) and (1.3b). Outside of H, the system is of mixed type.

When hypothesis (1.3a) and (1.3b) are satisfied, λ_1 and λ_4 are genuinely nonlinear and λ_2 and λ_3 are linearly degenerate. The Riemann problem has been analyzed in [CRS], [KK], and [Sh], and the solution is unique for every state given in H, a domain that is invariant for this problem. Therefore, the domain H is invariant by Glimm's scheme, but not necessarily invariant by Godunov's scheme [Go] or by the schemes studied in [Gi] because it is not convex.

Let us summarize the solution of the RP (Riemann problem), which is a very important tool for the application of Glimm's scheme.

Given a left state $U_l = (u_1, u_2, u_3, u_4)_l$ and a right state $U_r = (u_1, u_2, u_3, u_4)_r$, the solution of the RP consists of three intermediate constant states U_-, U_0, and U_+, which are disposed as on Fig. 1.

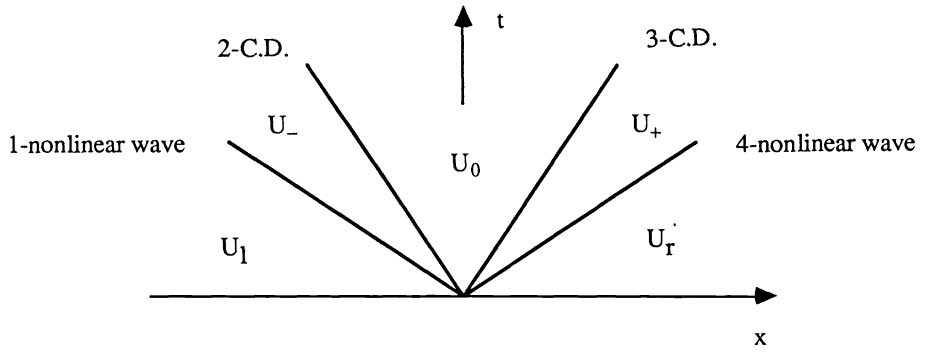


FIG. 1. *The Riemann problem for the system* (2.1).

By taking $r^* e^{i\theta^*} = u_1^* + iu_2^*$ for $* \in \{l, +, 0, -, r\}$ it is proved in [CRS] that $\theta^l = \theta^-$, $\theta^r = \theta^+$ (θ remains constant through a nonlinear wave) and $r^+ = r^0 = r^-$ (r remains constant through a contact discontinuity) and r^0 is the unique solution of the following equation:

$$J(r, U_l, U_r) = 0 \quad \text{with}$$

$$\begin{aligned} J(r, U_l, U_r) = {} & \tfrac{1}{4}[K_3^r(r) - K_3^l(r) + \sqrt{rT(r)}(\cos\theta^l + \cos\theta^r)]^2 \\ & + \tfrac{1}{4}[K_4^r(r) - K_4^l(r) + \sqrt{rT(r)}(\sin\theta^l + \sin\theta^r)]^2 - rT(r) \end{aligned} \tag{2.2}$$

where

$$K_3^*(r) = u_3^* + \operatorname{sgn}(*)\cos\theta^* \int_r^{r^*} \sqrt{T'(t)}\, dt$$

$$K_4^*(r) = u_4^* + \operatorname{sgn}(*)\sin\theta^* \int_r^{r^*} \sqrt{T'(t)}\, dt \qquad \text{if } \operatorname{sgn}(r^* - r) = \operatorname{sgn} T'',$$

and

$$K_3^{\#}(r) = u_3^{\#} + \operatorname{sgn}(\#)\cos\theta^{\#}(r^{\#}-r)\sqrt{(T(r^{\#})-T(r))/(r^{\#}-r)}$$

$$\text{if } \operatorname{sgn}(r^{\#}-r) \neq \operatorname{sgn} T'',$$

$$K_4^{\#}(r) = u_4^{\#} + \operatorname{sgn}(\#)\sin\theta^{\#}(r^{\#}-r)\sqrt{(T(r^{\#})-T(r))/(r^{\#}-r)}$$

$$\# \in \{l, r\}, \quad \operatorname{sgn}(\# = l) = -, \quad \operatorname{sgn}(\# = r) = +.$$

Then θ^0 satisfies

$$\tan\theta^0 = \frac{K_4^r(r) - K_4^l(r) + \sqrt{rT(r)}(\sin\theta^l + \sin\theta^r)}{K_3^r(r) - K_3^l(r) + \sqrt{rT(r)}(\cos\theta^l + \cos\theta^r)},$$

and we have

$$u_3^0 = u_3^l + r^0\sqrt{S(r^0)}(\cos\theta^0 - \cos\theta^l), \qquad u_4^0 = u_4^l + r^0\sqrt{S(r^0)}(\sin\theta^0 - \sin\theta^l),$$

or

$$u_3^0 = u_3^r + r^0\sqrt{S(r^0)}(\cos\theta^r - \cos\theta^0), \qquad u_4^0 = u_4^r + r^0\sqrt{S(r^0)}(\sin\theta^r - \sin\theta^0).$$

The Riemann problem is completely solved when the unique zero of $J(r, U_l, U_r)$ is computed.

From a numerical point of view, note that we tried to eliminate instabilities that would come from the method used to solve $J(r, U_l, U_r) = 0$. We used the bisection method because J is neither differentiable in r^r, nor in r^l, and therefore Newton's method is inadequate. Furthermore, for this problem, the secant method converges more slowly than the bisection method, and all the modifications to speed up the secant method have led to instabilities near $r = 1$.

3. The test problem. We consider an elastic string that is helicoidal at each time t. The point $M(s, t) = y(s, t) \in \mathbb{R}^3$, the location at time t of reference point s on the string, is given by

$$(3.1) \qquad y(s,t) = \begin{cases} \rho(t)\cos(\alpha(s,t)) \\ \rho(t)\sin(\alpha(s,t)) \\ a(t)s \end{cases} \quad \text{with } \alpha(s,t) = \omega(t)s - \varphi(t) \text{ and } \rho \not\equiv 0,$$

$$(3.2) \qquad y(s,0) = \begin{cases} \rho_0\cos(\alpha_0), \\ \rho_0\sin(\alpha_0), \\ a_0 s, \end{cases} \qquad y_t(s,0) = \begin{cases} \rho_0'\cos(\alpha_0) - \rho_0\alpha_0'\sin(\alpha_0), \\ \rho_0'\sin(\alpha_0) + \rho_0\alpha_0'\cos(\alpha_0), \\ a_0' s. \end{cases}$$

We want $y(s, t)$ to satisfy

$$y_{tt} = \left(T(|y_s|)\cdot\frac{y_s}{|y_s|}\right)_s = (S(|y_s|)\cdot y_s)_s.$$

To simplify, we will omit the variable t and time derivation will be represented by $'$; we obtain

$$y_s = \begin{cases} -\rho\omega\sin\alpha, \\ \rho\omega\cos\alpha, \\ a. \end{cases}$$

Thus, $|y_s| = \sqrt{a^2+\rho^2\omega^2}$ is independent of s such that

$$y_{tt} = S(\sqrt{a^2+\rho^2\omega^2})y_{ss},$$

$$y_{ss} = \begin{cases} -\rho\omega^2\cos\alpha, \\ -\rho\omega^2\sin\alpha, \\ 0, \end{cases} \qquad y_t = \begin{cases} \rho'\cos\alpha - \rho\alpha'\sin\alpha, \\ \rho'\sin\alpha + \rho\alpha'\cos\alpha, \\ a's, \end{cases}$$

$$y_{tt} = \begin{cases} \rho''\cos\alpha - 2\rho'\alpha'\sin\alpha - \rho\alpha''\sin\alpha - \rho\alpha'^2\cos\alpha, \\ \rho''\sin\alpha - 2\rho'\alpha'\cos\alpha - \rho\alpha''\cos\alpha - \rho\alpha'^2\sin\alpha, \\ a''s. \end{cases}$$

We obtain the following three equations:

$$\rho'' - \rho\alpha'^2 + \rho\omega^2 S(\beta) = 0, \tag{3.3}$$

$$2\rho'\alpha' + \rho\alpha'' = 0 \quad \text{with } \beta = \sqrt{(a^2+\rho^2\omega^2)}, \tag{3.4}$$

$$a''s = 0. \tag{3.5}$$

Equation (3.5) implies that a is a linear function of t and (3.4) leads to

$$\omega''\rho + 2\omega'\rho' = 0, \tag{3.6}$$

$$\varphi''\rho + 2\varphi'\rho' = 0, \tag{3.7}$$

and from (3.3), we have

$$\rho'' - \rho\varphi'^2 + \rho\omega^2 S(\beta) = 0, \tag{3.8}$$

$$\omega' = 0. \tag{3.9}$$

The frequency ω is constant (cf. (3.9)) and thus (3.6) is trivially verified. Equation (3.7) is equivalent to

$$\rho^2\varphi' = \mu, \qquad \mu = \text{constant}, \tag{3.10}$$

and finally, we have

$$\begin{aligned} &\rho'' - \frac{\mu^2}{\rho^3} + \rho\omega^2 S(\beta) = 0, \\ &\beta = \sqrt{a^2+\rho^2\omega^2}, \\ &a(t) = a_1 t + a_2. \end{aligned} \tag{3.11}$$

The Cauchy conditions give ω, a_0, a_0', ρ_0, ρ_0', φ_0, φ_0' and allow us to determine a_1, a_2, and μ. $a(t)$ is fixed by the initial conditions. Now assume that $a(t)$ is constant and take $R = \rho\omega$; (3.11) becomes

$$R'' + f(R) = 0 \quad \text{with } f(R) = \omega^2\left[RS(\sqrt{a^2+R^2}) - \frac{\mu^2\omega^2}{R^3}\right]. \tag{3.12}$$

In case of equilibrium, we have

$$S(\sqrt{a^2+R^2}) = \frac{\mu^2\omega^2}{R^4} \geqq 0.$$

From the stress properties (1.3a) and (1.3b), the elongation $(a^2+R^2)^{1/2}$ is greater than or equal to 1; according to this result, the equilibrium seems to be stable. Furthermore, f is an increasing function of $R \in \mathbb{R}^+$ and there exists $R^* > 0$ such that

$F(R) = \int_{R^*}^{R} f(r)\,dr$ is positive for $R \in \mathbb{R}^+$. After integration of (3.12), we have

(3.13) $\frac{1}{2}R'^2 + F(R) = C_0$, C_0 is a constant that depends only on the initial conditions.

Two cases arise.

Case 1. $\mu \neq 0$. We have

$$F(R) \to +\infty \quad \text{and} \quad F(R) \approx \frac{\mu^2\omega^4}{R^2} \to +\infty$$

$$R \to +\infty \qquad\qquad R \to 0^+.$$

Equation (3.13) then gives $0 < R_{\min} \leqq R \leqq R_{\max}$, for all $t \in \mathbb{R}^+$ with $R_{\min} \leqq R_* \leqq R_{\max}$. The motion is then periodic and the solution can leave the hyperbolic domain if $R_{\min} < 1/\omega$.

Case 2. $\mu = 0$, $a \neq 0$. R can be negative; this is not awkward from a theoretical point of view because we can extend f as an odd function. We take

(3.14) $F(R) = \omega^2 \int_0^R r \cdot S(\sqrt{a^2 + r^2})\,dr$ F is even and (3.13) gives $|R| \leqq R_{\max}$.

For numerical applications, we restrict ourselves to the case of an elastic string whose shape is a circle ($a = 0$); (3.11) becomes

(3.15) $$\rho'' - \frac{\mu^2}{\rho^3} + \rho\omega^2 S(\beta) = 0, \qquad \beta = \sqrt{\rho^2\omega^2}.$$

Furthermore, the computations described in the following sections will always correspond to $T(r) = r^2 - 1$ and $\omega = 1$. Assuming $\mu = 0$, the differential equation satisfied by ρ is

$$\rho'' = \frac{\mu^2}{\rho^3} + 1 - \rho^2,$$

and finally

$$\rho'^2 = \frac{-2}{3}\rho^3 + 2\rho - \frac{\mu^2}{\rho^2} + \rho_0'^2 + \frac{2\rho_0^3}{3} - 2\rho_0 + \frac{\mu^2}{\rho_0^2}.$$

If we take $\gamma = \rho_0'^2 + (2\rho_0^3/3) - 2\rho_0 + (\mu^2/\rho_0^2)$, ρ' is zero if

$$-\tfrac{2}{3}\rho^5 + 2\rho^3 + \gamma\rho^2 - \mu^2 = 0.$$

The real roots can be computed as soon as the initial data are known. For a two-dimensional problem, the five initial data that we can choose are: ω, ρ_0, ρ_0', φ_0, and φ_0'. (We recall that R, the radius of the circle (the shape of the string) is given by $R = \rho\omega$ and that $|y_s| = \rho\omega = R$.)

The two tests computed in the following sections are:

Test A.

$$\omega = 1, \quad \rho_0 = 2, \quad \rho_0' = -2, \quad \varphi_0 = 0, \quad \varphi_0' = 1.$$

The radius of the circle oscillates periodically between $R_{\min} = 1.231422$ and $R_{\max} = 2.645$; the solution remains in the hyperbolic domain.

Test B.

$$\omega = 1, \quad \rho_0 = 2, \quad \rho_0' = -10, \quad \varphi_0 = 0, \quad \varphi_0' = 1.$$

The radius of the circle oscillates periodically between $R_{\min} = 0.388384$ and $R_{\max} = 5.582529$; the solution leaves the hyperbolic domain (when R is less than 1).

4. Glimm's scheme. Glimm's scheme consists of two steps: a first step of exact resolution followed by a step of interpolation. These two steps are repeated at each timestep.

Let Δs be a fixed space step. We denote by $U_{\Delta s}(s, t) = (u_1, u_2, u_3, u_4)_{\Delta s}(s, t)$ the numerical approximation by Glimm's scheme at time t. This approximation is built with a random sequence a_n in $[-\frac{1}{2}, \frac{1}{2}]$ [Gl] or one that is equidistributed in $[-\frac{1}{2}, \frac{1}{2}]$ [Liu].

Let $U^R(U_l, U_r, \xi)$ be the Riemann problem solution (see Fig. 1) with Cauchy data $U_l = (u_1, u_2, u_3, u_4)_l$ and $U_r = (u_1, u_2, u_3, u_4)_r$. At time $t = 0$, the initial data is approximated by

$$U_{\Delta s}(s, 0) = U_{\Delta s}(j\Delta s, 0) \quad \text{if } s - j\Delta s \in [-\Delta s/2, \Delta s/2].$$

There are essentially two ways to handle the interpolation.

(1) The interpolation is built on a fixed grid. If the approximation is known at time $t = t^n$ and if $t^{n+1} = t^n + \Delta t^n$ is fixed, the solution of the Riemann problems determined by the discontinuities at time t^n is computed in the strip

$$t^n \leqq t < t^{n+1} \quad \text{and with } t = t^n + \tau,\ \tau \in [0, \Delta t^n[;$$

we have

$$U_{\Delta s}(s, t) = U^R\left(U_j^n, U_{j+1}^n, \frac{s - (j+\frac{1}{2})\Delta s}{t - t^n}\right), \qquad s - \left(j + \frac{1}{2}\right)\Delta s \in [-\Delta s/2, \Delta s/2].$$

The numerical approximation is built at time $t = t^{n+1}$ as follows:

$$U_j^{n+1} = U_{\Delta s}((j + a_{n+1})\Delta s,\ t^{n+1} - 0)$$

for a sequence a_n (see Fig. 2).

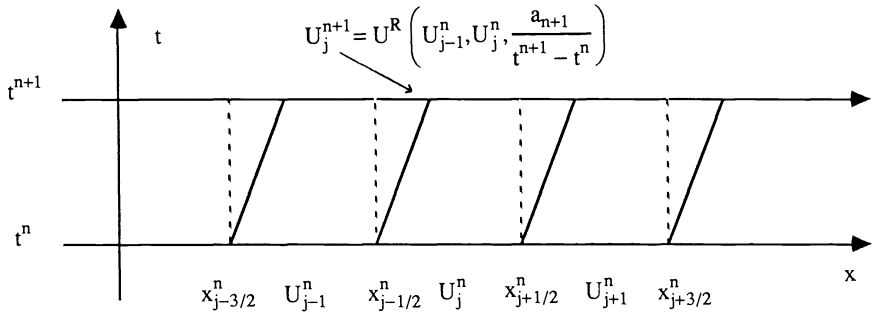


FIG. 2. *Glimm's scheme with sequence* a_n.

(2) The approximation is built on a staggered grid (the mesh is shifted by $\Delta s/2$ at each timestep). The nodes of the mesh are located at $(j+\frac{1}{2})\Delta s$ for n even and at $j\Delta s$ for n odd.

If the approximation is computed for $t = t^n$ and if $t^{n+1} = t^n + \Delta t^n$ is fixed, the solution of the Riemann problems determined by the discontinuities at time t^n is computed in the strip:

$$t^n \leqq t < t^{n+1} \quad \text{and with } t = t^n + \tau,\ \tau \in [0, \Delta t^n[;$$

we have

$$U_{\Delta s}(s, t) = U^R\left(U_j^n, U_{j+1}^n, \frac{s-(j+\frac{1}{2})\Delta s}{t-t^n}\right)$$

for n even and $s-(j+\frac{1}{2})\Delta s \in [-\Delta s/2, \Delta s/2]$,

$$U_{\Delta s}(s, t) = U^R\left(U_{j-1/2}^n, U_{j+1/2}^n \frac{s-j\Delta s}{t-t^n}\right) \text{for } n \text{ odd and } s-j\Delta s \in [-\Delta s/2, \Delta s/2].$$

The numerical approximation is built at time $t = t^{n+1}$ as follows:

$$U_{j+1/2}^{n+1} = U_{\Delta s}((j+\tfrac{1}{2}+a_{n+1,j})\Delta s, t^{n+1}-0) \quad \text{for } n \text{ even},$$

$$U_j^{n+1} = U_{\Delta s}((j+a_{n+1,j})\Delta s, t^{n+1}-0) \quad \text{for } n \text{ odd}$$

for a sequence $a_{n,j}$ (see Fig. 3).

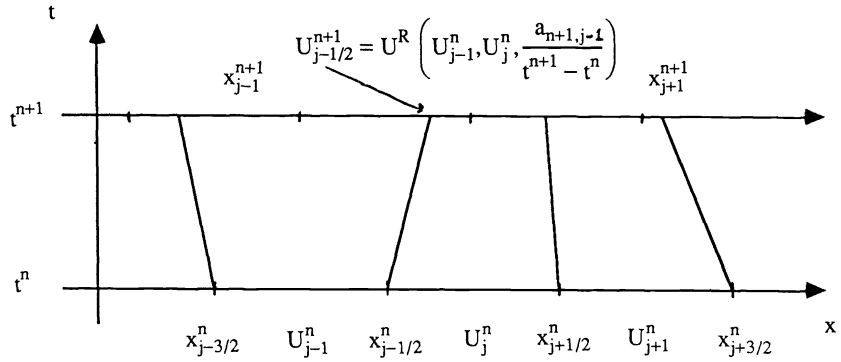


FIG. 3. *Glimm's scheme with sequence* $a_{n,j}$.

For the two-dimensional test problem, we shall notice that (2.1) and the initial data are invariant under the action of the group SO $(2, \mathbb{R})$. By taking a fixed timestep $\Delta t = \Delta s/\lambda$ and a sequence a_n (see Fig. 2), we can prove by induction that the numerical approximation remains invariant under the action of a discrete subgroup [PS]. It is then possible to solve only one RP at each timestep. When we take

$$K(r_1, r_2) = \begin{cases} \displaystyle\int_{r_1}^{r_2} \sqrt{T'(s)}\, ds & \text{if } \operatorname{sgn}(r_2-r_1) = \operatorname{sgn} T'', \\ \displaystyle -(r_2-r_1)\sqrt{\frac{T(r_2)-T(r_1)}{r_2-r_1}} & \text{if } \operatorname{sgn}(r_2-r_1) \neq \operatorname{sgn} T'' \end{cases}$$

with $u_1 + iu_2 = re^{i\theta}$ and $u_3 + iu_4 = v = -ize^{i\theta}$, (2.2) becomes

$$J(r, U_l, U_r) = \tfrac{1}{4}|v_r - v_l + K(r, r_l)\, e^{i\theta_l} + K(r, r_r)\, e^{i\theta_r} + \sqrt{rT(r)}(e^{i\theta_l} + e^{i\theta_r})|^2 - rT(r).$$

By induction we get

$$r_r^n = r_l^n = r^n, \quad v_r^n = -iz^n e^{i\theta_r^n}, \quad v_l^n = -iz^n e^{i\theta_l^n}$$

where z^n depends only on r^n with $\theta_l^n = (j\omega\Delta s - \varphi^n)$ and $\theta_r^n = ((j+1)\omega\Delta s - \varphi^n)$; and r^{n+1} is the unique solution of

$$J(r, r^n) = 0$$

where

$$J(r, r^n) = \left| z^n \sin \frac{\omega \Delta s}{2} + (K(r, r^n) + \sqrt{rT(r)}) \cos \frac{\omega \Delta s}{2} \right|^2 - rT(r).$$

For the test problem, in the case of an approximation using Glimm's scheme with a sequence a_n and for a fixed $\lambda = \Delta t / \Delta s$, the following proposition holds [PS]:

PROPOSITION [PS]. *If* $r^0 = 1$ *and* $\operatorname{Re}(z^0) < 0$, *there are two positive reals* B *and* T_1 *such that, for any* $T > 0$,

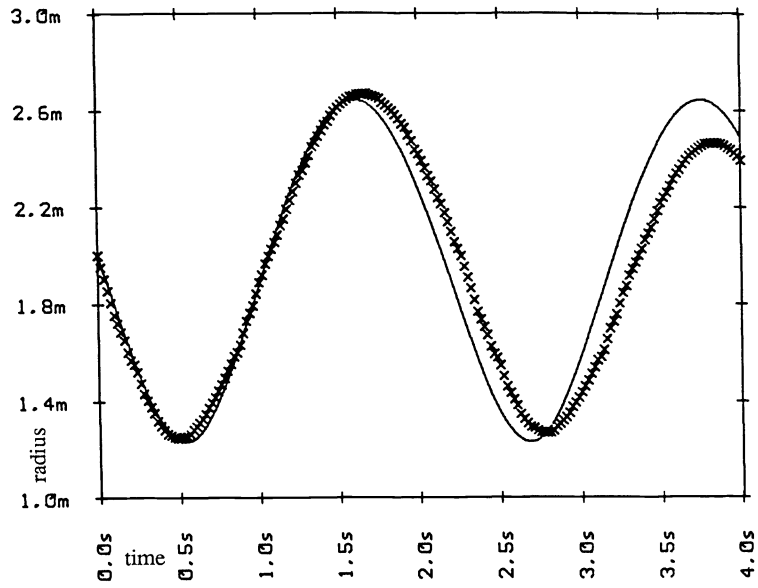


FIG. 4(a). *Glimm's scheme with sequence* a_n, *evolution of the radius* R *for* $\Delta s = 2\pi/200$.

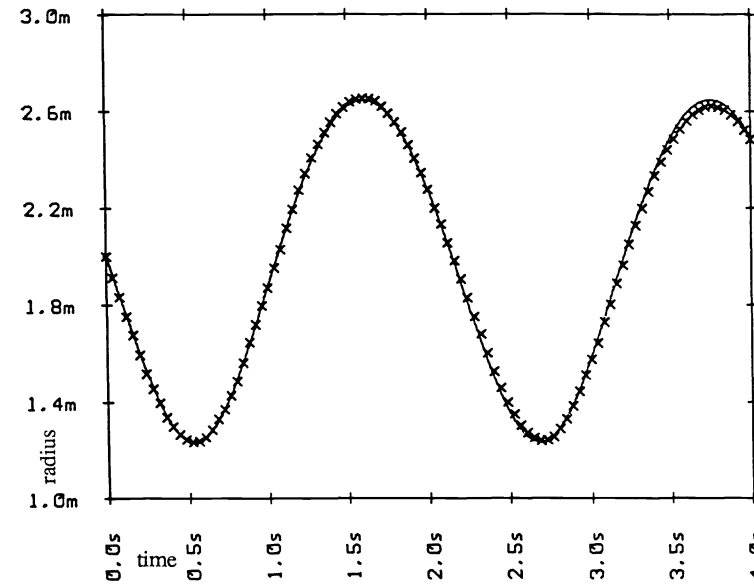


FIG. 4(b). *Glimm's scheme with sequence* a_n, *evolution of the radius* R *for* $\Delta s = 2\pi/500$.

$$\text{Prob}\{1 \leqq r^n \leqq 1 + B\Delta s^2 \text{ and } \operatorname{Re}(z^n) < 0, \forall 0 \leqq n \leqq T/\Delta t\} \to \exp(-T/T_1) \quad \Delta s \to 0.$$

That is to say that the initial data persist with a random lifetime distributed according to a Poisson distribution. This persistence of the initial condition corresponds to the time for which the numerical approximation satisfies $\operatorname{Re}(z^n) < 0$ and $r^n \sim 1$ (see Figs. 5(a), 5(b), 5(c)).

5. Numerical results with a sequence a_n. The exact solution of (2.1) for the 2D problem is invariant under the action of the group SO $(2, \mathbb{R})$, so it is for the numerical approximation obtained with the Glimm's scheme using a sequence a_n[PS]. The shape of the string remains a circle whose radius depends only on the time. We present in Figs. 4(a), 4(b), 5(a), 5(b), and 5(c) the evolution of the radius R of the numerical solution (represented with a cross), with the exact value (solid line) as reference.

Case A.

$$\omega = 1, \quad \rho_0 = 2, \quad \rho_0' = -2, \quad \varphi_0 = 0, \quad \varphi_0' = 1.$$

The values of the radius of the circle oscillate periodically between $R_{\min} = 1.231422$ and $R_{\max} = 2.645$; the exact solution always remains in the hyperbolic domain.

The numerical solution is very poor for $\Delta s = 2\pi/50$ and $\Delta s = 2\pi/100$; it slowly becomes better for $\Delta s = 2\pi/200$, but for $\Delta s = 2\pi/500$ it is an already good approximation and there is no special problem. The results become better as Δs vanishes.

Case B.

$$\omega = 1, \quad \rho_0 = 2, \quad \rho_0' = -10, \quad \varphi_0 = 0, \quad \varphi_0' = 1.$$

The values of the radius of the circle oscillate periodically between $R_{\min} = 0.388384$ and $R_{\max} = 5.582529$; the solution leaves the hyperbolic domain when R becomes less than 1.

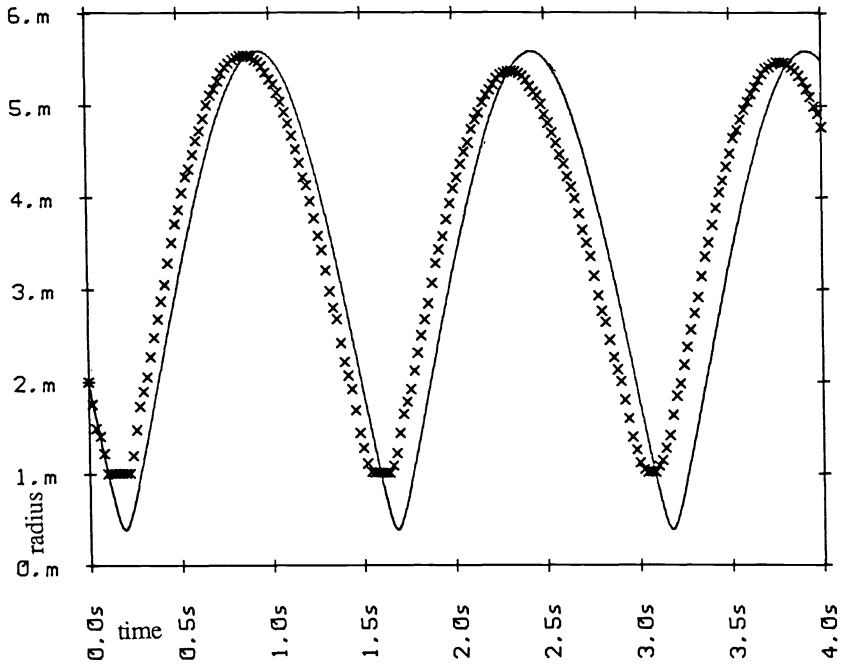


FIG. 5(a). *Glimm's scheme with sequence a_n, evolution of the radius R for $\Delta s = 2\pi/200$.*

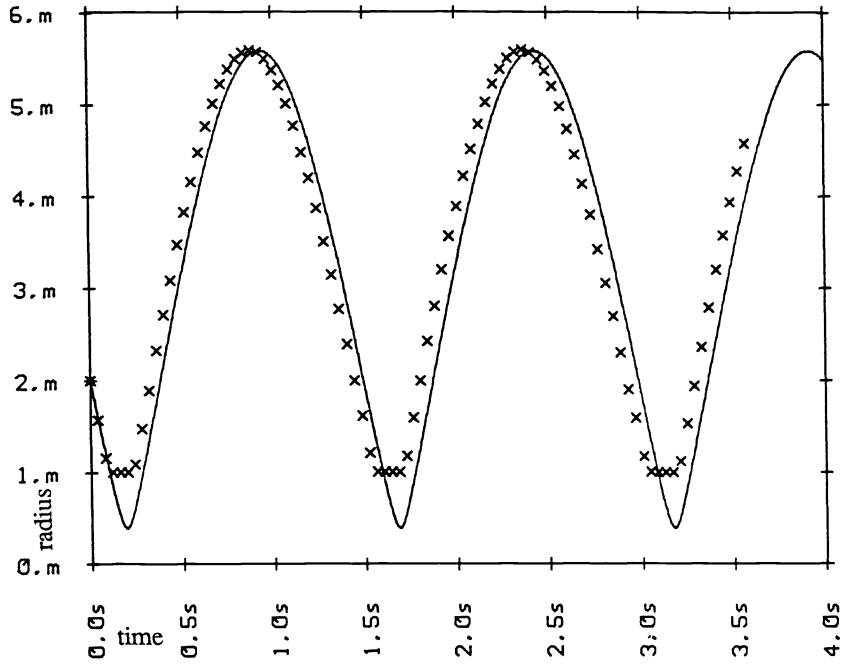


FIG. 5(b). *Glimm's scheme with sequence* a_n, *evolution of the radius* R *for* $\Delta s = 2\pi/500$.

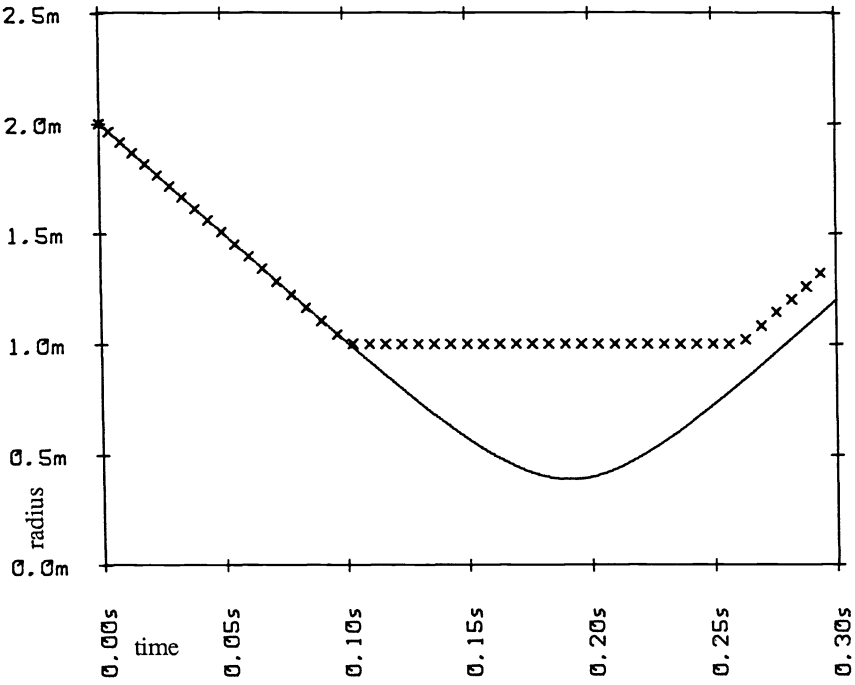


FIG. 5(c). *Glimm's scheme with sequence* a_n, *evolution of the radius* R *for* $\Delta s = 2\pi/2{,}000$.

As is proved in [PS], the numerical solution cannot leave the hyperbolic domain; it is shifted forward or backward from the exact one. This is due to the random lifetime during which $\mathrm{Re}\,(z^n) < 0$ and $r^n \sim 1$. We still remark that the maximum radius $R_{\max}$ is better approximated as Δs vanishes.

The accurate behavior of the numerical solution for $\Delta s = 2\pi/2{,}000$, near $R = 1$, is given on Fig. 5(c).

6. Numerical results with a sequence $a_{n,j}$. The numerical solution of (2.1) does not remain invariant under the action of the group SO $(2, \mathbb{R})$. We must solve as many RP as mesh points at each timestep. Two problems arise.

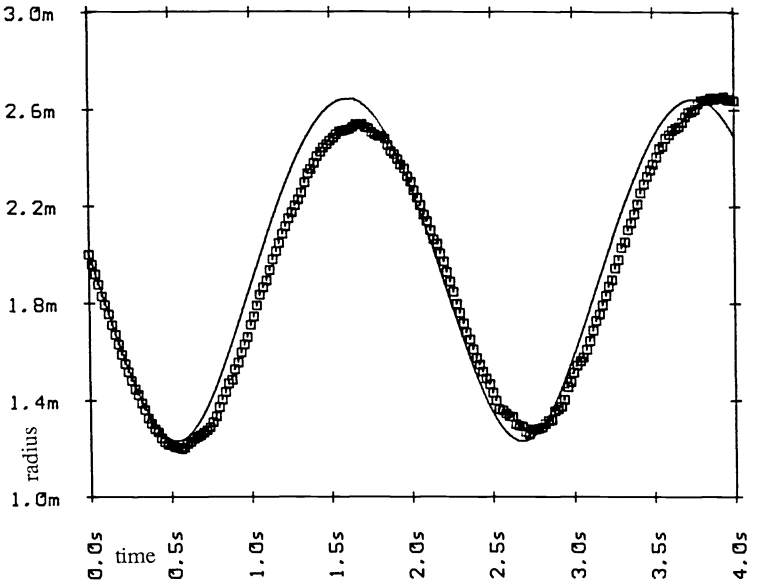


FIG 6(a). *Glimm's scheme with sequence $a_{n,j}$, evolution of the radius R for $\Delta s = 2\pi/200$.*

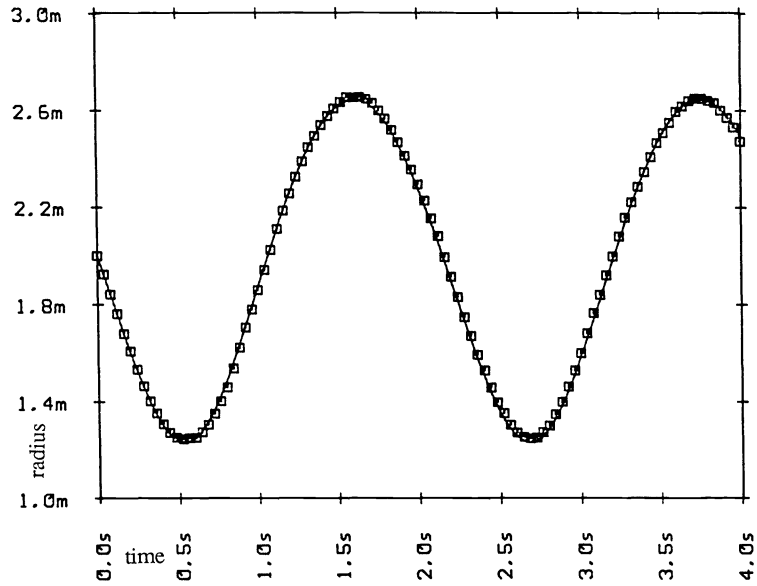


FIG. 6(b). *Glimm's scheme with sequence $a_{n,j}$, evolution of the radius R for $\Delta s = 2\pi/500$.*

(1) The computation of w_1 and w_2. We know $u_1 = \partial w_1/\partial s$ and $u_2 = \partial w_2/\partial s$ at each timestep; we must evaluate w_1 and w_2 from u_1 and u_2, and reconstruct the shape of the string. (See details of these computations in the Appendix.)

(2) The computation of the radius R. The shape of the string is no longer circular but the area defined by the string, which is a closed curve, is given by

$$A = \frac{1}{2}\int_0^{2\pi} \left(\frac{\partial w_1}{\partial s} w_2 - \frac{\partial w_2}{\partial s} w_1 \right) ds,$$

and the radius R of the circle, whose area would be A, is given by $R = \sqrt{A/\pi}$.

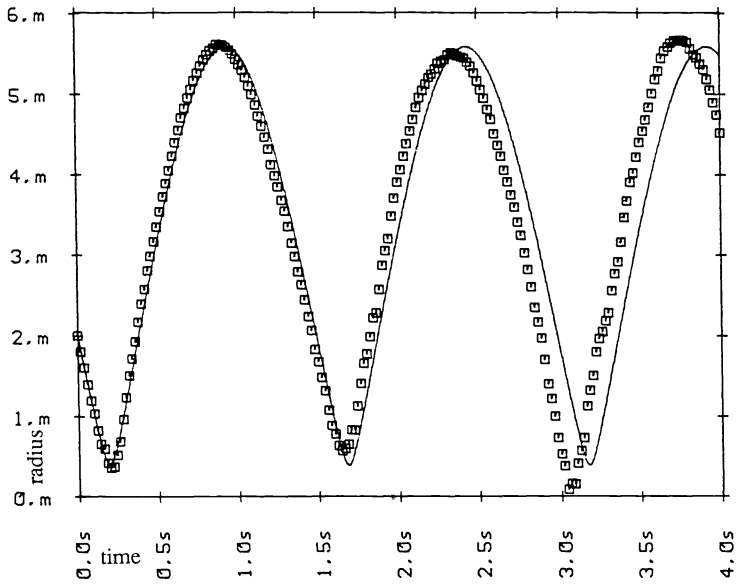


FIG. 7(a). *Glimm's scheme with sequence* $a_{n,j}$, *evolution of the radius* R *for* $\Delta s = 2\pi/200$.

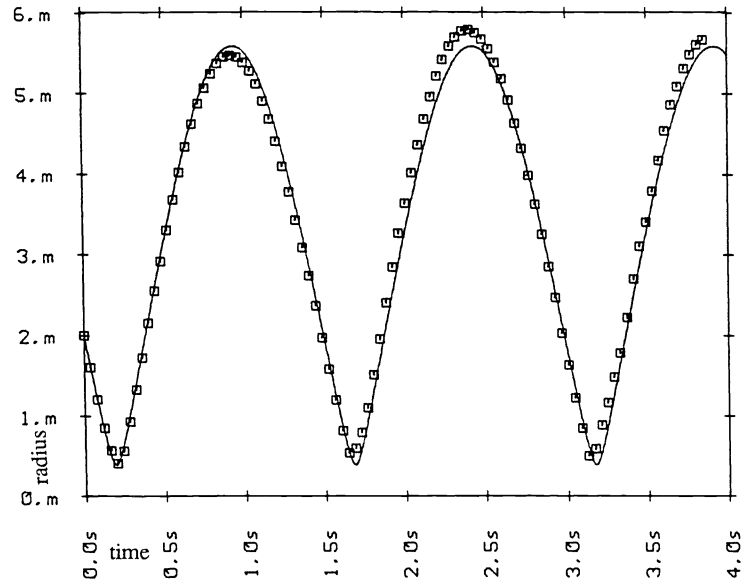


FIG. 7(b). *Glimm's scheme with sequence* $a_{n,j}$, *evolution of the radius* R *for* $\Delta s = 2\pi/500$.

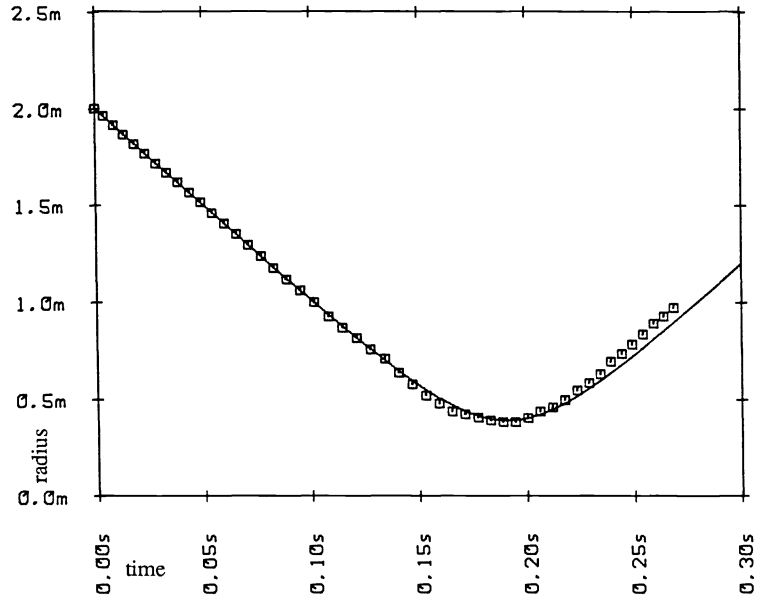


FIG. 7(c). *Glimm's scheme with sequence* $a_{n,j}$, *evolution of the radius* R *for* $\Delta s = 2\pi/2{,}000$.

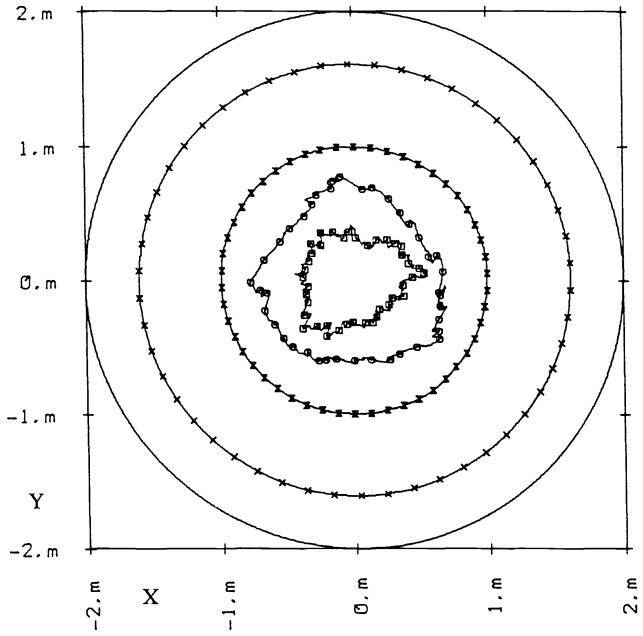


FIG. 7(d). *Case* B, *shape of the string for* $\Delta s = 2\pi/500$, *at time* t=: $-$ $t = 0$ *s*; $\times$ $t = 0.03932$ *s*; ⧖ $t = 0.08582$ *s*; ⊙ $t = 0.1$ *s*; ⊡ $t = 0.13932$ *s*.

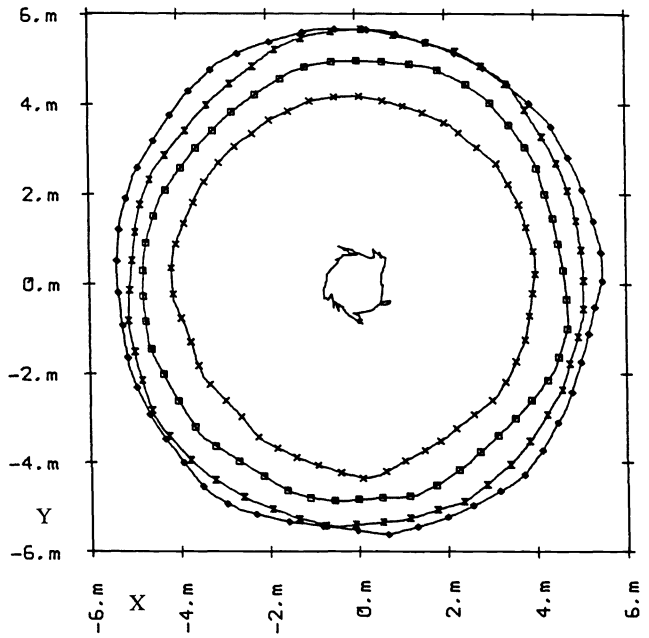


FIG 7(e). *Case* B, *shape of the string for* $\Delta s = 2\pi/500$, *at time* $t =$: $-$ $t = 0.28$ *s*; $\times$ $t = 0.6$ *s*; $\boxdot$ $t = 0.7$ *s*; $\boxtimes$ $t = 0.8$ *s*; $\odot$ $t = 0.9$ *s*.

Thus we have presented, in Figs. 6(a), 6(b), 7(a), 7(b), and 7(c), the evolution of this radius R (represented with a square), with the value of the exact solution (solid line) as reference.

(3) *Case* A.

$$\omega = 1, \quad \rho_0 = 2, \quad \rho_0' = -2, \quad \varphi_0 = 0, \quad \varphi_0' = 1.$$

The values of the radius oscillate periodically between $R_{\min} = 1.231422$ and $R_{\max} = 2.645$; the exact solution remains always in the hyperbolic domain.

The numerical solution is very poor for $\Delta s = 2\pi/50$ and $\Delta s = 2\pi/100$, but it becomes slowly better for $\Delta s = 2\pi/200$; for $\Delta s = 2\pi/500$ it is a good approximation very close to the exact solution. The cost of the computation is very high here because we have to solve as many RP as mesh nodes; but for general initial data the cost will be the same for the two schemes. The results are better when Δs vanishes.

(4) *Case* B.

$$\omega = 1, \quad \rho_0 = 2, \quad \rho_0' = -10, \quad \varphi_0 = 0, \quad \varphi_0' = 1.$$

The values of the radius oscillate periodically between $R_{\min} = 0.388384$ and $R_{\max} = 5.582529$; the solution leaves the hyperbolic domain when R is less than 1.

Now the numerical solution can leave the hyperbolic domain. With 50 and 100 grid points, the approximation is very poor; with 200 grid points, it becomes better; and with 500 grid points, it is a satisfying approximation.

This solution is much better than those obtained with a sequence a_n. We remark (see Fig. 7(d)) that the string departs from a circular shape the first time it leaves the hyperbolic domain but the string recovers this circular shape near $R = R_{\max}$ (see Fig. 7(e)). The accurate behavior of the approximation for 2,000 grid points ($\Delta s = 2\pi/2{,}000$), near $R = R_{\min}$, is given on Fig. 7(c).

7. Discussion. Glimm's scheme with a sequence a_n, does not allow the computation of a solution satisfying the system (2.1) when this solution leaves the hyperbolic domain; thus some conservation laws of (2.1) are violated.

To determine which of them are not satisfied:
The kinematic equations

$$\frac{\partial u_1}{\partial t}-\frac{\partial u_3}{\partial s}=0, \qquad \frac{\partial u_2}{\partial t}-\frac{\partial u_4}{\partial s}=0; \quad \text{or}$$

The dynamic equations,

$$\frac{\partial u_3}{\partial t}-\frac{\partial}{\partial s}(S(r)u_1)=0, \qquad \frac{\partial u_4}{\partial t}-\frac{\partial}{\partial s}(S(r)u_2)=0$$

we proceed as follows. Let us write (2.1) in the canonical way:

(7.1) $U_t+f(U)_s=0$ with $U=(u_1, u_2, u_3, u_4)$ and $f(U)=(u_3, u_4, S(r)u_1, S(r)u_2)$,

and consider, for each n

(7.2)
$$U_t+f(U)_s=0$$
$$(s, t)\in[0, 2\pi[\times[t^n, t^{n+1}[$$
$$\text{with } U(s, t^n)=U_j^n \text{ if } s-j\Delta s\in[-\Delta s/2, \Delta s/2].$$

The exact solution of (7.2), denoted by U, satisfies

$$0=\int_0^{2\pi}\int_{t^n}^{t^{n+1}}(U_t+f(U)_s)\,ds\,dt=\int_0^{2\pi}(U(s, t^{n+1})-U(s, t^n))\,ds.$$

Let $U_{\Delta s}$ be the approximate solution for $t=t^n$; then we have

$$0=\int_0^{2\pi}\int_{t^n}^{t^{n+1}}((U_{\Delta s})_t+f(U_{\Delta s})_s)\,ds\,dt=\int_0^{2\pi}(U_{\Delta s}(s, t^{n+1})-U_{\Delta s}(s, t^n))\,ds$$

with $U_{\Delta s}(s, t^n)=U(s, t^n)=U_j^n$ for $s-j\Delta s\in[-\Delta s/2, \Delta s/2]$, and thus

(7.3) $$\left|\int_0^{2\pi}\int_{t^n}^{t^{n+1}}((U_{\Delta s})_t+f(U_{\Delta s})_s)\,ds\,dt\right|\leqq\int_0^{2\pi}|(U_{\Delta s}(s, t^{n+1})-\bar{U}(s, t^{n+1}))|\,ds.$$

It is too expensive to compute

(7.4) $$\int_0^{2\pi}|(U_{\Delta s}(s, t^{n+1})-U(s, t^{n+1}))|\,ds \quad \text{for each } n;$$

however, we can obtain an estimate of this quantity with

(7.5)
$$\int_0^{2\pi}|(U_{\Delta s}(s, t^{n+1})-\tilde{U}(s, t^{n+1}))|\,ds=\mathrm{ERR}$$
$$\text{where } \tilde{U}(s, t^{n+1})=1/\Delta s\int_{(j-1/2)\Delta s}^{(j+1/2)\Delta s}U(s, t^{n+1})\,ds \quad \text{for } s-j\Delta s\in[-\Delta s/2, \Delta s/2].$$

We shall observe that ERR is no more than an estimation of (7.4), and can by no means be used as an upper bound of (7.4).

7.1. Evolution of ERR for a sequence a_n ($\Delta s=2\pi/500$). We present the evolution of ERR for the kinematic equations on Figs. 8(a) and 8(b), and for the dynamic equations on Figs. 8(c) and 8(d).

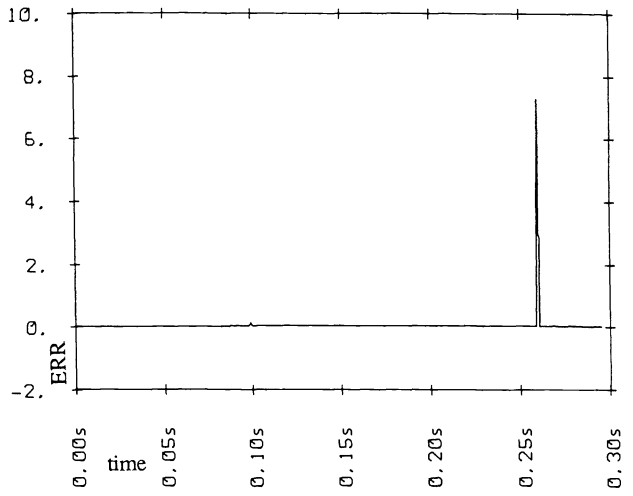


FIG 8(a). *Evolution of* ERR *for the sequence* a_n, *Case* B, $\Delta s = 2\pi/500$, *for the conservation law*: $\partial u_1/\partial t - \partial u_3/\partial s = 0$.

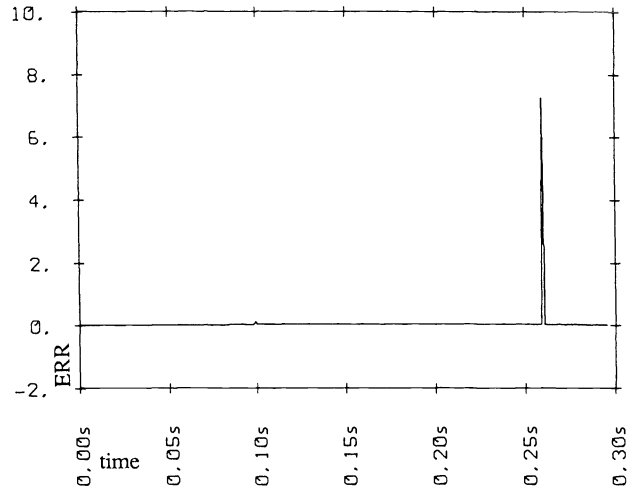


FIG. 8(b). *Evolution of* ERR *for the sequence* a_n, *Case* B, $\Delta s = 2\pi/500$ *for the conservation law*: $\partial u_2/\partial t - \partial u_4/\partial s = 0$.

We can see that the most poorly approximated conservation laws are the kinematic ones (this was predicted by the results of [PS]). The gap in Figs. 8(a) and 8(b) is 100 times bigger than in Figs. 8(c) and 8(d); this gap just arises when the exact solution enters the hyperbolic domain after having left it. The two little gaps in Figs. 8(c) and 8(d) result from the fact that the numerical solution waits on the frontier of the hyperbolic domain because it cannot leave it; the estimation of the error ERR is then zero because the computed solution does not move at all.

7.2. Evolution of ERR for a sequence $a_{n,j}$ ($\Delta s = 2\pi/500$). We present the evolution of the quantity ERR for the kinematic equations in Figs. 9(a) and 9(b), and for the dynamic equations in Figs. 9(c) and 9(d).

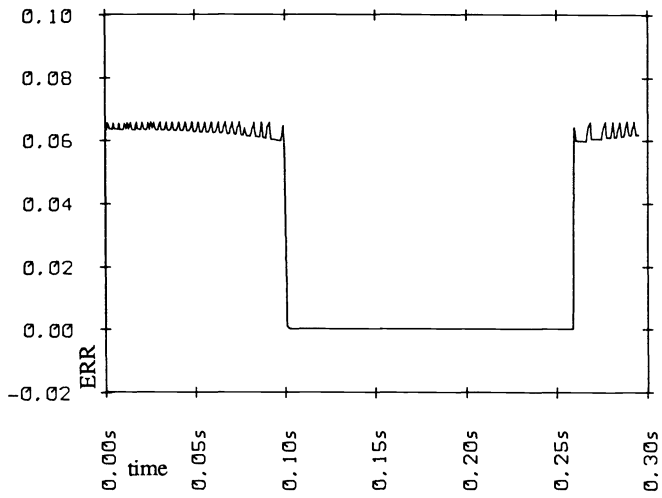


FIG. 8(c). *Evolution of* ERR *for the sequence* a_n, *Case* B, $\Delta s = 2\pi/500$ *for the conservation law*: $\partial u_3/\partial t - \partial/\partial s(S(r)u_1) = 0$.

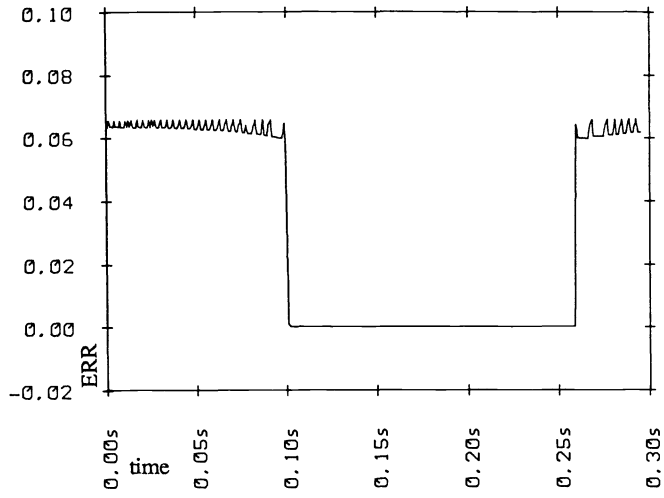


FIG. 8(d). *Evolution of* ERR *for the sequence* a_n, *Case* B, $\Delta s = 2\pi/500$ *for the conservation law*: $\partial u_4/\partial t - \partial/\partial s(S(r)u_2) = 0$.

The estimation of the numerical error ERR for the kinematic equations grows slowly and exhibits a small gap when the computed solution just leaves the hyperbolic domain ($t \cong 0.1s$); for the dynamic conservation laws, we can observe a rapid increase of ERR followed by a slow decrease. The conservation laws are better approximated than in the previous case.

8. Conclusion. Naturally, the exact solution of the problem becomes unstable when $R < 1$, because the equations are then partially elliptic. This is a well-known phenomenon (Hadamard) for a Cauchy problem. This solution is highly dependent on the small variations of the initial data (if we assume a solution exists for general

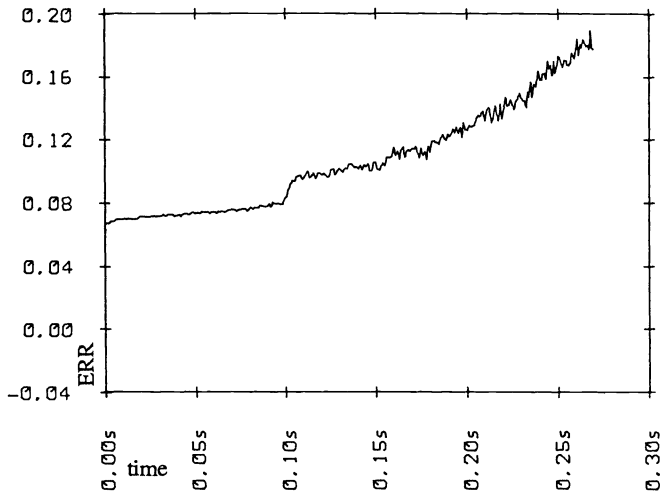


FIG. 9(a). *Evolution of* ERR *for the sequence* $a_{n,j}$, *Case* B, $\Delta s = 2\pi/500$ *for the conservation law*: $\partial u_1/\partial t - \partial u_3/\partial s = 0$.

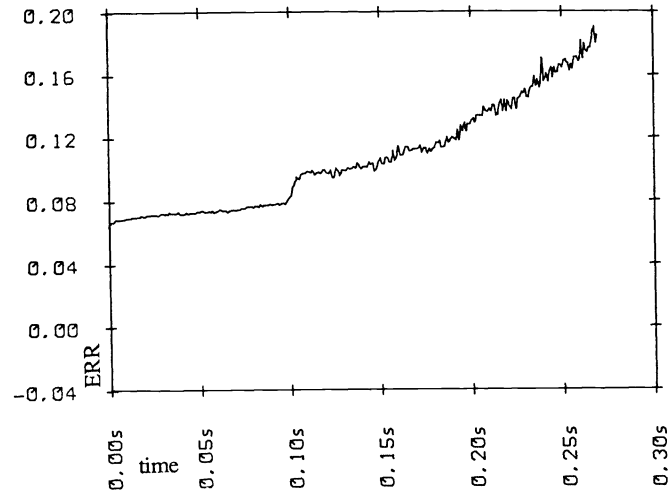


FIG. 9(b). *Evolution of* ERR *for the sequence* $a_{n,j}$, *Case* B, $\Delta s = 2\pi/500$ *for the conservation law*: $\partial u_2/\partial t - \partial u_4/\partial s = 0$.

initial data), at least as long as $R < 1$. In the same way the numerical solution is highly dependent on the random sequence a_n or $a_{n,j}$ or on the choice of Δs. This is immediate for a sequence a_n because it induces a random lifetime (that depends on a_n) during which $R = 1$. For a sequence $a_{n,j}$ the reconstructed shape of the string is random.

The important question is whether it is possible to compute a good approximation of the string when it recovers a stable state (after it returns to the hyperbolic domain). This is not the case of a sequence a_n, because of the random waiting lifetime. But Glimm's scheme with a sequence $a_{n,j}$ is sufficiently robust to allow the computations we have done. The cost for this robustness is well known: less precision than with the classical high-order schemes when the solution is regular and stable.

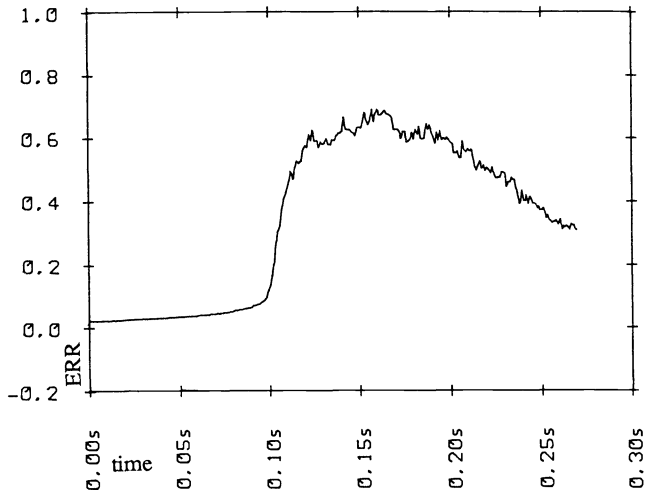


FIG. 9(c). *Evolution of* ERR *for the sequence* $a_{n,j}$, *Case* B, $\Delta s = 2\pi/500$ *for the conservation law*: $\partial u_3/\partial t - \partial/\partial s(S(r)u_1) = 0$.

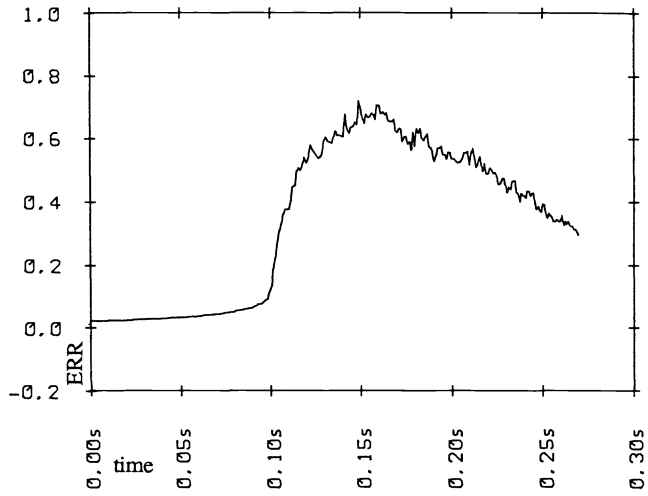


FIG. 9(d). *Evolution of* ERR *for the sequence* $a_{n,j}$, *Case* B, $\Delta s = 2\pi/500$ *for the conservation law*: $\partial u_4/\partial t - \partial/\partial s(S(r)u_2) = 0$.

Appendix. We know $u_1 = \partial w^1/\partial s$ and $u_2 = \partial w^2/\partial s$; to compute w^1 and w^2 from $\partial w^1/\partial s$ and $\partial w^2/\partial s$, at each timestep t^n, we minimize in the least-square sense:

$$\sum_{j=1}^{N} \left(w_{j+1}^{\#} - w_j^{\#} - \Delta s \frac{\partial w_{j+1/2}^{\#}}{\partial s} \right)^2 \quad \text{under the condition}$$

$$\sum_{k=1}^{N} w_k^{\#} = 0, \qquad \# = 1 \text{ or } 2.$$

We obtain the system:

(A1) $$Mw^{\#} = \Delta s b^{\#},$$

(A2) $$\sum_{k=1}^{N} w_k^{\#} = 0$$

where

$$M = \begin{bmatrix} 2 & -1 & 0 & \cdot & 0 & -1 \\ -1 & 2 & -1 & \cdot & 0 & 0 \\ 0 & -1 & 2 & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 2 & -1 \\ -1 & 0 & 0 & \cdot & -1 & 2 \end{bmatrix} \quad \text{and} \quad b_k^{\#} = \frac{\partial w_{k/2-1}^{\#}}{\partial s} - \frac{\partial w_{k/2+1}^{\#}}{\partial s}.$$

Equation (A1) has an infinite number of solutions that are defined up to an additive constant; but (A1)-(A2) has a unique solution.

By taking $w_N^{\#} = \alpha$, we obtain

(A3) $$M' w'^{\#} = \Delta s b'^{\#},$$

(A4) $$\sum_{k=1}^{N} w_k'^{\#} = 0 \quad \text{with } M' = \begin{bmatrix} 2 & -1 & \cdot & 0 & 0 \\ -1 & 2 & \cdot & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 2 & -1 \\ 0 & 0 & \cdot & -1 & 2 \end{bmatrix}$$

(A5) $$w_N'^{\#} = \alpha, \quad \text{and } b_1'^{\#} = b_1^{\#} + \alpha,\ b_{N-1}'^{\#} = b_{N-1}^{\#} + \alpha, \text{ and } b_k'^{\#} = b_k^{\#}.$$

When the system (A3) is solved by Cholesky decomposition, we have $\sum_{k=1}^{N-1} w_k'^{\#} + \alpha = \beta$. Then we take $C = \beta / N$, and the solution of (A1)-(A2) is given by $w_k^{\#} = w_k'^{\#} - C$.

REFERENCES

[CRS] C. CARASSO, M. RASCLE, AND D. SERRE, *Etude d'un modèle hyperbolique en dynamique des câbles*, Math. Model. Numer. Anal., 19 (1985), pp. 573-599.

[Gi] H. GILQUIN, *Une famille de schémas numériques* T.V.D. *pour les lois de conservation hyperboliques*, Math. Model. Numer. Anal., 20 (1986), pp. 429-460.

[Gl] J. GLIMM, *Solutions in the large for nonlinear hyperbolic systems of equations*, Comm. Pure Appl. Math., 18 (1965), pp. 697-715.

[Go] S. K. GODUNOV, *A finite difference method for the numerical computation of discontinuous solutions of the equations of fluid dynamics*, Mat. Sb. (N.S.), 47 (1959), pp. 271-290.

[HS] R. HAGAN AND M. SLEMROD, *The viscosity-capillarity admissibility criterion for shocks and phase transitions*, Arch. Rational Mech. Anal., 83 (1983), pp. 333-361.

[KK] B. L. KEYFITZ AND H. C. KRANZER, *A system of non-strictly hyperbolic conservation laws arising in elasticity theory*, Arch. Rational Mech. Anal., 72 (1980), pp. 219-241.

[Liu] T. P. LIU, *The deterministic version of the Glimm scheme*, Comm. Math. Phys., 57 (1977), pp. 135-148.

[PS] R. L. PEGO AND D. SERRE, *Instabilities in Glimm's scheme for two systems of mixed type.* Preprint.

[Sh] M. SHEARER, *The Riemann problem for a class of conservation laws of mixed type*, J. Differential Equations, 46 (1982), pp. 426-443.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 154-174, January 1989
© 1989 Society for Industrial and Applied Mathematics
012

STABILITY OF A RUNGE-KUTTA METHOD FOR THE EULER EQUATIONS ON A SUBSTRUCTURED DOMAIN*

KURT OTTO† AND MICHAEL THUNÉ†‡

Abstract. A Runge-Kutta-type time marching scheme for the Euler equations is considered. The scheme was originally suggested by Gary and has been used by Rizzi and Eriksson in airflow simulations. The stability of the method is analyzed, assuming that the computational domain is substructured, and several stable sets of exterior boundary conditions and overlapping conditions are designed. In performing the analyses, use is made of the software tool Ibstab. Several sets of numerical experiments support the Ibstab results.

Key words. stability, overlapping grids, Runge-Kutta method, Euler equations

AMS(MOS) subject classification. 65M10

1. Introduction. Currently, an important field of research in computational physics is airflow simulations based on Euler's equations. Successful calculations have been performed, e.g., using modified Runge-Kutta methods for timestepping. The Runge-Kutta approach leads to a partial difference operator that is wide in the space direction. This raises the important question of how to choose numerical boundary conditions that preserve stability.

Another difficulty in the simulation of airflow around aeroplanes is the complex geometry of the computational domain. A way to solve this is to substructure the domain and use overlapping computational grids connected to each other by some overlapping conditions (such as interpolation between the grids). Such a substructuring is also motivated by the desire to use a finer grid, e.g., near shock waves. The overlaps may also affect the stability of the solution procedure, and thus must be taken into account when performing a stability analysis. Results on stability in connection with overlapping grids have been reported by, e.g., Ciment [2], Starius [6], Berger [1], and Trefethen [9].

Our aim in this report is to study a modified Runge-Kutta method formulated by Gary [3]. It has been used by Rizzi and Eriksson [5] in airflow simulations. We will analyze the stability of the method for a substructured domain, implying that both exterior boundaries and overlaps must be considered. Stable boundary and overlapping conditions will be designed.

The analysis is much simplified by the theory in [8], which says that if each exterior boundary and each overlap is stable *separately*, then the composed approximation is also stable. The condition for this to hold is that if there is more than one overlap region, then the distances between the different overlap regions should not depend on the mesh sizes. Thus, our stability investigation can be divided into subproblems of standard type. The appropriate stability theory for our purpose is the normal mode theory as presented by Gustafsson, Kreiss, and Sundström [4] (henceforth called GKS). We assume the reader to be familiar with the GKS theory. The reader will then know that a GKS analysis can lead to rather painstaking algebraic manipulations already for small partial differential equation (pde) problems and low-order approximations. For many real-sized problems, performing a GKS analysis by hand is impossible. This is the case with the problems we are considering. So in the analysis we make use of

* Received by the editors October 7, 1987; accepted for publication (in revised form) May 12, 1988.
† Department of Scientific Computing, Uppsala University, S-752 23, Uppsala, Sweden.
‡ The work of this author was supported by the Swedish Natural Science Research Council.

the software tool Ibstab [7], which contains a numerical algorithm specially tailored for automatic GKS stability investigation.

The disposition of the report is as follows. In § 2 we present the basic approximation studied. In this context, we discuss how to simplify the analysis, and also make some general comments on the design of boundary and overlapping conditions. Then in § 3 we analyze exterior boundary conditions (at both inflow and outflow) and in § 4 overlapping conditions. Finally, the analysis is corroborated by numerical experiments in § 5.

2. The approximation.

2.1. The basic approximation. The pde problem we consider is the linearized Euler equation in one space dimension. These are

$$Y_t + A(x, t)\, Y_x = 0, \tag{1}$$

where $Y = (\rho \;\; v \;\; p)^T$ and

$$A(x, t) = \begin{pmatrix} \tilde{v} & \tilde{\rho} & 0 \\ 0 & \tilde{v} & \tilde{\rho}^{-1} \\ 0 & \gamma\tilde{p} & \tilde{v} \end{pmatrix}.$$

The density is denoted by ρ, the velocity by v, and the pressure by p. The tilded variables in $A(x, t)$ represent the given state around which the linearization has been made. The symbol γ denotes a thermodynamic entity closely related to the local speed of sound. The matrix $A(x, t)$ is variable. However, for the purpose of GKS stability analysis it is sufficient to consider constant coefficients (assuming that $A(x, t)$ is Lipschitz continuous). Thus, we consider the case $A(x, t) \equiv A$, where A is the matrix obtained by freezing $A(x, t)$ at the boundary or overlap under investigation.

The computational domain is depicted in Fig. 1. We assume that the domain is decomposed into two parts with separate grids, which here are denoted by G_1 and G_2. The mesh sizes are h_1 and h_2, respectively. x_{N_1,G_1} is the rightmost point on G_1, to the left of x_{0,G_2}. x_{-N_2,G_2} is defined analogously. We take N_1 and N_2 to be constants independent of h_1 and h_2. In [8] it is demonstrated that this is a more difficult case than assuming a constant overlapping *width* (in which case we have $N_1, N_2 \to \infty$ as $h_1, h_2 \to 0$).

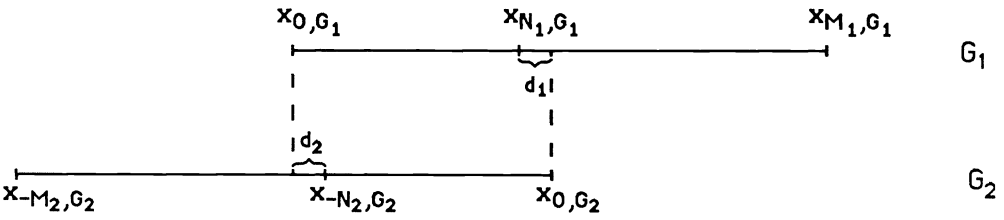


FIG. 1

In general, we would possibly have several subdomains, and thus several overlaps. However, as long as the distances between the overlaps are independent of h_1 and h_2, the stability of each interface separately is sufficient to ensure overall stability. This means that we do not lose in generality by considering only *one* overlap. The advantage of the situation depicted in Fig. 1 is that it is general enough to model the total complexity of one-dimensional problems on substructured domains, and still it is simple enough to describe without introducing too much notation.

We now turn to the basic approximation of (1). Let us then assume that we have a semidiscretization of the Euler equations:

$$\frac{d}{dt} U_\nu(t) = QU_\nu(t),$$

where $U_\nu(t) \approx Y(x_\nu, t)$ and Q is a linear difference operator with $QU_\nu(t) \approx -AY_x(x_\nu, t)$. To obtain a fully discrete approximation of (1), we apply a Runge–Kutta-type time-stepping procedure to the semidiscrete equations. We then get the multistage method

$$\begin{aligned} \check{U}_\nu &= U_\nu^n + kQU_\nu^n, \\ \check{\check{U}}_\nu &= U_\nu^n + \frac{k}{2} Q(U_\nu^n + \check{U}_\nu), \\ U_\nu^{n+1} &= U_\nu^n + \frac{k}{2} Q(U_\nu^n + \check{\check{U}}_\nu), \end{aligned} \tag{2}$$

where k denotes the stepsize in time. This method was formulated by Gary [3] and will henceforth be called *Gary's scheme.*[1] It can also be written in the one-stage form:

$$U_\nu^{n+1} = \left(I + kQ + \frac{k^2}{2} Q^2 + \frac{k^3}{4} Q^3 \right) U_\nu^n. \tag{3}$$

We will use both formulations, allowing the circumstances to decide which is the most suitable. By comparing (3) to the Taylor expansion of $U_\nu(t_{n+1})$ around t_n, we see that the method is of second order in time. The reason for not using a correct third-order Runge–Kutta formula is that by changing the coefficients slightly we get a method with better stability properties (see [5] for a more detailed discussion of this). Thus, by using (3) instead of a standard Runge–Kutta approximation we lose accuracy but gain stability.

In their code, Rizzi and Eriksson [5] use a centered finite volume approximation in space. For our simplified one-dimensional model problem we represent this by standard central differencing, and thus use

$$Q \equiv -AD_0. \tag{4}$$

Henceforth the approximation (2)/(3) with Q defined by (4) will be denoted BA (Basic Approximation).

2.2. Stability for the Cauchy problem. We now leave the substructured domain problem for a moment and instead consider the Cauchy problem for (1) on *one* grid with uniform mesh size h. The stability limit obtained when (3) is applied to this problem is of course a *necessary* stability limit for (3) on the substructured domain.

If we see (3) as an ordinary differential equation solver, then its stability region is given by

$$\left| 1 + w + \frac{w^2}{2} + \frac{w^3}{4} \right| \leqq 1.$$

An investigation of this inequality shows that the stability region is kidney-shaped, which is typical for Runge–Kutta methods. Especially, we find that the part of the imaginary axis included in the stability region is given by (cf. [5])

$$|\Im m(w)| \leqq 2. \tag{5}$$

[1] In fact, Gary formulated a p-stage formula for iterative solution of the implicit equations generated by the Crank–Nicholson approximation. The method (2) above is the particular case $p = 3$.

Next, consider BA applied to the Cauchy problem. When we use (5), the von Neumann condition leads to the stability condition

$$\frac{k}{2h}\rho(A) \leqq 1, \tag{6}$$

where $\rho(A)$ is the spectral radius of A.

2.3. General formulation of boundary and overlapping conditions. We now return to discussing the approximation BA on a substructured domain as in Fig. 1. BA will evidently require three points to the left and equally many points to the right. Thus, it cannot be applied at points close to the exterior boundaries or the overlaps. We need to specify numerical boundary and overlapping conditions.

Let us denote the approximation on the grid G_1 by U and the approximation on G_2 by V. The boundary conditions can then formally be written as follows:

$$B_\nu^{(1)} U_\nu^{n+1} = 0, \qquad \nu = M_1 - 2, M_1 - 1, M_1, \tag{7a}$$

$$B_\nu^{(2)} V_\nu^{n+1} = 0, \qquad \nu = -M_2 + 2, -M_2 + 1, -M_2. \tag{7b}$$

Here, $B_\nu^{(1)}$ and $B_\nu^{(2)}$ are difference operators in space *and* time that only need points up to the boundary. We remark that because conditions (7) are homogeneous this does not mean there is any loss of generality. Our results will be valid for nonhomogeneous boundary data as well.

Similarly, the overlapping conditions can be written as follows:

$$C_\nu^{(1)} U_\nu^{n+1} = L_\nu^{(1)} V_{-N_2}^{n+1}, \qquad \nu = 0, 1, 2, \tag{8a}$$

$$C_\nu^{(2)} V_\nu^{n+1} = L_\nu^{(2)} U_{N_1}^{n+1}, \qquad \nu = 0, 1, 2, \tag{8b}$$

where $C_\nu^{(1)}$ and $C_\nu^{(2)}$ are differences in space and time. $L_0^{(1)}$ and $L_0^{(2)}$ are interpolation operators, whereas $L_\nu^{(m)} \equiv 0$ for $\nu = 1, 2$. The complete approximation BA, (7), (8) will henceforth be denoted SDA (Substructured Domain Approximation).

Now, (1) is a hyperbolic system. Thus, there is a matrix T that diagonalizes A. Furthermore, the coefficients in BA are polynomials in A. Let us now assume the following:

(9a) The coefficients in $C_\nu^{(m)}$ are polynomials in A.

(9b) The coefficients in $B_\nu^{(1)}$, $\nu < M_1$ and $B_\nu^{(2)}$, $\nu > -M_2$ are polynomials in A.

It would *not* be reasonable to make the assumption (9b) for $\nu = M_1$ and $\nu = -M_2$, respectively. These values of ν correspond to the exterior boundaries, where some components will be given by the boundary conditions of the pde problem. At these points we make the assumption:

(9c) At the exterior boundaries the boundary conditions are given in terms of the characteristic variables, with uncoupled outflow conditions. The inflow conditions do not couple the inflow variables, but may couple inflow variables to outflow variables.

Finally, we assume

$$L_0^{(m)} = \sum_{j=-j_1}^{j_2} c_j^{(m)} E(h_m)^j, \tag{9d}$$

where $c_j^{(m)}$ are scalar coefficients and $E(h_m)$ is the translation operator in space with stepsize h_m, i.e., $E(h_m)w(x) = w(x + h_m)$. Under these assumptions SDA can obviously be transformed so that the stability analysis can be performed on scalar model problems.

2.4. Design considerations for boundary and overlapping conditions. We now focus attention on the design of boundary conditions and overlapping conditions. One aspect of this is choosing conditions that facilitate the stability analysis. From this point of view it is convenient to ensure that (9) is fulfilled. From now on we will refer to such conditions as being *quasi-scalar.*

Another aspect is ease of implementation. There are basically two possibilities. The first is the *one-level approach* represented by the formulations (7) and (8). (This corresponds to the formulation (3) of BA.) When this approach is adopted, the question of implementation will not affect the choice of boundary conditions in any essential way.

The second possibility is the *multilevel approach* corresponding to the formulation (2) of BA. When employing this approach it is convenient to give boundary and overlapping conditions on each level separately. As an example, in the multilevel approach the conditions (7a) should be given as follows:

$$\check{U}_{M_1} = \check{b}(U^n, \check{U}), \tag{10a}$$

$$\check{\check{U}}_{M_1} = \check{\check{b}}(U^n, \check{U}, \check{\check{U}}) \tag{10b}$$

$$U_{M_1}^{n+1} = b(U^n, \check{U}, \check{\check{U}}, U^{n+1}), \tag{10c}$$

where the functions $\check{b}$, $\check{\check{b}}$, and b are built up by difference operators.

The overlapping conditions that we are going to analyze are the following:

$$\check{U}_0 = \check{\check{U}}_0 = U_0^{n+1} = L_0^{(1)} V_{-N_2}^{n+1}, \qquad \check{V}_0 = \check{\check{V}}_0 = V_0^{n+1} = L_0^{(2)} U_{N_1}^{n+1}.$$

If the interpolation formulas only involve values at inner points, then these conditions can be used with the multilevel approach, provided that the computations are performed in the appropriate order. As this is the crucial part of the overlap scheme, we describe it in some detail. First, compute the solution on *all* three levels at *inner* points, i.e., $\check{U}_\nu$, $\nu = 1, 2, \cdots$, $\check{\check{U}}_\nu$, $\nu = 2, 3, \cdots$, and U_ν^{n+1}, $\nu = 3, 4, \cdots$ (similarly for the different levels of V). Then treat the overlapping conditions in a second pass. As we assume the interpolation formulas to involve only values at inner points, we can now compute $L_0^{(1)} V_{-N_2}^{n+1}$ and $L_0^{(2)} U_{N_1}^{n+1}$. This gives $\check{U}_0$, $\check{\check{U}}_0$, U_0^{n+1}, and subsequently $\check{\check{U}}_1$, U_1^{n+1}, and finally U_2^{n+1} (similarly for the different levels of V).

We remark that the strategy of computing the solution at inner points on all levels in a preparatory pass also allows for a more general form of the multilevel boundary conditions (10). The solution at boundary points on lower levels may depend on values at higher-level inner points. The conditions that will be analyzed below are all quasi-scalar. Most of them are of the multilevel type, but we also analyze some one-level boundary conditions.

2.5. Model problems. We have reduced the stability investigation to an analysis of scalar problems, using the assumptions (9). As a further simplification we will divide the analysis into two separate cases, one for the boundary conditions and one for the overlapping conditions. This is possible due to [8].

Our scalar model equation will be

$$u_t = \alpha u_x, \qquad \alpha \in \mathfrak{R}. \tag{11}$$

For convenience, we write BA when applied to (11).

The multistage form (2) will give

$$\check{u}_\nu = u_\nu^n + k\alpha D_0 u_\nu^n,$$

$$\check{\check{u}}_\nu = u_\nu^n + \frac{k}{2}\alpha D_0(u_\nu^n + \check{u}_\nu), \tag{12a}$$

$$u_\nu^{n+1} = u_\nu^n + \frac{k}{2}\alpha D_0(u_\nu^n + \breve{u}_\nu).$$

The one-stage form (3) will give

$$u_\nu^{n+1} = \left(I + k\alpha D_0 + \frac{k^2}{2}\alpha^2 D_0^2 + \frac{k^3}{4}\alpha^3 D_0^3\right) u_\nu^n. \tag{12b}$$

For the analysis of boundary conditions we now define *Model Problem* 1, which is BA applied to (11) on the domain $0 \leqq x < \infty$, $t \geqq 0$ using one single grid. We demand that $u \in \mathscr{L}_2(0, \infty)$. By allowing both negative and positive α-values, we can make this model problem cover both the inflow and outflow boundary case.

The analysis of overlapping conditions will be performed on *Model Problem* 2, which is BA applied to the Cauchy problem for (11). We demand that $u \in \mathscr{L}_2(-\infty, \infty)$. The overlap is depicted in Fig. 1. The grid functions on G_1 and G_2 are named u and v, respectively. In this case the sign of α does not matter. We will assume $\alpha > 0$.

According to our previous discussion, stability of both model problems will be sufficient for stability of SDA.

3. Analysis of boundary conditions.

3.1. The characteristic equation. We begin by analyzing the stability of Model Problem 1 in the outflow case, $\alpha > 0$. Here we employ the normal mode theory developed by Gustafsson, Kreiss, and Sundström [4]. Even though prior knowledge of the theory is assumed, rather extensive comments are given to enhance comprehension.

One prerequisite for stability of the initial boundary value problem is that the approximation of the Cauchy problem is stable. This condition is, according to (6), satisfied both at outflow and inflow when

$$0 < |\lambda| \leqq 1, \qquad \lambda = \frac{\alpha k}{2h}. \tag{13}$$

To simplify the algebra we define the difference operator $\Delta_0 = 2hD_0$. With λ defined by (13) the operator $k\alpha D_0$ in Gary's scheme is then the same as $\lambda\Delta_0$.

Insertion of the normal mode ansatz $u_\nu^n = z^n w_\nu$ into the difference scheme (12b) yields the resolvent equation:

$$(z-1)w_\nu = \left(\lambda\Delta_0 + \frac{\lambda^2}{2}\Delta_0^2 + \frac{\lambda^3}{4}\Delta_0^3\right) w_\nu.$$

By assuming $w_\nu = \kappa^\nu$ in the resolvent equation we get *the characteristic equation*:

$$z = 1 + \mu + \frac{\mu^2}{2} + \frac{\mu^3}{4}, \qquad \mu = \lambda(\kappa - \kappa^{-1}) \tag{14a}$$

or

$$\begin{aligned} P(z, \kappa) \equiv{}& (1-z)\kappa^3 + \lambda(\kappa^4 - \kappa^2) \\ &+ \frac{\lambda^2}{2}(\kappa^5 - 2\kappa^3 + \kappa) + \frac{\lambda^3}{4}(\kappa^6 - 3\kappa^4 + 3\kappa^2 - 1) = 0. \end{aligned} \tag{14b}$$

For every fixed value of z equation (14b) is a polynomial of degree 6, the roots of which determine the solution of the resolvent equation. The type of solution depends on whether or not the polynomial has multiple roots. The required multiplicity conditions are

$$P(z, \kappa) = 0, \qquad \frac{\partial P(z, \kappa)}{\partial \kappa} = 0.$$

By (14b) and after elimination of z, these conditions can be formulated as follows:

$$(\kappa^2+1)\left(\left(\kappa^2-1+\frac{2\kappa}{3\lambda}\right)^2+\frac{8\kappa^2}{9\lambda^2}\right)=0.$$

From this equation the multiple roots κ can be determined.

By using (14a) we then get the corresponding z-values

$$z=1-2\lambda^2\pm 2i\lambda(1-\lambda^2), \qquad z=\frac{13\pm i8\sqrt{2}}{27}$$

for which we have

$$|z|^2=1-4\lambda^4(1-\lambda^2), \qquad |z|^2=\tfrac{11}{27}.$$

But (13) then ensures that $|z|\leqq 1$, thus implying that the characteristic equation has *no multiple roots* for $|z|>1$. In order to characterize the roots further, the set M_I is defined as follows:

$$\kappa\in M_I \Leftrightarrow \kappa \text{ is a root of the characteristic equation, } |\kappa|<1.$$

Then we have the following:

For every fixed z satisfying $|z|>1$, there are three roots belonging to M_I.

The boundary condition $u^n\in l_2$ is by the normal mode ansatz transformed into $w\in l_2$. When $|z|>1$, that condition will be fulfilled only by the following solution of the resolvent equation:

$$w_\nu=\sum_{p=1}^{3}\sigma_p\kappa_p^\nu, \tag{15}$$

where

$$\kappa_p\in M_I, \qquad \sigma_p\in\mathfrak{R}.$$

The unknown coefficients σ_p are to be determined by insertion into the boundary conditions at $x=0$.

3.2. Outflow boundary conditions.

3.2.1. Boundary condition 1. When Gary's scheme is applied to the outflow problem, complementary conditions are required at the gridpoints x_2, x_1, x_0. To design the numerical boundary conditions (b.c.) we use two different strategies. One way is to extrapolate, which is done for b.c. 3 and 4. The other method, used for b.c. 1 and 2, is to replace the central differencing in the scheme by a one-sided difference operator. These strategies both lead to quasi-scalar boundary conditions, since the operations involved can be expressed as polynomials in α.

We now define the one-sided difference operator $\Delta_+=hD_+$. To substitute D_0 with D_+ will then correspond to replacing Δ_0 by $2\Delta_+$. One way to choose the complementary conditions is to carry out that substitution in the one-stage formulation (12b) of Gary's scheme. We then obtain

$$u_2^{n+1}=u_2^n+\lambda\Delta_0u_2^n+\frac{\lambda^2}{2}\Delta_0^2u_2^n+\frac{\lambda^3}{2}\Delta_+\Delta_0^2u_2^n, \tag{16a}$$

$$u_1^{n+1}=u_1^n+\lambda\Delta_0u_1^n+\lambda^2\Delta_+\Delta_0u_1^n+\lambda^3\Delta_+^2\Delta_0u_1^n \tag{16b}$$

$$u_0^{n+1}=u_0^n+2\lambda\Delta_+u_0^n+2\lambda^2\Delta_+^2u_0^n+2\lambda^3\Delta_+^3u_0^n. \tag{16c}$$

By inserting the normal mode ansatz, including (15), into these boundary conditions, we obtain a system of linear equations:

$$\underbrace{\begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix}}_{E(z)} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \end{pmatrix} = 0$$

where the elements of the matrix are given by

$$e_{1p} = (1-z)\kappa_p^2 + \lambda(\kappa_p^3 - \kappa_p) + \frac{\lambda^2}{2}(\kappa_p^2 - 1)^2 + \frac{\lambda^3}{2}(\kappa_p - 1)(\kappa_p^2 - 1)^2, \tag{17a}$$

$$e_{2p} = (1-z)\kappa_p + \lambda(\kappa_p^2 - 1) + \lambda^2(\kappa_p - 1)(\kappa_p^2 - 1) + \lambda^3(\kappa_p - 1)^2(\kappa_p^2 - 1) \tag{17b}$$

$$e_{3p} = 1 - z + 2\lambda(\kappa_p - 1) + 2\lambda^2(\kappa_p - 1)^2 + 2\lambda^3(\kappa_p - 1)^3. \tag{17c}$$

The condition for stability is (cf. [4, Lemma 10.3]):

$$\det(E) \neq 0 \quad \forall |z| \geqq 1. \tag{18}$$

The elements of the matrix $E(z)$ are derived by the same algebraic technique for all the boundary conditions, and we do not state the matrix elements in the subsequent sections.

3.2.2. Boundary condition 2. In contrast to b.c. 1 we can design boundary conditions of the form (10), which means that the multilevel approach is applied. We now replace Δ_0 by $2\Delta_+$ in the multistage form (12a) of Gary's scheme:

$$u_2^{n+1} = u_2^n + \frac{\lambda}{2}\Delta_0(u_2^n + \breve{\breve{u}}_2), \tag{19a}$$

$$u_1^{n+1} = u_1^n + \frac{\lambda}{2}\Delta_0(u_1^n + \breve{\breve{u}}_1), \tag{19b}$$

$$u_0^{n+1} = u_0^n + \lambda\Delta_+(u_0^n + \breve{\breve{u}}_0) \tag{19c}$$

where

$$\breve{\breve{u}}_0 = u_0^n + \lambda\Delta_+(u_0^n + \breve{u}_0), \qquad \breve{u}_0 = u_0^n + 2\lambda\Delta_+ u_0^n.$$

3.2.3. Boundary condition 3. Yet another way to design complementary conditions would be to perform linear extrapolation. This can be done as follows:

$$u_2^{n+1} = u_2^n + \frac{\lambda}{2}\Delta_0(u_2^n + \breve{\breve{u}}_2), \tag{20a}$$

$$u_1^{n+1} = u_1^n + \frac{\lambda}{2}\Delta_0(u_1^n + \breve{\breve{u}}_1), \tag{20b}$$

$$u_0^{n+1} = 2u_1^{n+1} - u_2^{n+1} \tag{20c}$$

where

$$\breve{\breve{u}}_0 = \breve{u}_0 = u_0^{n+1}.$$

3.2.4. Boundary condition 4. The extrapolation strategy can also be modified to convenient boundary conditions for the multilevel approach, i.e., explicit application

of the multistage form of the scheme. In that case the 'tilded' variables have to be treated separately. One possibility is to perform linear extrapolation at *each* stage.

$$u_2^{n+1} = u_2^n + \frac{\lambda}{2}\Delta_0(u_2^n + \check{\check{v}}_2), \tag{21a}$$

$$u_1^{n+1} = u_1^n + \frac{\lambda}{2}\Delta_0(u_1^n + \check{\check{u}}_1), \tag{21b}$$

$$u_0^{n+1} = 2u_1^{n+1} - u_2^{n+1} \tag{21c}$$

where

$$\check{\check{u}}_0 = 2\check{\check{u}}_1 - \check{\check{u}}_2, \qquad \check{u}_0 = 2\check{u}_1 - \check{u}_2.$$

3.3. The algebraic stability equations.

3.3.1. Stability criteria. The characteristic equation (14b) and the determinant condition (18) for the relevant matrix constitute the algebraic equations, the solutions of which determine whether the numerical boundary conditions are stable or not. Only those values of λ that satisfy constraint (13) are considered. The stability conditions can be formulated as follows.

The approximation (12b) is GKS-stable$\Leftrightarrow$(*) *and* (**) hold

(*)

$$P(z, \kappa) = 0, \qquad \det(E) = 0 \tag{22}$$

have *no solutions* where

$$|z| \geqq 1, \quad |\kappa_p| < 1, \quad p = 1, 2, 3,$$

(**) z is *not* a generalized eigenvalue.

Here $P(z, \kappa)$ is the characteristic polynomial (14b) and $E(z)$ is the matrix, the elements of which correspond to the set of boundary conditions under consideration.

When a solution $\{z, \kappa_1, \kappa_2, \kappa_3\}$ to (22) satisfies *both* the conditions below, z is called a generalized eigenvalue:

$$\sim|z| = 1, \quad \kappa_p \in \mathcal{G} \vee |\kappa_p| < 1, \quad p = 1, 2, 3$$

$$\sim\kappa_p \in \mathcal{G} \quad \text{for } \textit{at least one } p$$

where the set $\mathcal{G}$ is defined by

$$\kappa \in \mathcal{G} \Leftrightarrow (|\kappa| = 1, |z| \to 1_+ \Rightarrow |\kappa| \to 1_-).$$

By analytical means it is almost impossible to determine if condition (*) holds, whereas criterion (**), despite the extensive algebra, can be treated with analytical methods (§ 3.3.2). Therefore it is necessary to resort to numerical methods to decide for which λ-values the stability conditions are fulfilled. To this end we make use of the software tool Ibstab [7]. The Ibstab algorithm is specially designed for automatic GKS stability analysis. By exploiting the fact that the system (22) originates from a difference approximation, it is possible to show that it suffices to look for solutions for which $1 \leqq |z| \leqq 1 + \delta$, where δ is a small positive constant. This gives a much more efficient algorithm than, e.g., using a standard type continuation method, which would find *all* solutions to (22), including those with $|z| \ll 1$, which are of no interest in the present context. Futhermore, the Ibstab algorithm varies λ pseudocontinuously, and thus claims to analyze stability not for certain *discrete* λ-values but for a continuous *interval* $0 < \lambda \leqq \lambda_s$, where λ_s is a stop-value given by the Ibstab user.

3.3.2. Generalized eigenvalues for boundary condition 1. To find generalized eigenvalues, if there are any, we consider those roots of the characteristic equation that lie on the unit circle when the following constraint on λ holds. That is,

$$\kappa = e^{i\xi}, \qquad 0 < \lambda < 1.$$

The constraint is a somewhat restricted version of the stability condition (13) for the Cauchy problem combined with $|\lambda| = \lambda$, which is valid at outflow. A short investigation shows that we now have $|z| = 1$ *only* for $\xi = 0, \pm\pi$. These ξ-values together with $\kappa = e^{i\xi}$ and (14a) give $\{z_0 = 1,\ \kappa = \pm 1\}$ as the only solutions of the characteristic equation, for which $|z| = 1$, $|\kappa| = 1$. This means that the only *possible* generalized eigenvalue is z_0.

By analyzing the effect of a small perturbation of z_0, we can determine whether any of the relevant κ-values belong to the set $\mathcal{G}$:

(1)
$$z = 1 + \delta, \qquad \delta > 0,$$
$$\kappa = 1 + \varepsilon$$

inserted into the characteristic equation (14b) yields

$$-\delta(1+\varepsilon)^3 + \lambda(1+\varepsilon)^2(2\varepsilon+\varepsilon^2) + \frac{\lambda^2}{2}(1+\varepsilon)(2\varepsilon+\varepsilon^2)^2 + \frac{\lambda^3}{4}(2\varepsilon+\varepsilon^2)^3 = 0.$$

By neglecting high-order terms we obtain

$$\varepsilon \approx \frac{\delta}{2\lambda}.$$

Now $\delta, \lambda > 0$ implies $\varepsilon > 0$, which leads to

$$|z| \to 1_+ \Rightarrow |\kappa| \to 1_+ \quad \text{for } \kappa_4 = 1$$
$$\Downarrow$$
$$\kappa_4 \notin \mathcal{G}.$$

(2)
$$z = 1 + \delta, \qquad \delta > 0,$$
$$\kappa = -(1+\varepsilon)$$

inserted into the characteristic equation yields

$$\delta(1+\varepsilon)^3 + \lambda(1+\varepsilon)^2(2\varepsilon+\varepsilon^2) - \frac{\lambda^2}{2}(1+\varepsilon)(2\varepsilon+\varepsilon^2)^2 + \frac{\lambda^3}{4}(2\varepsilon+\varepsilon^2)^3 = 0.$$

By considering only low-order terms we obtain

$$\varepsilon \approx \frac{-\delta}{2\lambda}.$$

Now $\delta, \lambda > 0$ implies $\varepsilon < 0$, which leads to

$$|z| \to 1_+ \Rightarrow |\kappa| \to 1_- \quad \text{for } \kappa_1 = -1$$
$$\Downarrow$$
$$\kappa_1 \in \mathcal{G}.$$

For $z = 1$ the roots of the characteristic equation (14a) are given by

$$\mu\left(1 + \frac{\mu}{2} + \frac{\mu^2}{4}\right) = 0,$$

$\mu = 0$ yielding the roots κ_1 and κ_4, whereas the other solutions are obtained from

$$\mu^2 + 2\mu + 4 = 0$$
$$\Updownarrow$$
$$\mu = -1 \pm i\sqrt{3}$$
$$\Updownarrow$$
$$\kappa^2 + \frac{(1 \pm i\sqrt{3})}{\lambda}\kappa - 1 = 0,$$

which has two solutions where $|\kappa| < 1$, corresponding to κ_2, κ_3.

To sum up, $\{z_0, \kappa_1, \kappa_2, \kappa_3\}$ is a solution of *the characteristic equation* fulfilling the conditions for z_0 being a generalized eigenvalue. Hence, z_0 is a generalized eigenvalue if and only if the determinant condition holds.

When $z = 1$, $\kappa_1 = -1$ the elements (17a)-(17c) of the matrix for b.c 1 satisfy

$$e_{11} = e_{21} = 0, \qquad e_{31} \neq 0,$$

$$\left.\begin{aligned} e_{1p} &= \lambda(\kappa_p^2 - 1)\left(\kappa_p + \frac{\lambda}{2}(\kappa_p^2 - 1) + \frac{\lambda^2}{2}(\kappa_p - 1)(\kappa_p^2 - 1)\right) \\ e_{2p} &= \lambda(\kappa_p^2 - 1)(1 + \lambda(\kappa_p - 1) + \lambda^2(\kappa_p - 1)^2) \end{aligned}\right\} p = 2, 3. \tag{23}$$

The determinant condition then assumes the form

$$e_{12}e_{23} - e_{22}e_{13} = 0.$$

Insertion of (23) and reduction-transform the condition into

$$\lambda^4 + 2\lambda^2 - \lambda - \theta(2\lambda - \lambda^2 + 2\lambda^3) + \theta^2(1 - \lambda + \lambda^2) + \chi(\chi + \theta(1 - 2\lambda) + 2\lambda^2 - 1) = 0, \tag{24}$$

$$\chi = \lambda^2\kappa_2\kappa_3 - 1, \qquad \theta = \lambda(\kappa_2 + \kappa_3) + 1.$$

A close examination of the roots κ_2, κ_3 shows that

$$\kappa_2 + \kappa_3 = \sqrt{2(\psi + 1 - (1/2\lambda^2))} - \frac{1}{\lambda}, \tag{25a}$$

$$\lambda\kappa_2\kappa_3 = \frac{1}{\lambda} + \lambda\psi - \sqrt{\tfrac{1}{2}(\psi + 1 - (1/2\lambda^2))} - \sqrt{\tfrac{3}{2}(\psi - 1 + (1/2\lambda^2))}, \tag{25b}$$

$$\psi = \sqrt{1 - (1/\lambda^2) + (1/\lambda^4)}. \tag{25c}$$

By using (25a)-(25c) in (24) and then performing some rather extensive algebra, we can restate the determinant condition:

$$R(\lambda) \neq 0 \quad \forall\lambda : 0 < \lambda < 1 \Rightarrow$$

$\{z_0, \kappa_1, \kappa_2, \kappa_3\}$ does *not* satisfy the determinant condition,

where

$$R(\lambda) = P_1 - P_2,$$

$$P_1 \equiv 2460\lambda^{12} + 456\lambda^9 + 5280\lambda^8 + 704\lambda^5 + 2824\lambda^4 + 168\lambda + 153,$$

$$P_2 \equiv 128\lambda^{14} + 128\lambda^{13} + 584\lambda^{11} + 3872\lambda^{10} + 568\lambda^7 + 4676\lambda^6 + 528\lambda^3 + 904\lambda^2.$$

The result when computing the roots with a Numerical Algorithms Group-routine shows that the polynomial R has *no* zeros in the interval $0 < \lambda < 1$, which implies that $\{z_0, \kappa_1, \kappa_2, \kappa_3\}$ does *not* satisfy the determinant condition. Hence z_0 is not a generalized eigenvalue and then *there are no generalized eigenvalues at all for b.c.* 1.

We do not perform a corresponding calculation for the remaining sets of boundary conditions. By now the reader should be convinced of the need for a software tool like Ibstab for the analysis. Ibstab will find all eigenvalues z with $|z| \geqq 1$, including the generalized ones.

3.4. Inflow boundary conditions.

3.4.1. Boundary condition 5. We now investigate the stability for Model Problem 1 in the inflow case, $\alpha < 0$. The characteristic equation is the same (14b) as in the case of outflow and the same stability criteria as in § 3.3.1 are valid. The situation at the boundary, however, is slightly different since numerical boundary conditions now are required only at the gridpoints x_2, x_1. At x_0 the solution can be prescribed with an arbitrary function of time, but without loss of generality we can assume that this boundary condition is *homogeneous*, i.e., $u_0^{n+1} \equiv 0$.

At the inflow boundary we now use a strategy similar to the design of b.c. 2:

$$u_2^{n+1} = u_2^n + \frac{\lambda}{2}\Delta_0(u_2^n + \breve{\breve{u}}_2), \tag{26a}$$

$$u_1^{n+1} = u_1^n + \frac{\lambda}{2}\Delta_0(u_1^n + \breve{\breve{u}}_1), \tag{26b}$$

$$u_0^{n+1} = 0 \tag{26c}$$

where

$$\breve{\breve{u}}_0 = u_0^n + \lambda\Delta_+(u_0^n + \breve{u}_0), \qquad \breve{u}_0 = u_0^n + 2\lambda\Delta_+ u_0^n.$$

3.4.2. Boundary condition 6. A most natural way to formulate complementary conditions at the inflow boundary is to utilize the design of b.c. 3 adapted to inflow:

$$u_2^{n+1} = u_2^n + \frac{\lambda}{2}\Delta_0(u_2^n + \breve{\breve{u}}_2), \tag{27a}$$

$$u_1^{n+1} = u_1^n + \frac{\lambda}{2}\Delta_0(u_1^n + \breve{\breve{u}}_1), \tag{27b}$$

$$u_0^{n+1} = 0 \tag{27c}$$

where

$$\breve{\breve{u}}_0 = \breve{u}_0 = u_0^{n+1}.$$

3.5. Results of the normal mode analysis. The results of the Ibstab computations for the various boundary conditions are presented in Table 1. Ibstab returns two values, λ_L and λ_R. λ_L is the largest value of $|\lambda|$ for which Ibstab concluded the scheme to be stable. λ_R denotes the smallest value for which an instability was found. The length of the uncertainty interval $]\lambda_L, \lambda_R[$ (i.e., the interval for which Ibstab does not make any

TABLE 1
Ibstab-*results.*

Set of b.c.	λ_L	λ_R	Type of instability
1	0.460	0.485	GKS
2	0.985	1.01	von Neumann
3	0.985	1.01	von Neumann
4	0.985	1.01	von Neumann
5	0.878	0.898	GKS
6	0.985	1.01	von Neumann

statement) is assured to be within a tolerance, which in the present Ibstab version is set to 0.025. The expression 'von Neumann instability' refers to the situation where there are κ-values lying on the unit circle for which $|z|>1$. Note that independent of the choice of boundary conditions there will be von Neumann instability when $|\lambda|>1$, since in that case not even the Cauchy problem is stable.

From Table 1 we may conclude that b.c. 2, 3, 4, and 6 are GKS-stable for $0<|\lambda|\leqq 1$. According to (13), this interval is also the domain of stability for the associated Cauchy problem, and thus the least restrictive one attainable. For b.c. 1 we have GKS stability when $0<|\lambda|\leqq 0.460$, whereas b.c. 5 is GKS-stable for $0<|\lambda|\leqq 0.878$. Note that even though similar strategies were used in the design of outflow b.c 2 and inflow b.c. 5, they have different domains of stability. Another interesting observation is that b.c. 1 and b.c. 2 have completely different stability regions. This is remarkable since the same strategy was employed in both cases, the only difference being that b.c. 1 used the one-level approach and b.c. 2 used the multilevel approach.

4. Analysis of overlapping conditions.

4.1. The conditions. We now proceed to analyze the stability of Model Problem 2. First, the overlapping conditions are defined. The choice that seems most natural to us is

$$\check{u}_0=\check{\check{u}}_0=u_0^{n+1}=L_0^{(1)}v_{-N_2}^{n+1}, \tag{28a}$$

$$\check{v}_0=\check{\check{v}}_0=v_0^{n+1}=L_0^{(2)}u_{N_1}^{n+1} \tag{28b}$$

where $L_0^{(1)}$ and $L_0^{(2)}$ are interpolation operators.

We wish to use Ibstab as a tool in this case as well, but Ibstab can only handle quarter-plane problems. Thus, we fold the v-part of the problem (cf. [2]). By making a suitable variable transformation on G_2, we can consider u and v as approximating ψ and χ, respectively, where

$$\begin{pmatrix}\psi\\ \chi\end{pmatrix}_t=\begin{pmatrix}\alpha & 0\\ 0 & -\alpha\end{pmatrix}\begin{pmatrix}\psi\\ \chi\end{pmatrix}_x, \qquad 0\leqq x<\infty, \quad t\geqq 0.$$

This is an ordinary quarter-plane problem and the overlapping conditions (28) will by the same transformation turn into boundary conditions. Keeping the notation v (despite the transformation) the transformed conditions will simply be

$$\check{u}_0=\check{\check{u}}_0=u_0^{n+1}=L_0^{(1)}v_{N_2}^{n+1}, \tag{29a}$$

$$\check{v}_0=\check{\check{v}}_0=v_0^{n+1}=L_0^{(2)}u_{N_1}^{n+1}. \tag{29b}$$

Next, we dispose of the complication of having two different mesh sizes involved in the problem. This is done by introducing the parameter

$$\varphi=h_1/h_2.$$

If we let λ be defined by $\lambda\equiv\alpha k/2h_1$, then the characteristic equation corresponding to u will be precisely (14b), whereas the characteristic equation for v is obtained from (14b) by replacing λ by $-\varphi\lambda$. We now have the problem on a form suitable for treatment by Ibstab. Next, we will derive the determinant condition corresponding to (29). In the present case, we have two resolvent equations, one for u and one for v, with solutions $w^{(1)}$ and $w^{(2)}$, respectively. Using notation analogous to that introduced in (15), we obtain

$$w_j^{(m)}=\sum_{p=1}^{3}\sigma_{mp}\kappa_{mp}^j, \qquad m=1,2,$$

where

$$\kappa_{mp} \in M_I^{(m)}, \qquad m = 1, 2.$$

The unknown coefficients σ_{mp} are to be determined by insertion into the overlapping conditions. This gives the coefficient matrix $E(z)$ in the determinant condition.

The matrix has the following structure:

$$E(z) = \begin{pmatrix} -1 & -1 & -1 & e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} & -1 & -1 & -1 \\ e_{31} & e_{32} & e_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & e_{41} & e_{42} & e_{43} \\ e_{51} & e_{52} & e_{53} & 0 & 0 & 0 \\ 0 & 0 & 0 & e_{61} & e_{62} & e_{63} \end{pmatrix}.$$

The elements e_{1p} and e_{2p} depend explicitly on our choice of interpolation operators, which will be discussed later. The other elements are the following:

$$e_{3p} = -\tfrac{1}{2}\lambda(1+z) + \tfrac{1}{4}\lambda^3 + (1 - z - \tfrac{1}{2}\lambda^2)\kappa_{1p} + (\lambda - \tfrac{1}{2}\lambda^3)\kappa_{1p}^2 + \tfrac{1}{2}\lambda^2\kappa_{1p}^3 + \tfrac{1}{4}\lambda^3\kappa_{1p}^4,$$

$$e_{4p} = \tfrac{1}{2}\varphi\lambda(1+z) - \tfrac{1}{4}\varphi^3\lambda^3 + (1 - z - \tfrac{1}{2}\varphi^2\lambda^2)\kappa_{2p} + (-\varphi\lambda + \tfrac{1}{2}\varphi^3\lambda^3)\kappa_{2p}^2 + \tfrac{1}{2}\varphi^2\lambda^2\kappa_{2p}^3 - \tfrac{1}{4}\varphi^3\lambda^3\kappa_{2p}^4,$$

$$e_{5p} = \tfrac{1}{4}\lambda^2(1+z) + (-\lambda + \tfrac{1}{2}\lambda^3)\kappa_{1p} + (1 - z - \lambda^2)\kappa_{1p}^2 + (\lambda - \tfrac{3}{4}\lambda^3)\kappa_{1p}^3 + \tfrac{1}{2}\lambda^2\kappa_{1p}^4 + \tfrac{1}{4}\lambda^3\kappa_{1p}^5,$$

$$e_{6p} = \tfrac{1}{4}\varphi^2\lambda^2(1+z) + (\varphi\lambda - \tfrac{1}{2}\varphi^3\lambda^3)\kappa_{2p} + (1 - z - \varphi^2\lambda^2)\kappa_{2p}^2 + (-\varphi\lambda + \tfrac{3}{4}\varphi^3\lambda^3)\kappa_{2p}^3 + \tfrac{1}{2}\varphi^2\lambda^2\kappa_{2p}^4 - \tfrac{1}{4}\varphi^3\lambda^3\kappa_{2p}^5.$$

We have stability for those values of λ for which (18) holds.

4.2. The interpolation operators. We will analyze linear and quadratic interpolation. In the interpolation formulae we will make use of the parameters β_1 and β_2, where

$$\beta_j = d_j / h_j, \qquad j = 1, 2,$$

h_1, h_2 are the mesh sizes, and d_1, d_2 are the overlap quantities shown in Fig. 1. We define d_j so that $d_j < h_j$, implying $0 \leqq \beta_j < 1$, $j = 1, 2$.

In the case of linear interpolation it is natural to use u_{N_1}, u_{N_1+1} and v_{N_2}, v_{N_2+1}, respectively. We then obtain the interpolation operators

$$L_0^{(1)} v_{N_2} = (1 - \beta_2) v_{N_2} + \beta_2 v_{N_2+1},$$
$$L_0^{(2)} u_{N_1} = (1 - \beta_1) u_{N_1} + \beta_1 u_{N_1+1},$$

which yield

$$e_{1p} = (1 - \beta_2 + \beta_2\kappa_{2p})\kappa_{2p}^{N_2},$$
$$e_{2p} = (1 - \beta_1 + \beta_1\kappa_{1p})\kappa_{1p}^{N_1}.$$

When we turn to quadratic interpolation, an additional grid function value must be involved. We have chosen u_{N_1-1} and v_{N_2-1}, respectively, and the resulting interpolation operators are:

$$L_0^{(1)} v_{N_2} = \tfrac{1}{2}\beta_2(\beta_2 - 1) v_{N_2-1} + (1 - \beta_2^2) v_{N_2} + \tfrac{1}{2}\beta_2(\beta_2 + 1) v_{N_2+1},$$
$$L_0^{(2)} u_{N_1} = \tfrac{1}{2}\beta_1(\beta_1 - 1) u_{N_1-1} + (1 - \beta_1^2) u_{N_1} + \tfrac{1}{2}\beta_1(\beta_1 + 1) u_{N_1+1},$$

yielding

$$e_{1p} = (\tfrac{1}{2}\beta_2(\beta_2 - 1) + (1 - \beta_2^2)\kappa_{2p} + \tfrac{1}{2}\beta_2(\beta_2 + 1)\kappa_{2p}^2)\kappa_{2p}^{N_2 - 1},$$
$$e_{2p} = (\tfrac{1}{2}\beta_1(\beta_1 - 1) + (1 - \beta_1^2)\kappa_{1p} + \tfrac{1}{2}\beta_1(\beta_1 + 1)\kappa_{1p}^2)\kappa_{1p}^{N_1 - 1}.$$

4.3. Generalized eigenvalues. We will now prove that for certain values of β_1 and N_1 there is a generalized eigenvalue. By the same argument as in § 3.3.2, we get that $z_0 = 1$, $\kappa_{11} = -1$ and $z_0 = 1$, $\kappa_{21} = 1$ are solutions of *the characteristic equations* fulfilling the conditions for z_0 being a generalized eigenvalue. This means that z_0 is a generalized eigenvalue if and only if the determinant condition holds. Furthermore, z_0 is the only possible generalized eigenvalue.

When $z = 1$, $\kappa_{11} = -1$, $\kappa_{21} = 1$ the elements of the matrix $E(z)$ satisfy

$$e_{11} = 1,$$

$$e_{j1} = 0, \qquad j = 3, \cdots, 6,$$

$$e_{21} = \begin{cases} (1-2\beta_1)(-1)^{N_1} & \text{linear interpolation,} \\ (1-2\beta_1^2)(-1)^{N_1} & \text{quadratic interpolation.} \end{cases} \tag{30}$$

The determinant condition now is reduced to

$$(1 - e_{21}) \det (B) = 0 \tag{31}$$

where

$$B = \begin{pmatrix} e_{32} & e_{33} & 0 & 0 \\ 0 & 0 & e_{42} & e_{43} \\ e_{52} & e_{53} & 0 & 0 \\ 0 & 0 & e_{62} & e_{63} \end{pmatrix}.$$

Equation (31) holds especially for $e_{21} = 1$. From (30), and since $0 \leqq \beta_1 < 1$, we get $e_{21} = 1$ only when $\beta_1 = 0$ and N_1 is even. Then z_0 is a generalized eigenvalue, which implies instability.

We conclude that when $\beta_1 = 0$ and N_1 is *even*, the overlapping conditions (28) are *unstable* for both linear and quadratic interpolation.

4.4. Results of the normal mode analysis. For each kind of interpolation, we used Ibstab to analyze the following cases:

$$\varphi = 0.5, \qquad \beta_1 = 0, 0.5, \tag{32a}$$

$$\varphi = 1, \qquad \beta_1 = 0, 0.5, \tag{32b}$$

$$\varphi = 2, \qquad \beta_1 = 0, 0.25. \tag{32c}$$

In each case we chose the smallest value of N_1 that gives explicit overlapping conditions. In general, this is $N_1 = 3$ for $\varphi = 1, 2$ and $N_1 = 6$ for $\varphi = 0.5$. However, when quadratic interpolation is used with $\beta_1 = 0.5$, then we must take $N_1 = 4$ and $N_1 = 8$, respectively. The values of β_2 and N_2 are uniquely defined by φ, β_1, and N_1.

We believe that the values of φ included in (32) are representative. Larger or smaller values of φ are of less interest, since we assume that a global timestep k is used. According to the stability limit (13), we must choose $k \leqq 2h/\alpha$, where h is the smaller mesh size. This means that on the coarse grid, we have to take an unnecessarily small k. This can be accepted for values of φ as those in (32). If the difference between the mesh sizes is larger, then we should consider the possibility of taking local timesteps on the different grids (see, e.g., Berger [1]).

By combining the Ibstab results and the conclusion from § 4.3 regarding generalized eigenvalues, the results of the stability analysis can be summarized as follows:

> When $\beta_1 = 0$ and N_1 is even, the overlapping conditions (28) are unstable. For all other cases described by (32), both linear and quadratic interpolation yield overlapping conditions (28) that are stable up to the von Neumann stability limit, i.e., for $\lambda \leqq \min(1, \varphi^{-1})$.

We point out that for $\beta_1 = 0$, overlapping condition (28b) reduces to a *straight injection* of a value from grid G_1 to G_2: $\check{v}_0 = \check{\check{v}}_0 = v_0^{n+1} = u_{N_1}^{n+1}$. In our model problem, the direction of the flow is from right to left, i.e., from G_1 to G_2. Thus, our result says that when N_1 is even, straight injection from the inflow side to the outflow side is unstable.

It still remains to explain this difference between odd and even. One way to do this is to study the critical eigensolutions $u_j = (-1)^j$ and $v_j = 1$ that we get for $z = 1$, $\kappa_{11} = -1$, and $\kappa_{21} = 1$. With an odd value of N_1 and straight injection we cannot have both eigensolutions at the same time, but with an even N_1 this is possible, which causes the instability.

5. Numerical experiments.

5.1. Experiments testing the boundary conditions. To check the stability conditions stated in §§ 3.5 and 4.4, we have performed numerical experiments with Gary's scheme. We first made a set of stability tests on Model Problem 1, checking the boundary conditions 1-6. In the inflow case we used $\alpha = -1$, in the outflow case $\alpha = +1$ in (11). We specified initial data

$$u(x, 0) = f(x) \equiv e^{-x} \sin(4\pi x)$$

and in the inflow case boundary data

$$u(0, t) = f(-t),$$

whereby the analytical solution is given by

$$u(x, t) = f(x + \alpha t).$$

Model Problem 1 is a quarter-plane problem, so we had to introduce an artificial boundary, which we took to be at $x = 1$. Then the numerical scheme needs boundary conditions in the three points closest to this boundary. In order to make the artificial boundary transparent we gave the analytical solution values in these points. The computational domain $0 \leqq x \leqq 1$, $t \geqq 0$ was discretized as follows:

$$\begin{aligned} &x_j = jh, \qquad j = 0, 1, \cdots, N, \\ &h = 1/N, \\ &t_n = nk, \qquad n = 0, 1, \cdots, \\ &k = 2|\lambda|h. \end{aligned}$$

For the experimental stability check we introduced the norm

$$\|u^n\|_h = \left(\frac{h}{2}((u_0^n)^2 + (u_N^n)^2) + h \sum_{j=1}^{N-1} (u_j^n)^2\right)^{1/2},$$

which is a discrete version of the $\mathscr{L}_2(0, 1)$ norm. (The norm used in the definition of GKS stability [4, Def. 3.3] is more complicated, but in experiments like the ones we are considering it is common to use the $\mathscr{L}_2$ norm.)

We used the following strategy. For a fixed λ, choose a spatial mesh size $h = h_0$, which implies $k = k_0 \equiv 2|\lambda|h_0$. Then compute the solution at time t, i.e., u^{n_0} where $n_0 = t/k_0$, for successive values of t. The calculations are repeated using $h = h_0/2$, which implies $k = k_0/2$. The time t will be time level $2n_0$ for $h = h_0/2$. Thus, in the case of a numerical instability, we should expect

$$\|u^{2n_0}\|_{h_0/2} \gg \|u^{n_0}\|_{h_0}.$$

Numerical tests according to this strategy were performed on a MicroVax II at Uppsala University, using the standard Vax FORTRAN compiler. As a check of program

correctness the rate of convergence obtained in each test was computed by introducing the control quotient θ, defined by

$$\theta = \ln\left(\frac{\|u(\cdot, t_{n_0}) - u^{n_0}\|_{h_0}}{\|u(\cdot, t_{2n_0}) - u^{2n_0}\|_{h_0/2}}\right) \Big/ \ln 2.$$

Thus, linear convergence would correspond to $\theta = 1$ and quadratic convergence to $\theta = 2$. Theoretically, in the stable cases our tests should show quadratic convergence. Any serious programming errors would be reflected in the computed value of θ being much different from 2. All the experiments gave $\theta \approx 2$, which thus indicates program correctness.

For each set of boundary conditions two experiments were performed, one with $\lambda \approx \lambda_L$ and one with $\lambda \approx \lambda_R$, where λ_L and λ_R are given in Table 1. We thus expected stability for the first choice of λ and instability for the second. The experiments support the Ibstab results. It is of special interest to note that the stability limits claimed by Ibstab for boundary conditions 1 and 5 are correct. This can be seen from Tables 2 and 3.

According to the definitions of h_0 and n_0 and to the discussion earlier in this section it is evident that boundary condition 1 is stable for $|\lambda| = 0.46$ but unstable for $|\lambda| = 0.47$, and that boundary condition 5 is stable for $|\lambda| = 0.87$ but unstable for $|\lambda| = 0.91$. The fact that we get growing norms for $|\lambda| = 0.87$ in Table 3 is *not* a sign of instability. In fact the norm of the corresponding analytical solution increases with

TABLE 2
Stability check for b.c. 1 with $h_0 = 0.02$.

$\lvert\lambda\rvert$	n_0	$\|u^{n_0}\|_{h_0}$	$\|u^{2n_0}\|_{h_0/2}$
0.46	100	$8.00 \cdot 10^{-2}$	$7.76 \cdot 10^{-2}$
	200	$1.25 \cdot 10^{-2}$	$1.15 \cdot 10^{-2}$
	400	$1.66 \cdot 10^{-3}$	$3.39 \cdot 10^{-4}$
	800	$6.78 \cdot 10^{-4}$	$4.40 \cdot 10^{-5}$
	1600	$1.08 \cdot 10^{-4}$	$5.31 \cdot 10^{-6}$
0.47	100	$7.56 \cdot 10^{-2}$	$7.23 \cdot 10^{-2}$
	200	$2.03 \cdot 10^{-2}$	$1.30 \cdot 10^{-2}$
	400	$7.52 \cdot 10^{-2}$	0.126
	800	1.43	50.0
	1600	516	$7.89 \cdot 10^{6}$

TABLE 3
Stability check for b.c. 5 with $h_0 = 0.02$.

$\lvert\lambda\rvert$	n_0	$\|u^{n_0}\|_{h_0}$	$\|u^{2n_0}\|_{h_0/2}$
0.87	20	0.927	0.960
	30	1.21	1.26
	50	2.54	2.64
	100	14.6	15.2
	200	482	501
0.91	20	0.931	0.972
	30	1.29	1.34
	50	2.59	3.50
	100	28.7	$3.09 \cdot 10^{4}$
	200	$2.50 \cdot 10^{5}$	$4.83 \cdot 10^{12}$

time, so a growing norm in the numerical solution is expected. The rate of growth of the solution for $|\lambda| = 0.87$ closely agrees with analytical results, whereas $|\lambda| = 0.91$ gives an uncontrolled growth of the solution. Furthermore, for $|\lambda| \leqq 0.87$ and $h_0 = 0.01$, the relative error $\|u(\cdot, t_n) - u^n\|_{h_0} / \|u(\cdot, t_n)\|_{h_0}$ stays within 5 percent. The observation of importance for the stability check is that the norm for a fixed time does not change significantly when refining the mesh in space, keeping λ constant.

5.2. Experiments on the complete approximation. In our second set of experiments, we turned to the complete approximation including the overlap, and to our original problem of solving the linearized Euler equations. However, according to the discussion in § 2, we could as well consider the scalar model problem. The problem that we solved was (11), with $\alpha = +1$, on $0 \leqq x \leqq 1$. As initial data we took

$$u(x, 0) = f(x) \equiv \sin(4\pi x)$$

and the analytical solution is given by

$$u(x, t) = f(x + t).$$

The computational domain was substructured as in Fig. 1. The grid G_1 was fixed, with $x_{0,G_1} = 0.4$ and $x_{M_1,G_1} = 1$. Furthermore we set $x_{-M_2,G_2} = 0$. The parameters that we specified were h_1, φ, β_1, and N_1. We then used the following algorithm to compute x_{0,G_2}, h_2, β_2, and N_2:

$$\begin{aligned}
&x_{0,G_2} \leftarrow x_{0,G_1} + (N_1 + \beta_1) h_1, \\
&M_2 \leftarrow \lfloor (x_{0,G_2} - x_{-M_2,G_2}) \varphi / h_1 \rfloor, \\
&h_2 \leftarrow (x_{0,G_2} - x_{-M_2,G_2}) / M_2, \\
&\varphi \leftarrow h_1 / h_2, \\
&N_2 \leftarrow \lfloor (N_1 + \beta_1) \varphi \rfloor, \\
&\beta_2 \leftarrow (N_1 + \beta_1) \varphi - N_2
\end{aligned}$$

(here $\lfloor \cdot \rfloor$ denotes truncation). Thus we may get a slightly changed value of φ, but this is necessary if we want to keep x_{-M_2,G_2} fixed. Starting out with the parameter values given by (32), the procedure described above results in β_1 and N_1 exactly as in (32), whereas φ, β_2, and N_2 come close to the values defined by (32).

We did one experiment for each kind of interpolation and each combination of parameter values (32). To check stability, we used the strategy described in § 5.1. For each combination of parameter values given above, we ran a test involving the λ-values indicated by the Ibstab results. In the case $\varphi \approx 1$, we also included $\lambda = 0.5$, for a program-correctness check (see below).

The experiments confirm the stability analysis. The results for u and v are similar, so we only discuss those for u. Furthermore, we do not include results for $n_0 > 400$ unless they contribute some new information. Tables 4 and 5 show the results for $\varphi \approx 1$

TABLE 4
Case $\varphi \approx 1$, $\beta_1 = 0/\beta_1 = 0.5$, $N_1 = 3$, $h_0 = 0.025$.

λ	n_0	$\|u^{n_0}\|_{h_0}$	$\|u^{2n_0}\|_{h_0/2}$	$\|u(\cdot, t) - u^{2n_0}\|_{h_0/2}$
0.5	100	0.530/0.522	0.535/0.537	0.017/0.021
	400	0.529/0.521	0.535/0.537	0.017/0.021
0.9	100	0.498/0.471	0.534/0.533	0.031/0.036
	400	0.498/0.471	0.534/0.533	0.030/0.036
1.1	100	$\sim 10^{13}/10^{15}$	*overflow*	

TABLE 5
Case $\varphi \approx 2$, $\beta_1 = 0/\beta_1 = 0.25$, $N_1 = 3$, $h_0 = 0.025$.

λ	n_0	$\|u^{n_0}\|_{h_0}$	$\|u^{2n_0}\|_{h_0/2}$
0.45	100	0.535/0.533	0.538/0.537
	400	0.536/0.532	0.538/0.536
0.55	100	$\sim 10^{12}/10^{14}$	*overflow*

and $\varphi \approx 2$, respectively, in the case of linear interpolation. For comparison, we mention that the norm of the analytical solution has the value 0.537. Later, we will also discuss the norm of the error and in that context we come back to the last column of Table 4.

Table 6 is perhaps the most interesting one. It shows that the case $\varphi \approx 0.5$, $\beta_1 = 0$, $N_1 = 6$ with linear interpolation, really *is* unstable. We have included $\|u^{4n_0}\|_{h_0/4}$ to show the explosion clearly. The corresponding case with nonmatching gridpoints ($\beta_1 = 0.5$) is stable, as is shown by Table 7. The quadratic interpolation results are similar to those shown in Tables 4–7, and thus are not tabulated.

Let us now give some further comments. This set of experiments had three built-in program-correctness checks. The first concerns the parameter θ, which reflects how the norm of the error depends on the mesh size. θ was computed in our experiments and was found to be in the interval [1.8, 2.4] in all stable cases. Second, for $\varphi \approx 1$ the value $\lambda = 0.5$ should be the best choice, because it implies that the mesh size in time equals the mesh size in space, which for our problem means that the gridpoints in space and time align with the characteristics of the pde. The last column of Table 4 shows that $\lambda = 0.5$ *did* actually give a better solution than did $\lambda = 0.9$. The third program-correctness check was that for $\beta_1 = 0$ linear and quadratic interpolation should give equal results. This was fulfilled. In particular, the case $\varphi \approx 0.5$, $\beta_1 = 0$, $N_1 = 6$ with quadratic interpolation gave exactly the same results as in Table 6. The three checks that we have now described indicate that our program was correct.

Finally, we did an experiment to show that straight injection with $\varphi \approx 1$ and an even value of N_1 is unstable. However, with reasonable computing efforts and using

TABLE 6
Case $\varphi \approx 0.5$, $\beta_1 = 0$, $N_1 = 6$, $h_0 = 0.025$.

λ	n_0	$\|u^{n_0}\|_{h_0}$	$\|u^{2n_0}\|_{h_0/2}$	$\|u^{4n_0}\|_{h_0/4}$
0.1	100	0.543	0.538	0.537
	400	0.586	0.536	0.536
	3200	9.8	30.7	$\sim 10^5$

TABLE 7
Case $\varphi \approx 0.5$, $\beta_1 = 0.5$, $N_1 = 6$, $h_0 = 0.025$.

λ	n_0	$\|u^{n_0}\|_{h_0}$	$\|u^{2n_0}\|_{h_0/2}$	$\|u^{4n_0}\|_{h_0/4}$
0.9	100	0.451	0.528	0.532
	400	0.453	0.528	0.533
	3200	0.457	0.528	0.524
1.1	100	$\sim 10^{13}$	*overflow*	

the same initial data as in the earlier experiments, we were not able to make the instability show up. Instead, we took the eigensolutions $u_j = (-1)^j$, $v_j = 1$ as initial data and added homogeneous inflow data. On top of this we introduced on every time level a random perturbation at the overlap and at the outflow boundary. The perturbations were in the interval $[-0.1, 0.1]$. The results shown in Table 8 indicate that with $N_1 = 4$ the scheme is unstable, whereas with $N_1 = 3$ it is stable. The instability would probably show up clearer in a nonlinear problem or a problem with variable coefficients. As we see from Table 6, the growth was also much stronger in the mesh-refinement case.

TABLE 8
Case $\varphi \approx 1$, $\beta_1 = 0$, $N_1 = 3/N_1 = 4$, $h_0 = 0.025$.

λ	n_0	$\|u^{n_0}\|_{h_0}$	$\|u^{2n_0}\|_{h_0/2}$	$\|u^{4n_0}\|_{h_0/4}$
0.1	6400	0.530/1.32	0.362/1.35	0.270/1.42

6. Conclusions. We have analyzed the stability of Gary's scheme, which is a Runge–Kutta-type time marching method, for the Euler equations, or more generally, for first-order hyperbolic systems. We studied the approximation on a substructured domain and designed several stable boundary and overlapping conditions. The results of the stability analyses are summarized in §§ 3.5 and 4.4. When we use any combination of stable boundary and overlapping conditions, this will give a stable composed approximation on the substructured domain, provided the distances between different overlap regions are independent of the mesh sizes.

We have presented a series of experiments that fully support the results of the stability analyses. The conclusion from the analysis of overlapping conditions based on interpolation is that if the gridpoints in the overlap region do not match exactly between the grids, then the same interpolation strategy can be used for both incoming and outgoing characteristic variables. However, if the gridpoints match and straight injection is used, then it is crucial that the value injected into the grid on the outflow side be taken from a gridpoint with odd index on the inflow side. Injection from a point with even index is unstable. This especially means that straight injection is unstable in the case of mesh refinement for those components of the solution that move from the fine to the coarse grid. It might be worth investigating this phenomenon closer. Another direction of future research is to extend the stability analysis of Gary's scheme to multidimensional problems, primarily the two-dimensional case.

REFERENCES

[1] M. J. BERGER, *Stability of interfaces with mesh refinement*, Math. Comp., 45 (1985), pp. 301–318.
[2] M. CIMENT, *Stable difference schemes with uneven mesh spacings*, Math. Comp., 25 (1971), pp. 219–227.
[3] J. GARY, *On certain finite difference schemes for hyperbolic systems*, Math. Comp., 18 (1964), pp. 1–18.
[4] B. GUSTAFSSON, H.-O. KREISS, AND A. SUNDSTRÖM, *Stability theory of difference approximations for mixed initial boundary value problems.* II, Math. Comp., 26 (1972), pp. 649–686.
[5] A. RIZZI AND L.-E. ERIKSSON, *Computation of flow around wings based on the Euler equations*, J. Fluid. Mech., 148 (1984), pp. 45–71.
[6] G. STARIUS, *On composite mesh difference methods for hyperbolic differential equations*, Numer. Math., 35 (1980), pp. 241–245.

[7] M. THUNÉ, *Automatic* GKS *stability analysis*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 959-977.
[8] ———, *Stability of difference approximations of hyperbolic systems on substructured domains*, Report 106, Department of Scientific Computing, Uppsala University, 1986.
[9] L. N. TREFETHEN, *Stability of finite-difference models containing two boundaries or interfaces*, Math. Comp., 45 (1985), pp. 279-300.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 175–186, January 1989
© 1989 Society for Industrial and Applied Mathematics
013

A DUAL METHOD FOR CERTAIN POSITIVE SEMIDEFINITE QUADRATIC PROGRAMMING PROBLEMS*

KRZYSZTOF C. KIWIEL†

Abstract. This paper presents a dual active set method for minimizing a sum of piecewise linear functions and a strictly convex quadratic function, subject to linear constraints. It may be used for direction finding in nondifferentiable optimization algorithms and for solving exact penalty formulations of (possibly inconsistent) strictly convex quadratic programming problems. An efficient implementation is described extending the Goldfarb and Idnani algorithm, which includes Powell's refinements. Numerical results indicate excellent accuracy of the implementation.

Key words. convex optimization, quadratic programming, active set methods, dual algorithms, matrix factorizations, successive quadratic programming methods, nondifferentiable optimization methods

AMS(MOS) subject classification. 90C20

1. Introduction. We consider the following problem. Given an $n \times n$ symmetric positive definite matrix G, an m-vector c with positive components c_i, a set $J = \{1, 2, \cdots, p\}$ of cardinality $p < \infty$ partitioned into $m+1$ disjoint nonempty subsets J_i for $i = 0, 1, \cdots, m$, a p-vector a, and an $n \times p$ matrix P, find an n-vector x to

$$\text{minimize} \quad \frac{1}{2} x^T G x + \sum_{i=1}^{m} c_i \phi_i(x) \quad \text{over all } x \in X, \tag{1.1}$$

where

$$\phi_i(x) = \max \{P_j^T x - a_j : j \in J_i\} \quad \text{for } i = 1, \cdots, m,$$

$$X = \{x \in R^n : P_j^T x \leqq a_j \quad \text{for all } j \in J_0\},$$

P_j is column j of P, and superscript T denotes transposition.

Efficient methods for solving problem (1.1) are required in many nondifferentiable optimization algorithms (see, e.g., Kiwiel [K2], Lemarechal [L1], Mifflin [M2], Ruszczyński [R1] for general-purpose methods, and Fletcher [F2], Kiwiel [K1], [K3] for minimax and related methods), and in variable metric methods for nonlinearly constrained optimization (see, e.g., the review of Powell [P2]). Problem (1.1) may also arise as an exact penalty formulation of a (possibly inconsistent) quadratic programming (QP) problem (see § 5).

An equivalent QP formulation of (1.1) is to

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2} x^T G x + c^T v \quad \text{over all } (x, v) \in R^n \times R^m \\ &\text{satisfying} && P^T x - E^T v \leqq a, \end{aligned} \tag{1.2}$$

where the $m \times p$ matrix E has elements $E_{ij} = 1$ if $j \in J_i$ and $i \geqq 1$, $E_{ij} = 0$ otherwise. The dual problem is to

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2} |P\lambda|^2_{G^{-1}} + a^T \lambda \quad \text{over all } \lambda \in R^p \\ &\text{satisfying} && E\lambda = c, \qquad \lambda \geqq 0, \end{aligned} \tag{1.3}$$

* Received by the editors January 12, 1987; accepted for publication (in revised form) March 28, 1988. This research was supported by Project CPBP.02.15.

† Systems Research Institute, Polish Academy of Sciences, Newelska 6, 01-447 Warsaw, Poland.

where $|y|^2_{G^{-1}} = y^T G^{-1} y$ for all y in R^n, and $\lambda \geqq 0$ means $\lambda_j \geqq 0$ for all $j \in J$. The primal solution is recovered from a dual solution λ^* as

$$x^* = -G^{-1} P \lambda^*, \quad v_i^* = \phi_i(x^*) \quad \text{for } i = 1, \cdots, m.$$

In this paper we present a highly specialized active set method for problem (1.3). Its implementation combines the reduced gradient approach (see, e.g., Murtagh and Saunders [M3]) with the matrix factorizations and updating techniques of Goldfarb and Idnani [G6], including the refinements of Powell [P3]. In effect, it seems to inherit the excellent efficiency and accuracy of the Goldfarb and Idnani [G6] algorithm (see the discussions in Goldfarb [G5] and Powell [P3]).

The paper is organized as follows. The algorithm is described in § 2 and its matrix factorizations in § 3. Some modifications and extensions are given in § 4. Preliminary computational experience is reported on in § 5. The algorithm is compared with competing methods in § 6. Finally, we have a conclusion section.

We use the following notation. Given $x \in R^n$ and $q \geqq 1$, $\|x\|_q = (\sum_{i=1}^n |x_i|^q)^{1/q}$ denotes the l_q-norm of x. All vectors are column vectors. However, for convenience, $(x^T, v^T)^T$ is sometimes denoted as (x, v). Given a matrix Z, Z_j and Z_{ij}, denote the jth column and the (i, j)th element of Z, respectively. I denotes an identity matrix.

This paper is a condensed version of [K6], which establishes additional properties of the algorithm.

2. A method. We start with a few observations. For (x^*, v^*) to solve (1.2) and for λ^* to solve (1.3) it is both necessary and sufficient that the following Kuhn–Tucker conditions hold:

$$\begin{aligned} &E\lambda^* = c, \quad \lambda^* \geqq 0, \qquad x^* = -G^{-1} P \lambda^*, \\ &\sigma^* := a - P^T x^* + E^T v^* \geqq 0, \qquad (\sigma^*)^T \lambda^* = 0, \end{aligned} \tag{2.1}$$

where $-v^*$ and σ^* are the Lagrange multiplier vectors of (1.3). Note that either such x^*, v^*, and λ^* exist, or the constraints of (1.2) indexed by J_0 are inconsistent and the optimal value of (1.3) is unbounded from below.

Our algorithm for problem (1.3) follows the active set strategy (see Fletcher [F2], Gill et al. [G4]) by solving a sequence of equality-constrained subproblems of the following form:

$$\text{minimize} \quad f(\lambda) := \tfrac{1}{2}|P\lambda|^2_{G^{-1}} + a^T \lambda \tag{2.2a}$$

$$\text{subject to} \quad E\lambda = c, \ \lambda_j = 0 \quad \text{for } j \in J_N, \tag{2.2b}$$

where the active set $J_N \subset J$ is revised until the solution of (2.2) solves (1.3).

The explicit form of the constraints of (2.2)

$$\sum_{j \in J_i} \lambda_j = c_i \quad \text{for } i = 1, \cdots, m, \quad \lambda_j = 0 \text{ for } j \in J_N \tag{2.3}$$

suggests that we may eliminate certain variables. To this end, we shall use two lists of variables $\hat{L} = \{l(i)\}_{i=1}^m$ and $\hat{J} = \{j(k)\}_{k=1}^s$ such that $l(i) \in J_i$ for $i = 1, \cdots, m$, $\hat{J} \cup \hat{L} = J \backslash J_N$ and $\hat{J} \cap \hat{L} = \varnothing$, where $s = p - m - |J_N| \geqq 1$ and $s \leqq n + 1$. Identifying the components of λ indexed by $\hat{J}$ and $\hat{L}$ with vectors $\mu \in R^s$ and $\nu \in R^m$, respectively, we may partition the variables as follows:

$$\textit{superbasics}: \quad \lambda_{j(k)} = \mu_k \quad \text{for } k = 1, \cdots, s, \tag{2.4a}$$

$$\textit{basics}: \quad \lambda_{l(i)} = \nu_i \quad \text{for } i = 1, \cdots, m, \tag{2.4b}$$

$$\textit{nonbasics}: \quad \lambda_j = 0 \quad \text{for } j \in J_N = J \backslash (\hat{J} \cup \hat{L}), \tag{2.4c}$$

and equations (2.3) will hold if and only if

$$\nu_i = c_i - \sum_{k:j(k)\in J_i} \mu_k \quad \text{for } i = 1, \cdots, m. \tag{2.4d}$$

Thus each feasible λ is uniquely specified by its superbasic component μ, and we may write $\lambda = \lambda(\mu)$, $\mu = \mu(\lambda)$ to stress their connection. Hence λ solves (2.2) if and only if μ solves the following:

$$\text{minimize } w(\mu) \quad \text{over all } \mu \in R^s, \tag{2.5}$$

where

$$w(\mu) := f(\lambda(\mu)) = \tfrac{1}{2}|g + A\mu|^2_{G^{-1}} + b^T\mu + \bar{a}, \tag{2.6}$$

$$g + A\mu = P\lambda(\mu), \tag{2.7}$$

$$g = \sum_{i=1}^{m} c_i P_{l(i)}, \qquad \bar{a} = \sum_{i=1}^{m} c_i a_{l(i)} \tag{2.8}$$

and the $n \times s$ matrix A and the s-vector b are given by

$$\begin{aligned} &A_k = P_{j(k)} - P_{l(i)} \quad \text{and} \quad b_k = a_{j(k)} - a_{l(i)} \quad \text{if } j(k) \in J_i, \quad i \geqq 1, \\ &A_k = P_{j(k)} \quad \text{and} \quad b_k = a_{j(k)} \quad \text{if } j(k) \in J_0, \end{aligned} \tag{2.9}$$

for $k = 1, \cdots, s$.

Since in the absence of strict convexity (2.2) and (2.5) need not have finite solutions, it is more useful to consider feasible minimizing directions. More specifically, having an approximation $\hat{\lambda}$ to λ^* satisfying (2.2b), we shall need to find a direction $\hat{d} \in R^p$ such that the point $\bar{\lambda}(t) = \hat{\lambda} + t\hat{d}$ satisfies (2.2b) for each $t \in R$ and such that $\bar{\lambda}(t)$ solves (2.2) for some $t = t_{\text{opt}} \geqq 0$ or $f(\bar{\lambda}(t)) \downarrow -\infty$ as $t \to +\infty$. Equivalently (cf. (2.4)), the superbasic and basic components $\hat{\mu}$, $\hat{\nu}$, $\bar{\mu}(t)$, $\bar{\nu}(t)$, $\hat{d}^\mu$ and $\hat{d}^\nu$ of $\hat{\lambda}$, $\bar{\lambda}(t)$, and $\hat{d}$, respectively, should satisfy $\bar{\mu}(t) = \hat{\mu} + t\hat{d}^\mu$, $\bar{\nu}(t) = \hat{\nu} + t\hat{d}^\nu$ and

$$\begin{aligned} &\hat{d}_{j(k)} = \hat{d}^\mu_k \quad \text{for } k = 1, \cdots, s, \\ &\hat{d}_{l(i)} = \hat{d}^\nu_i \quad \text{for } i = 1, \cdots, m, \\ &\hat{d}_j = 0 \quad \text{for } j \in J_N, \\ &\hat{d}^\nu_i = -\sum_{k:j(k)\in J_i} \hat{d}^\mu_k \quad \text{for } i = 1, \cdots, m, \end{aligned} \tag{2.10}$$

and we only need to find $\hat{d}^\mu$ and $t_{\text{opt}} \geqq 0$ such that

$$\begin{aligned} &\bar{\mu}(t) = \hat{\mu} + t\hat{d}^\mu \quad \text{minimizes } w \text{ for } t = t_{\text{opt}} < +\infty, \quad \text{or} \\ &t_{\text{opt}} = +\infty \quad \text{and} \quad w(\bar{\mu}(t)) \downarrow -\infty \quad \text{as } t \to +\infty. \end{aligned} \tag{2.11}$$

As in Goldfarb and Idnani [G6], it will be convenient to choose $\hat{d}$ of the form

$$\hat{d} = \begin{bmatrix} \tilde{d}^\mu \\ 1 \end{bmatrix} \begin{matrix} \}s-1 \\ \}1 \end{matrix}, \tag{2.12}$$

and to associate with $\hat{\mu}$ and $\bar{\mu}(t)$ points $\hat{x} = x(\hat{\mu})$ and $x(\bar{\mu}(t))$, where

$$x(\mu) := -G^{-1}(g + A\mu) = -G^{-1}P\lambda(\mu) \tag{2.13}$$

(cf. (2.1), (2.7)). Partition A and b as follows:

$$A = [\tilde{A}, A_s] \quad \text{and} \quad b = \begin{bmatrix} \tilde{b} \\ b_s \end{bmatrix}, \tag{2.14}$$

where $\tilde{A}$ is $n\times(s-1)$ and $\tilde{b}$ is $(s-1)\times 1$, and let

$$\sigma_l(x)=b_s-A_s^Tx \tag{2.15}$$

for $l=j(s)$, so that the gradient

$$\nabla w(\hat{\mu})=\begin{bmatrix}\tilde{b}-\tilde{A}^T\hat{x}\\ \hat{\sigma}_l\end{bmatrix} \tag{2.16}$$

for $\hat{x}=x(\hat{\mu})$ and $\hat{\sigma}_l=\sigma_l(\hat{x})$ (see (2.6)). Consider the following assumptions:

$$\tilde{A}^Tx=\tilde{b}, \tag{2.17}$$

$$\hat{\sigma}_l<0, \tag{2.18}$$

$$\text{rank }(\tilde{A})=s-1,\quad \text{i.e., } \tilde{A} \text{ has full rank.} \tag{2.19}$$

The above relations and the Taylor expansion of (2.6) yield

$$w(\hat{\mu}+t\hat{d}^{\mu})=w(\hat{\mu})+t\hat{\sigma}_l+\frac{t^2}{2}|A_s+\tilde{A}\tilde{d}^{\mu}|^2_{G^{-1}},$$

so (2.11) is satisfied by

$$\tilde{d}^{\mu}=-\tilde{A}^*A_s, \tag{2.20}$$

$$\tilde{A}^*=(\tilde{A}^TG^{-1}\tilde{A})^{-1}\tilde{A}^TG^{-1}, \tag{2.21}$$

$$t_{\text{opt}}=-\hat{\sigma}_l/|A_s-\tilde{A}\tilde{A}^*A_s|^2_{G^{-1}} \tag{2.22}$$

with $t_{\text{opt}}=+\infty$ if the denominator vanishes; moreover,

$$t_{\text{opt}}<+\infty \quad \text{if and only if rank } (A)=s \tag{2.23}$$

(since $A_s=\tilde{A}\tilde{d}$ for some $\tilde{d}$ means that $\tilde{d}=\tilde{A}^*\tilde{A}\tilde{d}=\tilde{A}^*A_s=-\tilde{d}^{\mu}$ because $\tilde{A}^*\tilde{A}=I$), $t_{\text{opt}}>0$ and

$$w(\hat{\mu}+t\hat{d}^{\mu})<w(\hat{\mu}) \quad \text{for all } t\in(0;\, t_{\text{opt}}]. \tag{2.24}$$

When the algorithm moves from $\hat{\mu}$ to $\bar{\mu}(\hat{t})=\hat{\mu}+\hat{t}\hat{d}^{\mu}$ with some $\hat{t}\leqq t_{\text{opt}}$, we will save work by exploiting the following relations. By (2.12)-(2.14) and (2.20), $\bar{x}(\hat{t})=x(\mu(\hat{t}))$ and $\hat{x}=x(\hat{\mu})$ are related by

$$\bar{x}(\hat{t})=\hat{x}-\hat{t}z, \tag{2.25}$$

$$z=\tilde{H}A_s, \tag{2.26}$$

$$\tilde{H}=G^{-1}(I-\tilde{A}\tilde{A}^*). \tag{2.27}$$

Since $\tilde{A}^*\tilde{A}=I$, $\tilde{H}^T=\tilde{H}$, and $\tilde{H}\tilde{A}=0$, (2.17) yields

$$\tilde{A}^T\bar{x}(\hat{t})-b=-\hat{t}\tilde{A}^THA_s=0. \tag{2.28}$$

Next, since $z=\tilde{H}A_s$, $\tilde{H}=\tilde{H}^T$, and $\tilde{H}\tilde{A}=0$,

$$|(I-\tilde{A}\tilde{A}^*)A_s|^2_{G^{-1}}=z^T(I-\tilde{A}\tilde{A}^*)A_s=z^TA_s-A_s\tilde{H}\tilde{A}\tilde{A}^*A_s=z^TA_s,$$

so the expression (2.22) for t_{opt} gives

$$t_{\text{opt}}=-\hat{\sigma}_l/z^TA_s \tag{2.29}$$

and by (2.15) $\bar{\sigma}_l(\hat{t})=\sigma_l(\bar{x}(\hat{t}))$ and $\hat{\sigma}_l=\sigma_l(\hat{x})$ satisfy

$$\bar{\sigma}_l(\hat{t})=\hat{\sigma}_l+\hat{t}z^TA_s=\hat{\sigma}_l(1-\hat{t}/t_{\text{opt}}). \tag{2.30}$$

We may now state the method.

ALGORITHM 2.1.

Step 0. *Initialization.*

0.1. *Find an initial constrained stationary point* $\hat{\lambda}$: Set $\hat{\lambda}=0$; select $l(i)\in J_i$ and set $\hat{\lambda}_{l(i)}=c_i$ for $i=1,\cdots,m$; set $\hat{L}=\{l(i)\}_{i=1}^{m}$, $\hat{J}=\varnothing$, $s=0$, $J_N=J\setminus\hat{L}$, $\hat{\mu}=\varnothing$ (so that $\hat{\lambda}$ solves (2.2) with $J_N=\{j\in J\colon \hat{\lambda}_j=0\}$, $\hat{\nu}=c$ and there are no superbasics).

0.2. *Find the corresponding primal vector* $\hat{x}=x(\hat{\lambda})$: Set g by (2.8), $\tilde{H}=G^{-1}$, and $\hat{x}=-\tilde{H}g$ (cf. (2.13) and (2.7)).

Step 1. *Optimality test.*

1.1. *Find Lagrange multipliers of* (2.2): Set

$$\begin{aligned}
&\hat{v}_i=-a_{l(i)}+P_{l(i)}^T\hat{x} && \text{for } i=1,\cdots,m,\\
&\sigma_j=a_j-P_j^T\hat{x}+\hat{v}_i && \text{for } j\in J_i\cap J_N,\quad i=1,\cdots,m,\\
&\sigma_j=a_j-P_j^T\hat{x} && \text{for } j\in J_0\cap J_N.
\end{aligned}\tag{2.31}$$

1.2. *Check dual optimality* (*primal feasibility*): If $\sigma_j\geqq 0$ for all $j\in J_N$, stop (($\hat{x}$, $\hat{v}$) solves (1.2) and $\hat{\lambda}$ solves (1.3)). Otherwise, continue.

Step 2. *Active-set reduction.*

2.1. *Choose constraint to drop*: Find $l\in J_N$ such that $\sigma_l<0$. (Here it is possible to reduce the objective value of (2.2) by deactivating the constraint $\lambda_l=0$.) Delete l from J_N.

2.2. *Superbasic addition*: Set $\hat{\sigma}_l=\sigma_l$, increase s by 1, append l to $\hat{J}$ and 0 to $\hat{\mu}$, and update $\tilde{A}^*$ and $\tilde{H}$.

Step 3. *Search direction finding.* Find the superbasic direction $\hat{d}^{\mu}=\begin{bmatrix}\tilde{d}^{\mu}\\ 1\end{bmatrix}$, where $\tilde{d}^{\mu}=-\tilde{A}^*A_s$ if $s>1$ and $\hat{d}=[1]$ if $s=1$, and compute the dual direction $\hat{d}$ via (2.10). Find the primal direction $z=\tilde{H}A_s$.

Step 4. *Stepsize calculation.* Find the maximum feasible stepsize $t_{\max}=\max\{t\colon \bar{\lambda}(t)\geqq 0\}$ as

$$t_{\max}=\min[\infty,\min\{-\hat{\lambda}_j/\hat{d}_j\colon \hat{d}_j<0, j\in\hat{J}\cup\hat{L}\}].$$

Set the optimal stepsize t_{opt} (that minimizes $f(\bar{\lambda}(t))$ over all $t\geqq 0$) by (2.29). Set the actual stepsize $\hat{t}=\min\{t_{\max},t_{\text{opt}}\}$ (that minimizes $f(\bar{\lambda}(t))$ subject to $\bar{\lambda}(t)\geqq 0$). If $\hat{t}=+\infty$, stop ((1.3) is unbounded from below and (1.2) is infeasible). Otherwise, take steps in the primal and dual spaces: set $\bar{\mu}(t)=\hat{\mu}+\hat{t}\hat{d}^{\mu}$, $\bar{\nu}(t)=\hat{\nu}+\hat{t}\hat{d}^{\nu}$; set $\bar{x}(\hat{t})$ and $\bar{\sigma}_l(\hat{t})$ by (2.25) and (2.30); replace $\hat{\mu}$, $\hat{\nu}$, $\hat{x}$, $\hat{\sigma}_l$ by $\bar{\nu}(\hat{t})$, $\bar{\nu}(\hat{t})$, $\bar{x}(\hat{t})$, $\bar{\sigma}_l(\hat{t})$; and recover $\hat{\lambda}$ via (2.4) (so that $\hat{\lambda}=\bar{\lambda}(\hat{t})$ is feasible for (1.3) and (2.2)).

Step 5. *Active-set augmentation.* (Here J_N will be augmented if $\hat{t}=t_{\max}\leqq t_{\text{opt}}$ with all $\tilde{j}\in J_N$ such that $\hat{\lambda}_{\tilde{j}}=0$, i.e., the next subproblem (2.2) will include all the constraints of (1.3) that are active at $\hat{\lambda}$.)

5.1. If $\hat{\lambda}_j>0$ for all $j\in\hat{J}\cup\hat{L}$, go to Step 6.

5.2. Choose a variable $\hat{\lambda}_{\tilde{j}}=0$ to be made nonbasic that corresponds via (2.4) either to a superbasic $\hat{\mu}_q=0$ with $q<s$ or to a basic $\hat{\nu}_{\tilde{i}}=0$ with $1\leqq\tilde{i}\leqq m$. Augment J_N with $\tilde{j}$. If $\hat{\lambda}_{\tilde{j}}$ corresponds to $\hat{\mu}_q$, go to Step 5.4; otherwise, continue.

5.3. *Make the basic* $\hat{\nu}_{\tilde{i}}=0$ *nonbasic*: Find the smallest q such that $j(q)\in J_{\tilde{i}}$ and replace the $\tilde{i}$th basic with qth superbasic by setting $\hat{\nu}_{\tilde{i}}=\hat{\mu}_q$ and $l(\tilde{i})=j(q)$. (This choice of q facilitates updating certain matrix factorizations; see the next section.)

5.4. *Make the* q*th superbasic nonbasic.* Delete the qth entries from $\hat{J}$ and $\hat{\mu}$, and decrease s by 1. Update $\tilde{A}^*$ and $\tilde{H}$. Go to Step 5.1.

Step 6. *Subproblem optimality test.* If $\hat{\lambda}$ solves (2.2), i.e., if $\hat{t} = t_{\text{opt}}$, $s = 0$ or $j(s) \neq l$ (i.e., the last superbasic became basic), go to Step 1; otherwise, go to Step 3.

We have the following convergence result.

THEOREM 2.2 (i) *Conditions* (2.17)-(2.19) *hold at each execution of Step* 3, *and at Step* 1 $A^T\hat{x} = b$, $\nabla w(\hat{\mu}) = 0$, *and* rank $(A) = s$.

(ii) *Algorithm* 2.1 *terminates after a finite number of steps with either solutions* $(\hat{x}, \hat{v})$ *for* (1.2) *and* $\hat{\lambda}$ *for* (1.3), *or the information that* (1.2) *is infeasible and* (1.3) *is unbounded from below.*

Proof. (i) Use induction based on (2.23), (2.28), and (2.30) (see [K6] for details).

(ii) Invoke the classical argument that no J_N can enter Step 1 more than once, because $\hat{\lambda}$ solves (2.2), and then $f(\bar{\lambda}(\hat{t})) < f(\hat{\lambda})$ at Step 4 (cf. (2.24)) with $\hat{t} > 0$ and $t_{\max} > 0$ due to Step 5.

3. Updating the matrix factorization. Algorithm 2.1 requires the calculation of $\hat{d}^{\mu} = -\tilde{A}^* A_s$ and $z = \tilde{H} A_s$. The operators $\tilde{A}^*$ and $\tilde{H}$ need not be explicitly computed if we use the matrix-factorization technique of Goldfarb and Idnani [G6] with a modification of Powell [P3]. For completeness, we shall review it below.

The technique is based on the factorizations

$$G^{-1} = ZZ^T, \tag{3.1}$$

$$Z^T A = \begin{bmatrix} R \\ 0 \end{bmatrix} \begin{matrix} \}\tilde{s} \\ \}n - \tilde{s} \end{matrix}, \tag{3.2}$$

where Z is an $n \times n$ matrix, R is an $\tilde{s} \times s$ upper trapezoidal matrix, and $\tilde{s} = \min\{s, n\}$ (A is $n \times s$, $s \leqq n+1$). Partition Z as follows:

$$Z = [\tilde{Z}^{(1)}, \tilde{Z}^{(2)}], \tag{3.3}$$

where $\tilde{Z}^{(1)}$ and $\tilde{Z}^{(2)}$ consist of the first $s-1$ and last $n-s+1$ columns of Z. Consider the partitions

$$R = \begin{bmatrix} \tilde{R} & r \\ 0 & \rho \end{bmatrix} \quad \text{if } s \leqq n, \tag{3.4a}$$

$$R = [\tilde{R}, r] \quad \text{and} \quad \rho = 0 \quad \text{if } s = n+1, \tag{3.4b}$$

where $\tilde{R}$ is $(s-1) \times (s-1)$ upper triangular, r is an $(s-1)$-vector, and ρ is a scalar. Note that Z and $\tilde{R}$ are nonsingular, since G is positive definite, $\tilde{R} = \tilde{Z}^{(1)T} \tilde{A}$, and rank $(\tilde{A}) = s-1$. Substituting (3.1)-(3.4) in (2.21) and (2.27), we obtain $\tilde{A}^* = \tilde{R}^{-1} \tilde{Z}^{(1)T}$ and $\tilde{H} = \tilde{Z}^{(2)} \tilde{Z}^{(2)T}$, whereas (2.20) and (2.26) imply that

$$\tilde{R} \tilde{d}^{\mu} = -r, \tag{3.5}$$

$$z = \rho Z_{\tilde{s}}. \tag{3.6}$$

Note that, by (3.2), (3.4), and (3.6),

$$z^T A_s = \rho^2. \tag{3.7}$$

In the next section we shall also require the partition

$$Z = [Z^{(1)}, Z^{(2)}], \quad Z^{(1)} \in R^{n \times s}, \quad Z^{(2)} \in R^{n \times (n-s)},$$

and the vectors $y^{(g)} \in R^n$ and $y^{(b)} \in R^s$ defined by

$$y^{(g)} = Z^T g = \begin{bmatrix} Z^{(1)T} g \\ Z^{(2)T} g \end{bmatrix} =: \begin{bmatrix} y^{(g1)} \\ y^{(g2)} \end{bmatrix} \begin{matrix} \}s \\ \}n - s \end{matrix}, \tag{3.8}$$

$$R^T y^{(b)} = b. \tag{3.9}$$

Whenever the active set is revised the factors R and Z are updated, as computing them ab initio would be much too expensive. Initialization is trivial, since R is vacuous if $s=0$, whereas Z may be set to L^{-T} to satisfy (3.1), where LL^T is the Cholesky factorization of G. It turns out (see [G6], [K6], [P3]) that updating the factorization (3.2) is formally the same as updating the classical QR factorization of A, for which numerically stable methods with orthogonal Givens transformations are well known (see, e.g., Daniel et al. [D1], Gill et al. [G1]). Hence we skip details, noting only that the costs of updating $y^{(g)}$ and $y^{(b)}$ are negligible (cf. [K4], [R1]).

4. Modification and extensions. In this section we describe some extensions that are included in our Fortran implementation of the method.

First, we replace the objective function of (1.2) by

$$F(x, v)=\frac{u}{2}x^TGx+c^Tv,$$

where u is a positive weight. This form of F arises, for instance, in nondifferentiable optimization methods (see Kiwiel [K5]), when problem (1.2) is solved for several values of u. In theory, this extension amounts to replacing G by uG in all the preceding formulae. However, for efficiency and stability (especially when u is very small), it is better to handle u explicitly. Thus only relations (2.29), (2.25), and (2.30) are replaced by

$$\begin{aligned} &t_{\text{opt}}=-\hat{\sigma}_l u/z^TA_s=-\hat{\sigma}_l u/\rho^2,\\ &\bar{x}(\hat{t})=\hat{x}-(\hat{t}/u)z=\hat{x}-(\hat{t}\rho/u)Z_{\bar{s}},\\ &\bar{\sigma}_l(\hat{t})=\hat{\sigma}_l+\hat{t}z^TA_s/u=\hat{\sigma}_l+\hat{t}\rho^2/u, \end{aligned} \tag{4.1}$$

where we use (3.6) and (3.7) to express z and z^TA_s.

Second, to prevent cycling due to computer rounding errors when the constraints of (1.2) are ("nearly") degenerate at the solution, we may monitor the value of $f(\hat{\lambda})$ and terminate when it stops to decrease. To save work, we prefer to test for an increase in $F(\hat{x}, \hat{v})$ at Step 1 on every third major iteration, since $F(\hat{x}, \hat{v})=-f(\hat{\lambda})$ (by duality and the fact that $\hat{\lambda}$ solves (2.2)) and the calculation of $F(\hat{x}, \hat{v})$ is cheaper than that of $f(\hat{\lambda})$. Here a major iteration denotes a sequence of steps beginning and ending at Step 1, whereas a minor iteration starts at Step 3 and ends at Step 6.

Third, following Powell [P3], we observe that updating $\hat{x}$ by (4.1) may result in instability when the change in $\hat{x}$ is relatively large. Therefore, if the sum of l_1-norms of corrections to $\hat{x}$ exceeds $1+10\|\hat{x}\|_1$, we recalculate $\hat{x}$ and $\hat{\mu}$ at Step 1 as follows. We deduce from Theorem 2.2(i) and (2.6) that $\hat{x}$ and $\hat{\mu}$ satisfy the Kuhn-Tucker conditions

$$uG\hat{x}+A\hat{\mu}=-g \quad \text{and} \quad A^T\hat{x}=b \tag{4.2}$$

(for the problem $\min_x\{ux^TGx/2+g^Tx-\bar{a}: A^Tx=b\}$), so they can be computed (in about $n^2+s^2/2$ multiplications) as follows:

$$\begin{aligned} &\hat{x}=-\frac{1}{u}Z^{(2)}y^{(g2)}+Z^{(1)}y^{(b)},\\ &\hat{\mu}=-R^{-1}(y^{(g1)}-uy^{(b)}) \end{aligned} \tag{4.3}$$

(see Powell [P3] or Kiwiel [K6]). In practice $\hat{x}$ and $\hat{\mu}$ are recalculated only at earlier iterations.

Fourth, we use the quantities

$$\gamma_j = \frac{a_j - P_j^T \hat{x} + \hat{v}_i}{1 + |a_j| + \sum_{k=1}^n |P_{kj}\hat{x}_k| + |\hat{v}_i|}, \qquad j \in J_i, \quad i \geqq 1,$$

$$\gamma_j = \frac{a_j - P_j^T \hat{x}}{1 + |a_j| + \sum_{k=1}^n |P_{kj}\hat{x}_k|}, \qquad j \in J_0, \tag{4.4}$$

which are scaled versions of σ_j (cf. (2.31)), for the main stopping criterion. Step 1 calculates

$$\sigma_l = \min\{\sigma_j : j \in J_N\} \quad \text{and} \quad \gamma_r = \min\{\gamma_j : j \in J_N\}, \tag{4.5}$$

and terminates if $\gamma_r \geqq -\varepsilon_s$, where $\varepsilon_s \geqq 0$ is a user-provided tolerance. This test is less sensitive to problem scaling than $\sigma_l \geqq -\varepsilon_s$, but it requires about twice as many additions.

Fifth, the accuracy of the final solutions is improved by iterative refinement based on (4.2) (with corrections computed by a generalization of (4.3), (3.8), and (3.9); see Kiwiel [K6], and also Bartels et al. [B1], Powell [P3]).

Another extension may save work when the algorithm is repeatedly called to solve related problems. Thus suppose that, having solved (1.2), we want to solve a modified version of (1.2) in which certain columns of P and a^T corresponding to $j \in J_N$ have been deleted and some other columns have been added. If $\hat{J}$ and $\hat{L}$ are updated, the algorithm may start at Step 1. If the remaining components of a or u have been changed and $s > 0$, the algorithm may start at Step 3, provided the following modification of Steps 3 and 4 is used until Step 1 is entered. At Step 3 calculate the "Newton" multiplier $\bar{\mu} = -R^{-1}(y^{(g1)} + uy^{(b)})$ (see (4.3) and note that $y^{(b)} = R^{-T}b$ must be recalculated if b has changed), set $\hat{d}^\mu = \bar{\mu} - \hat{\mu}$ and compute $\hat{d}^\nu$ by (2.10). At Step 4 set $t_{\text{opt}} = 1$ (and do not update $\hat{x}$ and $\hat{\sigma}_l$). When Step 1 is reached (e.g., the initial $s = 0$), calculate $\hat{x}$ by (4.3). The same modification is used if c has changed, but first $\hat{\mu}$ and $\hat{v}$ are scaled so that $\hat{\lambda}$ is feasible for (2.2), and $y^{(g)} = Z^T g$ is recalculated. Moreover, if G and P have changed but the preceding $\hat{J}^0 = \hat{J}$ and $\hat{L}^0 = \hat{L}$ are good estimates for the final $\hat{J}$ and $\hat{L}$ (as, e.g., in successive QP methods for constrained optimization; see Goldfarb and Idnani [G6]), then the algorithm may first solve (1.2) with constraints $j \notin \hat{J}^0 \cup \hat{L}^0$ temporarily ignored, and then the full problem.

We shall now give the amount of work required by each step of the algorithm as the highest-order terms in the expressions for the number of multiplications. We assume the three-multiplication form of a Givens transformation (Daniel et al. [D1]).

Step 1 takes $n(p-s)$ multiplications to form $\hat{v}$ and σ, and n^2 to compute $F(\hat{x}, \hat{v})$ on every third large iteration. Step 2 takes n^2 to form $h = Z^T A_s$, and $3n(n-s)$ to update Z and R. Step 3 takes $s^2/2$ to solve $\tilde{R}\tilde{d}^\mu = -r$. The costs of Steps 4 and 6 are negligible. Step 5.4 takes $3(\tilde{s}-q)^2/2 + 3n(\tilde{s}-q)$ to update Z and R; thus on average Step 5.4 takes $\tilde{s}(3n+\tilde{s})/2$ if q is randomly, uniformly chosen between 1 and $\tilde{s}$.

Now suppose a typical major iteration involves two minor iterations, i.e., one superbasic addition and one deletion. Then the total number of multiplications has the value

$$W = np + 13n^2/3 + 3s^2/2 - 5ns/2.$$

Averaging over s gives an "average" operational count of

$$\bar{W} = np + 43n^2/12 \approx np + 3.58n^2.$$

The np term could be reduced in an extension that stops calculating further σ_j at Step 1 once a substantially negative σ_l has been found (cf. Goldfarb and Idnani [G6]).

The algorithm requires n^2 storage locations for Z and $\tilde{p}(\tilde{p}+1)/2$ for R, where $\tilde{p} = \min\{p-m, n+1\}$. Thus the total workspace required is of order $1.5n^2$.

5. Numerical experiments. In this section we describe numerical experiments with our method on some test problems.

First, consider the strictly convex QP problem

$$\text{(5.1a)} \qquad \text{minimize} \quad \tfrac{1}{2}x^T G x + P_1^T x - a_1$$

$$\text{(5.1b)} \qquad \text{subject to} \quad P_j^T x \leqq a_j \quad \text{for } j = 2, \cdots, q+1,$$

$$\text{(5.1c)} \qquad x^L \leqq x \leqq x^U,$$

where $q \geqq 1$ and $x^L \leqq x^U$. Although (5.1) is a special case of (1.1), for testing purposes we may consider an equivalent l_∞ exact penalty formulation of (5.1), which has the form of (1.1) with $X = \{x: x^L \leqq x \leqq x^U\}$, $m = 2$, $\phi_1(x) = P_1^T x - a_1$, $\phi_2(x) = \max\{P_j^T x - a_j: j = 2, \cdots, q+2\}$ with $P_{q+2} = 0$ and $a_{q+2} = 0$, $c_1 = 1$ and $c_2 = \bar{c} > 0$, where $\bar{c}$ is a penalty coefficient greater than the sum of Lagrange multipliers of (5.1b) (see, e.g., Mangasarian [M1]).

Our first examples are l_∞ exact penalty formulations of problems 1-20 of Powell [P3], with penalty coefficient $\bar{c} = 100$, 10, 10^4, 100, and 10^4 for problems 1-10, 11-13, 14-18, 19, and 20, respectively, except that we use larger $n = 9$, 12, 15 for problems 11-13, $n = 6$, 10, 15, 25, 50 for problems 14-18, and our problems 21-24 are obtained by setting $n = 20$, 30, 50, 70, and $\bar{c} = 10^4$ in problem 20 of Powell [P3]. Problems 25 and 26 have $n = 10$ and 15, respectively, $G_{ij} = 1/(i+j-1)$ for $i, j = 1, \cdots, n$, $P_1 = 0$, $q = 0$, $x_i^L = 1$ and $x_i^U = 10^{10}$ for $i = 1, \cdots, n$, with solution $x^* = (1, \cdots, 1)^T$. Moreover, if G seems to be (numerically) singular, each G_{ii} is increased by at most $\varepsilon_G = 2.2 \times 10^{-16}$ $(= 2.2E-16)$.

In order to measure the relative accuracy in the satisfaction of the Kuhn-Tucker conditions (2.1), we calculate γ_r,

$$\varepsilon_{KT1} = \max\{|\gamma_j|: j \in \hat{J}\},$$

$$\varepsilon_{KT2} = \max_{1 \leqq i \leqq n} \frac{|\sum_{j=1}^n G_{ij}\hat{x}_j + \sum_{j=1}^p P_{ij}\hat{\lambda}_j|}{\sum_{j=1}^n |G_{ij}\hat{x}_j| + \sum_{j=1}^p |P_{ij}\hat{\lambda}_j|}$$

(cf. (4.4), (4.5)), where $\hat{x}$, $\hat{v}$, and $\hat{\lambda}$ are the final primal and dual variables, and $\hat{J}$ is the final superbasic set.

The results of our computational tests are reported in Table 1. The algorithm was coded in Fortran in double precision on an IBM PC clone microcomputer with the relative machine accuracy $\varepsilon_M \approx 2.2E-16$. The accuracy tolerance ε_s of the stopping criterion was set to ε_M. Table 1 also gives the numbers of minor iterations and operations (multiplications + divisions + 10* square roots) performed by the algorithm, excluding the costs of the initial factorization of G.

The accuracy reported in Table 1 seems to be very good, except perhaps for the value $\varepsilon_{KT2} = 1$ in problem 2, for which $G\hat{x} = (0.04, -2.4E-35)^T$, $P\hat{\lambda} = (-0.04, 0)^T$, and only one component of $\hat{\lambda}$ is nonzero.

We may add that the results for l_1 exact penalty formulations (cf. Mangasarian [M1]) of problems 1-24 differ from those in Table 1 only by slightly increased operational counts.

Next, the algorithm was applied to the sequences of related problems given in Kiwiel [K4]. The final relative accuracy was about 10 times higher than that in [K4]. More importantly, the results showed virtually no accumulation of rounding errors. For example, for $n = 45$ the algorithm cycled 10 times through 92 problems (with $m = 1$

TABLE 1
Results for $l_\infty - QP$ test problems.

Problem	n	$-\gamma_r$	ε_{KT1}	ε_{KT2}	Operations	Iterations
1	2	0.0	0.0	0.0	146	1
2	2	0.0	0.0	1.0	213	3
3	3	0.0	0.0	$2.8E-17$	247	1
4	3	0.0	0.0	$7.4E-17$	275	1
5	5	$4.9E-17$	0.0	$8.3E-17$	621	1
6	5	0.0	0.0	$1.5E-33$	579	1
7	5	$3.4E-17$	$3.4E-17$	$9.5E-17$	1,465	5
8	5	0.0	0.0	$8.3E-17$	1,465	5
9	4	$8.1E-17$	$8.1E-17$	$7.4E-17$	750	4
10	15	0.0	0.0	$3.1E-19$	25,115	23
11	9	$8.5E-17$	$8.5E-17$	$6.7E-17$	15,248	36
12	12	$8.5E-17$	$8.5E-17$	$6.0E-17$	40,904	65
13	15	$4.9E-17$	$4.9E-17$	$8.1E-17$	101,522	107
14	6	$8.9E-17$	$5.4E-17$	$6.8E-17$	3,085	10
15	10	$5.4E-17$	$8.6E-17$	$1.2E-16$	7,643	9
16	15	$2.3E-16$	$1.5E-16$	$1.6E-16$	16,598	9
17	25	$1.7E-16$	$2.0E-16$	$2.0E-16$	59,633	16
18	50	$3.8E-16$	$2.3E-16$	$2.2E-16$	301,786	21
19	2	0.0	0.0	$1.8E-17$	803	10
20	10	$5.6E-17$	$5.6E-17$	$6.1E-17$	4,574	8
21	20	$8.9E-17$	$8.9E-17$	$6.2E-17$	29,487	19
22	30	0.0	0.0	$6.3E-17$	87,375	28
23	50	$6.7E-17$	$6.7E-17$	$1.1E-16$	371,108	48
24	70	$8.9E-17$	$8.9E-17$	$9.0E-17$	981,030	68
25	10	0.0	0.0	$1.1E-16$	9,311	24
26	15	0.0	0.0	$1.9E-16$	39,039	57

and $p=92$), terminating with $\gamma_r=-2.0E-16$, $\varepsilon_{KT1}=1.3E-16$, and $\varepsilon_{KT2}=1.4E-16$ after 4,462 iterations.

The algorithm is used as a QP subroutine in the author's software package for nondifferentiable optimization. Until now it has solved thousands of QP problems, and the results have been uniformly good. For instance, when the method of [K2, § 2.5] solved the TR48 problem of Lemarechal and Mifflin [L2] to the accuracy of 14 digits in the objective value, 150 QP problems with $n=48$ and $p\approx 60$ were solved in 717 iterations; thus on average each problem required only five iterations, although final problems had 48 active constraints of (1.2).

6. A comparison with related methods. Problems (1.2) and (1.3) are, of course, positive semidefinite QP problems, for which there exist several algorithms, e.g., Best and Ritter [B3], Fletcher [F1], Gill and Murray [G2], van de Panne and Whinston [V1], and Dantzig [D2] (see Pang [P1]). We shall call them primal (dual) when they are applied to (1.2) (to (1.3)). Best [B2] shows that under exact arithmetic all these algorithms are equivalent versions of a "general" active set method, which differ only in the manner in which they solve equality-constrained subproblems. A similar analysis shows that our algorithm is mathematically, but not computationally, equivalent to all these dual methods.

The discussion of advantages of dual methods over primal ones given by Goldfarb [G5], Goldfarb and Idnani [G6], and Powell [P3] may be extended to our setting. First, constraint degeneracies of (1.2) may upset the primal methods much more severely than the dual ones (for which the active constraints of (1.3) are always linearly

independent). Second, the primal methods cannot incorporate the facilities of § 4 for solving related problems. Therefore, the dual methods seem to be more suitable for solving sequences of related problems (1.2) with (possibly) degenerate constraints, such as those arising in nondifferentiable optimization algorithms (see, e.g., Kiwiel [K2]).

To compare our algorithm with the remaining dual methods, we note that they cannot exploit the "least-squares" structure of the Hessian $P^TG^{-1}P$ of (1.3), and this may cause an unnecessary loss of accuracy (see, e.g., Gill and Murray [G2], Gill et al. [G4], Fletcher [F2]). (The "squaring effect" usually associated with P^TP when $G=I$ may be even more harmful for $P^TG^{-1}P$ when G is ill conditioned.) Moreover, none of these algorithms seems to be competitive with our method in terms of workspace and computational effort per iteration. For example, subroutine QPSOL of Gill et al. [G3], which implements the algorithm of Gill and Murray [G2], would require additional storage of E and workspace of order $2p^2$, which is much larger than our $1.5n^2$ when $p=n+2m$ and $m \geqq n$ as considered by Ruszczyński [R1].

The few algorithms due to Kiwiel [K4], Ruszczyński [R1], and Stoer [S1] that take account of the least-squares structure of (1.3) are restricted to $G=I$. In comparison with our method, Stoer's algorithm requires significantly more workspace and work per iteration, Ruszczyński's method requires more effort per iteration (if more than one reorthogonalization step occurs) and uses fewer stable updates of $\hat{x}$, whereas the method of [K4] can only handle $m=1$. Moreover, these three algorithms require the determination of potential singularity of certain matrices, and in practice this may pose problems due to rounding errors.

To sum up, it seems that our algorithm can compete with both the general-purpose methods and the specialized ones for the case $G=I$.

7. Conclusions. We have presented a highly specialized dual active set method for certain QP programs that arise as subproblems in many algorithms for nondifferentiable and nonlinearly constrained optimization. Its implementation, which combines the reduced gradient technique with the ideas of Goldfarb and Idnani [G6] and Powell [P3], seems to be both accurate and efficient.

We hope that the work of this paper will facilitate the implementation of many nondifferentiable optimization algorithms.

Acknowledgments. I thank the associate editor and two referees for their useful suggestions.

REFERENCES

[B1] R. H. BARTELS, G. H. GOLUB, AND M. A. SAUNDERS, *Numerical techniques in mathematical programming*, in Nonlinear Programming, J. B. Rosen, O. L. Mangasarian, and K. Ritter, eds., Academic Press, New York, 1970, pp. 123-176.

[B2] M. J. BEST, *Equivalence of some quadratic programming algorithms*, Math. Programming, 30 (1984), pp. 71-87.

[B3] M. J. BEST AND K. RITTER, *An effective algorithm for quadratic minimization problems*, MRC Technical Summary Report 1961, University of Wisconsin, Madison, WI, 1976.

[D1] J. W. DANIEL, W. B. GRAGG, L. C. KAUFMAN, AND G. W. STEWART, *Reorthogonalization and stable algorithms for updating the Gram-Schmidt* QR *factorization*, Math. Comp., 30 (1976), pp. 772-795.

[D2] G. B. DANTZIG, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[F1] R. FLETCHER, *A general quadratic programming algorithm*, J. Inst. Math. Appl., 7 (1971), pp. 76-91.

[F2] ———, *Practical Methods of Optimization* II: *Constrained Optimization*, John Wiley, New York, 1981.

[G1] P. E. GILL, G. H. GOLUB, W. MURRAY, AND M. A. SAUNDERS, *Methods for modifying matrix factorizations*, Math. Comp., 28 (1974), pp. 505-535.

[G2] P. E. GILL AND W. MURRAY, *Numerically stable methods for quadratic programming*, Math. Programming, 14 (1978), pp. 349-372.

[G3] P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND M. H. WRIGHT, *User's guide for* SOL/QPSOL: *a Fortran package for quadratic programming*, Report SOL 83-7, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA, 1983.

[G4] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, London, 1981.

[G5] D. GOLDFARB, *Numerically stable approaches to linearly constrained optimization*, in Nonlinear Optimization 1981, M. J. D. Powell, ed., Academic Press, London, 1982, pp. 141-146.

[G6] D. GOLDFARB AND A. IDNANI, *A numerically stable dual method for solving strictly convex quadratic programs*, Math. Programming, 27 (1983), pp. 1-33.

[K1] K. C. KIWIEL, *A quadratic approximation method for minimizing a class of quasidifferentiable functions*, Numer. Math., 45 (1984), pp. 411-430.

[K2] ———, *Methods of Descent for Nondifferentiable Optimization*, Lecture Notes in Mathematics 1133, Springer-Verlag, Berlin, 1985.

[K3] ———, *A linearization method for minimizing certain quasidifferentiable functions*, Math. Programming Stud., 29 (1986), pp. 85-94.

[K4] ———, *A method for solving certain quadratic programming problems arising in nonsmooth optimization*, IMA J. Numer. Anal., 6 (1986), pp. 137-152.

[K5] ———, *An ellipsoid trust region bundle method for nonsmooth convex minimization*, Tech. Report 586, Institut National de Recherche en Informatique et en Automatique, Le Chesnay, Rocquencourt, France, 1986; SIAM J. Control Optim., 27 (1989), to appear.

[K6] ———, *A dual method for certain positive semidefinite quadratic programming problems*, Prace IBS PAN 145, Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland, 1987.

[L1] C. LEMARECHAL, *Nonsmooth optimization and descent methods*, Report RR-78-4, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1977.

[L2] C. LEMARECHAL AND R. MIFFLIN, EDS., *Nonsmooth Optimization*, Pergamon Press, Oxford, 1978.

[M1] O. L. MANGASARIAN, *Sufficiency of exact penalty minimization*, SIAM J. Control Optim., 23 (1985), pp. 30-37.

[M2] R. MIFFLIN, *Better than linear convergence and safeguarding in nonsmooth optimization*, in System Modelling and Optimization, P. Thoft-Christensen, ed., Lecture Notes in Control and Information Sciences 59, Springer-Verlag, Berlin, 1984, pp. 321-330.

[M3] B. A. MURTAGH AND M. A. SAUNDERS, *Large-scale linearly constrained optimization*, Math. Programming, 14 (1978), pp. 41-72.

[P1] J. C. PANG, *An equivalence between two algorithms for quadratic programming*, Math. Programming, 20 (1981), pp. 152-165.

[P2] M. J. D. POWELL, *Variable metric methods for constrained optimization*, in Mathematical Programming, The State of the Art, Bonn 1982, A. Bachem, M. Grötschel, and B. Korte, eds., Springer-Verlag, Berlin, 1983, pp. 288-311.

[P3] ———, *On the quadratic programming algorithm of Goldfarb and Idnani*, Math. Programming Stud., 25 (1985), pp. 46-61.

[R1] A. RUSZCZYŃSKI, *A regularized decomposition method for minimizing a sum of polyhedral functions*, Math. Programming, 35 (1986), pp. 309-333.

[S1] J. STOER, *On the numerical solution of constrained least-squares problems*, SIAM J. Numer. Anal., 8 (1971), pp. 382-411.

[V1] C. VAN DE PANNE AND A. WHINSTON, *The symmetric formulation of the simplex method for quadratic programming*, Econometrica, 37 (1969), pp. 507-527.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 187-204, January 1989
© 1989 Society for Industrial and Applied Mathematics
014

MONTE CARLO, CONTROL VARIATES, AND STOCHASTIC ORDERING*

GEORGE S. FISHMAN†

Abstract. Using prior information about the stochastic orderings between a phenomenon of interest and other ancillary phenomena whose population parameters are known in a Monte Carlo experiment, this paper derives an unbiased control variate estimator and an exact $100\times(1-\alpha)$ percent confidence interval for a proportion and, more generally, for a distribution function for finite sample size K. These results substantially improve on the more traditional control variate variance-reduction method for which, in the case of finite K, neither an unbiased point estimator is generally available nor, except in the case of multinormal data, is an exact $100\times(1-\alpha)$ percent confidence interval available.

The paper also derives upper bounds on the estimator's variance and coefficient of variation and a lower bound on the achievable variance reduction in terms of the known parameters for the ancillary phenomena. Computing these bounds prior to sampling provides useful guidance rarely available with other variance reduction techniques. In the case of estimating a discrete distribution function, the paper also shows how to derive a variance-reduced estimator of the population mean from the estimated distribution function, and gives the variance of this sample mean to order $1/K$. An example illustrates the technique for estimating the distribution of maximal flow in a flow-network problem with random arc capacities.

Key words. control variates, maximal flow, Monte Carlo, multistate reliability, variance reduction

AMS(MOS) subject classifications. 60, 90

Introduction. When estimating a parameter of a problem by the Monte Carlo method, available prior information usually enables us to improve the statistical efficiency of the estimation procedure. Techniques for achieving this improvement are called *variance-reduction methods*, and they differ considerably in the way they gain their advantages. For example, a user of the *importance sampling technique* draws data from a sampling distribution designed, on the basis of prior information, to reduce the variance of each observation while preserving its mean. The user of the *stratified sampling technique* draws observations from *partitions* of the sample space, and then forms a linear combination of the resulting sample means as the estimator. By using prior information to determine the optimal relative number of observations per partition, we achieve a smaller variance for the estimator than the variance that crude Monte Carlo allows. The *antithetic variate technique* derives its benefit by inducing negative correlation between sample outcomes taken in pairs, when it is known a priori that certain monotone relationships hold.

This paper concerns the *control variate technique* of variance reduction. Instead of gaining an advantage by modifying the sampling procedure, on each trial this technique collects ancillary sample data on phenomena whose true means are known, and then uses a regression method to derive an estimator of the parameter of interest with reduced variance. Since the only additional work in using the technique is the collection of the additional data and, at the end of sampling, deriving the control variate estimator, the incremental cost is usually relatively small.

Originally, Fieller and Hartley [1] proposed using the control variate technique in a Monte Carlo study designed to estimate the relative frequencies in an unknown population. More recently, Wilson [8] has summarized the known theoretical results for the method, noting that for the finite sample size case control variate estimators are generally not unbiased, and only in the case of normally distributed observations

* Received by the editors December 15, 1986; accepted for publication (in revised form) May 20, 1988. This research was supported by the Air Force Office of Scientific Research under grant AFOSR-84-0140.

† Department of Operations Research, University of North Carolina, Chapel Hill, North Carolina 27514.

does an exact distribution theory exist for deriving confidence intervals for the parameter of interest.

The present paper represents a contribution to the theory of control variates as applied to the estimation of a proportion, or more generally, a distribution function, and when information is available to the experimenter, prior to sampling, on *stochastic orderings* between the phenomenon of interest and ancillary phenomena with known population parameters. In particular, the paper derives an unbiased point estimator (§ 2) and a $100\times(1-\alpha)$ percent confidence interval (§ 3), for the parameter, that hold for finite sample size K.

Section 2 also uses this prior information to derive upper bounds on the variance and coefficient of variation of the estimator and a lower bound on the achievable variance reduction. This information is especially valuable before sampling begins. It tells the experimenter what the least possible benefit of the control variate technique is and it enables him to estimate an upper bound on the sample size needed to achieve, say, a specified variance or coefficient of variation.

The results for a single parameter extend easily to the multiparameter case. In particular, § 4 describes how they apply to the estimation of a distribution function (d.f.). A variance reduction is achieved for all estimated ordinates of the d.f., a notable improvement over most earlier applications of Monte Carlo variance-reducing methods that focused on estimating a single ordinate of the d.f.

The proposed technique offers yet an additional benefit. For a discrete d.f., § 5 shows how we can use the variance-reduced ordinates of the d.f. to derive an unbiased estimator of the population mean. It also gives the variance of this estimator to order $1/K$.

The proposed technique can be especially useful in network problems where the formal specification of a stochastic network often makes ordering information readily accessible to an analyst. Examples include binary-state reliability systems (Fishman [2]) and multistate reliability systems whose performances are measured in terms of flow (Fishman [3]). To illustrate the technique, § 6 describes the estimation of the complementary distribution function of maximal flow in a flow network of 10 nodes and 25 arcs, where the capacities of the arcs are subject to stochastic variation.

1. The control variate method. Consider a random variable Z_0 whose unknown mean μ_0 we wish to estimate. Let Z_1 and Z_2 denote random variables with known means μ_1 and μ_2, respectively, and observe that, for any fixed β_1 and β_2,

$$Y(\beta_1,\beta_2)=Z_0+\beta_1(Z_1-\mu_1)+\beta_2(Z_2-\mu_2) \tag{1}$$

is an unbiased estimator of μ_0. Here Z_1 and Z_2 are the *control variates* and β_1 and β_2 are *free* parameters whose values can be chosen to meet a specified objective.

Let $\sigma_{ij}=\operatorname{cov}(Z_i,Z_j)$, $i,j=0,1,2$. Then the choice

$$\begin{aligned}\beta_1^*&=(\sigma_{02}\sigma_{12}-\sigma_{01}\sigma_{22})/(\sigma_{11}\sigma_{22}-\sigma_{12}^2),\\ \beta_2^*&=(\sigma_{01}\sigma_{12}-\sigma_{02}\sigma_{11})/(\sigma_{11}\sigma_{22}-\sigma_{12}^2)\end{aligned} \tag{2}$$

minimizes var $Y(\beta_1,\beta_2)$. If the σ_{ij} were known, then using β_1^* and β_2^* in (2) with K independent samples $\{Z_0^{(k)},Z_1^{(k)},Z_2^{(k)};\ k=1,\cdots,K\}$ would lead to the estimator

$$\bar{Y}_K(\beta_1^*,\beta_2^*)=\bar{Z}_{0K}+\beta_1^*(\bar{Z}_{1K}-\mu_1)+\beta_2^*(\bar{Z}_{2K}-\mu_2), \tag{3}$$

where

$$\bar{Z}_{iK}=\frac{1}{K}\sum_{k=1}^{K}Z_i^{(k)},\qquad i=0,1,2$$

with

$$\text{var } \bar{Y}_K(\beta_1^*, \beta_2^*) = \text{var } Y(\beta_1^*, \beta_2^*)/K = \sigma_{00}(1-\rho^2)/K,$$

ρ denoting the multiple correlation coefficient between Z_0 and the control variates Z_1 and Z_2. Thus var $\bar{Y}_K(0,0)/\text{var } \bar{Y}_K(\beta_1^*, \beta_2^*) = 1/(1-\rho^2)$ gives the number of observations that we would need in crude Monte Carlo sampling to achieve the same variance that obtains with one observation using the control variate estimator (3). Observe that high correlation is the key to success here.

When, as is common, the covariances are unknown, they too are estimated from the sample data and substituted into (2). Being random, the resulting estimators $\hat{\beta}_1^*$ and $\hat{\beta}_2^*$ generally induce a bias in $\bar{Y}_K(\hat{\beta}_1^*, \hat{\beta}_2^*)$ as an estimator of μ_0. To reduce this bias, Tocher [7] suggests using the jackknife method. However, it is not at all clear that the associated work and increased variance that the jackknife method induces leave us better off than we are with crude Monte Carlo sampling.

In the case of multinormal data, Lavenberg and Welch [5] give an exact distribution theory for each K that enables us to compute a confidence interval for μ_0. Rubinstein and Marcus [6] extend these results to the multinormal data/multivariable response case, where more than one parameter is to be estimated. Wilson [8] gives asymptotically ($K \to \infty$) valid confidence regions for the multivariable case for nonnormal data.

2. Binary control variates with stochastic ordering. Now consider the case in which Z_0, Z_1, and Z_2 are binary (0-1) random variables so that $\sigma_{11} = \mu_1(1-\mu_1)$ and $\sigma_{22} = \mu_2(1-\mu_2)$. While reducing the number of parameters that need to be estimated in (2), this specification by itself offers little benefit. However, the results of Lemma 1.1 show that the presence of an ordering relationship among Z_0, Z_1, Z_2 can be of considerable value.

LEMMA 1.1. *Let Ω denote the space of all elementary events ω and let $Z_0 = Z_0(\omega)$, $Z_1 = Z_1(\omega)$, and $Z_2 = Z_2(\omega)$ denote binary 0-1 random variables with means μ_0, μ_1, and μ_2, respectively, and suppose that the stochastic ordering $Z_1 \leqq Z_0 \leqq Z_2$ holds with probability one. Then*

(i) $\mu_1 \leqq \mu_0 \leqq \mu_2$;

(ii) $\sigma_{01} = \mu_1(1-\mu_0)$, $\sigma_{02} = \mu_0(1-\mu_2)$, *and* $\sigma_{12} = \mu_1(1-\mu_2)$.

Proof. Since $E(Z_1|Z_0) = 0$ and $E(Z_1|Z_0) = \lambda_1 \leqq 1$, $EZ_1 = \mu_1 = \lambda_1\mu_0 \leqq \mu_0$. Analogous results hold for $E(Z_0|Z_2 = 0)$ and $E(Z_0|Z_2 = 1)$ so that $\mu_0 \leqq \mu_2$, establishing (i). Since $Z_0Z_1 = \min(Z_0, Z_1) = Z_1$, $Z_0Z_2 = \min(Z_0, Z_2) = Z_0$ and $Z_1Z_2 = \min(Z_1, Z_2) = Z_1$, result (ii) follows.

Observe that we have

$$\beta_1^* = (\mu_0 - \mu_2)/(\mu_2 - \mu_1), \qquad \beta_2^* = (\mu_1 - \mu_0)/(\mu_2 - \mu_1), \tag{4}$$

where, since the use of control variates requires that μ_1 and μ_2 be known, μ_0 is the only *unknown* parameter. We now show how this reduction in the number of unknown parameters greatly facilitates the use of the control variate technique.

THEOREM 1. *Consider the estimators*

$$\bar{Z}_{0K} = \bar{Z}_{0K} + \hat{\beta}_1^*(\bar{Z}_{1K} - \mu_1) + \hat{\beta}_2^*(\bar{Z}_{2K} - \mu_2), \tag{5}$$

$$\ddot{Z}_{0K} = [K(\mu_2 - \mu_1)\tilde{Z}_{0K} - \mu_1]/[(K+1)(\mu_2 - \mu_1) - 1] \tag{6}$$

where

$$\hat{\beta}_1^* = (\bar{Z}_{0K} - \mu_2)/(\mu_2 - \mu_1), \qquad \hat{\beta}_2^* = (\mu_1 - \bar{Z}_{0K})/(\mu_2 - \mu_1). \tag{7}$$

Then

$$\text{(8)} \quad \text{(i)} \quad E\tilde{Z}_{0K} - \mu_0 = [\mu_0 - (\mu_0 - \mu_1)/(\mu_2 - \mu_1)]/K,$$

$$\text{(9)} \quad \text{(ii)} \quad \operatorname{var} \tilde{Z}_{0K} = (a_1\mu_0^2 + a_2\mu_0 + a_3)/K + (b_1\mu_0^2 + b_2\mu_0 + b_3)/K^2 + (c_1\mu_0^2 + c_2\mu_0 + c_3)/K^3$$

where

$$\begin{aligned} a_1 &= -1/(\mu_2 - \mu_1), \\ a_2 &= (\mu_1 + \mu_2)/(\mu_2 - \mu_1), \\ a_3 &= -\mu_1\mu_2/(\mu_2 - \mu_1), \\ b_1 &= (1 - \mu_2 + \mu_1)(3 - 2\mu_2 + 2\mu_1)/(\mu_2 - \mu_1)^2, \\ b_2 &= -(1 - \mu_2 + \mu_1)(5\mu_1 + \mu_2)/(\mu_2 - \mu_1)^2, \\ b_3 &= [\mu_1^2 + 2\mu_1\mu_2(1 - \mu_2 + \mu_1)]/(\mu_2 - \mu_1)^2, \\ c_1 &= (1 - \mu_2 + \mu_1)[6(\mu_2 - \mu_1) - 4]/(\mu_2 - \mu_1)^2, \\ c_2 &= (9\mu_1 - 3\mu_2 + 1 + 10\mu_1^2 + 2\mu_2^2 - 12\mu_1\mu_2)/(\mu_2 - \mu_1)^2, \\ c_3 &= -[\mu_1(1 - 2\mu_2) + 4\mu_1^2]/(\mu_2 - \mu_1)^2, \end{aligned} \tag{10}$$

$$\text{(11)} \quad \text{(iii)} \quad E\ddot{Z}_{0K} = \mu_0,$$

$$\text{(12)} \quad \text{(iv)} \quad \operatorname{var} \ddot{Z}_{0K} = \{K(\mu_2 - \mu_1)/[(K+1)(\mu_2 - \mu_1) - 1]\}^2 \operatorname{var} \tilde{Z}_{0K} \geqq \operatorname{var} \tilde{Z}_{0K},$$

$$\text{(13)} \quad \text{(v)} \quad V(\tilde{Z}_{0K}) = \frac{1}{d^2 + a}[a\tilde{Z}_{0K}^2 + (bd - 2ae)(\tilde{Z}_{0K} - e) + d^2c + ae^2]$$

is an unbiased estimator of $\operatorname{var} \tilde{Z}_{0K}$, *where*

$$\begin{aligned} a &= a_1/K + b_1/K^2 + c_1/K^3, \qquad b = a_2/K + b_2/K^2 + c_2/K^3, \\ c &= a_3/K + b_3/K^2 + c_3/K^3, \qquad d = 1 + 1/K - 1/K(\mu_2 - \mu_1), \\ e &= \mu_1/K(\mu_2 - \mu_1). \end{aligned} \tag{14}$$

Proof. Write $\Delta\bar{Z}_{iK} = \bar{Z}_{iK} - \mu_i$, $i = 0, 1, 2$ so that

$$\tilde{Z}_{0K} - \mu_0 = \Delta\bar{Z}_{0K} + \Delta\bar{Z}_{1K}(\mu_0 - \mu_2)/(\mu_2 - \mu_1) + \Delta\bar{Z}_{2K}(\mu_1 - \mu_0)/(\mu_2 - \mu_1) + \Delta\bar{Z}_{0K}(\Delta\bar{Z}_{1K} - \Delta\bar{Z}_{2K})/(\mu_2 - \mu_1).$$

Since we have $Z_i^{(s)}Z_j^{(s)} = Z_i^{(s)}$, $Z_i^{(s)}Z_j^{(s)}Z_k^{(s)} = Z_i^{(s)}$, and $Z_i^{(s)}Z_j^{(s)}Z_k^{(s)}Z_m^{(s)} = Z_i^{(s)}$ for $i \leqq j \leqq k \leqq m$, and $s = 1, \cdots, K$, we have

$$E(\Delta\bar{Z}_{iK}\Delta\bar{Z}_{jK}) = \mu_i(1 - \mu_j)/K \quad \text{if } Z_i \leqq Z_j \text{ w.p. 1},$$

$$E(\Delta\bar{Z}_{iK}\Delta\bar{Z}_{jK}\Delta\bar{Z}_{kK}) = \mu_i(1 - 2\mu_j)(1 - \mu_k)/K^2 \quad \text{if } Z_i \leqq Z_j \leqq Z_k \text{ w.p. 1},$$

$$E(\Delta\bar{Z}_{iK}\Delta\bar{Z}_{jK}\Delta\bar{Z}_{kK}\Delta\bar{Z}_{mK}) = [\mu_i(1 - \mu_j)\mu_k(1 - \mu_m) + 2\mu_i(1 - \mu_k)\mu_j(1 - \mu_m)]/K^2 + \mu_i(1 - \mu_m)[1 - 2\mu_k + 2\mu_j(3\mu_k - 2)]/K^3$$

$$\text{if } Z_i \leqq Z_j \leqq Z_k \leqq Z_m \text{ w.p. 1}.$$

Then (i) and (ii) follow from substitution into $E\tilde{Z}_{0K} - \mu_0$ and $E(\tilde{Z}_{0K} - \mu_0)^2$, rearrangement of terms, and by noting that

$$\operatorname{var} \tilde{Z}_{0K} = E(\tilde{Z}_{0K} - \mu_0)^2 - (E\tilde{Z}_{0K} - \mu_0)^2.$$

Result (iii) follows by substituting $E\tilde{Z}_{0K}$ in (i) into

$$E\ddot{Z}_{0K} = [K(\mu_2 - \mu_1)E\tilde{Z}_{0K} - \mu_1]/[(K+1)(\mu_2 - \mu_1) - 1] \tag{15}$$

and (iv) is obvious. Taking expectations with respect to $\tilde{\tilde{Z}}_{0K}^2$ and $\tilde{\tilde{Z}}_{0K}$, we obtain (v). Observe that $V(\ddot{Z}_{0K}) = \{K(\mu_2-\mu_1)/[(K+1)(\mu_2-\mu_1)-1]\}^2 V(\tilde{\tilde{Z}}_{0K})$ is an unbiased estimator of var $\ddot{Z}_{0K}$.

The results in Theorem 1 enable us to describe the properties for var $\ddot{Z}_{0K}$, the coefficients of variation

$$\begin{aligned} &\gamma_K(\ddot{Z}_{0K}) = (\text{var } \ddot{Z}_{0K})^{1/2}/E\ddot{Z}_{0K}, \\ &\gamma_K(1-\ddot{Z}_{0K}) = (\text{var } \ddot{Z}_{0K})^{1/2}/(1-E\ddot{Z}_{0K}), \end{aligned} \tag{16}$$

and the variance ratio

$$\Lambda(\mu_0, K) = \text{var } \bar{Z}_{0K}/\text{var } \ddot{Z}_{0K} \tag{17}$$

for finite K and as $K \to \infty$. In particular, the ratio (17) compares the accuracies of the crude Monte Carlo estimator and the control variate estimator $\ddot{Z}_{0K}$. A ratio greater than unity favors the control variate technique. Theorem 2 concerns the asymptotic case and Theorems 3 and 4, the finite case.

THEOREM 2. *For* $\ddot{Z}_{0K}$ *in* (6)

$$\text{(i)} \quad \lim_{K\to\infty} K \text{ var } \ddot{Z}_{0K} = (\mu_2-\mu_0)(\mu_0-\mu_1)/(\mu_2-\mu_1) \leqq (\mu_2-\mu_1)/4, \tag{18a}$$

$$\text{(ii)} \quad \lim_{K\to\infty} K\gamma_K^2(\ddot{Z}_{0K}) \leqq (\mu_2-\mu_1)/4\mu_1\mu_2, \tag{18b}$$

$$\text{(iii)} \quad \lim_{K\to\infty} K\gamma_K^2(1-\ddot{Z}_{0K}) \leqq (\mu_2-\mu_1)/4(1-\mu_1)(1-\mu_2), \tag{18c}$$

$$\begin{aligned} \text{(iv)} \quad \lim_{K\to\infty} \Lambda(\mu_0, K) &\geqq [(1-\mu_1)^{1/2}\mu_2^{1/2} + \mu_1^{1/2}(1-\mu_2)^{1/2}]/[\mu_2^{1/2}(1-\mu_1)^{1/2} \\ &\quad -\mu_1^{1/2}(1-\mu_2)^{1/2}] \\ &\geqq 1. \end{aligned} \tag{18d}$$

Proof. Result (i) is immediate. For a_1, a_2, and a_3 as defined in (10), observe that

$$\lim_{K\to\infty} K \text{ var } \ddot{Z}_{0K} = a_1\mu_0^2 + a_2\mu_0 + a_3,$$

which gives the equality in (i). The upper bound follows for the worst case $\mu_0 = (\mu_1+\mu_2)/2$. The results in (ii) and (iii) follow from maximizing $(\mu_2-\mu_0)\times(\mu_0-\mu_1)/\mu_0^2(\mu_2-\mu_1)$ and $(\mu_2-\mu_0)(\mu_0-\mu_1)(1-\mu_0)^2(\mu_2-\mu_1)$ with respect to μ_0, subject to $\mu_1 \leqq \mu_0 \leqq \mu_2$. To obtain (iv), we minimize $\mu_0(1-\mu_0)(\mu_2-\mu_1)/(\mu_2-\mu_0)(\mu_0-\mu_1)$ with respect to μ_0, subject to the same constraints.

Results (ii), (iii), and (iv) are of most interest. For example, if we wish to guarantee a coefficient of variation $\gamma_K(\ddot{Z}_{0K}) \leqq \lambda$, then for sufficiently small λ, we would take

$$K = (\mu_2-\mu_1)/4\mu_1\mu_2\lambda^2$$

replications. For $\gamma_K(1-\ddot{Z}_{0K}) \leqq \lambda$ we would take

$$K = (\mu_2-\mu_1)/4(1-\mu_1)(1-\mu_2)\lambda^2$$

replications. These are worst-case results and assume that the asymptotic results (ii) and (iii) hold. A more precise evaluation is possible for finite K using the results to be presented in Theorem 3. Regardless of approach, the computed sample sizes provide useful information to an analyst prior to the undertaking of any sampling.

Result (iv) shows the smallest possible variance ratio that this control variate method can induce. Observe that as K becomes large, we would have to take $K\Lambda(\mu_0, K)$ replications to achieve the same variance for $\bar{Z}_{0K}$ that we achieve for $\ddot{Z}_{0K}$ with just K replications using the control variate technique.

Since Theorem 1 makes clear that for some μ_1 and μ_2 the rate of convergence to the asymptotic results in Theorem 2 can be slow, it is of interest to study the behavior of $\ddot{Z}_{0K}$ for finite K.

THEOREM 3. *For a, b, and c in* (14) *and for every fixed K, we have the following*:

(i) *If* $a<0$, var $\ddot{Z}_{0K}$ *has its maximum at*

$$\begin{aligned}\mu_0 &= -b/2a \quad \text{if } \mu_1 \leqq -b/2a \leqq \mu_2,\\ &= \mu_1 \quad \text{if } -b/2a < \mu_1,\\ &= \mu_2 \quad \text{if } -b/2a > \mu_2.\end{aligned}$$

Otherwise, the maximum is $\max_{\mu_0=\mu_1,\mu_2} \operatorname{var} \ddot{Z}_{0K}$.

(ii) *If* $c<0$, $\gamma_K(\ddot{Z}_{0K})$ *has its maximum at*

$$\begin{aligned}\mu_0 &= -2c/b \quad \text{if } \mu_1 \leqq -2c/b \leqq \mu_2,\\ &= \mu_1 \quad \text{if } -2c/b < \mu_1,\\ &= \mu_2 \quad \text{if } -2c/b > \mu_2.\end{aligned}$$

Otherwise, the maximum is $\max_{\mu_0=\mu_1,\mu_2} \gamma(\ddot{Z}_{0K})$.

(iii) *If* $a+b+c<0$, $\gamma_K(1-\ddot{Z}_{0K})$ *has its maximum at*

$$\begin{aligned}\mu_0 &= -(b+2c)/(2a+b) \quad \text{if } \mu_1 \leqq -(b+2c)/(2a+b) \leqq \mu_2,\\ &= \mu_1 \quad \text{if } -(b+2c)/(2a+b) < \mu_1,\\ &= \mu_2 \quad \text{if } -(b+2c)/(2a+b) > \mu_2.\end{aligned}$$

Otherwise, the maximum is $\max_{\mu_0=\mu_1,\mu_2} \gamma(1-\ddot{Z}_{0K})$.

Proof. Let $w=\{K(\mu_2-\mu_1)/[(K+1)(\mu_2-\mu_1)-1]\}^2$. The results follow from maximizing var $\ddot{Z}_{0K}=w(a\mu_0^2+b\mu_0+c)$, $\gamma_K^2(\ddot{Z}_{0K})=w(a\mu_0^2+b\mu_0+c)/\mu_0^2$ and $\gamma_K^2(1-\ddot{Z}_{0K})=w(a\mu_0^2+b\mu_0+c)/(1-\mu_0)^2$ subject to $\mu_1 \leqq \mu_0 \leqq \mu_2$. Again, these are worst-case bounds. Since they depend only on the known μ_1 and μ_2, we can evaluate them for several values of K to assess when the asymptotic bounds of Theorem 2 come to prevail.

For some μ_1 and μ_2, the variance ratio $\Lambda(\mu_0, K)$ may not exceed unity for small K, indicating the poorer accuracy of the control variate technique in such cases. Therefore, it is of interest to scrutinize the worst-case ratio $\min_{\mu_0 \in [\mu_1,\mu_2]} \Lambda(\mu_0, K)$ for sample sizes K of interest to ensure that the proposed sampling experiment does have a variance-reducing effect. Theorem 4 provides a basis for doing this.

THEOREM 4. *For a, b, and c as in* (14), *let $\omega_1 \leqq \omega_2$ denote the solutions to the quadratic equation*

$$(a+b)\mu_0^2+2c\mu_0-c=0$$

and define

$$\Lambda'(x)=\partial\Lambda(\mu_0, K)/\partial\mu_0|_{\mu_0=x}.$$

Then for $\mu_1>0$, $\Lambda(\mu_{0,K})$ *has its minimum at*

$$\begin{aligned}\mu_0 &= \omega_1 \quad \text{if } \Lambda'(\mu_1)<0,\ \Lambda'(\mu_2)>0,\ \omega_1 \in [\mu_1, \mu_2],\\ &= \omega_2 \quad \text{if } \Lambda'(\mu_1)>0,\ \Lambda'(\mu_2)>0,\ \omega_1, \omega_2 \in [\mu_1, \mu_2]\\ &\qquad \text{with } \Lambda(\mu_1, K)>\Lambda(\omega_2, K),\\ &= \mu_1 \quad \text{if } \Lambda'(\mu_1)>0,\ \Lambda'(\mu_2)>0\end{aligned}$$

$$
\begin{aligned}
&\textit{and either } \omega_1 \notin [\mu_1, \mu_2] \textit{ or } \omega_2 \notin [\mu_1, \mu_2], \\
&= \mu_1 \quad \textit{if } \Lambda'(\mu_1) > 0,\ \Lambda'(\mu_2) > 0,\ \omega_1, \omega_2 \in [\mu_1, \mu_2] \\
&\qquad \textit{with } \Lambda(\mu_1, K) \leqq \Lambda(\omega_2, K), \\
&= \mu_2 \quad \textit{if } \Lambda'(\mu_1) > 0,\ \Lambda'(\mu_2) < 0, \\
&\qquad \textit{or } \Lambda'(\mu_1) < 0,\ \Lambda'(\mu_2) < 0,\ \omega_1 \notin [\mu_1, \mu_2] \\
&= \{\mu\colon \mu \in \{\omega_1, \mu_2\},\ \Lambda(\mu, K) = \min_{\mu_0 = \omega_1, \mu_2} \Lambda(\mu_0, K)\} \\
&\qquad \textit{if } \Lambda'(\mu_1) < 0,\ \Lambda'(\mu_2) < 0 \textit{ and } \omega_1 \in [\mu_1, \mu_2]
\end{aligned}
$$

and for $\mu_1 = 0$

$$\min_{0 \leqq \mu_0 \leqq \mu_2} \Lambda(\mu_0, K) = \min [1/bK, (1-\mu_2)/K(a\mu_2 + b)].$$

Proof for $\mu_1 > 0$. The quantities ω_1 and ω_2 are the roots of $\partial \Lambda(\mu_0, K)/\partial \mu_0 = 0$. To determine where the minimum occurs we first solve for ω_1 and ω_2 and check to see which if either of these roots gives the minimum. As the example in § 6 shows, computing this worst-case variance-reduction ratio for several values of K and comparing it with the worst-case asymptotic variance-reduction ratio in (18d) enables us to determine when the asymptotic result holds with incidental error.

Proof for $\mu_1 = 0$. In this case

$$\Lambda(\mu_0, K) = (1-\mu_0)/K(a\mu_0 + b)$$

and

$$\partial V(\mu_0, K)/\partial \mu_0 = -(a+b)/(a\mu_0 + b)^2.$$

If $a + b < 0$, then the minimum occurs at $\mu_0 = 0$. If $a + b > 0$, then it occurs at $\mu_0 = \mu_2$.

3. Confidence intervals. The proposed control variate method also leads to shorter confidence intervals. Let

$$\bar{W}_{1K} = \bar{Z}_{0K} - \bar{Z}_{1K}, \qquad \bar{W}_{2K} = \bar{Z}_{2K} - \bar{Z}_{1K}$$

and

$$\theta = (\mu_0 - \mu_1)/(\mu_2 - \mu_1)$$

so that (3) has the representation

$$\bar{Y}_K(\beta_1^*, \beta_2^*) - \mu_0 = \bar{W}_{1K} - \theta \bar{W}_{2K}$$

with

$$\operatorname{var} \bar{Y}_K(\beta_1^*, \beta_2^*) = (\mu_2 - \mu_1)\theta(1-\theta)/K.$$

Application of the central limit theorem gives

$$\lim_{K \to \infty} \operatorname{pr} \left\{ \frac{|\bar{W}_{1K} - \theta \bar{W}_{2K}|}{[(\mu_2 - \mu_1)\theta(1-\theta)/K]^{1/2}} \geqq \zeta \right\} = \alpha \tag{19a}$$

where

$$\zeta = \zeta(\alpha) = \left\{ z\colon (2\pi)^{-1/2} \int_{-\infty}^{z} e^{-x^2/2}\, dx = 1 - \alpha/2 \right\}, \tag{19b}$$

so that $[\mu_1+\theta_1(\mu_2-\mu_1), \mu_1+\theta_2(\mu_2-\mu_1)]$ asymptotically covers μ_0 with probability $1-\alpha$, where $\theta_1 \leqq \theta_2$ are the roots of the equation

$$[\bar{W}_{2K}^2+(\mu_2-\mu_1)\zeta^2/K]\theta^2-[2\bar{W}_{1K}\bar{W}_{2K}+(\mu_2-\mu_1)\zeta^2/K]\theta+\bar{W}_{1K}^2=0. \tag{20}$$

To order $K^{-1/2}$, the resulting interval width is

$$(\theta_2-\theta_1)(\mu_2-\mu_1)\approx 2\zeta[(\mu_2-\mu_1)^3(\bar{Z}_{0K}-\bar{Z}_{1K})(\bar{Z}_{2K}-\bar{Z}_{0K})/K(\bar{Z}_{2K}-\bar{Z}_{1K})^4]^{1/2}. \tag{21}$$

By contrast, a normal confidence interval based on $(\bar{Z}_{0K}-\mu_0)/[\mu_0(1-\mu_0)/K]^{1/2}$ has approximate width $2\zeta[\bar{Z}_{0K}(1-\bar{Z}_{0K})/K]^{1/2}$, clearly establishing the benefit of the control variate approach.

Unfortunately, convergence to normality is not uniform in μ_0, μ_1, and μ_2 so that putting bounds on the error of the normal approximation superimposed on the error of estimation remains a challenge. A second alternative, based on Chebyshev's inequality, replaces ζ as defined in (19b) by $\zeta=1/(1-\alpha)^{1/2}$. Then $(\mu_1+\theta_1(\mu_2-\mu_1), \mu_1+\theta_2(\mu_2-\mu_1))$ covers μ_0 with probability $>1-\alpha$ for each K, but with width considerably greater than the width for the normal interval. Therefore, we propose a third alternative free of the approximating error that normality induces and leads to a shorter confidence interval than Chebyshev's inequality induces.

THEOREM 5. *Let* $\theta=(\mu_0-\mu_1)/(\mu_2-\mu_1)$ *and assume* $0<\theta<1$. *Let*

$$R(\theta, \varepsilon, h, \mu_1, \mu_2)=e^{-h\varepsilon}\{1-\mu_2+\mu_1+(\mu_2-\mu_1)[\theta\, e^{(1-\theta)h}+(1-\theta)\, e^{-\theta h}]\} \tag{22}$$

for $0\leqq\mu_1<\mu_0<\mu_2\leqq 1$ *and* $h\geqq 0$.

(i) *For each fixed* $\varepsilon>0$

$$\text{pr}\,(\bar{W}_{1K}-\theta\bar{W}_{2K}\geqq\varepsilon)\leqq[R(\theta, \varepsilon, h^*, \mu_1, \mu_2)]^K$$

where

$$R(\theta, \varepsilon, h^*, \mu_1, \mu_2)=\min_{h\geqq 0} R(\theta, \varepsilon, h, \mu_1, \mu_2) \tag{23}$$

and $0\leqq\varepsilon\leqq 1-\theta$.

(ii) *For each fixed* $\varepsilon>0$

$$\text{pr}\,[\bar{W}_{2K}-\bar{W}_{1K}-(1-\theta)\bar{W}_{2K}\geqq\varepsilon]\leqq[R(1-\theta, \varepsilon, h^{**}, \mu_1, \mu_2)]^K$$

where

$$R(1-\theta, \varepsilon, h^{**}, \mu_1, \mu_2)=\min_{h\geqq 0} R(1-\theta, \varepsilon, h, \mu_1, \mu_2) \tag{24}$$

and $0<\varepsilon\leqq\theta$.

(iii) *Let* θ_3 *denote the solution to the equation*

$$R(\theta, \bar{W}_{1K}-\theta\bar{W}_{2K}, h^*, \mu_1, \mu_2)=(\alpha/2)^{1/K}, \qquad 0<\theta\leqq\bar{W}_{1K}/\bar{W}_{2K}, \quad 0<\alpha<1$$

for the case in which $\bar{W}_{1K}>0$. *Let*

$$\theta_*=\theta_3 \quad \textit{if } \bar{W}_{1K}>0$$
$$=0 \quad \textit{if } \bar{W}_{1K}=0.$$

Then

$$\text{pr}\,[\mu_0>\mu_1+\theta_*(\mu_2-\mu_1)]>1-\alpha/2. \tag{25}$$

(iv) *Let* θ_4 *denote the solution to the equation*

$$R(1-\theta, \bar{W}_{2K}-\bar{W}_{1K}-(1-\theta)\bar{W}_{2K}, h^{**}, \mu_1, \mu_2)=(\alpha/2)^{1/K},$$
$$\bar{W}_{1K}/\bar{W}_{2K}\leqq\theta<1, \quad 0<\alpha<1$$

for $\bar{W}_{1K} < \bar{W}_{2K}$. *Let*

$$\theta^* = \theta_4 \quad \textit{if } \bar{W}_{1K} < \bar{W}_{2K},$$
$$= 1 \quad \textit{if } \bar{W}_{1K} = \bar{W}_{2K}.$$

Then

$$\text{pr}\,[\mu_0 < \mu_1 + \theta^*(\mu_2 - \mu_1)] > 1 - \alpha/2. \tag{26}$$

(v)

$$\text{pr}\,[\mu_1 + \theta_*(\mu_2 - \mu_1) < \mu_0 < \mu_1 + \theta^*(\mu_2 - \mu_1)] > 1 - \alpha. \tag{27}$$

The Appendix contains the proof.

Bisection provides a convenient method for computing (23) and (24) for specified θ, ε, μ_1, and μ_2. Observe that

$$\frac{\partial R}{\partial h} = e^{-h\varepsilon}[-\varepsilon(1-\mu_2+\mu_1) + (\mu_2-\mu_1)\theta(1-\theta-\varepsilon)\, e^{(1-\theta)h} - (\mu_2-\mu_1)(1-\theta)(\theta+\varepsilon)\, e^{-\theta h}]$$

so that

$$\frac{\partial R}{\partial h} \leqq e^{-h\varepsilon}[-\varepsilon(1-\mu_2+\mu_1) + (\mu_2-\mu_1)\theta(1-\theta-\varepsilon)\, e^{(1-\theta)h}]. \tag{28}$$

Let

$$h_L = \frac{1}{1-\theta} \ln\,[\varepsilon(1-\mu_2+\mu_1)/\theta(1-\theta-\varepsilon)(\mu_2-\mu_1)],$$

which results from equating the right side of (28) to zero. Since R is convex in h, as shown in the Appendix, $h^* \geqq h_L$. Conversely, since $e^{-h\theta} \leqq 1$

$$\frac{\partial R}{\partial h} \geqq e^{-h\varepsilon}[-\varepsilon(1-\mu_2+\mu_1) - (\mu_2-\mu_1)(1-\theta)(\theta+\varepsilon) + (\mu_2-\mu_1)\theta(1-\theta-\varepsilon)\, e^{(1-\theta)h}]$$

so that

$$h_U = \frac{1}{1-\theta} \ln\,[1 + \varepsilon/\theta(1-\theta-\varepsilon)(\mu_2-\mu_1)]$$

provides an upper bound on h^*. Observe that

$$h_U - h_L = \frac{1}{1-\theta} \ln\,[1 + (\varepsilon+\theta)(1-\theta)(\mu_2-\mu_1)/\varepsilon(1-\mu_2+\mu_1)].$$

Algorithm A describes steps, based on bisection, for computing the lower bound $\Psi_1 = \mu_1 + \theta_3(\mu_2 - \mu_1)$. It also applies for the upper point $\Psi_2 = \mu_1 + \theta_4(\mu_2 - \mu_1)$ if we set $y = \bar{Z}_{2K} - \bar{Z}_{0K}$ in step (2a) and set $\Psi_2 = \Psi_1 + (1-\theta)(\mu_2 - \mu_1)$ in step (3).

ALGORITHM A.

Purpose: To compute Ψ_1 where $\text{pr}\,(\mu_0 > \Psi_1) > 1 - \alpha/2$ for $0 < \alpha < 1$.

Input: Sample data $0 < y = \bar{Z}_{0K} - \bar{Z}_{1K} < x = \bar{Z}_{2K} - \bar{Z}_{1K}$; bounds μ_1, μ_2; confidence level α; sample size K; error tolerances δ_1 and δ_2.

Output: Ψ_1.

Method:

Define:

$$R = e^{-h\varepsilon}\{1 - \mu_2 + \mu_1 + (\mu_2 - \mu_1)[\theta\, e^{(1-\theta)h} + (1-\theta)\, e^{-\theta h}]\}$$

and

$$\frac{\partial R}{\partial h} = -\varepsilon R + e^{-h\varepsilon}(\mu_2 - \mu_1)\theta(1-\theta)(e^{(1-\theta)h} - e^{-\theta h}).$$

(1) Set $\theta_L = 0$ and $\theta_U = y/x$.
(2) Until $\alpha/2 - \delta_1 \leqq R^K \leqq \alpha/2$:
(2a) $\theta = (\theta_L + \theta_U)/2$; $\varepsilon = y - \theta x$.
Computer $R^* = \min_{h \geqq 0} R$.
(2b) Compute bounds on h.

$$h_L = \frac{1}{1-\theta} \ln [\varepsilon(1 - \mu_2 + \mu_1)/\theta(1 - \theta - \varepsilon)(\mu_2 - \mu_1)],$$

$$h_U = \frac{1}{1-\theta} \ln [1 + \varepsilon/\theta(1 - \theta - \varepsilon)(\mu_2 - \mu_1)].$$

(2c) Until $|\partial R/\partial h| \leqq \delta_2$:

$$h = (h_L + h_U)/2; \text{ compute } \partial R/\partial h.$$

If $\partial R/\partial h < 0$, $h_L = h$.
If $\partial R/\partial h \geqq 0$, $h_U = h$.
(2d) Compute R^*.
If $(R^*)^K \leqq \alpha/2$, $\theta_L = \theta$.
If $(R^*)^K > \alpha/2$, $\theta_U = \theta$.
(3) Deliver $\Psi_1 = \mu_1 + \theta(\mu_2 - \mu_1)$.

4. Estimating a distribution function. The extension of these techniques to the estimation of a probability distribution function (d.f.) is direct. Consider the stochastically ordered random variables $X_1 \leqq X_0 \leqq X_2$ with complementary d.f.'s

$$\mu_i(x) = \text{pr}\,(X_i > x), \qquad i = 0, 1, 2, \quad -\infty < x < \infty$$

where $\{\mu_1(x)\}$ and $\{\mu_2(x)\}$ are known and $\{\mu_0(x)\}$ is to be estimated at all points $x \varepsilon \mathscr{X}$. Let

$$Z(w, x) = 1 \quad \text{if } w > x, \qquad = 0 \quad \text{otherwise.} \tag{29}$$

Since $Z(X_1, x) \leqq Z(X_0, x) \leqq Z(X_2, x)$ it is clear that $\mu_1(x) \leqq \mu_0(x) \leqq \mu_2(x)$. As before,

$$Y(x) = Z(X_0, x) + \beta_1^*(x)[Z(X_1, x) - \mu_1(x)] + \beta_2^*(x)[Z(X_2, x) - \mu_2(x)] \tag{30}$$

has mean $\mu_0(x)$.

Let

$$\bar{Z}_{ik}(x) = \frac{1}{K} \sum_{k=1}^{K} Z(X_i^{(k)}, x), \qquad i = 0, 1, 2 \tag{31}$$

where $(X_0^{(1)}, X_1^{(1)}, X_2^{(1)}), \cdots, (X_0^{(K)}, X_1^{(K)}, X_2^{(K)})$ denote K independent samples. Then if we estimate $\{\mu_0(x)\, x \varepsilon \mathscr{X}\}$ by

$$\tilde{Z}_{0K}(x) = \bar{Z}_{0K}(x) + \hat{\beta}_1^*(x)[\bar{Z}_{1K}(x) - \mu_1(x)] + \hat{\beta}_2^*(x)[\bar{Z}_{2K}(x) - \mu_2(x)]\, x \varepsilon \mathscr{X} \tag{28}$$

where

$$\hat{\beta}_1^*(x) = [\bar{Z}_{0K}(x) - \mu_2(x)]/[\mu_2(x) - \mu_1(x)] \tag{32a}$$

and

$$\hat{\beta}_2(x) = [\mu_1(x) - \bar{Z}_{0K}(x)]/[\mu_2(x) - \mu_1(x)], \tag{32b}$$

then all the earlier results apply. In particular,

$$\ddot{Z}_{0K}(x) = \{K[\mu_2(x) - \mu_1(x)]\tilde{Z}_{0K}(x) - \mu_1(x)\}/\{(K+1)[\mu_2(x) - \mu_1(x)] - 1\} \tag{33}$$

is an unbiased estimator of $\mu_0(x)$. Although other methods of estimating $\{\beta_1(x), \beta_2(x)\}$ are possible, all appear to introduce additional unknown covariance terms that preclude direct application of the appealing earlier results.

5. Estimating the mean. The control variate technique of distribution function estimation also enables us to derive an estimate of the sample mean in at least one important case. If $X_1 \leqq X_0 \leqq X_2$ are discrete random variables on the set of integers $\{a, a+1, \cdots, b\}$, then the corresponding means are

$$\lambda_i = a + \sum_{x=a}^{b-1} \mu_i(x), \qquad i = 0, 1, 2. \tag{34}$$

This suggests the control variate estimators

$$\tilde{\lambda}_{0K} = a + \sum_{x=a}^{b-1} \tilde{Z}_{0K}(x) \tag{35}$$

and

$$\ddot{\lambda}_{0K} = a + \sum_{x=a}^{b-1} \ddot{Z}_{0K}(x) \tag{36}$$

for λ_0, which are to be compared with the conventional sample mean

$$\hat{\lambda}_{0K} = \frac{1}{K} \sum_{k=1}^{K} X_0^{(k)} = a + \sum_{x=a}^{b-1} \bar{Z}_{0K}(x) \tag{37}$$

for accuracy. We base this comparison on variances. An unbiased estimator of $\operatorname{var} \hat{\lambda}_{0K}$ is

$$V(\hat{\lambda}_{0K}) = \frac{1}{K(K-1)} \sum_{k=1}^{K} (X_0^{(k)} - \hat{\lambda}_{0K})^2. \tag{38}$$

Observe that

$$\operatorname{var} \tilde{\lambda}_{0K} \leqq \Gamma(\tilde{\lambda}_{0K}) = \left\{ \sum_{x=a}^{b-1} [\operatorname{var} \tilde{Z}_{0K}(x)]^{1/2} \right\}^2 \tag{39}$$

and

$$\operatorname{var} \ddot{\lambda}_{0K} \leqq \Gamma(\ddot{\lambda}_{0K}) = \left\{ \sum_{x=a}^{b-1} [\operatorname{var} \ddot{Z}_{0K}(x)]^{1/2} \right\}^2. \tag{40}$$

As estimates of these bounds, we have

$$\hat{\Gamma}(\tilde{\lambda}_{0K}) = \left\{ \sum_{x=a}^{b-1} [V(\tilde{Z}_{0K}(x))]^{1/2} \right\}^2 \tag{41}$$

and

$$\hat{\Gamma}(\ddot{\lambda}_{0K}) = \left\{ \sum_{x=a}^{b-1} [V(\ddot{Z}_{0K}(x))]^{1/2} \right\}^2, \tag{42}$$

$V(\tilde{Z}_{0K}(x))$ being defined in (13) with $\tilde{Z}_{0K}(x)$ replacing $\tilde{Z}_{0K}$ and with

$$V(\ddot{Z}_{0K}(x)) = \{K[\mu_2(x)-\mu_1(x)]/\{(K+1)[\mu_2(x)-\mu_1(x)]-1\}\}^2 V(\tilde{Z}_{0K}(x)). \tag{43}$$

Here $V(\hat{\lambda}_{0K})/\hat{\Gamma}(\tilde{\lambda}_{0K})$ and $V(\hat{\lambda}_{0K})/\hat{\Gamma}(\ddot{\lambda}_{0K})$ give estimates of lower bounds on the variance reductions that using control variates for distribution-function estimation induces. Observe that the computation of the right sides of (39) and (40) each takes $O(b-a)$ time.

A direct estimate of var $\tilde{\lambda}_{0K}$ is also possible. Let $\Delta\bar{Z}_{iK}(j) = \bar{Z}_{iK}(j) - \mu_i(j)$, $i = 0, 1, 2$ and write

$$\begin{aligned}\Delta\tilde{Z}_{0K}(j) = \tilde{Z}_{0K}(j) - \mu_0(j) = \Delta\bar{Z}_{0K}(j) + \beta_1^*(j)\Delta\bar{Z}_{1K}(j) - [1+\beta_1^*(j)]\Delta\bar{Z}_{2K}(j) \\ + \bar{Z}_{0K}(j)[\Delta\bar{Z}_{1K}(j) - \Delta\bar{Z}_{2K}(j)]/[\mu_2(j)-\mu_1(j)], \qquad a \leqq j < b.\end{aligned} \tag{44}$$

Then to order $1/K$ we have

$$\begin{aligned}\text{var } \tilde{\lambda}_{0K} = \frac{1}{K}\sum_{j=a}^{b-1}[\mu_2(j)-\mu_0(j)][\mu_0(j)-\mu_1(j)]/[\mu_2(j)-\mu_1(j)] \\ + \frac{2}{K}\sum_{a\leqq j<k\leqq b-1}\{\mu_0(k)[1-\mu_0(j)] + \beta_1^*(k)\mu_1(k)[1-\mu_0(j)] \\ + \beta_1^*(j)\beta_1^*(k)\mu_1(k)[1-\mu_1(j)] \\ - [1+\beta_1^*(j)]\mu_0(k)[1-\mu_2(j)] \\ - [1+\beta_1^*(j)]\beta_1^*(k)\mu_1(k)[1-\mu_2(j)] \\ + [1+\beta_1^*(j)][1+\beta_1^*(k)]\mu_2(k)[1-\mu_2(j)]\} \\ + 2\sum_{a\leqq j<k\leqq b-1}\{-[1+\beta_1^*(k)]\,\text{cov}\,[\bar{Z}_{0K}(j), \bar{Z}_{2K}(k)] \\ + \beta_1^*(j)\,\text{cov}\,[\bar{Z}_{1K}(j), \bar{Z}_{0K}(k)] \\ - \beta_1^*(j)[1+\beta_1^*(k)]\,\text{cov}\,[\bar{Z}_{1K}(j), \bar{Z}_{2K}(k)]\},\end{aligned} \tag{45}$$

for which an estimate is derived by substituting $\ddot{Z}_{0K}(j)$ for $\mu_0(j)$, $\hat{\beta}_1^*(j)$, for $\beta_1^*(j)$, and $T_{02}(j,k)$, $T_{10}(j,k)$ and $T_{12}(j,k)$, where

$$\begin{aligned}T_{rs}(j,k) = \frac{1}{K(K-1)}\sum_{m=1}^{K}[Z(X_r^{(m)}, j) - \bar{Z}_{rK}(j)][Z(X_s^{(m)}, k) - \bar{Z}_{sK}(k)], \\ a \leqq j < k \leqq b-1, \quad r=0, s=2, \quad r=1, s=0, \quad r=1, s-2,\end{aligned} \tag{46}$$

for $\text{cov}\,[\bar{Z}_{0K}(j), \bar{Z}_{2K}(k)]$, $\text{cov}\,[\bar{Z}_{1K}(j), \bar{Z}_{0K}(k)]$, and $\text{cov}\,[\bar{Z}_{1K}(j), \bar{Z}_{2K}(k)]$, respectively. That is, all but these three covariances are expressible in terms of the three means $\mu_1(j)$, $\mu_0(j)$, and $\mu_2(j)$. Since the computation of $\{T_{rs}(j,k),\ a \leqq j < k \leqq b-1\}$ takes $O((b-a)(b-a-1)/2)$ time, this direct estimation of var $\tilde{\lambda}_{0K}$ to order $1/K$ may become costly for large $b-a$. In such a case the upper bounds (39) and (40) with linear cost in $(b-a)$ may be more appealing.

6. Example. An example illustrates how the proposed technique works in practice. Consider a binary-state component, multistate system reliability problem whose system is representable as a stochastic flow network. Let $G = (\mathcal{V}, \mathcal{E})$ denote a directed network with node set $\mathcal{V}$, arc set $\mathcal{E} = \{1, \cdots, n\}$, source node s and terminal node t, $s, t \in V$. Arc capacities are independent random variables where arc i has flow capacity $D_i = M_i$ with probability p_i and $D_i = 0$ flow capacity with probability $1-p_i$, $i = 1, \cdots, n$. System reliability is characterized by the probability distribution function (d.f.) of maximal

$s-t$ flow in the network. Although well-established techniques exist for computing maximal flow in a network with deterministic capacities, no comparable technique exists for the stochastic capacity specification. In fact, the computation of this d.f. belongs to the class of NP-hard problems, implying that no polynomial time (as a function of $|\mathcal{V}|$ and $|\mathcal{E}|$) algorithm is known to exist for computing the d.f.

Figure 1 shows a stochastic flow network of 10 nodes, 25 arcs, $s=1$, $t=10$ and where the notation j/k on an arc indicates arc j with upper bound on capacity $= M_j = k$. For example, arc 1, from nodes 1 to 2, has $M_1 = 12$. Also, let

$$p_i = \text{pr (capacity of arc } i = M_i) = .9, \qquad i = 1, \cdots, 25.$$

The minimal flow for this network is zero, and the maximal flow, with all arc capacities at their upper values is 71.

Let X_0 denote the maximal flow on a realization of the network. We wish to estimate the complementary distribution function

$$\mu_0(x) = \text{pr}\,(X_0 > x), \qquad x = 1, \cdots, 70.$$

To derive lower and upper bounds on $\{\mu_0(x)\}$, we use a technique described in Fishman [2] for binary-state reliability problems. Consider the edge-disjoint minimal $s-t$ paths $\mathcal{P}_1 = \{1, 5, 13, 19, 23, 25\}$, $\mathcal{P}_2 = \{2, 9, 20\}$, and $\mathcal{P}_3 = \{3, 12, 14, 16, 18, 24\}$. Together, $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ form a directed network G_1 in which arcs in $\mathcal{E} - \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_3$ are treated as having zero capacity. If X_1 is the maximal $s-t$ flow in G_1, then pr $(X_0 - X_1 \geqq 0) = 1$, implying $Z(X_0, x) \geqq Z(X_1, x)$ w.p.1 and pr $(X_1 > x) =$ pr $[Z(X_1, x) = 1] = \mu_1(x) \leqq \mu_0(x)$. Most importantly,

$$\mu_1(x) = \text{pr}\left(\sum_{j=1}^{3} \min_{i \in \mathcal{P}_j} D_i > x\right),$$

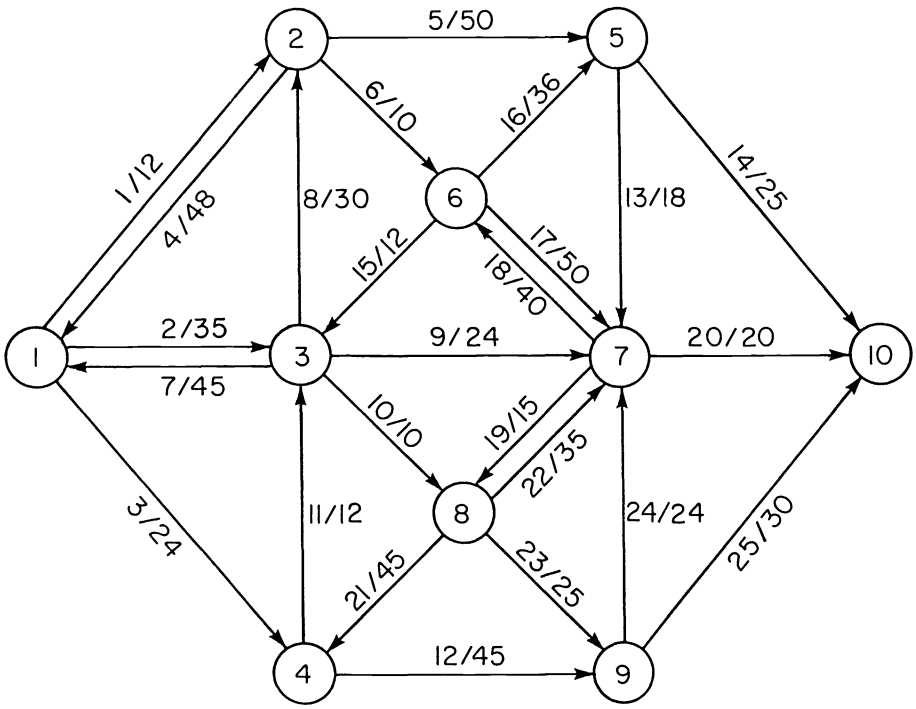


FIG. 1

which we can compute exactly for a discrete flow system G_1 with edge-disjoint minimal $s-t$ paths and independently distributed capacities. (See Fishman [3].)

To derive an upper bound on $\{\mu_0(x)\}$, we use the edge-disjoint minimal $s-t$ cutsets $\mathscr{C}_1=\{1, 2, 3\}$, $\mathscr{C}_2=\{14, 20, 25\}$, and $\mathscr{C}_3=\{5, 6, 9, 10, 12\}$. Together, $\mathscr{C}_1$, $\mathscr{C}_2$, and $\mathscr{C}_3$ form an $s-t$ network G_2 in which arcs in $\mathscr{E}-\mathscr{C}_1\cup\mathscr{C}_2\cup\mathscr{C}_3$ are assumed to have infinite capacity. If X_2 is the maximal $s-t$ flow in G_2, then $\text{pr}(X_2-X_0\geqq 0)=1$, implying $Z(X_0, x)\leqq Z(X_2, x)$ w.p.1 and $\text{pr}(X_2>x)=\text{pr}[Z(X_2, x)=1]=\mu_2(x)\geqq\mu_0(x)$. Also

$$\mu_2(x)=\text{pr}\left(\min_{1\leqq j\leqq 3}\sum_{i\in\mathscr{C}_j}D_i>x\right)$$

$$=\prod_{j=1}^{3}\text{pr}\left(\sum_{i\in\mathscr{C}_j}D_i>x\right),$$

which we can compute exactly for a discrete flow system G_2 with edge-disjoint minimal $s-t$ cutsets and independently distributed capacities. (See Fishman [3].)

Table 1 gives the lower and upper bounds for $\{\mu_0(x)\}$ together with the lower bound on the minimal achievable variance ratios for $K=128$ and for $K\to\infty$. Clearly the asymptotic results apply for $K=128$. Table 2 lists point and 99 percent confidence intervals for $\mu_0(x)$, based on (27), together with an estimate of the induced variance ratio for a sample of $K=65{,}536$ observations.

TABLE 1
Bounds and variance ratios.

Maximal flow x	Bounds		Minimal achievable variance ratio[1]	
	Lower $\mu_1(x)$	Upper $\mu_2(x)$	$\min_{\mu_1\leqq\mu_0\leqq\mu_2}\Lambda(\mu_0(x), K)$ $(K=128)$	$\lim_{K\to\infty}\min_{\mu_1\leqq\mu_0\leqq\mu_2}\Lambda(\mu_0(x), K)$
0-9	.9405	.9980	1.480	1.434
10-11	.9405	.9978	1.485	1.458
12-19	.8730	.9888	1.764	1.773
20-23	.7130	.9791	1.605	1.598
24	.6455	.9701	1.626	1.620
25-29	.6455	.9613	1.745	1.742
30-31	.6455	.9525	1.861	1.863
32-33	.4640	.9525	1.529	1.525
34	.4640	.9509	1.540	1.536
35	.4640	.9422	1.603	1.599
36-43	.3874	.8637	1.924	1.924
44	.2059	.8574	1.527	1.524
45-46	.2059	.7859	1.727	1.724
47-49	.2059	.7144	1.950	1.950
50-54	.2059	.6494	2.191	2.195
55	.2059	.5835	2.492	2.510
56-58	0.	.5835	1.006	1.000
59	0.	.5252	1.007	1.000
60-64	0.	.5243	1.007	1.000
65-68	0.	.5204	1.007	1.000
69	0.	.5200	1.007	1.000
70	0.	.5161	1.007	1.000

[1] See Theorems 2(iv) and 4.

TABLE 2
Estimate of $\{\mu_0(x)\}$.[2]
($K = 65536$)

x	$\bar{Z}_{0K}(x)$	$\tilde{Z}_{0K}(x)$	$\ddot{Z}_{0K}(x)$	$10^6 \times V(\ddot{Z}_{0K}(x))$	Variance ratio $\Lambda(\mu_0(x), K)$	Individual .99 confidence intervals Lower	Upper
0-9	.9977	.9975	.9975	.7776	4.94	.9971	.9977
10-11	.9959	.9956	.9956	3.237	2.06	.9949	.9961
12-14	.9861	.9860	4.142	4.142	5.07	.9853	.9866
15-19	.9860	.9859	.9860	4.253	4.96	.9852	.9865
20-21	.9736	.9733	.9733	8.619	4.59	.9723	.9742
22-23	.9718	.9716	11.16	11.16	3.77	.9705	.9726
24	.9559	.9550	.9550	22.01	2.98	.9534	.9564
25-27	.9452	.9446	.9446	24.18	3.30	.9429	.9461
28-29	.9450	.9444	.9444	24.41	3.28	.9427	.9459
30-31	.9354	.9345	.9345	25.84	3.61	.9328	.9360
32-33	.9339	.9331	.9331	28.42	3.35	.9313	.9347
34	.9214	.9207	.9207	43.34	2.57	.9185	.9227
35	.9113	.9104	.9104	45.30	2.75	.9082	.9125
36	.8454	.8453	.8453	27.07	7.38	.8435	.8468
37	.8453	.8451	.8451	27.26	7.33	.8434	.8467
38-39	.8447	.8446	.8446	28.03	7.15	.8429	.8462
40-41	.8430	.8429	.8429	30.37	6.65	.8411	.8446
42-43	.8410	.8408	.8408	33.26	6.14	.8389	.8426
44	.8236	.8229	.8229	49.87	4.46	.8205	.8251
45	.7401	.7414	.7414	62.64	4.67	.7388	.7439
46	.7272	.7285	.7285	78.93	3.82	.7255	.7313
47	.6688	.6706	.6706	61.09	5.52	.6680	.6731
48	.6677	.6695	.6695	62.53	5.40	.6669	.6720
49	.6666	.6684	.6684	63.90	5.30	.6657	.6709
50-51	.6016	.6036	.6036	62.74	5.82	.6009	.6061
52-53	.5992	.6011	.6011	65.71	5.57	.5984	.6037
54	.5987	.6007	.6007	66.22	5.53	.5980	.6033
55	.5433	.5430	.5430	55.21	6.86	.5405	.5453
56-57	.5432	.5429	.5429	57.72	6.56	.5403	.5452
58	.5324	.5321	.5321	71.52	5.31	.5293	.5348
59	.4576	.4566	.4566	91.01	4.16	.4534	.4596
60-6fi	.4498	.4487	.4487	98.74	3.82	.4454	.4519
62-63	.4433	.4423	.4422	105.6	3.56	.4388	.4455
64-65	.4430	.4419	.4419	106.0	3.55	.4385	.4452
66-67	.4390	.4380	.4380	105.9	3.55	.4346	.4412
68	.4115	.4105	.4105	132.3	2.79	.4067	.4142
69	.4015	.4004	.4004	140.5	2.61	.3965	.4042
70	.3744	.3731	157.8	157.8	2.26	.3690	.3771

[2] The $\Lambda(\mu_0(x), K)$ were estimated with $\ddot{Z}_{0K}(x)$ for $\mu_0(x)$ in (18b); the confidence intervals were computed using Algorithm A.

With regard to the mean λ_0 we have

$$\hat{\lambda}_{0K} = 56.40, \quad \tilde{\lambda}_{0K} = 56.39, \quad \ddot{\lambda}_{0K} = 56.39$$

with variance estimates

$$V(\hat{\lambda}_{0K}) = .3633 \times 10^{-2} \qquad V(\tilde{\lambda}_{0K}) = V(\ddot{\lambda}_{0K}) = .4621 \times 10^{-3}$$

implying a variance ratio of

$$V(\hat{\lambda}_{0K})/V(\ddot{\lambda}_{0K}) = 7.86.$$

Here $V(\tilde{\lambda}_{0K})$ is based on (46) with estimates replacing unknown quantities.

Time also plays a role in evaluating a variance-reduction technique. The control variate Monte Carlo experiment required 1.51 milliseconds of CPU time per replication on an IBM 4381-P2 at the University of North Carolina, Chapel Hill, North Carolina. A crude Monte Carlo using the same random numbers required 1.41 milliseconds per replication. Presumably, the need to compute the sample maximal flows in G_1 and G_2 on each replication accounted for the .09 millisecond difference. This $100 \times .09/1.42 = 6.3$ percent increase in time does little to reduce the attraction of the control variate method.

Since G is planar, the algorithm of Itai and Shiloach [4] enabled us to compute $X_0^{(k)}$, the sample maximal $s-t$ flow in G, on replication k in $O(|\mathscr{E}| \log |\mathscr{E}|)$ time. However, $X_1^{(k)}$ and $X_2^{(k)}$ the sample maximal $s-t$ flows in G_1 and G_2, respectively, were computed more directly as

$$X_1^{(k)} = \sum_{m=1}^{3} \min_{j \in \mathscr{P}_m} D_j^{(k)}, \qquad X_2^{(k)} = \min_{1 \leqq m \leqq 3} \sum_{j \in \mathscr{P}_m} D_j^{(k)}$$

where $D_j^{(k)}$ denotes the capacity of arc j on replication k.

Appendix.

Proof of Theorem 5. Proof of (i) and (ii). Observe that for $k = 1, \cdots, K$

$$\begin{aligned} Z_1^{(k)} - \theta Z_2^{(k)} &= -\theta \qquad \text{w.p. } \mu_2 - \mu_1, \\ &= 0 \qquad \text{w.p. } 1 - \mu_2, \\ &= 1 - \theta \qquad \mu_1, \end{aligned}$$

and define the indicator function

$$\begin{aligned} I_{[a,b)}(x) &= 1 \quad \text{if } a \leqq x < b, \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

Then for $h \geqq 0$

$$\begin{aligned} I_{[0,\infty)}(\bar{W}_{1K} - \theta \bar{W}_{2K} - \varepsilon) &= I_{[0,\infty)}\left(\sum_{k=1}^{K} [Z_0^{(k)} - Z_1^{(k)} - \theta(Z_2^{(k)} - Z_1^{(k)})] - K\varepsilon \right) \\ &\leqq \exp\left\{ h\left(\sum_{k=1}^{K} [Z_0^{(k)} - Z_1^{(k)} - \theta(Z_2^{(k)} - Z_1^{(k)})] - K\varepsilon \right) \right\}. \end{aligned}$$

Therefore

$$\begin{aligned} \text{pr}\,(\bar{W}_{1K} - \theta \bar{W}_{2K} \geqq \varepsilon) &\leqq \{E \exp(h[Z_0^{(k)} - Z_1^{(k)} - \theta(Z_1^{(k)} - Z_2^{(k)}) - \varepsilon])\}^K \\ &= [R(\theta, \varepsilon, h, \mu_1, \mu_2)]^K. \end{aligned}$$

Since this inequality holds for all h, it must hold at h^*. To see that a unique minimum exists, we note that $\partial^2 R/\partial h^2 > 0$, implying that R is convex in h. This minimum occurs at an $h < \infty$, provided that $\varepsilon < 1 - \theta$. The proof of (ii) follows analogously.

Proof of (iii), (iv), *and* (v). Consider the sets

$$\Omega(\alpha, K) = \{(T, \varepsilon): 0 < \varepsilon \leqq 1 - T, 0 \leqq T \leqq 1, R(T, \varepsilon, h^*, \mu_1, \mu_2) = (\alpha/2)^{1/K}\}$$

and

$$\mathcal{H}(x, y) = \{(T, \varepsilon(x, y, T)): \varepsilon(x, y, T) = y - Tx, 0 < T \leqq y/x, 0 < y \leqq x \leqq 1\}.$$

Then $\Omega(\alpha, K) \cap \mathcal{H}(x, y)$ is the set of all (T, ε) that satisfy

$$\text{(A1)} \qquad R^* = R(T, \varepsilon(x, y, T), h^*(T, \varepsilon(x, y, T)), \mu_1, \mu_2) = (\alpha/2)^{1/K}.$$

Observe that

$$R(0, y, h^*(0, y), \mu_1, \mu_2) = 0$$

and

$$R(y/x, 0, h^*(y/x, 0), \mu_1, \mu_2) = 1.$$

Also, observe that

$$\text{(A2)} \qquad \frac{dR^*}{dT} = \frac{\partial R}{\partial T} + \frac{\partial R}{\partial \varepsilon}\frac{d\varepsilon(x, y, T)}{dT} + \frac{\partial R}{\partial h}\left[\frac{\partial h^*}{\partial T} + \frac{\partial h^*}{\partial \varepsilon}\frac{d\varepsilon(x, y, T)}{dT}\right]$$

where all partial derivatives of R are taken at the point $(T, \varepsilon, h, \mu_1, \mu_2) = (T, y - Tx, h^*(T, y - Tx), \mu_1, \mu_2)$ and all partial derivatives of h^* are taken at the point $(T, \varepsilon) = (T, y - Tx)$. Since $\partial R(T, \varepsilon, h^*(T, \varepsilon), \mu_1, \mu_2)/\partial h = 0$ for all T and ε,

$$\text{(A3)} \qquad \frac{dR}{dT} = xh^*R^* + (\mu_2 - \mu_1)\{(1 - Th^*)\, e^{h^*} - [1 + (1 - T)h^*]\}\, e^{-(T+\varepsilon)h^*} > 0.$$

Combining (A1), (A2), and (A3), we see that (A1) has a unique solution $\theta_3 = \theta_3(x, y, \alpha, K)$ for each (x, y) such that $0 < y \leqq x \leqq 1$. That is, $\Omega(\alpha, K) \cap \mathcal{H}(x, y) = (\theta_3(x, y, \alpha, K), \varepsilon(x, y, \theta_3(x, y, \alpha, K)))$.

Let $(T, \varepsilon(T))$ denote a point in $\Omega(\alpha, K)$. Since for every $0 < T \leqq y/x$ and $0 < y \leqq x \leqq 1$, the curve $\{\varepsilon(T)\}$ intersects the line $\{\varepsilon(x, y, T) = y - Tx\}$ exactly once, there must exist an $\varepsilon(T) \leqq \varepsilon(x, y, \theta_3(x, y, \alpha, K))$ for $T \leqq \theta_3(x, y, \alpha, K)$. Since $\partial R(T, \varepsilon, h^*, \mu_1, \mu_2)/\partial \varepsilon < 0$, it follows that

$$\text{(A4)} \qquad \begin{aligned}(\alpha/2)^{1/K} &= R(T, \varepsilon(T), h^*(T, \varepsilon(T)), \mu_1, \mu_2) \\ &\geqq R(T, \varepsilon(x, y, \theta_3(x, y, \alpha, K)), h^*(T, \varepsilon(T)), \mu_1, \mu_2).\end{aligned}$$

Recall that

$$\begin{aligned}\theta_* &= \theta_3(\bar{W}_{2K}, \bar{W}_{1K}, \alpha, K) \quad \text{if } \bar{W}_{1K} > 0, \\ &= 0 \qquad\qquad\qquad\qquad\quad \text{otherwise,}\end{aligned}$$

and observe that

$$\begin{aligned}\text{pr}(\theta \leqq \theta_*) &= \text{pr}(\theta \leqq \theta_* \mid \theta_* > 0)\,\text{pr}(\theta_* > 0) + \text{pr}(\theta \leqq \theta_* \mid \theta_* = 0)\,\text{pr}(\theta_* = 0) \\ &= \text{pr}(\theta \leqq \theta_* \mid \theta_* > 0)\,\text{pr}(\theta_* > 0) \quad \text{since pr}(\theta \leqq 0) = 0 \\ &= \text{pr}[\theta \leqq \theta_3(\bar{W}_{2K}, \bar{W}_{1K}, \alpha, K) \mid \bar{W}_{1K} > 0]\,\text{pr}(\bar{W}_{1K} > 0) \\ &= \text{pr}[\bar{W}_{1K} - \theta\bar{W}_{2K} \geqq \bar{W}_{1K} - \theta_3(\bar{W}_{2K}, \bar{W}_{1K}, \alpha, K)\bar{W}_{2K} \mid \bar{W}_{1K} \\ &\quad > 0]\,\text{pr}(\bar{W}_{1K} > 0).\end{aligned}$$

Let $\bar{\theta}_3 = \theta_3(\bar{W}_{2K}, \bar{W}_{1K}, \alpha, K)$ and $\bar{\varepsilon} = \varepsilon(\bar{W}_{2K}, \bar{W}_{1K}, \theta_3(\bar{W}_{2K}, \bar{W}_{1K}, \alpha, K))$. Observe that if $\bar{\theta}_3 \geqq \theta$, then (A4) implies that

$$(\alpha/2)^{1/K} = R(\theta, \varepsilon(\theta), h^*(\theta, \varepsilon(\theta)), \mu_1, \mu_2) \geqq R(\theta, \bar{\varepsilon}, h^*(\theta, \varepsilon(\theta)), \mu_1, \mu_2).$$

Since part (ii) of the theorem gives

$$\begin{aligned}\operatorname{pr}(\bar{W}_{1K} - \theta\bar{W}_{2K} \geqq \bar{\varepsilon} \mid \bar{W}_{1K} > 0) &\leqq R(\theta, \bar{\varepsilon}, h^*(\theta, \bar{\varepsilon}), \mu_1, \mu_2)^K \\ &\leqq R(\theta, \bar{\varepsilon}, h^*(\theta, \varepsilon(\theta)), \mu_1, \mu_2)^K,\end{aligned}$$

it follows that

$$\operatorname{pr}(\bar{W}_{1K} - \theta\bar{W}_{2K} \geqq \bar{\varepsilon} \mid \bar{W}_{1K} > 0) = \operatorname{pr}[\theta \leqq \theta_3(\bar{W}_{2K}, \bar{W}_{1K}, \alpha, K) \mid > \bar{W}_{1K} > 0] \leqq \alpha/2$$

so that, finally,

$$\operatorname{pr}(\theta \leqq \theta_*) = \operatorname{pr}[\theta \leqq \theta_3(\bar{W}_{2K}, \bar{W}_{1K}, \alpha, K) \mid \bar{W}_{1K} > 0] \operatorname{pr}(\bar{W}_{1K} > 0) \leqq \alpha/2.$$

Since $\theta = (\mu_0 - \mu_1)/(\mu_2 - \mu_1)$, (25) follows directly. An analogous argument establishes (26), and (27) is merely a consequence of (25) and (26).

Acknowledgments. I am grateful to Mr. Christos Alexopoulos for his programming assistance and to the reviewers for their constructive comments.

REFERENCES

[1] E. C. FIELLER AND H. O. HARTLEY, *Sampling with control variables*, Biometrika, 41 (1954), pp. 494–501.

[2] G. S. FISHMAN, *A Monte Carlo sampling plan for estimating network reliability*, Operations Res., 34 (1986), pp. 581–594.

[3] ———, *A Monte Carlo sampling plan for estimating the distribution of maximal flow in a system with discrete component flows*, Tech. Report UNC/OR/TR-86/18, Department of Operations Research, University of North Carolina, Chapel Hill, NC, 1986.

[4] A. ITAI AND Y. SHILOACH, *Maximum flow in planar networks*, SIAM J. Comput., 8 (1979), pp. 135–150.

[5] S. S. LAVENBERG AND P. D. WELCH, *A perspective on the use of control variables to increase the efficiency of Monte Carlo simulations*, Management Sci., 27 (1981), pp. 332–335.

[6] R. Y. RUBINSTEIN AND R. MARCUS, *Efficiency of multivariate control variates in Monte Carlo simulation*, Operations Res., 33 (1985), pp. 661–677.

[7] K. TOCHER, *The Art of Simulation*, van Nostrand, New York, 1963.

[8] J. WILSON, *Variance reduction techniques for digital simulation*, American J. Math. Management Sci., 4 (1984), pp. 277–312.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 1, pp. 205-209, January 1989

© 1989 Society for Industrial and Applied Mathematics
015

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 this journal with an accelerated reviewing process. It will be possible for the note to appear approximately six months after the date of acceptance.

BISECTION IS NOT OPTIMAL ON VECTOR PROCESSORS*

HORST D. SIMON†

Abstract. Recently there has been revived interest in the bisection method for computing eigenvalues of symmetric tridiagonal matrices, since this method lends itself easily to a parallel implementation. A natural extension of the bisection method is the multisection method. The relative advantages of these two methods have been discussed in several publications ([H. Bernstein and M. Goldstein, *SIAM J. Sci. Statist. Comput.*, 9(1988), pp. 601-602], [I. Ipsen and E. Jessup, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, Tech. Report YALEU/DCS/RR-548, Yale Univ., Dept. of Computer Science, July 1987], [S. Lo, B. Philippe, and A. Sameh, *SIAM J. Sci. Statist. Comput.*, 8(1987), pp. s155-s165]). The purpose of this note is to contribute another argument in favor of using the multisection method, which did not arise explicitly in the past discussion. A simple analysis and some numerical examples show that the bisection method is in general not optimal in the class of multisection methods for the extraction of one eigenvalue on vector processors. Numerical results on a CRAY-2, a Convex C1-XP, and an Alliant FX/8 show that the optimal multisection section method can be several times faster than the bisection method.

Key words. symmetric tridiagonal eigenvalue problem, bisection method, multisection method, parallel bisection, vector processors

AMS(MOS) subject classification. 65F15

1. Introduction.

Several researchers have recently investigated parallel algorithms for the symmetric tridiagonal eigenvalue problem [2], [5], [7], [8]. Dongarra and Sorensen [5] implemented the divide-and-conquer algorithm by Cuppen [3], and obtained considerable speedups over TQL2 from EISPACK [6],[12] on a CRAY X-MP/4 and an Alliant FX/8. Lo, Philippe, and Sameh [8] used a combination of bisection and inverse iteration and demonstrated an even better performance of their method than the divide-and-conquer algorithm on the Alliant FX-8, when computing all the eigenvalues and vectors. Ipsen and Jessup [7] investigated the same algorithms on a hypercube, and arrived at the conclusion that bisection and multisection are more efficient than Cuppen's divide-and-conquer method on distributed-memory parallel processors, with bisection faster than multisection.

The authors in [8] state that they prefer multisection over bisection in what they call the isolation phase. They give two reasons: (a) multisection creates more tasks than does bisection and thus achieves a better load balancing, and (b) there are several eigenvalues in one interval. In a recent timely communication Bernstein and Goldstein [1] argue against these two reasons for preferring multisection in the isolation phase. The purpose of this communication is to add another argument in favor of multisection to the current discussion. This argument has so far not been explicitly stated in any

* Received by the editors July 18, 1988; accepted for publication September 22, 1988.

† Numerical Aerodynamic Simulation (NAS) Systems Division, National Aeronautics and Space Administration Ames Research Center, Mail Stop 258-5, Moffett Field, California 94035. (The author is an employee of Boeing Computer Services.)

of the references: even for the computation of a single eigenvalue, bisection is not optimal on a vector processor.

In addition to introducing a new argument into the ongoing bisection versus multisection discussion, this observation has several other potential consequences. Two of the second-generation distributed-memory parallel computers, which have been introduced recently, provide vector processing [10],[11]. Thus the observation of this note also may change the outcome of Ipsen and Jessup's [7] numerical studies in favor of multisection, when carried out on a hypercube with vector processing at the nodes. Multisection is also an option considered in the design of LAPACK [4]. In [4] multisection is considered only as an option for a parallel environment. Based on the results here, multisection is also a valid option for eigenvalue extraction on a single vector processor.

2. Analysis of multisection on vector processors. Given a symmetric tridiagonal $n \times n$ matrix T, with

$$T = \begin{pmatrix} \alpha_1 & \beta_2 & 0 & \dots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ 0 & \dots & 0 & \beta_n & \alpha_n \end{pmatrix},$$

The eigenvalues λ_i, $i = 1 \cdots n$ can be determined as the zeros of the so-called bottom-pivot function $\delta_n(\lambda)$, which is defined as follows:

$$\begin{aligned} \delta_1(\lambda) &= \alpha_1 - \lambda, \\ \delta_j(\lambda) &= \alpha_j - \lambda - \frac{\beta_j^2}{\delta_{j-1}(\lambda)}. \end{aligned}$$

The δ_j's are the elements of the diagonal matrix D in the LDL^T factorization of $T - \lambda I$. For a given point ς, the bottom pivot is commonly evaluated in the following loop (see [9]):

```
BISECTION ALGORITHM: δ ← α_1 − ς
                     ν ← 0
                     for j = 2 ··· n do
                         δ ← α_j − ς − β_j^2/δ
                         if (δ = 0) then δ ← ε
                         if (δ < 0) then ν ← ν + 1
                     end do
```

Here ϵ is a small multiple of the round-off unit of the machine used. Using Sylvester's inertia theorem, it follows that ν is equal to the number of eigenvalues less than ς (for details see [9]). Using ν the implementation of a bisection algorithm for computing an eigenvalue of T is straightforward.

Because of the nonlinear recurrence, the loop evaluating δ does not vectorize. However, if the value of ν is to be evaluated for p distinct trial points ς_k, $k = 1 \cdots p$, then the two loops can simply be interchanged. The additional cost are three arrays of the length p for the storage of the ς, δ, and ν values. This is trivial on today's vector machines. The benefit is that the evaluation of multiple δ's and ν's now vectorizes.

A simple analysis shows that on a scalar machine bisection is optimal, and no extra speedup is gained by using a multisection scheme. However, on a vector processor the simple loop interchange above allows us to evaluate ν for several points p at some higher speed. This may compensate for the additional arithmetic, which is performed by a multisection algorithm. That this is indeed the case will now be shown.

Suppose we want to locate an eigenvalue in the interval $[a_0, b_0]$, with length $l_0 = b_0 - a_0$, using a multisection algorithm with p points. At each step of a multisection

TABLE 1
Optimal number of multisection points.

$\frac{s}{r}$	p	$\frac{s}{r}$	p
0.0	1	9.7	7
1.3	2	11.8	8
2.6	3	14.1	9
4.1	4	16.4	10
5.8	5	18.9	11
7.7	6	21.5	12

algorithm, the length of the interval including the desired eigenvalue is reduced by a factor of $(p+1)^{-1}$. In order to reduce the initial interval to a length l_j, with $l_j/l_0 < \epsilon_0$, where ϵ_0 is some given tolerance, we thus have to carry out at least j multisection steps, with

$$j = \lceil \frac{-\log(\epsilon_0)}{\log(p+1)} \rceil.$$

On the other hand, the evaluation of the recurrence for p points at each of the j steps can be carried out in vector mode in time proportional to

$$s + rp, \tag{1}$$

where s stands for the start-up cost of the vector loop and r the asymptotic rate. Hence the total cost C of the multisection algorithm is given by

$$C = (s + rp)\lceil \frac{-\log(\epsilon)}{\log(p+1)} \rceil.$$

Ignoring the ceiling function for the moment, it easily checked that C is minimized for a value p_0 satisfying the equation

$$(p+1)\log(p+1) - p = \frac{s}{r}. \tag{2}$$

Thus the optimal number of multisection points is a function of the ratio s/r, i.e., the ratio of start-up time and asymptotic rate for the loop described in the bisection algorithm above. Note that s/r is equal to the parameter $n_{1/2}$, which is used to denote the half-performance length in vector processing. Solving equation (2) for p one obtains Table 1, which lists the optimal number of multisection points as a function of s/r. For a start-up time $s = 0$, the known fact that bisection is optimal for scalar processors is recovered. For vector processors, we expect a value of s/r of about 10. This would indicate that a multisection algorithm with about seven or

eight points is optimal on such a machine.

3. Numerical results. In order to validate the above analysis, a multisection algorithm has been implemented on several vector processors. In a first numerical experiment the inner loop of the multisection algorithm was timed. Execution times were fitted to the linear model given by (1) by a least-squares fit. The resulting measurements for the ratio s/r are listed in Table 2, together with the asymptotic rate in MFLOPS, and the resulting optimal number of points according to (2). The implementation was in FORTRAN, and all compilers vectorized the two if-statements

TABLE 2
Measurements of s/r for several vector processors.

Machine	Compiler	MFLOPS	$\frac{s}{r}$	p_{opt}
CRAY-2	cft2	29.5	9.8	7
CRAY-2	cft77	28.1	17.4	10
Alliant FX/8 (1CE)	fortran -Ogv	1.0	7.6	6
Alliant FX/8 (4CE)	fortran -O	3.4	60.0	25
Convex C1-XP	fc -O2	1.5	10.3	7

TABLE 3
Comparison bisection versus multisection.

Machine	Bisection	Multisection	p_{opt}	Ratio
CRAY-2 (cft2)	0.299	0.098	13	3.05
CRAY-2 (cft77)	0.293	0.092	18	3.18
Alliant FX/8 (1CE)	4.102	1.894	6	2.17
Alliant FX/8 (4CE)	7.164	1.881	37	3.81
Convex C1-XP	3.532	1.285	13	2.75

within the loop. The multisection loop was timed for lengths up to 2,048 iterations (i.e., multisection points). All times are averages over 50 runs, and all results reported are for 64 bit arithmetic. All machines are at the NASA Ames Research Center in Moffett Field, California.

Table 2 also lists the results for four processors of an Alliant FX-8. These are included because the model above carries over easily to this situation and yields an example of a machine with a high ratio s/r. The CRAY-2 in the benchmark is a newer model with the faster dynamic random access memory (DRAM). Results on an older CRAY-2 with the slower DRAM were not significantly different to warrant their inclusion here.

The results in Table 2 are only timing information for the simple inner loop of the multisection algorithm. They do not indicate the potential speedup using a multisection algorithm on these machines. This information is provided in Table 3. The execution times in Table 3 were obtained from running the multisection algorithm on a random tridiagonal matrix of order 3,000. The multisection algorithm was used to compute the smallest eigenvalue of the matrix. Table 3 lists both the execution time (in seconds) for bisection ($p = 1$) and for multisection with the optimal p.

The results in Table 3 show the multisection algorithm with optimal number of bisection points to be about three times faster than the bisection algorithm. The optimal p_{opt} in Table 3 has been obtained by inspection from runs of the multisection

algorithm, with p varying from 1 to 60. The p_{opt} from Table 3 is different from the predicted value in Table 2. The execution times in Table 3 take the whole subroutine into account, whereas Table 2 is based on the evaluation of the simple multisection loop alone. The theoretical analysis gives an approximate prediction of the number of optimal multisection points, with the numbers in Table 2 giving an underestimate.

The results in Table 3 demonstrate that multisection is considerably more efficient than bisection for vector processors and therefore should be considered in the design of LAPACK. Some other closely related questions, for example, the use of asymptotically faster methods such as zeroin and when to switch from multisection to QR, are beyond the scope of this timely communication, but also require reevaluation in the light of our new results.

REFERENCES

[1] H. BERNSTEIN AND M. GOLDSTEIN, *Optimizing Givens' algorithm for multiprocessors*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 601 – 602.

[2] ———, *Parallel implementation of bisection for the calculation of eigenvalues of tridiagonal symmetric matrices*, Tech. Rep., Courant Institute, New York University, 1985.

[3] J. J. CUPPEN, *A divide and conquer method for the symmetric tridiagonal eigenvalue problem*, Numer. Math., 36 (1981), pp. 177 – 195.

[4] J. DEMMEL, J. DU CROZ, S. HAMMARLING, AND D. SORENSEN, *Guidelines for the design of symmetric eigenroutines,* SVD, *and iterative refinement and condition estimation for linear systems*, Tech. Report MCSD 111, Argonne National Laboratory, February 1988.

[5] J. J. DONGARRA AND D. C. SORENSEN, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s139 – s154.

[6] B. S. GARBOW, J. M. BOYLE, J. J. DONGARRA, AND C. B. MOLER, *Matrix Eigensystem Routines* - EISPACK *Guide Extension*, Lecture Notes in Computer Sciences, Vol. 51, Springer-Verlag, Berlin, New York, 1977.

[7] I. IPSEN AND E. JESSUP, *Solving the symmetric tridiagonal eigenvalue problem on the hypercube*, Tech. Report YALEU/DCS/RR-548, Yale Univ., Dept. of Computer Science, July 1987.

[8] S. LO, B. PHILIPPE, AND A. SAMEH, *A multiprocessor algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Statist.. Comput., 8 (1987), pp. s155 – s165.

[9] B. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ, 1980.

[10] D. SCOTT AND J. RATTNER, *The scalability of Intel concurrent super computers*, in Proc. Third International Conference on Supercomputing, International Supercomputing Institute, Boston, 1988, pp. 121 – 125.

[11] Y. SHIH AND A. KERNEK, *A new generation in parallel processing systems*, SuperComputing, Winter (1988), pp. 5 – 28.

[12] 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*, Lecture Notes in Computer Sciences, Vol. 6, Springer-Verlag, Berlin, New York, 1976.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 211-226, March 1989
© 1989 Society for Industrial and Applied Mathematics

A DOMAIN DECOMPOSITION LAPLACE SOLVER FOR INTERNAL COMBUSTION ENGINE MODELING*

CHRISTOPH BÖRGERS† AND OLOF B. WIDLUND‡

Abstract. Discrete elliptic problems can often naturally be partitioned into subproblems corresponding to subregions into which the region has been partitioned, or from which it was originally assembled. Fast conjugate gradient methods have been developed to handle the continuity conditions, across the interfaces, that need to be satisfied by the solution of the entire problem.

This paper focuses on a special family of problems that arises in combustion engine modeling. It is demonstrated that efficient algorithms can conveniently be built using standard software to solve elliptic problems on simple regions. A convergence analysis is given that is much simpler than the general theory for domain decomposition algorithms. It is also shown that the use of multigrid cycles to solve the subproblems inexactly can be very efficient.

Key words. Laplace solver, domain decomposition, multigrid methods

AMS(MOS) subject classifications. 65N20, 65F10

1. Introduction. The interest in domain decomposition methods has increased considerably in recent years, to a large extent because of their promise on parallel computers. There are frequent conferences (cf. Glowinski et al. (1987)). Much has been learned about how to design optimal or almost optimal iterative algorithms to handle the interaction between solvers on the subregions. In this paper, we will examine the use of one of these methods for a particular problem arising in combustion engine modeling. We note, however, that the ideas explored here can be adopted to other classes of problems in fluid dynamics, etc. By focusing on a quite specific problem, we are able to demonstrate that the theory sometimes can be simplified, that the algorithms can be speeded up by using special features, and that standard software can be used to assemble a solver on relatively simple but nontrivial regions.

We describe a Laplace solver on a domain Ω of the shape shown in Fig. 1. This solver will be used to compute the potential flow component in a random vortex method applied to a problem with a moving piston (see Sethian (1987)). Related work is described, e.g., in Sethian (1984) and Sethian and Ghoniem (1986).

In Fig. 1, the large rectangle $\Omega^{(2)}$ represents a cylinder, and the small attached quadrilaterals, the union of which we denote by $\Omega^{(1)}$, represent so-called inlets. There are no inlets attached to the lower side of $\Omega^{(2)}$. The number of inlets attached to the left, right, and upper sides of $\Omega^{(2)}$, their positions, and the angles they form with $\Omega^{(2)}$ are left variable in our program. Sethian's calculations are time-dependent with the lower side of $\Omega^{(2)}$ moving. It is a fractional step method, and one of the steps requires the solution of a Neumann problem of the following form:

$$-\Delta\phi = 0 \quad \text{on } \Omega, \tag{1.1}$$

* Received by the editors August 24, 1987; accepted for publication (in revised form) May 31, 1988.

† Department of Mathematics, University of Michigan, Ann Arbor, Michigan 48109. The work of this author 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, at the Lawrence Berkeley Laboratory, University of California, Berkeley, California.

‡ Courant Institute of Mathematical Sciences, 251 Mercer Street, New York University, New York, New York 10012. The work of this author was supported in part by the National Science Foundation under grant NSF-CCR-8703768 and by the U.S. Department of Energy under contract DE-AC02-76ER03077-V at the Courant Mathematics and Computing Laboratory, New York University, New York.

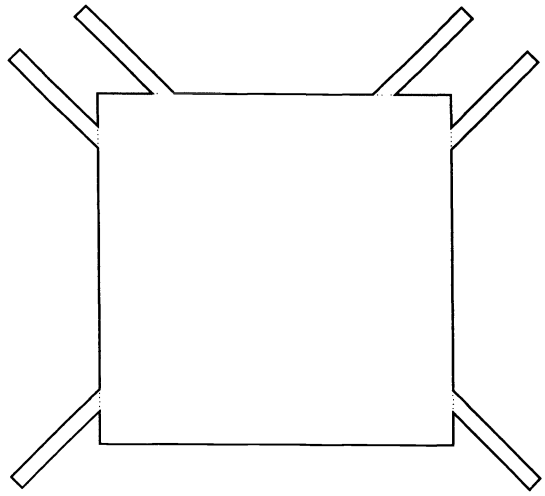


FIG. 1. *The region* $\Omega = \Omega^{(1)} \cup \Omega^{(2)}$. $\Omega^{(1)}$ *is the union of the small pieces, and* $\Omega^{(2)}$ *is the rectangle.*

$$\frac{\partial \phi}{\partial n} = g \quad \text{on } \partial\Omega. \tag{1.2}$$

Here ϕ is a velocity potential. The Neumann data g and the position of the lower side of $\Omega^{(2)}$, representing the top of the piston, vary with time. The lowest inlet always remains above the lower side of $\Omega^{(2)}$.

Equations (1.1) and (1.2) are discretized with piecewise linear finite elements on a quasi-uniform mesh as described in § 2. The region has re-entrant corners, and this is known to limit the regularity of the solution (cf. Grisvard (1985)). As a consequence, the discretization will yield only a crude approximation to ϕ, unless the mesh width is very small. On the other hand, the resulting system of linear equations has a relatively simple structure and can be solved very efficiently. We will focus on this case in this paper, but will discuss possible remedies through local refinement of the mesh in the last section. One of the domain decomposition methods studied by Bjørstad and Widlund (1984), (1986), the so-called "excellent" or Neumann-Dirichlet algorithm, will be used. A careful motivation of this choice is given in those papers. For the problem at hand, we use a fast Poisson solver for the rectangular region and band Cholesky for the relatively narrow inlets. The general convergence analysis of this method requires tools from the theory of elliptic boundary value problems (see Bjørstad and Widlund (1986), Widlund (1987b).) However, for the geometry and meshes under consideration, a much simpler convergence analysis, using elementary mappings and reflections, can be given. This results in quantitative and realistic bounds (see § 4, where results of numerical experiments are also presented).

Recently there has been an increased interest in the use of inexact solvers for the problems defined on the subregions (see Bramble, Pasciak, and Schatz (1986), Widlund (1987b)). In § 5, we demonstrate that the exact solver for $\Omega^{(2)}$ can be replaced by a multigrid cycle. The number of outer iterations grows modestly, but the cost per step is considerably decreased. We also give bounds for the rate of convergence of this variant of the algorithm. In § 6, we collect a number of ideas that could considerably further enhance the performance of the iterative method as well as the accuracy of the discretization.

The code used in the numerical experiments is available from the authors.

2. Triangulation of the region. The region is the union of a rectangle $\Omega^{(2)}$ and N_{inlets} small quadrilaterals called inlets. $\Omega^{(2)}$ is covered by a regular rectangular mesh, with mesh widths h_x in the x-direction and h_y in the y-direction. We use logically rectangular meshes on the inlets (see Fig. 2) with common mesh points on the interface between any inlet and the rectangle.

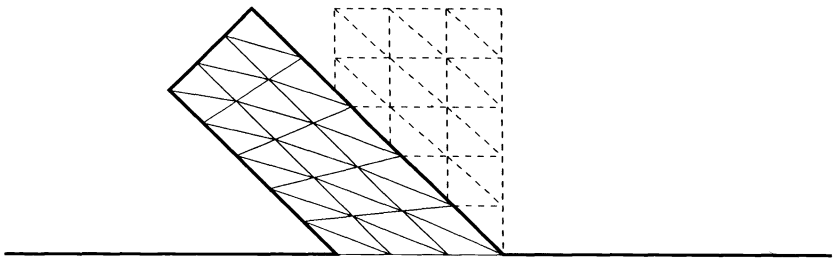


FIG. 2. *Solid rule*: I_i *and its triangulation. Dashed rule*: R_i *and its triangulation.*

The triangulation of the rectangle is obtained by cutting each cell of the rectangular mesh into two triangles. We do not need to specify the direction of these cuts, since the same five-point formula results when we use piecewise linear finite elements.

We triangulate an inlet by defining a bilinear mapping ψ_i from a rectangle R_i onto the inlet I_i, as indicated in Fig. 2. On the rectangle R_i, a regular, rectangular mesh is defined. While the mesh width in one coordinate direction is determined by the mesh on the large rectangle, the other can be chosen freely. We triangulate R_i by dividing each of its mesh cells into two triangles. This triangulation induces a triangulation of the inlet as shown in Fig. 2.

3. A preconditioned conjugate gradient method. Let $A \in R^{n\times n}$ be a symmetric, positive semidefinite matrix, and let $\mathbf{b} \in R^n$ be a vector orthogonal to the kernel of A. Let $\tilde{A} \in R^{n\times n}$ be symmetric and positive semidefinite, with $ker\,(\tilde{A}) \subset ker\,(A)$. Using the preconditioner $\tilde{A}$, there are several possible preconditioned conjugate gradient algorithms for

$$A\mathbf{u} = \mathbf{b} \tag{3.1}$$

which are mathematically equivalent but algorithmically different. The version given below is particularly useful for domain decomposition methods. The matrix A only appears in a vector-matrix multiplication with $A - \tilde{A}$. In our application, this is much cheaper than multiplying with A. In addition, we need a procedure that gives us the product of the Moore–Penrose pseudoinverse $\tilde{A}^+$ of $\tilde{A}$ with an arbitrary vector.

PRECONDITIONED CONJUGATE GRADIENT ALGORITHM.

Let $\mathbf{u}^{(0)} := \mathbf{0} \in R^n$.

$\mathbf{g}^{(0)} := \mathbf{b}$

Replace $\mathbf{g}^{(0)}$ by its orthogonal projection onto the range of A.

$\mathbf{d}^{(0)} := \mathbf{g}^{(0)}$

$\tilde{\mathbf{g}}^{(0)} := \tilde{A}^+\mathbf{g}^{(0)}$

$\tilde{\mathbf{d}}^{(0)} := \tilde{\mathbf{g}}^{(0)}$

For $j = 0, 1, 2, \cdots$:

$\alpha^{(j)} := [\mathbf{g}^{(j)T}\tilde{\mathbf{g}}^{(j)}]/[\tilde{\mathbf{d}}^{(j)T}((A-\tilde{A})\tilde{\mathbf{d}}^{(j)}+\mathbf{d}^{(j)})]$
$\mathbf{u}^{(j+1)} := \mathbf{u}^{(j)} + \alpha^{(j)}\tilde{\mathbf{d}}^{(j)}$
$\mathbf{g}^{(j+1)} := \mathbf{g}^{(j)} - \alpha^{(j)}[(A-\tilde{A})\tilde{\mathbf{d}}^{(j)}+\mathbf{d}^{(j)}]$
Replace $\mathbf{g}^{(j+1)}$ by its orthogonal projection onto the range of A.
$\tilde{\mathbf{g}}^{(j+1)} := \tilde{A}^{+}\mathbf{g}^{(j+1)}$
$\beta^{(j)} := [\tilde{\mathbf{g}}^{(j+1)T}\mathbf{g}^{(j+1)}]/[\tilde{\mathbf{g}}^{(j)T}\mathbf{g}^{(j)}]$
$\mathbf{d}^{(j+1)} := \mathbf{g}^{(j+1)} + \beta^{(j)}\mathbf{d}^{(j)}$
$\tilde{\mathbf{d}}^{(j+1)} := \tilde{\mathbf{g}}^{(j+1)} + \beta^{(j)}\tilde{\mathbf{d}}^{(j)}$.

The vectors $\mathbf{u}^{(j)}$ converge to a solution of (3.1).

The orthogonal projections onto the range of A have no effect in exact arithmetic, but these steps are sometimes necessary to obtain convergence in floating point arithmetic.

4. The domain decomposition algorithm. In this section, we describe the Neumann-Dirichlet domain decomposition method as given in Bjørstad and Widlund (1984), (1986). Because of the simple structure of our problem, we can give a simplified convergence analysis. We also present numerical results.

4.1. Notation and preliminary remarks. The finite element approximation of (1.1), (1.2) can be written as a system of linear equations of the form

$$K\mathbf{x} = \mathbf{b}, \tag{4.1}$$

with K, $\mathbf{x}$, and $\mathbf{b}$ partitioned as follows:

$$K = \begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & K_{22} & K_{23} \\ K_{13}^T & K_{23}^T & K_{33} \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix}. \tag{4.2}$$

Here the subscript 1 refers to the nodes of the inlets not belonging to the interfaces between the inlets and the rectangle, the subscript 2 to the nodes in the rectangle not belonging to the interfaces, and the subscript 3 refers to the nodes on the interfaces.

The entries of the stiffness matrix K are integrals of the form

$$\int_\Omega \nabla\phi \cdot \nabla\psi \, d\mathbf{x}, \tag{4.3}$$

where ϕ and ψ are canonical basis functions of the space of piecewise linear finite elements. These integrals can be written in the form

$$\int_\Omega = \int_{\Omega^{(1)}} + \int_{\Omega^{(2)}}. \tag{4.4}$$

We obtain a corresponding splitting $K = K^{(1)} + K^{(2)}$, where

$$K^{(1)} = \begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & 0 & 0 \\ K_{13}^T & 0 & K_{33}^{(1)} \end{pmatrix}, \tag{4.5}$$

and

$$K^{(2)} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & K_{22} & K_{23} \\ 0 & K_{23}^T & K_{33}^{(2)} \end{pmatrix}, \tag{4.6}$$

and thus

$$K_{33} = K_{33}^{(1)} + K_{33}^{(2)}. \tag{4.7}$$

We note that a system of the form

$$\begin{pmatrix} K_{22} & K_{23} \\ K_{23}^T & K_{33}^{(2)} \end{pmatrix} \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix} \tag{4.8}$$

is a discrete Neumann problem on $\Omega^{(2)}$.

The following matrix, the so-called Schur complement, plays a central role below:

$$S := K_{33} - K_{13}^T K_{11}^{-1} K_{13} - K_{23}^T K_{22}^{-1} K_{23}. \tag{4.9}$$

Since S can be shown to be a diagonal block in a block Cholesky factorization of K, it is symmetric and positive semidefinite, and its kernel is *span* (**1**), the set of constant vectors. The two subregions each contribute

$$S^{(j)} := K_{33}^{(j)} = K_{j3}^T K_{jj}^{-1} K_{j3}, \qquad j = 1, 2 \tag{4.10}$$

to S. $S^{(1)}$ and $S^{(2)}$ are symmetric and positive semidefinite with $ker\,(S^{(j)}) = span\,(\mathbf{1})$, and

$$S = S^{(1)} + S^{(2)}. \tag{4.11}$$

The linear system (4.8) is easy to solve, using standard fast Laplace solvers for rectangular domains. It is easy to see that the solution $\mathbf{x}_3$ of

$$S^{(2)}\mathbf{x}_3 = \mathbf{r}_3 \tag{4.12}$$

is a subvector of the solution of

$$\begin{pmatrix} K_{22} & K_{23} \\ K_{23}^T & K_{33}^{(2)} \end{pmatrix} \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{r}_3 \end{pmatrix}. \tag{4.13}$$

Therefore (4.12) can be solved with a fast Poisson solver. The application of

$$S^{(1)} = K_{33}^{(1)} - K_{13}^T K_{11}^{-1} K_{13} \tag{4.14}$$

to a vector $\mathbf{y}_3$ is also easy and inexpensive. The main work required is the solution of the system

$$K_{11}\mathbf{y}_1 = K_{13}\mathbf{y}_3. \tag{4.15}$$

We use band Gaussian elimination for this purpose, observing that K_{11} is block diagonal with N_{inlets} blocks. To reduce the band width of a block of K_{11}, we order the unknowns as indicated in Fig. 3. We denote the number of nodes on the interface by m_i.

From the point of view of band width, it does not matter whether the nodes closest to the rectangle $\Omega^{(2)}$ are numbered first or last. However, we order them last for the following reason. During the iteration, the right-hand side in (4.15) is nonzero only

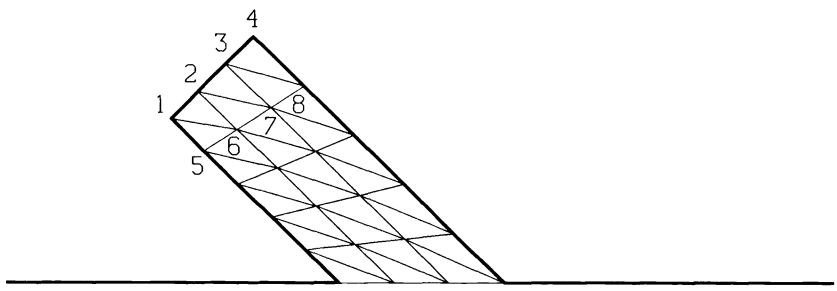


FIG. 3. *Ordering of the unknowns in the inlets.*

in the components corresponding to the mesh layer in the inlets closest to $\Omega^{(2)}$, and we only need the same components of $\mathbf{x}_1$ to form $S^{(1)}\mathbf{r}_3$. If we order these nodes last, most of the work in the solution phase of the Cholesky algorithm becomes unnecessary. In fact, only the bottom right m_i-by-m_i submatrices of the triangular factors of the stiffness matrices for the inlets need to be saved. The amount of work per iteration on the ith inlet will therefore be proportional to m_i^2, once the stiffness matrices corresponding to the inlets have been factored. Since K_{11} is unaffected by the position of the bottom of $\Omega^{(2)}$, which changes with time, this matrix is factored only once.

4.2. Statement of the algorithm. When solving (4.1), we first reduce the right-hand side $\mathbf{b}$ to the form

$$\begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{\rho}_3 \end{pmatrix}. \tag{4.16}$$

This is accomplished by first solving a problem with the matrix

$$\begin{pmatrix} K_{22} & K_{23} \\ K_{23}^T & K_{33}^{(2)} \end{pmatrix}, \tag{4.17}$$

i.e., a discrete Neumann problem, and then a problem with the matrix K_{11}. A problem of the form

$$\begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & K_{22} & K_{23} \\ K_{13}^T & K_{23}^T & K_{33} \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \boldsymbol{\rho}_3 \end{pmatrix} \tag{4.18}$$

results, which, after expressing $\mathbf{y}_1$ and $\mathbf{y}_2$ in terms of $\mathbf{y}_3$, is seen to be equivalent to

$$S\mathbf{y}_3 = \boldsymbol{\rho}_3. \tag{4.19}$$

This equation is solved by using the preconditioned conjugate gradient method, with the preconditioner $S^{(2)}$.

Once the solution $\mathbf{y}_3$ of (4.19) is known, the complete solution of (4.18) is obtained by solving

$$K_{11}\mathbf{y}_1 = -K_{13}\mathbf{y}_3 \tag{4.20}$$

for $\mathbf{y}_1$, and then

$$\begin{pmatrix} K_{22} & K_{23} \\ K_{23}^T & K_{33}^{(2)} \end{pmatrix} \begin{pmatrix} \mathbf{y}_2 \\ \mathbf{y}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \boldsymbol{\rho}_3 - K_{13}^T\mathbf{y}_1 - K_{33}^{(1)}\mathbf{y}_3 \end{pmatrix} \tag{4.21}$$

for $\mathbf{y}_2$.

This concludes the description of the algorithm.

We note that we never need to compute the elements of S and $S^{(2)}$. The straightforward computation of S would require the solution of p linear systems with the matrix K_{11}, and the same number of systems with the matrix K_{22}, where p is the total number of nodes on the interfaces between $\Omega^{(1)}$ and $\Omega^{(2)}$. Linear systems with K_{22} are not as easy to solve as those with the matrix (4.17), since the matrix K_{22} represents discretized boundary value problems with Dirichlet conditions along the interfaces, and Neumann conditions elsewhere. Such a system is, in general, nonseparable and fast direct Poisson solvers are not available.

If the elements of $S^{(2)}$ were required, they could also be computed by first computing $(S^{(2)})^{-1}$, by using a fast Poisson solver p times (cf. (4.12) and (4.13)). Thereafter $(S^{(2)})^{-1}$ is inverted. As we have noted, these computations are unnecessary for our algorithm.

4.3. Convergence analysis. From the theory of the conjugate gradient method, it is known that the number of steps needed to reach a prescribed accuracy is bounded above by a function of

$$\kappa = \max \frac{\mathbf{x}_3^T S \mathbf{x}_3}{\mathbf{x}_3^T S^{(2)} \mathbf{x}_3} \Big/ \min \frac{\mathbf{x}_3^T S \mathbf{x}_3}{\mathbf{x}_3^T S^{(2)} \mathbf{x}_3}, \tag{4.22}$$

where the maximum and minimum are taken over the orthogonal complement of **1** (see, e.g., Luenberger (1969, p. 296)). This function grows linearly with the square root of κ for large κ.

The condition number κ is estimated in two steps (cf., e.g., Bjørstad and Widlund (1986)). In the first step, given in Lemma 1, only linear algebra is needed. The second step, given in Lemma 2, is in general more difficult and requires tools from analysis (see Widlund (1987a)). However, in the present case, this argument can be very much simplified.

LEMMA 1. *Let C be the smallest constant such that, for all $\binom{\mathbf{x}_2}{\mathbf{x}_3}$, there is an $\mathbf{x}_1$ such that*

$$\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}^T \begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & K_{22} & K_{23} \\ K_{13}^T & K_{23}^T & K_{33} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} \leqq C \cdot \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}^T \begin{pmatrix} K_{22} & K_{23} \\ K_{23}^T & K_{33}^{(2)} \end{pmatrix} \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}. \tag{4.23}$$

Then

$$\kappa \leqq C. \tag{4.24}$$

This estimate is not necessarily sharp.

Proof. We first note that the denominator in (4.22) is always $\geqq 1$. Therefore

$$\kappa \leqq \max \frac{\mathbf{x}_3^T S \mathbf{x}_3}{\mathbf{x}_3^T S^{(2)} \mathbf{x}_3}. \tag{4.25}$$

Inequality (4.25) is not necessarily sharp. Consider, e.g., a case where Ω is a rectangle, divided into two halves. Then, by symmetry, $S^{(1)} = S^{(2)}$, and therefore the right-hand side in (4.25) becomes 2, but $\kappa = 1$.

The right-hand side in (4.25) is bounded by the constant C referred to in Lemma 1. To see this, note that, from the definitions of S and $S^{(2)}$,

$$\max \frac{\mathbf{x}_3^T S \mathbf{x}_3}{\mathbf{x}_3^T S^{(2)} \mathbf{x}_3} = \max \frac{\begin{pmatrix} -K_{11}^{-1} K_{13} \mathbf{x}_3 \\ -K_{22}^{-1} K_{23} \mathbf{x}_3 \\ \mathbf{x}_3 \end{pmatrix}^T \begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & K_{22} & K_{23} \\ K_{23}^T & K_{23}^T & K_{33} \end{pmatrix} \begin{pmatrix} -K_{11}^{-1} K_{13} \mathbf{x}_3 \\ -K_{22}^{-1} K_{23} \mathbf{x}_3 \\ \mathbf{x}_3 \end{pmatrix}}{\begin{pmatrix} -K_{22}^{-1} K_{23} \mathbf{x}_3 \\ \mathbf{x}_3 \end{pmatrix}^T \begin{pmatrix} K_{22} & K_{23} \\ K_{23}^T & K_{33}^{(2)} \end{pmatrix} \begin{pmatrix} -K_{22}^{-1} K_{23} \mathbf{x}_3 \\ \mathbf{x}_3 \end{pmatrix}} \tag{4.26}$$

The proof of Lemma 1 is completed by showing, by a straightforward computation, that

$$\begin{aligned} &\begin{pmatrix} -K_{11}^{-1} K_{13} \mathbf{x}_3 \\ -K_{22}^{-1} K_{23} \mathbf{x}_3 \\ \mathbf{x}_3 \end{pmatrix}^T \begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & K_{22} & K_{23} \\ K_{13}^T & K_{23}^T & K_{33} \end{pmatrix} \begin{pmatrix} -K_{11}^{-1} K_{13} \mathbf{x}_3 \\ -K_{22}^{-1} K_{23} \mathbf{x}_3 \\ \mathbf{x}_3 \end{pmatrix} \\ &\leqq \begin{pmatrix} \mathbf{x}_1 \\ -K_{22}^{-1} K_{23} \mathbf{x}_3 \\ \mathbf{x}_3 \end{pmatrix}^T \begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & K_{22} & K_{23} \\ K_{13}^T & K_{23}^T & K_{33} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ -K_{22}^{-1} K_{23} \mathbf{x}_3 \\ \mathbf{x}_3 \end{pmatrix} \end{aligned} \tag{4.27}$$

for all vectors $\mathbf{x}_1$ and $\mathbf{x}_3$. □

We note that the right-hand side in (4.25) is exactly equal to the constant C.

For Lemma 2, the following notation is needed. I_i is the ith inlet and R_i is the corresponding rectangle (cf. § 2 and Fig. 2). Recall that the mesh on I_i is the image of a regular rectangular mesh on R_i. Let h_i be the mesh width in R_i in the direction normal to the interface.

LEMMA 2. *Assume that the mesh ratios h_x/h_i and h_y/h_i are uniformly bounded. Then the constant C of Lemma 1 is uniformly bounded.*

Proof. Let

$$\begin{pmatrix} x_2 \\ x_3 \end{pmatrix} \tag{4.28}$$

be a vector representing a finite-element function ϕ on $\Omega^{(2)}$. The quadratic form on the right-hand side of (4.23) is the Dirichlet integral of ϕ. Similarly, the left-hand side of (4.23) is the Dirichlet integral of a finite-element function extending ϕ to all Ω. To estimate C, we should therefore study the extension of finite-element functions from $\Omega^{(2)}$ to Ω. A general extension theorem of this kind for conforming finite elements has been established by Widlund (1987a).

When studying the extension problem for the current geometry, we first make three simplifying assumptions: (1) The inlets are rectangular. (2) The mesh widths in the inlets are the same as those of $\Omega^{(2)}$. (3) The mirror images of the inlets are completely contained in $\Omega^{(2)}$ and do not overlap each other (see Fig. 4).

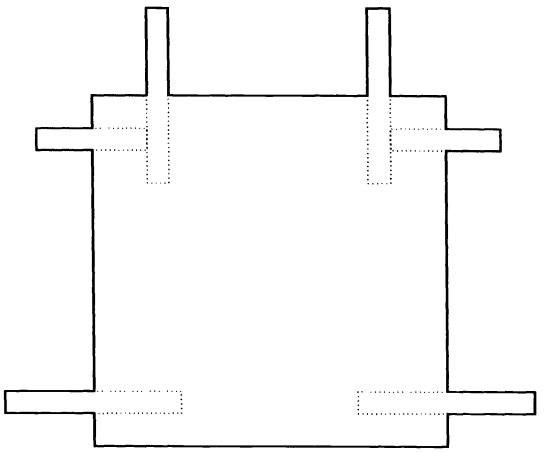


FIG. 4. *A domain satisfying assumptions* (1) *and* (3) *in the proof of Lemma* 2.

The finite-element function ϕ is extended into the inlets as an even function with respect to the direction normal to the interface. The Dirichlet integral over $\Omega^{(1)}$ of the extended function is bounded by the Dirichlet integral of ϕ over $\Omega^{(2)}$, and thus

$$C \leqq 2. \tag{4.29}$$

We note that this argument would fail if (1.2) were a Dirichlet condition, since the extended function would have to vanish on the three outer sides of each inlet.

The remainder of the proof consists of the step-by-step removal of the assumptions (1)-(3). We begin with assumption (3). If the reflected images of the inlets in $\Omega^{(2)}$

overlap each other, (4.29) is replaced by

$$C \leqq 4, \tag{4.30}$$

since no point in $\Omega^{(2)}$ can be covered by more than three reflected images.

If one of the inlets is so long that its reflected image is not entirely contained in $\Omega^{(2)}$, the reflection does not define the extension in the entire inlet, but only in a piece of it. However, the extension can be completed by repeated reflection. Let, for example, the left side of $\Omega^{(2)}$ be the line $x=0$, and the right side of $\Omega^{(2)}$ be the line $x=a$. If (x, y) is a point in a left inlet, with $-2a<x<-a$, then

$$\bar{\phi}(x, y) := \bar{\phi}(-2a-x, y), \tag{4.31}$$

and

$$\bar{\phi}(-2a-x, y) := \phi(2a+x, y). \tag{4.32}$$

This again leads to a larger, but mesh-independent C.

Suppose now that assumption (2) is violated, e.g., that the horizontal mesh width h in a left inlet differs from the horizontal mesh width h_x in $\Omega^{(2)}$. The extension function is given by

$$\bar{\phi}(-x, y) := \phi\left(x \cdot \frac{h_x}{h}, y\right). \tag{4.33}$$

The Dirichlet integral of $\bar{\phi}$ can easily be estimated in terms of the Dirichlet integral of ϕ. If $h_x \leqq h_i$ for all i, the previous estimate for C remains unchanged. Otherwise, it is enlarged by a factor q, where q is the maximum of all quotients h_x/h_i and h_y/h_i. This factor is, by the assumption of Lemma 2, bounded independently of the mesh.

Assumption (1) can be eliminated by using a similar argument. The inlets are images of rectangular regions, and the corresponding mapping functions ψ_i naturally enter. The estimate for C is then enlarged by the factor

$$\max_i \max_{\mathbf{x}} \rho[\det(D\psi_i(\mathbf{x}))(D\psi_i(\mathbf{x}))^{-T}(D\psi_i(\mathbf{x}))^{-1}] + o(1), \tag{4.34}$$

where $\rho[\cdots]$ denotes the spectral radius, and ψ_i are the bilinear mappings used in the mesh construction (see § 2). This follows from an estimate of the Dirichlet integral and a change of variables. □

From Lemmas 1 and 2, we obtain the following convergence theorem.

THEOREM. *Let the mesh ratios be bounded as in Lemma 2. Then the condition number κ defined in* (4.22) *is bounded independently of the mesh.*

This analysis and the numerical results below show that $S^{(2)}$ is an excellent preconditioner for S.

It is also possible to use $S^{(1)}$ as a preconditioner for S. As we noticed previously, there is a considerable difference between a Neumann and a Dirichlet problem. For a Dirichlet problem, $S^{(1)}$ would be the better choice (see, e.g., Fig. 8 in Anderson (1985)). The reason is that in the Dirichlet case, the extension of certain finite-element functions from $\Omega^{(2)}$ into $\Omega^{(1)}$ will introduce large gradients, since we have to impose homogeneous Dirichlet conditions on the outer sides of each inlet. This results in a large Dirichlet integral over $\Omega^{(1)}$. On the other hand, the extension from $\Omega^{(1)}$ to $\Omega^{(2)}$ can easily be accomplished in the Dirichlet case if assumptions (1)–(3) are satisfied. We use reflections and then extend the resulting function by zero to the rest of $\Omega^{(2)}$, obtaining an extended function with a Dirichlet integral, that is twice as large.

TABLE 1

Experiments for the region in Fig. 1. The size of the mesh, including the mesh points on the boundary, is 33μ *by* 33μ *in the rectangle* $\Omega^{(2)}$, *and* 3μ *by* 9μ *in each of the inlets.*

μ	Condition number	ε	Number of iterations	μ	Condition number	ε	Number of iterations
		10^{-1}	3			10^{-1}	3
		10^{-2}	4			10^{-2}	4
		10^{-3}	5			10^{-3}	6
		10^{-4}	5			10^{-4}	7
1	2.88	10^{-5}	6	3	3.12	10^{-5}	9
		10^{-6}	7			10^{-6}	10
		10^{-7}	7			10^{-7}	10
		10^{-8}	8			10^{-8}	11
		10^{-9}	9			10^{-9}	12
		10^{-10}	9			10^{-10}	13
		10^{-1}	3			10^{-1}	3
		10^{-2}	4			10^{-2}	4
		10^{-3}	6			10^{-3}	6
		10^{-4}	7			10^{-4}	7
2	3.07	10^{-5}	8	4	3.14	10^{-5}	9
		10^{-6}	8			10^{-6}	10
		10^{-7}	9			10^{-7}	11
		10^{-8}	10			10^{-8}	12
		10^{-9}	11			10^{-9}	13
		10^{-10}	13			10^{-10}	14

For Neumann problems, we are not able to establish that preconditioning with $S^{(1)}$ leads to a large or small condition number. From a practical point of view, the use of $S^{(2)}$ is clearly preferable, since using $S^{(1)}$ as a preconditioner would make it necessary to solve nonseparable problems on $\Omega^{(2)}$.

4.4. Numerical results. We have conducted experiments on the regions shown in Figs. 1 and 4. The size of the mesh, including the mesh points on the boundary, is 33μ by 33μ in the rectangle $\Omega^{(2)}$, and 3μ by 9μ in each of the inlets, with $\mu = 1, 2, 3$, or 4. Tables 1 and 2 give information on the rate of convergence of the iterative method. The choice of boundary data has no significant influence on the performance. In these runs, random boundary data are used.

We give the number of iterations required for

$$|\rho_3 - Sy_3|_2 \leqq \varepsilon |\rho_3|_2, \tag{4.35}$$

(cf. (4.19)), and also determine the condition number κ by using the coefficients generated by the conjugate gradient method (see, e.g., O'Leary and Widlund (1979)).

5. Using an inexact solver on the rectangle.

5.1. A different view of the domain decomposition algorithm. The iteration described in § 4 requires an application of $(S^{(2)})^+$, and thus a call to a fast Laplace solver on $\Omega^{(2)}$ in every step. It is not practical to replace this solver by an approximate solver without making other changes in the algorithm, for the following reason: The use of an inexact solver can be interpreted as replacing $(S^{(2)})^+$ by an approximation. It is easy to construct an inexact solver such that the approximation takes the form $(\tilde{S}^{(2)})^+$, with a symmetric, positive semidefinite matrix $\tilde{S}^{(2)}$ with $ker\,(\tilde{S}^{(2)}) = span\,(\mathbf{1})$. However, the resulting domain decomposition method requires, in each step, one application of

TABLE 2

Experiments for the region in Fig. 4. The size of the mesh, including the mesh points on the boundary, is 33μ *by* 33μ *in the rectangle* $\Omega^{(2)}$, *and* 3μ *by* 9μ *in each of the inlets.*

μ	Condition number	ε	Number of iterations	μ	Condition number	ε	Number of iterations
		10^{-1}	2			10^{-1}	2
		10^{-2}	3			10^{-2}	3
		10^{-3}	4			10^{-3}	4
		10^{-4}	4			10^{-4}	5
1	1.90	10^{-5}	5	3	2.00	10^{-5}	6
		10^{-6}	5			10^{-6}	7
		10^{-7}	5			10^{-7}	7
		10^{-8}	6			10^{-8}	8
		10^{-9}	7			10^{-9}	9
		10^{-10}	8			10^{-10}	9
		10^{-1}	2			10^{-1}	2
		10^{-2}	3			10^{-2}	3
		10^{-3}	4			10^{-3}	4
		10^{-4}	5			10^{-4}	5
2	2.00	10^{-5}	6	4	2.00	10^{-5}	6
		10^{-6}	6			10^{-6}	7
		10^{-7}	7			10^{-7}	7
		10^{-8}	8			10^{-8}	8
		10^{-9}	8			10^{-9}	9
		10^{-10}	9			10^{-10}	10

$S-\tilde{S}^{(2)}$ and one application of $(\tilde{S}^{(2)})^{+}$, or alternatively, one application of S and one of $(\tilde{S}^{(2)})^{+}$. This is not practical because applications of $S-\tilde{S}^{(2)}$ or S would require the exact solution of linear systems with the matrix K_{22}.

The following modification of the algorithm of § 4.2 makes possible the use of an inexact solver. We first reduce the right-hand side of (4.1) to the form

$$\begin{pmatrix} \mathbf{0} \\ \boldsymbol{\rho}_2 \\ \boldsymbol{\rho}_3 \end{pmatrix}. \tag{5.1}$$

This is accomplished by solving a problem with the matrix K_{11}, resulting in a problem of the form

$$\begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & K_{22} & K_{23} \\ K_{13}^T & K_{23}^T & K_{33} \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \mathbf{y}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \boldsymbol{\rho}_2 \\ \boldsymbol{\rho}_3 \end{pmatrix}. \tag{5.2}$$

This system can be reduced to

$$R \begin{pmatrix} \mathbf{y}_2 \\ \mathbf{y}_3 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\rho}_2 \\ \boldsymbol{\rho}_3 \end{pmatrix}, \tag{5.3}$$

with

$$R := \begin{pmatrix} K_{22} & K_{23} \\ K_{23}^T & K_{33} - K_{13}^T K_{11}^{-1} K_{13} \end{pmatrix}. \tag{5.4}$$

Equation (5.3) is solved by using the preconditioned conjugate gradient method, with the preconditioner

$$\tilde{R} := \begin{pmatrix} K_{22} & K_{23} \\ K_{23}^T & K_{33}^{(2)} \end{pmatrix}. \tag{5.5}$$

Once the solution of (5.3) is known, the complete solution of (5.2) is obtained by solving

$$K_{11}\mathbf{y}_1 = -K_{13}\mathbf{y}_3. \tag{5.6}$$

This concludes the description of the modified algorithm.

By replacing Lemma 1 by Lemma 3, the convergence analysis for the modified algorithm can be completed.

LEMMA 3. *The bound C of Lemma* 1 *is also a bound on*

$$\max \frac{\begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}^T R \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}}{\begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}^T \tilde{R} \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}} \Bigg/ \min \frac{\begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}^T R \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}}{\begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}^T \tilde{R} \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}}. \tag{5.7}$$

For the design of algorithms using inexact solvers on both subregions, see Bramble, Pasciak, and Schatz (1986), and Widlund (1987b).

5.2. Multigrid cycles with symmetric iteration matrices. In § 5.3, the fast solver on $\Omega^{(2)}$ will be replaced by a multigrid cycle. We will require the iteration matrix of the multigrid cycle to be symmetric. This can be accomplished as follows.

Consider a system of n linear equations

$$A\mathbf{x} = \mathbf{b} \tag{5.8}$$

with $A = A^T \geqq 0$. We first assume that A is invertible, and discuss the case of a nontrivial kernel later.

We first recall the definition of the well-known two-level correction cycle (see, e.g., Stüben and Trottenberg (1982)). Let m be an integer; m is usually smaller than the number n of equations in the system to be solved. Let $B \in R^{m \times m}$ be a symmetric matrix, which should be chosen as an m-dimensional approximation to A^{-1}. Let R be a linear mapping $R^n \to R^m$. A two-level correction step is defined by

$$\mathbf{x}_{\text{new}} := \mathbf{x}_{\text{old}} + R^T B R(\mathbf{b} - A\mathbf{x}_{\text{old}}). \tag{5.9}$$

The complete two-level cycle is obtained by supplementing (5.9) with a relaxation method. We consider Gauss–Seidel block relaxation here. Let I be any subset of $\{1, \cdots, n\}$. Given an approximation $\mathbf{x}$ for $A^{-1}\mathbf{b}$, we can determine $\boldsymbol{\delta} \in R^n$ with $\delta_i = 0$ for $i \notin I$, and such that the jth component of $\mathbf{b} - A(\mathbf{x} + \boldsymbol{\delta})$ is zero for all $j \in I$. We then say that $\mathbf{x} + \boldsymbol{\delta}$ is obtained from $\mathbf{x}$ by the relaxation of the block I. Let $I_1, I_2, \cdots, I_r$ be subsets of $\{1, \cdots, n\}$, the union of which is $\{1, \cdots, n\}$.

TWO-LEVEL CORRECTION CYCLE.

Choose $\mathbf{x}_0$ and generate $\mathbf{x}_1, \mathbf{x}_2, \cdots$ as follows.

$\mathbf{x}_{i+1}$ is obtained from $\mathbf{x}_i$ by relaxing over the blocks $I_1, I_2, \cdots, I_r$, carrying out one step of (5.9), and relaxing over the blocks $I_r, I_{r-1}, \cdots, I_1$.

The iteration (5.9) and each individual block relaxation step have symmetric iteration matrices with respect to the energy product, i.e., the inner product defined by A. Therefore we have the following lemma.

LEMMA 4. *The iteration matrix M of the two-level correction cycle is symmetric with respect to the energy product.*

A mapping $\tilde{G}$ is defined as follows. For $\mathbf{b} \in R^n$, $\tilde{G}\mathbf{b}$ is the result of a two-level correction cycle for the problem

$$A\mathbf{x} = \mathbf{b}, \tag{5.10}$$

starting with the initial guess $\mathbf{x} = \mathbf{0}$. Denoting

$$G := A^{-1}, \tag{5.11}$$

we obtain

$$\tilde{G} = (I - M)G. \tag{5.12}$$

LEMMA 5. *If M is symmetric in the energy inner product, and if $\rho(M) < 1$, then $\tilde{G}$ is symmetric in the Euclidean inner product and positive definite, and*

$$cond\ (\tilde{G}^{-1/2} G \tilde{G}^{-1/2}) \leqq \frac{1+\rho(M)}{1-\rho(M)}. \tag{5.13}$$

Proof. Since M is symmetric in the energy product, MG and thus $\tilde{G}$ are symmetric in the Euclidean inner product. Next we show that $\tilde{G}$ is positive definite if $\rho(M) < 1$. $\tilde{G}$ is congruent with $G^{-1/2}(I - M)G^{1/2}$, which is similar to $I - M$. Since M is symmetric in the energy inner product, M has only real eigenvalues, and if $\rho(M) < 1$, $I - M$ has only positive real eigenvalues. Using Sylvester's theorem, it follows that $\tilde{G}$ has only positive eigenvalues. We now estimate $cond\ (\tilde{G}^{-1/2} G \tilde{G}^{-1/2})$:

$$cond\ (\tilde{G}^{-1/2} G \tilde{G}^{-1/2}) = cond\ (G^{-1/2} \tilde{G} G^{-1/2}) = cond\ (G^{-1/2}(I - M)G^{1/2}) \tag{5.14}$$

$$= \frac{\lambda_{\max}(I-M)}{\lambda_{\min}(I-M)} = \frac{1-\lambda_{\min}(M)}{1-\lambda_{\max}(M)} \leqq \frac{1+\rho(M)}{1-\rho(M)}. \tag{5.15}$$

This concludes the proof of Lemma 5. □

Since we are interested in multigrid cycles for Neumann problems, we will now study the case of a constant null vector. We consider

$$A\mathbf{x} = \mathbf{b} \tag{5.16}$$

and assume that $\mathbf{b} \in R^n$ satisfies the compatibility condition $\mathbf{b} \cdot \mathbf{1} = 0$. Formula (5.9) remains unchanged. If I is a proper subset of $\{1, \cdot \ldots \cdot, n\}$, then the corresponding principal minor of A is invertible, and therefore the relaxation of the block I is well defined. The definition of the two-level cycle remains unchanged. The solution is defined only modulo constant, and therefore the iteration matrix M is defined as a mapping of the orthogonal complement of *span* $(\mathbf{1})$ into itself.

For $\mathbf{b} \in R^n$ with $\mathbf{b} \cdot \mathbf{1} = 0$, let $\mathbf{x}^{(1)}$ be the result of a two-level cycle for $A\mathbf{x} = \mathbf{b}$, with the initial guess $\mathbf{x}^{(0)} = \mathbf{0}$, and let $\tilde{G}b$ be the projection of $\mathbf{x}^{(1)}$ onto the orthogonal complement of *span* $(\mathbf{1})$.

LEMMA 6. *$\tilde{G}$ is symmetric with respect to the Euclidean inner product, and a positive definite mapping of the orthogonal complement of span $(\mathbf{1})$ onto itself. Denoting this restriction by $\tilde{G}^*$, we have*

$$cond\ [(\tilde{G}^*)^{-1/2} G^* (\tilde{G}^*)^{-1/2}] \leqq \frac{1+\rho(M)}{1-\rho(M)}, \tag{5.17}$$

where G^* is the restriction of G to the orthogonal complement of span $(\mathbf{1})$.

The proof is similar to that of Lemma 5.

The discussion in this section has been restricted to two-level cycles. However, it is easy to obtain analogous statements for arbitrarily many levels, using, e.g., *V*- or *W*-cycles (see, e.g., Stüben and Trottenberg (1982) for the definition of these cycles).

5.3. The domain decomposition algorithm using a multigrid cycle as an inexact solver. We have carried out numerical experiments with the modified domain decomposition algorithm described in § 5.1, where the exact fast solver is replaced by a multigrid V-cycle. The ratio of the mesh widths of consecutive levels is $\frac{1}{2}$. Piecewise bilinear interpolation is used to transfer the corrections from a given level to the next finer level, and residuals are transferred from a given level to the next coarser level by using the adjoint of the bilinear interpolation operator. The relaxation method is red-black Gauss-Seidel iteration. Three half sweeps (red-black-red) are carried out before and after each coarse grid-correction step.

We find numerically that the spectral radius of the iteration matrix of this cycle is not larger than 0.18 on $n+1$ by $n+1$ meshes, where n is a power of two. By rearranging the relaxation sweeps in a nonsymmetric way, we could obtain smaller multigrid convergence factors for the same amount of work. We have, however, used the symmetric arrangement since it is needed for the analysis of the method.

By using the framework of §§ 5.1 and 5.2, we can interpret this method as a preconditioned conjugate gradient method for (5.3), using a symmetric, positive definite preconditioner $\tilde{R}^*$ with $ker\,(\tilde{R}^*) = span\,(\mathbf{1})$, and with

$$\max \frac{\begin{pmatrix} x_2 \\ x_3 \end{pmatrix}^T R \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}}{\begin{pmatrix} x_2 \\ x_3 \end{pmatrix}^T \tilde{R}^* \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}} \Bigg/ \min \frac{\begin{pmatrix} x_2 \\ x_3 \end{pmatrix}^T R \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}}{\begin{pmatrix} x_2 \\ x_3 \end{pmatrix}^T \tilde{R}^* \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}} \leqq \frac{1+0.18}{1-0.18} C \approx 1.44C. \tag{5.18}$$

Here C is the constant defined in Lemma 1.

Table 3 repeats the tests of Table 1, using the method described above. The numbers of iterations in Table 3 are somewhat larger than those in Table 1. For a systematic comparison between the cost of a multigrid cycle and several fast direct solvers, see Foerster and Witsch (1981). In their study, a multigrid cycle, quite similar to the one we used, is shown to be faster than the best of the direct solvers considered, but the gain in speed does not exceed a factor 1.5. Considerations of convenience rather than speed might therefore determine the choice of a solver on the rectangular region. We note that multigrid methods can be applied successfully to many problems for which there are no fast direct solvers available. For a suggestion of how much faster methods might be developed for the problem at hand, cf. the next section.

6. Further directions. In this section, we will discuss the possibility of using faster solvers on the rectangle and mesh refinement in neighborhoods of the re-entrant corners.

We begin by noticing that during the iteration, the solution of the discrete elliptic problem on the rectangle is only required on and next to the interfaces, i.e., at relatively few points. Similarly, the right-hand side of the equation differs from zero only on the interfaces. Problems of this kind have been considered by Banegas (1978), and more recently by several Soviet scientists (cf., e.g., Bakhvalov and Orekhov (1982), Kuznetsov and Matsokin (1982)). When solving the discrete Poisson problem on an n-by-n mesh using Bakhvalov's method, only on the order of $n \log (n)^2$ operations are required to obtain the solution on one mesh line within a prescribed accuracy. When using this method, the cost of solving our problem on the entire region to prescribed accuracy would asymptotically, i.e., for sufficiently fine meshes, be dominated by the cost of the final step in which the solution is obtained at all mesh points.

The problem of the lack of accuracy caused by the re-entrant corners could be addressed systematically, using tools similar to those in this paper. In the work of Bai and Brandt (1987), Bramble, Ewing, Pasciak, and Schatz (1987), Hart and McCormick

TABLE 3

Experiments for the region in Fig. 1, using the method of § 5. The size of the mesh, including the mesh points on the boundary, is 33μ by 33μ in the rectangle $\Omega^{(2)}$, and 3μ by 9μ in each inlet.

μ	Condition number	ε	Number of iterations	μ	Condition number	ε	Number of iterations
		10^{-1}	3			10^{-1}	3
		10^{-2}	4			10^{-2}	5
		10^{-3}	6			10^{-3}	7
		10^{-4}	8			10^{-4}	9
1	3.51	10^{-5}	9	4	3.84	10^{-5}	11
		10^{-5}	10			10^{-6}	13
		10^{-7}	11			10^{-7}	15
		10^{-8}	12			10^{-8}	16
		10^{-9}	15			10^{-9}	18
		10^{-10}	15			10^{-10}	19
		10^{-1}	3				
		10^{-2}	5				
		10^{-3}	7				
		10^{-4}	8				
2	3.75	10^{-5}	10				
		10^{-6}	11				
		10^{-7}	13				
		10^{-8}	15				
		10^{-9}	17				
		10^{-10}	19				

(1987), McCormick (1984), and McCormick and Thomas (1986), the questions of designing accurate approximations on locally refined meshes and the effective solution of the resulting linear systems of equations are addressed. Some of these algorithms are multigrid methods, while others require exact solvers for the unrefined problems as well as the refined models on the patches of refinement. It has been demonstrated that certain of these methods work quite well for a considerable number of refinement levels. We note that the relationship between domain decomposition and iterative refinement methods is explored in Widlund (1988).

What comes to mind for our current problem would be to design a refinement of the mesh locally around the re-entrant corners, possibly using several successive refinements. If the refined patches of the rectangle were logically rectangular, then iterative methods could be designed using only fast Poisson solvers, as in the algorithm of § 4. We might prefer to handle the inlets by Gaussian elimination, while the finite-element problems on the locally refined meshes of $\Omega^{(2)}$ could be solved inexactly, e.g., by one step of an iterative refinement algorithm (cf. Hart and McCormick (1987) and Widlund (1988)). A suitable domain decomposition algorithm would be used as an outer iteration. Such an algorithm would be similar to the one outlined in § 5. It is also possible to use domain decomposition on the different mesh levels separately and an iterative refinement method as an outer iteration.

Acknowledgments. Much of this work was done while the second author was enjoying the hospitality of Alexandre Chorin and the Department of Mathematics of the University of California at Berkeley. The authors also thank Gene Golub for his interest and helpful comments, and James Sethian for proposing the problem.

REFERENCES

C. R. ANDERSON (1985), *On domain decomposition*, Manuscript Classic-85-09, Center for Large Scale Scientific Computation, Stanford University, Stanford, CA.

D. BAI AND A. BRANDT (1987), *Local mesh refinement multilevel techniques*, SIAM J. Sci. Stat. Comput., 8, pp. 109-134.

N. S. BAKHVALOV AND M. YU. OREKHOV (1982), *On fast methods of solving Poisson's equations*, U.S.S.R. Comput. Math. and Math. Phys., 22, pp. 107-114.

A. BANEGAS (1978), *Fast Poisson solvers for problems with sparsity*, Math. Comp., 32, pp. 441-446.

P. E. BJØRSTAD AND O. B. WIDLUND (1984), *Solving elliptic problems on regions partitioned into substructures*, in Elliptic Problem Solvers II, G. Birkhoff and A. Schoenstadt, eds., Academic Press, Orlando Beach, FL, pp. 245-256.

——— (1986), *Iterative methods for the solution of elliptic problems on regions partitioned into substructures*, SIAM J. Numer. Anal., 23, pp. 1097-1120.

J. H. BRAMBLE, R. E. EWING, J. E. PASCIAK, AND A. H. SCHATZ (1987), *A preconditioning technique for the efficient solution of problems with local grid refinement*, Comput. Methods Appl. Mech. Engrg.

J. H. BRAMBLE, J. E. PASCIAK, AND A. H. SCHATZ (1986), *An iterative method for elliptic problems on regions partitioned into substructures*, Math. Comp., 46, pp. 361-369.

H. FOERSTER AND K. WITSCH (1981), *On efficient multigrid software for elliptic problems on rectangular domains*, Math. Comput. Simulation, 23, pp. 293-298.

R. GLOWINSKI, G. H. GOLUB, G. A. MEURANT, AND J. PÉRIAUX, EDS. (1987), in Proc. 1st International Symposium on Domain Decomposition Methods for Partial Differential Equations, Paris, France, January 7-9, 1987, Society for Industrial and Applied Mathematics, Philadelphia, PA.

P. GRISVARD (1985), *Elliptic Problems in Nonsmooth Domains*, Pitman, Boston.

L. HART AND S. MCCORMICK (1987), *Asynchronous multilevel adaptive methods for solving partial differential equations on multiprocessors: Computational Analysis*, Tech. Report, Computational Mathematics Group, University of Colorado, Denver, CO.

YU. A. KUZNETSOV AND A. M. MATSOKIN (1982), *Partial solution of systems of linear algebraic equations*, in Numerical Methods in Applied Mathematics, Nauka, Sibirsk. Otdel., Novosibirsk, pp. 143-163. (In Russian.)

D. G. LUENBERGER (1969), *Optimization by Vector Space Methods*, John Wiley, New York.

S. MCCORMICK (1984), *Fast adaptive composite grid* (FAC) *methods: Theory for the variational case*, Computing (Suppl.), 5, pp. 115-121.

S. MCCORMICK AND J. THOMAS (1986), *The fast adaptive composite grid* (FAC) *method for elliptic equations*, Math. Comp., 46, pp. 439-456.

D. P. O'LEARY AND O. B. WIDLUND (1979), *Capacitance matrix methods for the Helmholtz equation on general three-dimensional regions*, Math. Comp., 33, pp. 849-879.

J. A. SETHIAN (1984), *Turbulent combustion in open and closed vessels*, J. Comp. Phys., 54, pp. 425-456.

——— (1987), *Development of large eddies in moving pistons*, J. Comput. Phys.

J. A. SETHIAN AND A. F. GHONIEM (1986), *Validation of vortex methods*, J. Comput. Phys.; also LBL-Report 21767, Lawrence Berkeley Laboratory, University of California, Berkeley, CA.

K. STÜBEN AND U. TROTTENBERG (1982), *Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications*, Lecture Notes in Mathematics 960, Springer-Verlag, Berlin.

O. B. WIDLUND (1987a), *An extension theorem for finite element spaces and three applications*, in Proc. 2nd GAMM-Seminar, Kiel, January 17-19, 1986; Notes Numer. Fluid Mech., 16, pp. 110-122.

——— (1987b), *Iterative substructuring methods: Algorithms and theory for elliptic problems in the plane*, in Proc. 1st International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Paris, France, January 7-9, 1987, Society for Industrial and Applied Mathematics, Philadelphia, PA 1987; Tech. Report 287, Computer Science Department, Courant Institute of Mathematical Sciences, New York University, NY.

——— (1988), *Optimal iterative refinement methods*, Tech. Report, Computer Science Department, Courant Institute of Mathematical Science, New York University, NY.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 227-252, March 1989

© 1989 Society for Industrial and Applied Mathematics
002

ADAPTIVE IMPLICIT CRITERIA FOR TWO-PHASE FLOW WITH GRAVITY AND CAPILLARY PRESSURE*

P. A. FORSYTH†

Abstract. Based on monotonicity conditions, the switching criteria for an adaptive implicit discretization for two-phase flow in a porous medium with gravity and capillary pressure are derived. These criteria can be applied to multidimensional flows, provided an easily checked condition is satisfied. Use of the monotonicity conditions is demonstrated for some example problems in one and two dimensions.

Key words. adaptive implicit, two-phase flow, porous medium

AMS(MOS) subject classifications. 65M05, 76S05

1. Introduction. Multiphase flow problems in porous media arise in oil reservoir simulation [1], contamination of groundwater with organic pollutants [2], and geothermal energy extraction. Recently, adaptive implicit methods have been used to solve reservoir simulation problems [3]–[9]. The basic idea of these techniques is to use a fully implicit discretization only in those cells undergoing large flow rates, while using an IMPES (implicit pressure, explicit saturation) method elsewhere. This permits large timesteps with less computational work than a fully implicit discretization. Full details of the adaptive implicit method can be obtained in references [5] and [10]. Various heuristic methods have been used as criteria for switching from IMPES to fully implicit and vice versa. In reference [5], a conservative strategy is proposed, based on changes observed over a timestep, and allowing only switching from IMPES to fully implicit. Other authors have proposed an alternative criterion that permits backward switching [8], but this criterion does not necessarily give a monotone discretization.

All the above methods for adaptive implicit switching are based on the concept of preventing stepwise instability. In other words, given a fixed grid size, these methods attempt to ensure that the solution remains bounded as the number of timesteps becomes large. However, as will be demonstrated, it is possible to obtain bounded solutions that produce nonphysical local maxima and minima. This problem can be avoided by using monotonicity requirements as adaptive implicit criteria.

Note that a true adaptive implicit discretization is used in [5], [8], [10]. This is not simply zeroing the elements of the Jacobian as suggested by several authors [4], [6].

The objective of this article is to examine some two-phase flow problems, and to determine the local monotonicity conditions. The spatial discretization used in this study involves phase upstream weighting. This method is widely used in practice, since it is easily applied to complex flow problems involving gravity and capillary pressure. However, upstream weighting has the disadvantage that shock fronts are smeared over several cells. Grid-orientation effects can also be severe in the case of adverse mobility ratio flows [2]. Front tracking methods [11] have been proposed as an alternative to upstream weighting, so that shocks remain sharp. Grid orientation can also be reduced by using high-order Godunov methods [12]. However, since this article is concerned with adaptive implicit criteria, attention will be restricted to upstream weighted spatial discretizations. Upstream weighting has the advantage that techniques for the solution

* Received by the editors January 25, 1988; accepted for publication (in revised form) June 29, 1988. This work was supported by the National Sciences and Engineering Research Council of Canada and by the Information Technology Research Center, which is funded by the Province of Ontario.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

of the nonlinear algebraic equations resulting from fully implicit timestepping are fairly well established. The primary goal of an adaptive implicit method is to obtain a locally monotone solution cheaply, while using timesteps comparable to a fully implicit method. Stability will generally follow from local monotonicity. A monotone discretization will prevent the appearance of nonphysical local maxima and minima.

Note that the adaptive implicit method is a timestepping technique. The spatial discretization is similar to that used with a standard upstream weighted IMPES or fully implicit method. Only the level of time evaluation of the flow terms (IMPES or fully implicit) is adjusted to preserve local monotonicity. Consequently, it is expected that the adaptive implicit solution will be close to a small timestep IMPES solution, or a fully implicit solution with the same timesteps. The solutions will not exactly coincide, however, due to the different time truncation errors. Some computational examples showing the difference between IMPES, fully implicit and adaptive implicit will be given in the following.

2. Formulation. Recently, Sammon [13] has used monotonicity arguments to demonstrate that phase upstream weighting used for one-dimensional, two-phase flow with gravity but zero capillary pressure, converges to the physically correct solution satisfying the entropy condition. Similar arguments will be extended in the following sections to determine the monotonicity conditions for two-phase flow with gravity and capillary pressure. Of course, the inclusion of a finite capillary pressure means that the system of equations, even in one dimension, is not purely hyperbolic. However, we are only interested in adaptive implicit criteria, not convergence issues. Since these conditions can be applied on a cell-by-cell basis, it is possible to determine criteria for two-dimensional problems if certain tests are satisfied.

For definiteness in the following, we take the two phases to be oil (o) and water (w). The equations for two-phase, incompressible flow are given by [1]:

$$\frac{\partial}{\partial t}(S_o\phi) = \nabla \cdot [\lambda_o \nabla P_o - \rho_o g \nabla D],$$

$$\frac{\partial}{\partial t}(S_w\phi) = \nabla \cdot [\lambda_w \nabla P_w - \rho_w g \nabla D], \tag{1}$$

where

S_l = saturation of phase l = oil, water,
ϕ = porosity,
$\lambda_l = KK_{rl}/\mu_l$,
K_{rl} = relative permeability of phase l,
μ_l = viscosity of phase l,
K = absolute permeability,
D = depth,
g = gravitational acceleration,
P_l = pressure of phase l,
ρ_l = density of phase l.

The oil and water pressures are related through the capillary pressure, which is a function of water saturation S_w:

$$P_w = P_o + P_c(S_w) \tag{2}$$

while the relative permeability K_{rl} is an experimentally determined function of phase

saturation S_l [1]. Equations (1) are discretized using a cell-centered finite-difference method on a Cartesian grid [14]. It is assumed that cell edges are perpendicular to the line joining the cell centers. With these assumptions, a compact expression can be obtained for the discretized equations which is valid for any number of dimensions. If cell i has a set of nearest neighbours N_i, then the discretized equations can be written as follows:

$$S_{oi}^{N+1} - S_{oi}^{N} = \sum_{j \in N_i} \frac{\theta_j \lambda_{o,j+1/2}^{M}}{\Delta x_{j+1/2}} [(P_{oj}^{N+1} - P_{oi}^{N+1}) - \rho_o g (D_j - D_i)] \tag{3}$$

$$S_{wi}^{N+1} - S_{wi}^{N} = \sum_{j \in N_i} \theta_j \frac{\lambda_{w,j+1/2}}{\Delta x_{j+1/2}} [(P_{oj}^{N+1} - P_{oi}^{N+1}) - \rho_w g (D_j - D_i) - (P_{cj}^{M} - P_{ci}^{M})] \tag{4}$$

where

$$\begin{aligned} \theta_j &= A_{j+1/2} \Delta t / V_i \phi_i, \\ V_i &= \text{volume of the } i\text{th cell}, \\ A_{j+1/2} &= \text{interface area between cell } i \text{ and cell } j, \\ \Delta x_{j+1/2} &= (\Delta x_i + \Delta x_j)/2, \\ \Delta x_i &= \text{cell width in the direction from cell } i \text{ to cell } j. \end{aligned}$$

For convenience, the phase potentials ψ_{lj} are defined as follows:

$$\begin{aligned} \psi_{oj} &= \{P_{oj}^{N+1} - P_{oi}^{N+1} - \rho_o g (D_j - D_i)\} / \Delta x_{j+1/2}, \\ \psi_{wj} &= \{P_{oj}^{N+1} - P_{oi}^{N+1} - \rho_w g (D_j - D_i) - (P_{cj}^{M} - P_{ci}^{M})\} / \Delta x_{j+1/2}. \end{aligned} \tag{5}$$

The notation

$$\lambda_{l,j+1/2}$$

refers to the value of λ_l evaluated at the upstream point, depending on the sign of the phase potential ψ_l. Note that in (3)-(4), if for all j, $M = N$, then an IMPES method is used, while if for all j, $M = N+1$, then a fully implicit discretization is specified.

To avoid a profusion of subscripts, the following convention will be observed. The notation λ_l will be used to signify

$$\lambda_l = \lambda_{l,j+1/2}^{M} \tag{6}$$

with the sub/superscripts understood. Similarly, the notation P_{oi} will be used to mean

$$P_{oi} = P_{oi}^{N+1} \tag{7}$$

and

$$P_{ci} = P_{ci}^{M}, \qquad \Delta x = \Delta x_{j+1/2}.$$

Consequently, if (3) and (4) are added together, and noting that

$$S_o + S_w = 1$$

then

$$0 = \sum_j \frac{\theta_j}{\Delta x} [(\lambda_o + \lambda_w)(P_{oj} - P_{oi}) - (\rho_o g \lambda_o + \rho_w g \lambda_w)(D_j - D_i) - \lambda_w (P_{cj} - P_{ci})] \tag{8}$$

where we have used the notation of (6)-(7). Equations (3) and (8) are the starting point for further analysis. If the system is one-dimensional, or if the function

$$\lambda_o / (\lambda_o + \lambda_w)$$

is constant over N_i (this will be discussed later), then the pressure can be eliminated from (3) using (8), and the system could, in principle, be solved for S_{oi}^{N+1}:

$$S_{oi}^{N+1} = g_i(S_{oi}^N, S_{oj}^N, S_{oj}^{N+1}), \qquad j \in N_i. \tag{9}$$

Consequently, if $g_i(S_{oi}^N, S_{oj}^N, S_{oj}^{N+1})$ is a monotone function of its arguments in the interval $[a, b]$, that is,

$$\begin{gathered} \frac{\partial g_i}{\partial S_{oi}^N} \geqq 0, \quad \frac{\partial g_i}{\partial S_{oj}^N} \geqq 0, \quad \frac{\partial g_i}{\partial S_{oj}^{N+1}} \geqq 0, \\ \{S_{oj}^N, S_{oj}^{N+1}, S_{oi}^N\} \in [a, b], \end{gathered} \tag{10}$$

then bounds for the maximum and minimum values of S_{oi}^{N+1} are given by

$$\max\{S_{oi}^{N+1}\} \leqq g_i(b, b, b), \qquad \min\{S_{oi}^{N+1}\} \geqq g_i(a, a, a). \tag{11}$$

(The inequality signs are necessary if the arguments of g_i are not independent.) If, for example, an IMPES type discretization is used, and

$$g_i(a, a, a) = a;$$

then

$$S_{oi}^{N+1} \in [a, b] \quad \text{if } S_{oi}^N, S_{oj}^N \in [a, b],$$

and hence the discretization is stable.

The desirability of a monotone scheme is easily demonstrated. Suppose that an IMPES scheme is being used with

$$S_{oi}^N = S_{oj}^N = a$$

and suppose that

$$(S_{oi}^{N+1})^* = g_i(a, a, a).$$

Now consider an initial state with

$$\begin{gathered} S_{oi}^N = a + \alpha, \qquad S_{oj}^N = a, \\ \alpha \ll 1, \qquad \alpha > 0, \end{gathered}$$

then

$$S_{oi}^{N+1} = (S_{oi}^{N+1})^* + \frac{\partial g_i}{\partial S_{oi}^N}\bigg|_a \alpha + O(\alpha^2).$$

Suppose that g_i is not monotone, with

$$\frac{\partial g_i}{\partial S_{oi}^N} < 0$$

then for α sufficiently small,

$$S_{oi}^{N+1} < (S_{oi}^{N+1})^*.$$

In other words, a small positive perturbation of S_{oi}^N has produced a value of S_{oi}^{N+1}, which is less than the unperturbed value. This is physically absurd.

In the following, the conditions for monotonicity will be determined. As will be demonstrated, a monotone discretization will generally lead to a stable scheme with desirable properties. It will also be shown that, under certain assumptions, the fully

implicit method with phase-upstream weighting is monotone, as is the IMPES method for a sufficiently small timestep. Consequently, we will attempt to derive switching criteria that will ensure that the adaptive implicit method also has this same monotonicity property.

3. Monotonicity conditions. To avoid having to work out the conditions for various special cases, the most general expressions will be obtained. This requires somewhat involved but straightforward algebra.

As discussed above, we will need the derivatives of (3) and (8) with respect to S^M_{oK}. Differentiating (3) and using the notation of (5)–(7) gives

$$\frac{\partial S^{N+1}_{oi}}{\partial S^L_{oK}} = \delta_{i,K}\delta_{L,N} + \sum_j \theta_j \left[\lambda'_o \psi_{oj} + \lambda_o \left(\frac{\partial P_{oj}}{\partial S^L_{oK}} - \frac{\partial P_{oi}}{\partial S^L_{oK}} \right) \Big/ \Delta x \right] \tag{12}$$

where $\delta_{i,j}$ is the Kronecker delta, and

$$\lambda'_l = \left. \frac{d\lambda_l}{dS_o} \right|_{S_o = S^M_{o,j+1/2}} \delta_{j+1/2,K}\delta_{M,L}. \tag{13}$$

Differentiating (8) with respect to S^L_{oK}, we obtain

$$\begin{aligned} 0 = \sum_j \frac{\theta_j}{\Delta x} \Bigg\{ & (\lambda'_o + \lambda'_w)(P_{oj} - P_{oi}) + (\lambda_o + \lambda_w)\left(\frac{\partial P_{oj}}{\partial S^L_{oK}} - \frac{\partial P_{oi}}{\partial S^L_{oK}} \right) \\ & - (\lambda'_o \rho_o g + \lambda'_w \rho_w g)(D_j - D_i) - \lambda'_w (P_{cj} - P_{ci}) - \lambda_w (P'_{cj} - P'_{ci}) \Bigg\} \end{aligned} \tag{14}$$

where

$$P'_{ci} = \left. \frac{dP_c}{dS_o} \right|_{S_o = S^L_{oK}} \delta_{i,K}\delta_{L,M}. \tag{15}$$

It is convenient to define the discrete total mobility:

$$\begin{aligned} \lambda_t &= \lambda_o + \lambda_w \\ &= \lambda^M_{o,j+1/2} + \lambda^M_{w,j+1/2}. \end{aligned} \tag{16}$$

Note that the discrete λ_t is not necessarily equal to the continuous λ_t evaluated at a point, since the upstream point for the water phase may not be the upstream point for the oil phase.

In one dimension, each term of the sum in (14) is identically zero, since (8) is a difference form of

$$\nabla \cdot V_t = 0,$$

$$V_t = \lambda_t \nabla P_o - (\rho_o g \lambda_o + \rho_w g \lambda_w) \nabla D - \lambda_w \nabla P_c,$$

and hence implies that $V_t = \text{constant}$, and thus

$$\frac{\partial V_t}{\partial S^M_{oK}} = 0.$$

Consequently, in one dimension, we can multiply each term of (14) under the summation sign by

$$\begin{aligned} f_{oj} &= \lambda_o / (\lambda_o + \lambda_w) \\ &= \lambda^M_{o,j+1/2} / (\lambda^M_{o,j+1/2} + \lambda^M_{w,j+1/2}) \end{aligned} \tag{17}$$

to obtain

$$\sum_j \frac{\theta_j}{\Delta x}\lambda_o\left(\frac{\partial P_{oj}}{\partial S^L_{oK}}-\frac{\partial P_{oi}}{\partial S^L_{oK}}\right)$$

$$(18)\qquad =\sum_j \frac{\theta_j}{\Delta x}\frac{\lambda_o}{\lambda_t}\{(\lambda'_o\rho_o g+\lambda'_w\rho_w g)(D_j-D_i)-(\lambda'_o+\lambda'_w)(P_{oj}-P_{oi})$$

$$+\lambda'_w(P_{cj}-P_{ci})+\lambda_w(P'_{cj}-P'_{ci})\}.$$

In two or more dimensions, (14) cannot in general be multiplied under the summation sign by f_{oj} (17) since each term of the sum in (18) is not necessarily zero, nor is f_{oj} a constant independent of j. However, in two or more dimensions, the monotonicity condition (for backward switching) will not be applied unless

$$(19)\qquad |\max\{f_{o,j+1/2}\}-\min\{f_{o,j+1/2}\}|<\beta,\qquad \beta\ll 1,\quad j\in N_i,$$

which implies that

$$f_{o,j+1/2}\sim\text{constant},\qquad j\in N_i.$$

In practice, this is not unduly restrictive, since f_{oj} represents the discrete fractional flow. If f_{oj} is rapidly changing over the set of nearest neighbour cells, then this indicates the presence of a shock front, and hence an implicit treatment will almost certainly be required. On the other hand, ahead or behind the shock, $f_{o,j+1/2}$ will be sensibly constant over the set of nearest neighbour cells.

Note that in general

$$\frac{\partial S^{N+1}_{oi}}{\partial S^M_{oK}}\neq 0,\qquad K\notin N_i$$

since the pressures P^{N+1}_{oj}, P^{N+1}_{oi} will depend on saturations not in N_i. However, if the system is one-dimensional, or if (19) is satisfied, then S^{N+1}_{oi} does not explicitly depend on S^M_{oK}, $K\notin N_i$.

Consequently, assuming that multiplication of (14) by $f_{o,j+1/2}$ within the summation sign is valid, then (18) can be used to eliminate the pressure derivatives from (12). This gives a compact expression for the derivatives

$$(20)\qquad \frac{\partial S^{N+1}_{oi}}{\partial S^L_{oK}}=\delta_{i,K}\delta_{L,N}+\sum_j\theta_j\left\{\frac{\lambda'_o\lambda_w\psi_{oj}}{\lambda_t}-\frac{\lambda'_w\lambda_o\psi_{wj}}{\lambda_t}+(P'_{cj}-P'_{ci})/\Delta x\right\}.$$

Noting that

$$\lambda_l\psi_l=\lambda^M_{l,j+1/2}\psi_{lj}$$

$$=\lambda^M_{lj}\max(0,\psi_{lj})+\lambda^M_{li}\min(0,\psi_{lj}),$$

which follows from the definition of upstream weighting, we have

$$\psi_{lj}\lambda'_l=\frac{\partial\lambda_l}{\partial S^L_{oK}}\psi_{lj}$$

$$=(\lambda^M_{lj})'\max(0,\psi_{lj})\delta_{j,K}\delta_{L,M}+(\lambda^M_{li})'\min(0,\psi_{lj})\delta_{i,K}\delta_{L,M}.$$

Consequently, it is easy to evaluate (20) for the various derivatives of interest. For example, if cell i is an IMPES cell, then

$$(21)\qquad \left.\frac{\partial S^{N+1}_{oi}}{\partial S^M_{oj}}\right|_{j\in N_i,j\neq i}=\theta_j\{\lambda'_{oj}\lambda_w\max(\psi_{oj},0)/\lambda_t-\lambda'_{wj}\lambda_o\max(\psi_{wj},0)/\lambda_t+P'_{cj}/\Delta x\}.$$

Physically realizable relative permeabilities and capillary pressure curves obey the following conditions [1]:

$$\frac{d\lambda_o}{dS_o} \geqq 0, \quad \frac{d\lambda_w}{dS_o} \leqq 0, \quad \frac{dP_c}{dS_o} \geqq 0. \tag{22}$$

Consequently, from (21), we have

$$\frac{\partial S_{oi}^{N+1}}{\partial S_{oj}^{M}} \geqq 0, \quad j \in N_i, \quad j \neq i, \quad \forall \theta_j, \quad \lambda_{lj} = \lambda_{lj}^{N}, \lambda_{lj}^{N+1}, \quad \lambda_{li} = \lambda_{li}^{N}. \tag{23}$$

If cell i is an implicit cell, then

$$\lambda_{li} = \lambda_{li}^{N+1}.$$

Consequently,

$$\begin{aligned} \frac{\partial S_{oi}^{N+1}}{\partial S_{oi}^{N}} &= 1 + \frac{\partial S_{oi}^{N+1}}{\partial S_{oi}^{N}} \\ &\quad \cdot \sum_j \theta_j \{\lambda_o' \lambda_w \min(0, \psi_{oj})/\lambda_t - \lambda_w' \lambda_o \min(0, \psi_{wj})/\lambda_t - P_{ci}'/\Delta x\}. \end{aligned} \tag{24}$$

When we recall (22), all the terms multiplying θ_j in (24) are always negative or zero. Consequently,

$$\frac{\partial S_{oi}^{N+1}}{\partial S_{oi}^{N}} \geqq 0, \quad \forall \theta_j, \quad \lambda_{li} = \lambda_{li}^{N+1}. \tag{25}$$

If cell i is implicit, then differentiating implicitly gives

$$\begin{aligned} \left.\frac{\partial S_{oi}^{N+1}}{\partial S_{oj}^{M}}\right|_{j \in N_i, j \neq i} &= \sum_i \theta_j \{\lambda_{oj}' \lambda_w \max(\psi_{oj}, 0)/\lambda_t - \lambda_{wj}' \lambda_o \max(\psi_{wj}, 0)/\lambda_t + P_{cj}'/\Delta x\}; \\ &\quad + \frac{\partial S_{oi}^{N+1}}{\partial S_{oj}^{M}} \sum_j \theta_j \{\lambda_o' \lambda_w \min(0, \psi_{oj})/\lambda_t - \lambda_w' \lambda_o \min(0, \psi_{wj})/\lambda_t - P_{ci}'/\Delta x\} \end{aligned}$$

using (22), the above equation also implies that

$$\frac{\partial S_{oi}^{N+1}}{\partial S_{oj}^{M}} > 0 \quad j \in N_i, \qquad j \neq i,$$

$$\forall \theta_j; \quad \lambda_{li} = \lambda_{li}^{N+1}.$$

If cell i is an IMPES cell with

$$\lambda_{li} = \lambda_{li}^{N},$$

then

$$\frac{\partial S_{oi}^{N+1}}{\partial S_{oi}^{N}} = 1 + \sum_j \theta_j \{\lambda_o' \lambda_w \min(0, \psi_{oj})/\lambda_t - \lambda_w' \lambda_o \min(0, \psi_{wj})/\lambda_t - P_{ci}'/\Delta x\}. \tag{26}$$

Since all terms multiplying θ_j in (26) are negative, the monotonicity condition is

$$\left| \sum_j \theta_j \{\lambda_o' \lambda_w \min(0, \psi_{oj})/\lambda_t - \lambda_w' \lambda_o \min(0, \psi_{wj})/\lambda_t - P_{ci}'/\Delta x\} \right| < 1. \tag{27}$$

4. Application of monotonicity conditions. The results from § 3 can be summarized as follows:

$$\frac{\partial S_{oi}^{N+1}}{\partial S_{oj}^{M}} \geqq 0, \qquad j \neq i, \quad M = N, N+1 \quad \forall \theta_j. \tag{28}$$

If cell i is a fully implicit cell, then

$$\frac{\partial S_{oi}^{N+1}}{\partial S_{oi}^{N}} \geqq 0 \quad \forall \theta_j \tag{29}$$

and if cell i is an IMPES cell, then

$$\frac{\partial S_{oi}^{N+1}}{\partial S_{oi}^{N}} \geqq 0 \tag{30}$$

if (27) is satisfied. In more than one dimension, these conditions are valid if

$$f_{o,j+1/2} \sim \text{constant}, \qquad j \in N_i. \tag{31}$$

In the following, we will assume that the flow is either one-dimensional or (31) holds. Consequently, a fully implicit cell always gives rise to a monotone discretization, while an IMPES cell is also monotone for a sufficiently small timestep (from (27)). We will require that the selection of fully implicit-IMPES cells in the adaptive implicit method also ensures that the discretization is monotone. We will demonstrate an important property of a monotone discretization in the following.

Recalling that S_{oi}^{N+1} can, in principle, be written as follows:

$$S_{oi}^{N+1} = g_i(S_{oi}^{N}, S_{oj}^{N}, S_{oj}^{N+1});$$

then if

$$S_{\min} = \min\{S_{oj}^{M}, S_{oi}^{N}\}, \qquad j \in N_i, \quad M = N, N+1,$$
$$S_{\max} = \max\{S_{oj}^{M}, S_{oi}^{N}\}, \qquad j \in N_i, \quad M = N, N+1$$

and if g_i is a monotone function of its arguments in $[S_{\min}, S_{\max}]$, then

$$g_i(S_{\min}, \cdots) \leqq S_{oi}^{N+1} \leqq g_i(S_{\max}, \cdots).$$

In the case of an IMPES cell ($\lambda_{oi} = \lambda_{oi}^{N}$), the bounds can be determined by setting

$$S_{oi}^{N} = S_o^* = \text{constant}$$

and setting all functions of S_{oi}^{N} and S_{oj}^{M} to constants in (3) and (4). This gives

$$S_{oi}^{N+1} = S_o^* + \frac{\lambda_o \lambda_w}{\lambda_t}(\rho_w - \rho_o) g \sum_j \frac{\theta_j}{\Delta x}(D_j - D_i). \tag{32}$$

On a regular Cartesian grid, the sum

$$\sum_j \frac{\theta_j}{\Delta x}(D_j - D_i) \tag{33}$$

vanishes for all interior cells. Consequently, for these cells

$$S_{\min} \leqq S_{oi}^{N+1} \leqq S_{\max}. \tag{34}$$

In the case of a fully implicit cell ($\lambda_{oi} = \lambda_{oi}^{N+1}$), the bounds can be determined by setting

$$S_{oi}^{N} = S_o^* = \text{constant} \tag{35}$$

and setting all functions of S_{oi}^{N} and S_{oj}^{M} to constants in (3) and (4), as for the IMPES case. However, in this situation, we have an implicit equation for S_{oi}^{N+1} since λ_{oi}^{N+1}, λ_{wi}^{N+1}, P_{ci}^{N+1} are functions of S_{oi}^{N+1}. Assuming that (33) is identically zero, then one

solution for S_{oi}^{N+1} is clearly given by

$$S_{oi}^{N+1} = S_o^*$$

as for the IMPES case (this is verified by direct substitution). If the system is one-dimensional, or f_{oi} = constant, for all j, $j \in N_i$, then (3) can be written as

$$S_{oi}^{N+1} = S_{oi}^{N} + \gamma(S_{oi}^{N+1}, S_{oj}^{M}) \tag{36}$$

where we have imagined using (4) to eliminate the pressure. From (12) and (24) it follows that

$$\frac{\partial \gamma}{\partial S_{oi}^{N+1}} \leqq 0$$

so that the right-hand side of (36) is a decreasing function of S_{oi}^{N+1}, and the left-hand side of (36) is a straight-line function of S_{oi}^{N+1}, so that there is at most one solution for S_{oi}^{N+1}. Consequently, (34) is valid for interior implicit cells, as well as IMPES cells.

Note that $S_{\min}$ and $S_{\max}$ can contain values of S_{oj}^{N+1}, as can condition (27). These are, of course, unknown values at the beginning of a timestep. However, as will be discussed later, estimates of these quantities are easily obtained in an adaptive implicit method.

Equation (34) implies that the value of S_{oi}^{N+1} is bounded by the maximum and minimum value $\{S_{oi}^{N}, S_{oj}^{M}\}$. This property means that in the absence of source terms, no new local maxima or minima can appear during the time evolution of the system.

If the values of S_{oi}^{N} are bounded by

$$0 \leqq S_{oi}^{N}, S_{oj}^{N} \leqq 1$$

and $M = N$ in (34), then (34) obviously implies stability. If S_{oj}^{N+1} appears in (34), then cell j is an implicit cell. Assuming that

$$\lambda_o \equiv 0, \qquad S_o \leqq 0,$$

$$\lambda_w \equiv 0, \qquad S_w \leqq 0,$$

then if (3) and (4) are written for cell j, and if

$$S_{oK}^{N}, S_{wK}^{N} \in [0, 1] \quad \forall K$$

then it is impossible to solve (3) and (4) unless

$$S_{oj}^{N+1} \in [0, 1];$$

consequently, the extreme values of $S_{\min}$ and $S_{\max}$ appearing in (34) are bounded by [0, 1], and hence monotonicity implies stability.

Equation (33) does not vanish for edge cells on a regular Cartesian grid. For these boundary cells, it is not possible to obtain a "maximum" principle as given by (34). Of course, this is because physically, new local maxima and minima can appear in boundary cells. (Imagine a trivial example with two vertical cells, each with $S_{oi}^{N} = .5$, and oil less dense than water.) However, we would still like to ensure that the adaptive implicit method is always monotone, since the fully implicit method is always monotone, and the IMPES method is also monotone for a sufficiently small timestep.

As far as stability for edge cells is concerned, if (27) is satisfied for $S_{oi}^{N}, S_{oj}^{M} \in [0, 1]$, then since

$$\lambda_o \lambda_w \sum_j \frac{\theta_j}{\Delta x}(D_j - D_i) = 0$$

at $S_o = 0$ or $S_o = 1$, then $S_{oi}^{N+1} \in [0, 1]$. However, satisfying (27) (for IMPES cells) over the entire range [0, 1] will be quite restrictive in general for practical problems. Of course, applying such a test over [0, 1] would also be computationally expensive.

An alternative for edge cells would be to apply (27), and then check that the bound on S_{oi}^{N+1} given by (32) is in [0, 1]. However, this bound is quite crude, and will probably be too pessimistic. Unfortunately, it is not possible to obtain sharper bounds for edge cells without additional assumptions.

The results derived above will be used to aid in the development of switching criteria. Some heuristic reasoning is required to develop practical tests. This will be described in the next section.

5. A practical stability test. The whole idea of the adaptive implicit method is to produce a stable, physically reasonable solution while saving computational work. Consequently, a simple, inexpensive method of applying the tests derived in this article is required. As discussed elsewhere [5], an extremely cautious approach is necessary for practical problems. In other words, it is desirable to err on the side of too much implicitness, rather than too little.

The following method is used to switch cells from fully implicit to IMPES. After each Newton iteration, only cells that satisfy

$$|\max(f_{o,j+1/2}) - \min(f_{o,j+1/2})| < \text{tol}_1, \qquad j \in N_i, \tag{37}$$

$$\max |S_{oi}^N - S_{oj}^M| < \text{tol}_2, \qquad j \in N_i, \tag{38}$$

$$|S_{oi}^{N+1} - S_{oi}^N| < \text{tol}_3 \tag{39}$$

are considered as candidates for switching from fully implicit to IMPES. Equation (37) ensures that multiplication of (14) by $f_{o,j+1/2}$ under the summation sign is valid (this is required only for multidimensional problems). Equation (38) ensures that

$$|S_{\min} - S_{\max}| < \text{tol}_2$$

so that (27) does not have to be evaluated over a range of saturation values. After each Newton iteration, (27) is evaluated for fully implicit cells satisfying the above conditions. Note that the most recent estimates of $\lambda_{l,j+1/2}^{N+1}$ are available if required in (27). If (27) is satisfied, then the cell is switched to the IMPES state. For edge cells where the gravity sum (33) is nonzero, the switched cell is monitored for the next few iterations to ensure that (39) is satisfied. The philosophy behind this latter check can be stated as follows: in implicit edge cells, if (39) is satisfied, this indicates that gravity equilibration has occurred. Consequently, if large changes are observed after switching to IMPES, this is symptomatic of nonphysical overshoot and possible instability. This approach is preferred since (32) tends to be unduly pessimistic.

The method for switching from IMPES to fully implicit is a combination of the method used previously [5] and the monotonicity condition described in this work. If either

$$|S_{oi}^{N+1} - S_{oi}^N| > \text{tol}_3 \tag{40}$$

or (27) is violated, then the cell is switched to a fully implicit state. The tolerance in (40) is set to a fraction of the timestep selector norm [5], and will catch rapidly growing instabilities. Condition (27) indicates a physically unreasonable nonmonotone discretization, which can be avoided by using a fully implicit method. Note that (27) is rigorously correct in only one dimension, or if (37) is satisfied. However, in accordance with our cautious approach, (27) is always checked.

Consequently, the strategy used previously [5] is augmented with an additional check to trigger IMPES to fully implicit switches. As well, a new capability is added to switch from implicit to IMPES. The switch from implicit to IMPES occurs only when the discretization is monotone.

6. One-dimensional example. This first example demonstrates the possibility of obtaining nonphysical, but bounded, solutions that cannot be detected by the previous switching criteria [5]. Consider the one-dimensional slab shown in Fig. 1, with the physical properties given in Table 1. The slab has an initial water saturation of $S_w = .2$, which is the critical saturation. Water is injected halfway up the slab, so that initially some of the water flows downward, while oil flows upward, establishing a counter current flow regime. The domain of Fig. 1 was discretized with 20 cells in the vertical direction.

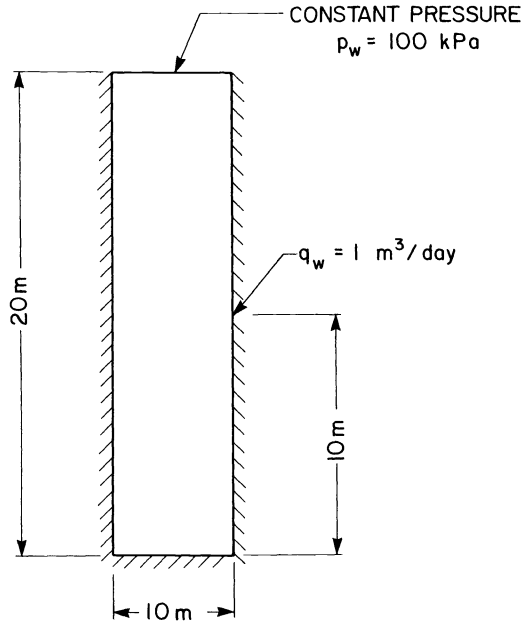


FIG. 1. *Region for one-dimensional example.*

TABLE 1
Data for the one-dimensional example.

Absolute permeability, K	10^{-12} m^2
Porosity, ϕ	.3
Initial saturations	$S_w = .2$, $S_o = .8$
Discretization	$nx = 1$, $\Delta x = 1.0$ m,
	$ny = 20$, $\Delta y = 1.0$ m
Densities	$\rho_o = 864$ kg/m^3,
	$\rho_w = 1000$ kg/m^3
Viscosities	$\mu_o = 1$ cp, $\mu_w = 1$ cp
Initial pressure	100 kpa

TABLE 1(a)
Relative permeability and capillary pressure data for the one-dimensional example.

S_w	K_{rw}	K_{ro}	P_{cow} (kpa)
.2	0.0	.68	103.0
.3	.04	.55	27.6
.4	.10	.43	10.3
.5	.18	.31	7.58
.6	.30	.20	7.44
.7	.44	.12	7.30
.8	.60	.05	7.17
.9	.80	0.0	7.03
1.0	1.0	0.0	6.89

Two runs were carried out. One run used the switching criterion described in a previous article [5]. In this case, a timestep selector norm of 20 percent (saturation change) was specified, while the implicit trigger (tol_3) was set at 5 percent. A norm of 20 percent would be typical for a problem of this type.

The saturation profile after 10 days of injection is shown in Fig. 2(a). The local maximum at a depth of 10 m is due to the injection of fluid at this point. However, the local maxima at depths of 4 m and 16 m are clearly nonphysical. These cells are just on the boundary of the fully implicit region. If the run is continued, these local maxima (which are at the 4 percent level) persist. The existence of these local maxima is easily explained. The large derivatives of the capillary pressure curve near $S_w = .2$ (see Table 1(a)) results in a nonmonotone IMPES discretization (from (27)). After a small local maximum evolves, the strongly nonlinear capillary pressure curve has a much smaller derivative, and hence the discretization becomes monotone, and the

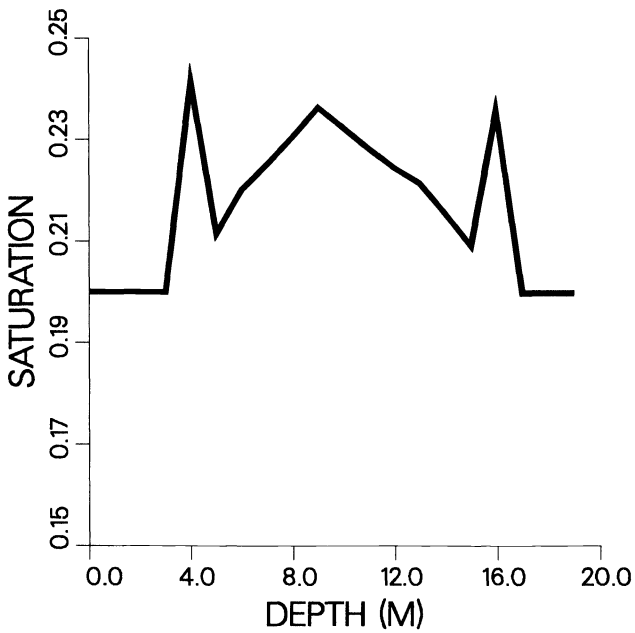


FIG. 2(a). *Saturation profile at* 10 *days, old switching criterion* [5].

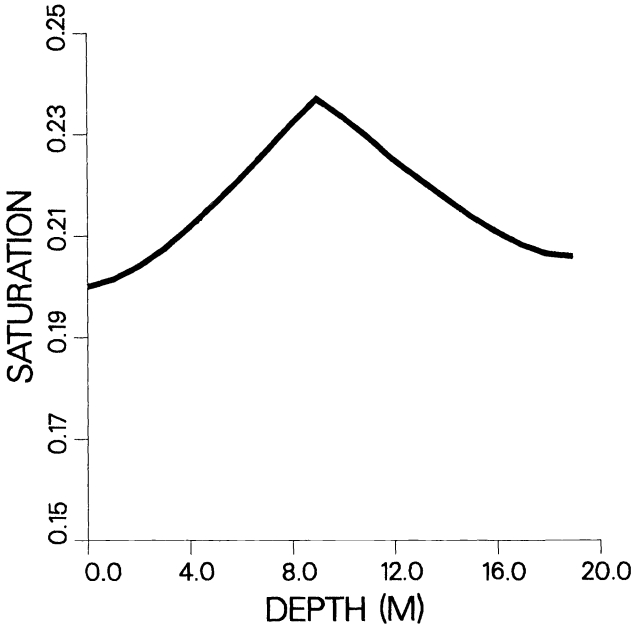


FIG. 2(b). *Saturation profile at* 10 *days, new switching criterion.*

solution remains bounded. A similar effect can also be observed in problems where the fractional flow curve has a large derivative over a small saturation range. This occurs in any problem where the ratio of the viscosities of the two fluids is large. Consequently, there is no disastrous instability in the stepwise sense; the variables always remain bounded due to the nonlinear nature of the problem. However, the nonphysical saturation maxima remain. Eventually, if a large enough timestep is taken, an implicit switch is triggered for the cells in question, and a better-behaved solution is obtained.

Figure 2(b) shows the simulation results for a run using the criteria suggested in this paper. Note the disappearance of the nonphysical local maxima. In this case, condition (27) was violated by an order of magnitude for the nonmonotone cells, and hence an implicit method was used. The results for the adaptive implicit run were virtually identical with the results for the fully implicit solution (agreement to three decimal places). The values used for the tolerances in (38)-(39) were $\text{tol}_1 = \text{tol}_2 = .20$.

The relative permeabilities and capillary pressure curves used were typical of those found in nonaqueous phase groundwater contamination [15], [16].

TABLE 2
Data for the one-quarter five-spot problem.

Absolute permeability, K	10^{-13} m^2
Porosity, ϕ	.3
Initial saturations	$S_w = .15$, $S_o = .85$
Discretization	$nx = 10$, $\Delta x = 25$ m,
	$ny = 10$, $\Delta y = 25$ m,
	$nz = 1$, $\Delta z = 10$ m
Depth	$D = 1000 - (i-1)2.5 + (j-1)2.5$ m, cell (1, 1) is injector
Densities	$\rho_o = 864$ kg/m^3
	$\rho_w = 1000$ kg/m^3
Viscosities	$\mu_o = 1$ cp, $\mu_w = 1$ cp

7. Waterflood example. A one-quarter five-spot [1], two-phase, incompressible water-injection problem is shown in Fig. 3. The relative permeabilities and capillary pressure data are taken from reference [8]. Water is injected at a constant rate in the top left corner, and a constant pressure is specified in the lower right corner. All other boundaries are no-flow. The initial water saturation is the critical saturation. Both 10×10 and 20×20 grids were used. The data for the 10×10 grid is summarized in Table 2. All cell dimensions were halved for the 20×20 grid.

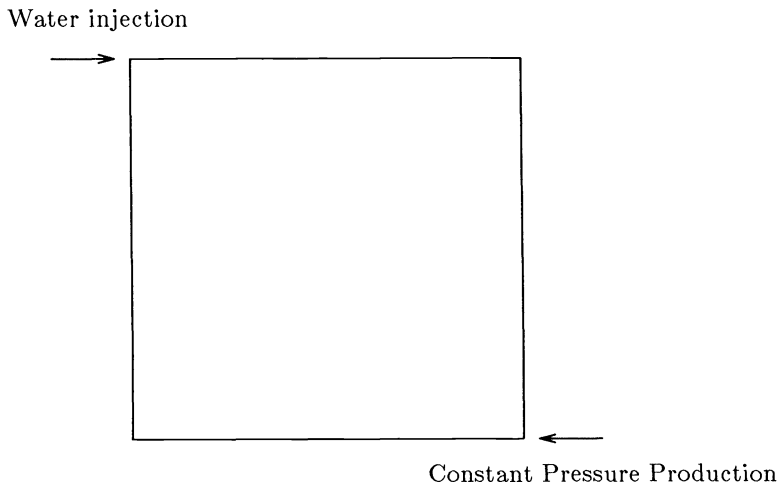


FIG. 3. *One-quarter five-spot water-injection problem.*

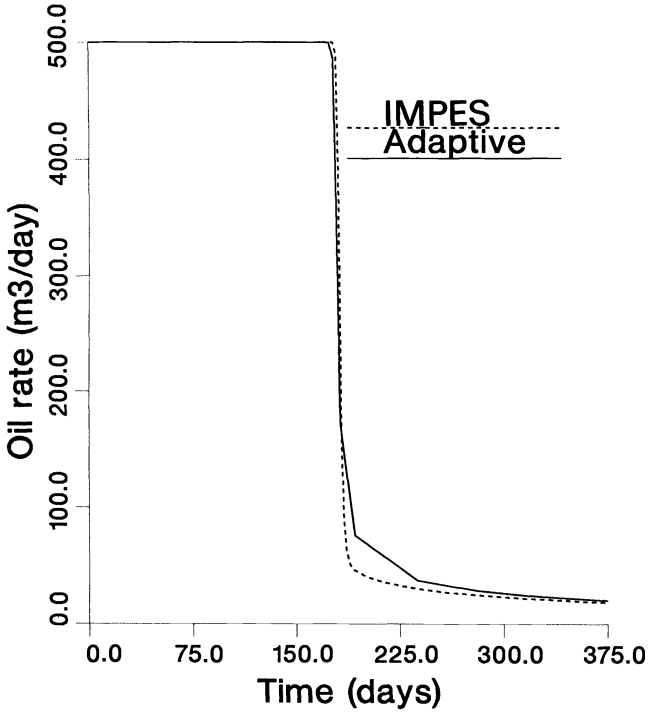


FIG. 4. *Comparison of computed oil rate for* IMPES *and adaptive implicit solutions of the one-quarter five-spot problem* (10×10 *grid*).

Figure 4 shows the oil rate versus time for an IMPES solution and an adaptive implicit solution on the 10×10 grid. The maximum stable timestep for the IMPES method was approximately 1.5 days, compared to the average timestep of about 15 days for the adaptive implicit method. The two solutions agree quite well, except near the knee of the oil rate curve. This discrepancy is simply due to the much larger timesteps taken by the adaptive implicit method after water breakthrough, and hence the much larger time truncation error. This is probably insignificant in practical problems. The IMPES solution required about three times the execution time of the adaptive implicit method.

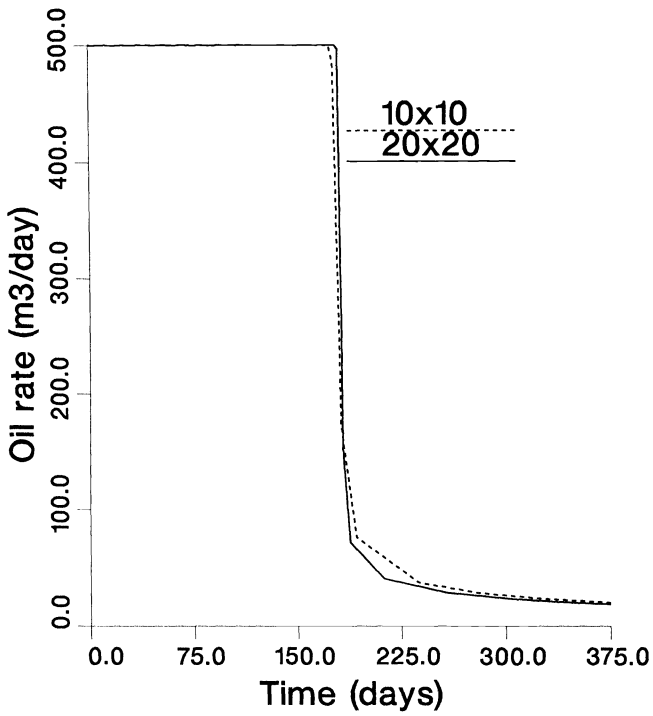


FIG. 5. *Comparison of adaptive implicit solutions for varying grid sizes for the one-quarter five-spot problem.*

Figure 5 compares the oil rates for the adaptive implicit solution on 10×10 and 20×20 grids. The results are very close, indicating that the effect of grid size, at least for oil-production rates, is not too severe. Again, there is some discrepancy in the knee of the curve, but this is probably due to the different timestep sizes at this point (the oil rates at each timestep are connected by a straight line).

TABLE 2(a)
Injection/production data for the one-quarter five-spot.

Injection	500 m^3/day
Production	constant pressure
	$P_w = 1000$ kpa

Relative permeability and capillary pressure from reference [8].

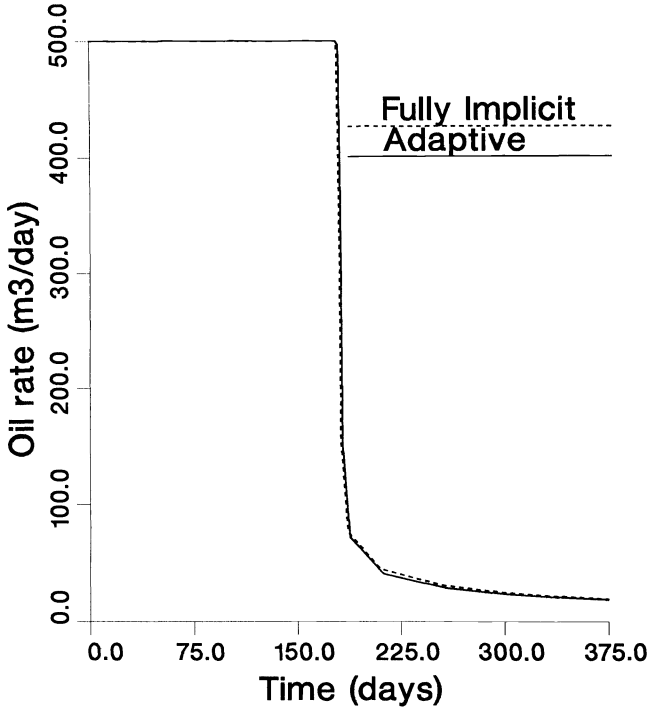


FIG. 6. *Comparison of fully implicit and adaptive implicit solutions for the one-quarter five-spot* (20×20 *grid*).

Figure 6 shows a comparison of results for the adaptive implicit and fully implicit solutions on a 20×20 grid. The results are virtually identical.

Figures 7(a)-(c) and 8(a)-(c) show the saturation contours at various times for both 10×10 and 20×20 grids (adaptive implicit method used). The contours at 375 days are well past water breakthrough, so the front has disappeared. Note that the fronts are quite rounded. This is due to the fact that the reservoir dips slightly in the direction from producer to injector (see Table 2(a)).

It is clear that the fronts are sharper and less smeared on the finer grids, as we would expect. However, the shock smearing did not greatly affect the oil rate (see Fig. 5), which is the important parameter from an engineering point of view.

Figures 9(a)-(c) show the location of the implicit cells for the adaptive implicit method, with the 20×20 grid. Note that the location of the implicit cells track the shock front quite closely.

Table 3 gives the run statistics for a fully implicit, adaptive implicit (forward switching only), and an adaptive implicit simulation with forward and back switching.

TABLE 3
Run statistics for the one-quarter five-spot water flood, 20×20 *grid.*

		Adaptive implicit	
	Fully implicit	Forward switching only	Forward and back switching
Average degree of implicitness	1.0	.47	.15
Normalized CPU time	1.0	.77	.61

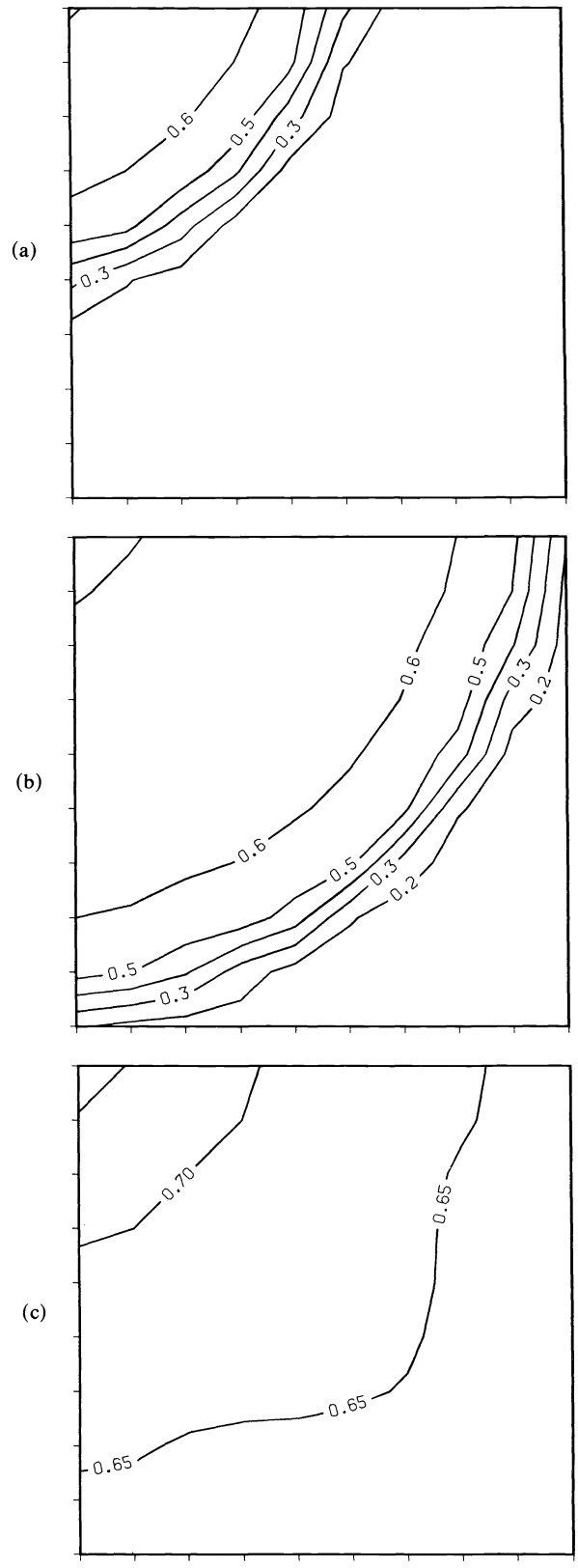


FIG. 7. *Water-saturation contours, one-quarter five-spot,* 10×10 *grid.* (a) 37.5 *days*; (b) 112.5 *days*; (c) 375 *days.*

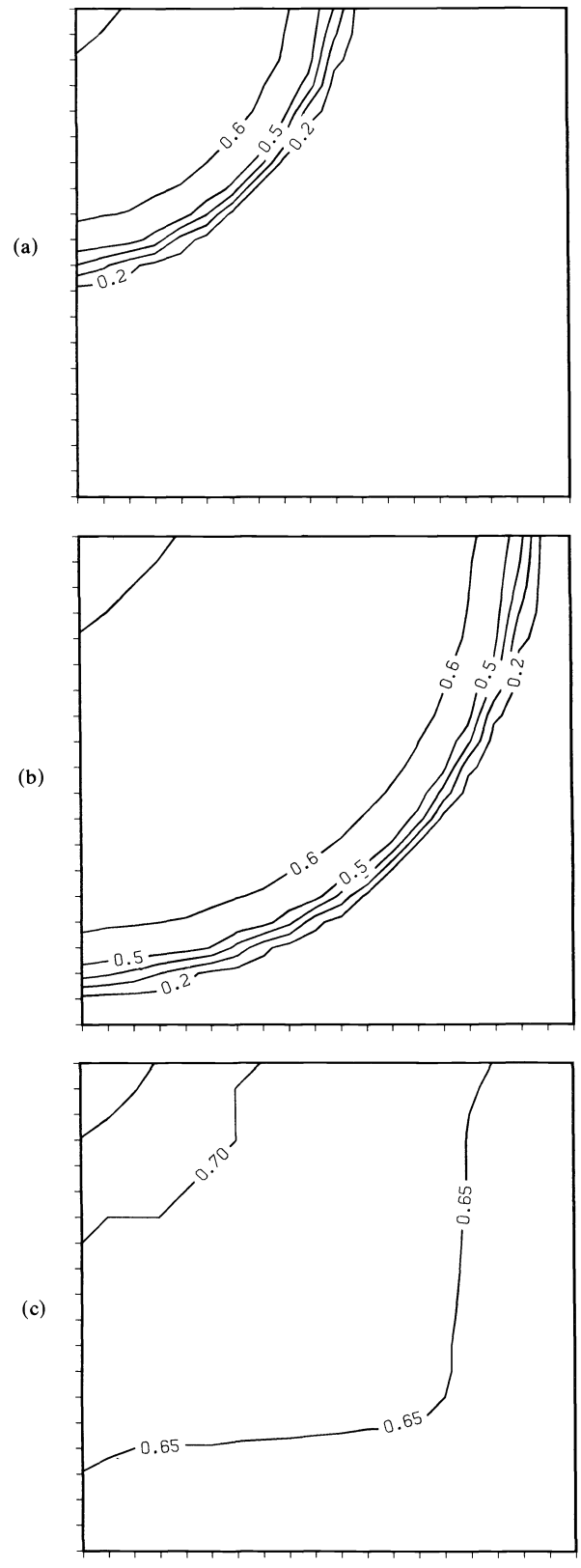


FIG. 8. *Water-saturation contours, one-quarter five-spot,* 20×20 *grid.* (a) 37.5 *days*; (b) 112.5 *days*; (c) 375 *days.*

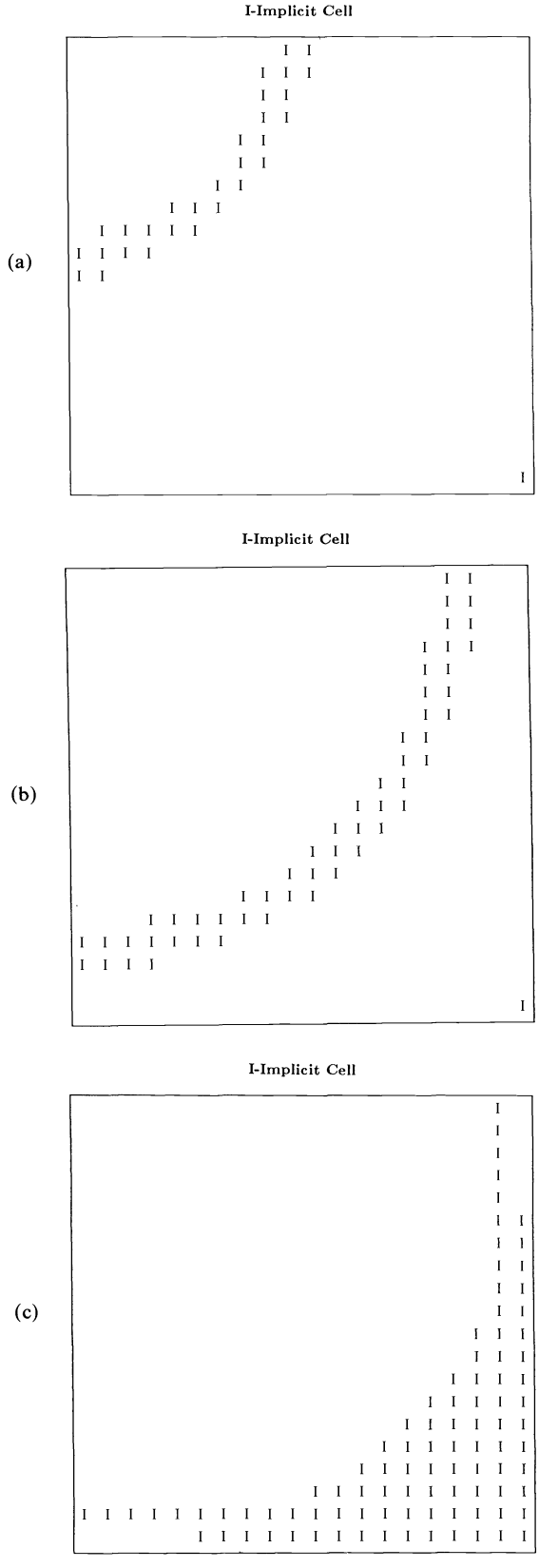


FIG. 9. *Implicit map, one-quarter five-spot,* 20×20 *grid.* (a) 37.5 *days*; (b) 112.5 *days*; (c) 375 *days.*

Both adaptive implicit runs used the criteria of this work. The values of the tolerances were the same as used in the one-dimensional example. The average degree of implicitness was 15 percent for the run with forward and back switching, which is significantly smaller than the 47 percent obtained with forward switching only. A reasonable reduction in CPU time is also obtained. It is expected that more careful coding of the Jacobian construction for IMPES cells could reduce the adaptive implicit times even further.

The backward switching was particularly useful when water broke through to the producer. Bacause of the large nonlinearity, small timesteps were required. Consequently, almost all cells, except those near the producer, reverted to the IMPES state. This makes the small timesteps during water breakthrough comparatively cheap. After breakthrough, the timesteps quickly increased, and many cells became fully implicit again.

8. Coning example. Figure 10 shows a two-dimensional coning problem in cylindrical coordinates. The discretized equations for cylindrical coordinates have the same form as (3)-(4), except that the area factor θ_j and node spacing Δx_i are redefined [1]. This problem was solved on both an 11×11 and a 22×22 grid. The data for the 11×11 grid are given in Table 4, with all cell sizes halved for the 22×22 grid. Note the very large changes in radial cell sizes: Δr ranges from 1 m—1024 m. The small radial cells are required to obtain adequate resolution near the wellbore. This type of grid would be typical of those used in practical problems [17]. Since the oil viscosity is five times greater than the water viscosity and the vertical permeability is equal to the horizontal permeability, coning effects will be very severe.

TABLE 4
Data for coning problem.

Absolute permeability	$(K_r = K_z)$, 10^{-13} m^2
Porosity, ϕ	.3
Discretization	$nr = 11$, $nz = 11$,
	Δr_i, $i = 1 - 11$
	$= 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024$ m,
	Δz_i, $i \neq 3, 4, 11 = 5$ m,
	Δz_i, $i = 3, 4 = 2.5$ m,
	Δz_i, $i = 11 = 10$ m
Densities	$\rho_o = 710$ kg/m^3
	$\rho_w = 1000$ kg/m^3
Viscosities	$\mu_o = 5$ cp
	$\mu_w = 1$ cp
Compressibilities	oil: $c_o = 3.10^{-6}$ kpa^{-1}
	water: $c_w = 4 \times 10^{-7}$ kpa^{-1}
	rock: $c_r = 5 \times 10^{-6}$ kpa^{-1}

In order to model a more realistic situation, the compressible two-phase equations are actually used [14]. Since the compressibilities of oil and water are very small (see Table 4), the analysis of § 2 can be assumed to hold with negligible error. In fact, the finite compressibility coupled with the very large outermost radial cell effectively forces a constant pressure on the outer boundary. Compressibility effects are very small for the inner cells. The sink term, corresponding to a well with radius of .08 m completed over a 5 m interval, is handled in the usual way [1], [14].

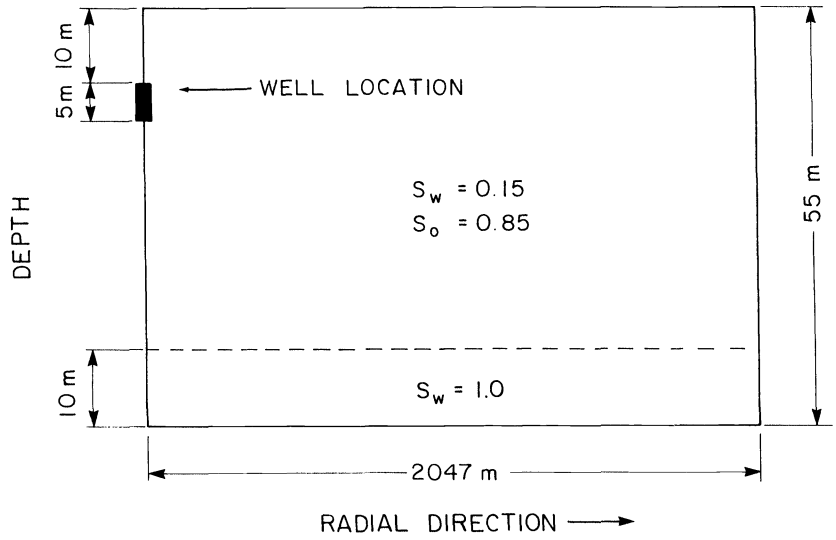


FIG. 10. *Domain for coning problem.*

Figure 11 shows the oil rate versus time for the 11×11 and 22×22 grids. The adaptive implicit method was used for the results shown. A fully implicit solution gives results very close to the adaptive implicit solution. The analysis of § 2 (which was also verified by a few computational experiments) indicated that the maximum stable

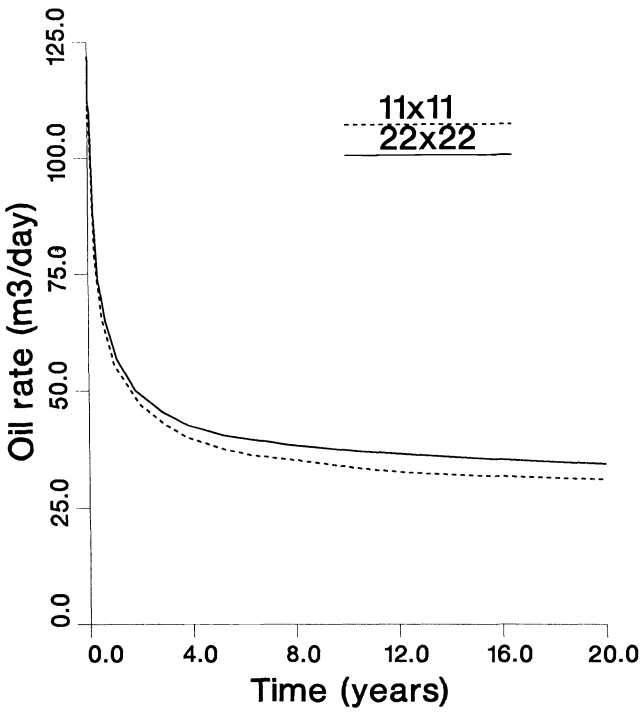


FIG. 11. *Comparison of computed oil rates for the coning problem, adaptive implicit method used.*

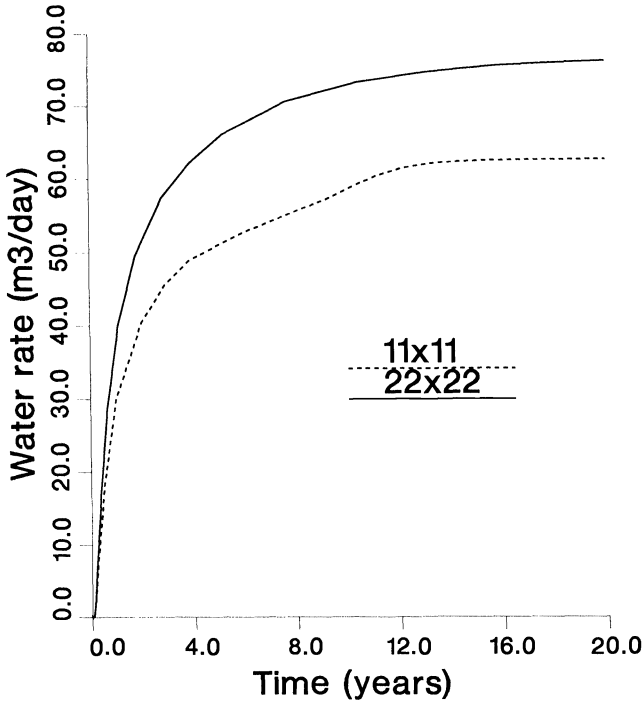


FIG. 12. *Comparison of computed water rates for the coning problem, adaptive implicit method used.*

IMPES timestep was approximately .01 days. This means that an IMPES method would require approximately 7×10^5 timesteps to complete the simulated 20 years. This is clearly impractical. Although the diffusive effects of upstream weighting do not affect the oil rate significantly, Fig. 12 shows the effect of grid size on the water rate. Since this is a compressible problem, unlike the quarter five-spot problem, the total fluid rate is not constant. In this case, we can see that the effect of grid size is quite pronounced. For example, at 20 years, the water rate on the 11×11 grid is approximately 60 m^3/day, while the rate for the 22×22 grid is approximately 75 m^3/day. However, since the oil rate is the most important parameter in practical situations, this type of error in the water rate is probably not too significant. This error is due entirely to the

TABLE 4(a)
Additional data for the coning problem.

Well data: $P_w = 500$ kpa, Well radius = .08 m, Completed interval = 5 m							
Relative permeability and capillary pressure							
S_w	K_{rw}	K_{ro}	P_{cow}	S_w	K_{rw}	K_{ro}	P_{cow}
.15	0.0	1.0	7.0	.60	.28	.01	3.2
.20	.003	.85	6.6	.70	.41	.001	2.3
.30	.03	.50	5.7	.80	.58	0.0	1.7
.40	.08	.30	4.8	.90	.65	0.0	.85
.50	.17	.10	4.1	1.0	1.0	0.0	0.0
Initial saturations as shown in Fig. 10.							
Initial pressure $P_o = 9000$ kpa.							

use of fairly coarse grids coupled with upstream weighting. This does not have anything to do, per se, with the use of adaptive implicit timestepping.

Figures 13 and 14 show the water-saturation contours for both the 11×11 and 22×22 grids at 1400 days. A straightforward analysis of the fractional flow curves generated by the relative permeability curves in Table 4(a) reveals that, for one-dimensional flow with no gravity or capillary pressure, the shock saturation should be about $S_w = .5$. Assuming that this simple analysis is indicative of the more complex

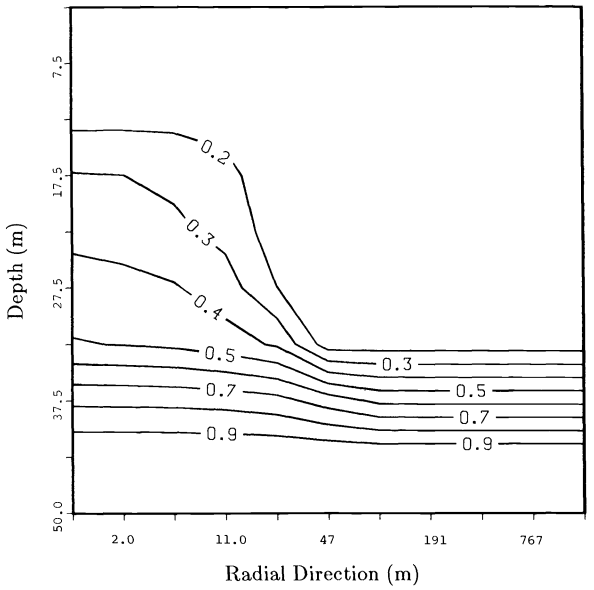


FIG. 13. *Water-saturation contours for the coning problem, time* = 1400 *days,* 11×11 *grid.*

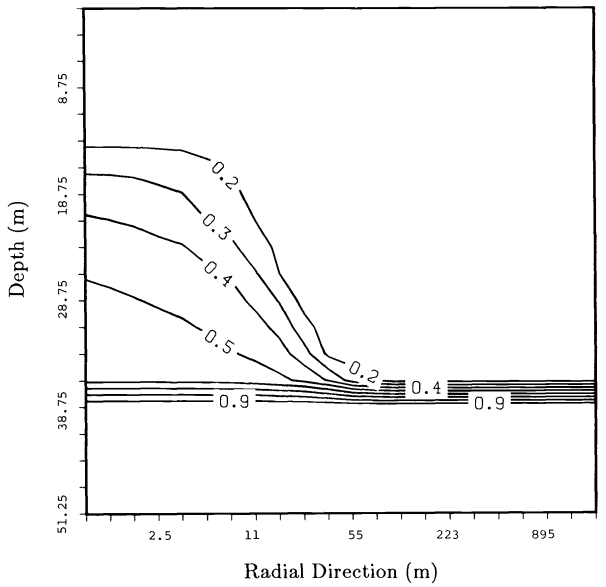


FIG. 14. *Water-saturation contours for the coning problem, time* = 1400 *days,* 22×22 *grid.*

phenomena modelled here, the location of the shock can be placed approximately along the $S_w = .5$ contour. The most dramatic change between Fig. 13 and Fig. 14 is that the $S_w = .5$ contour has become very much more cone shaped for the finer grid. This effect is even more pronounced than shown in the figures, because of the distortion in the radial direction. For visibility purposes, the contours are distorted in the radial direction (the cells are plotted as equally spaced). The location of the implicit cells at 1400 days for the 22×22 grid is given in Figure 15. At this point in the simulation, approximately one half the cells are implicit. However, only a few large timesteps (approximately three years) are required to complete the run after this point.

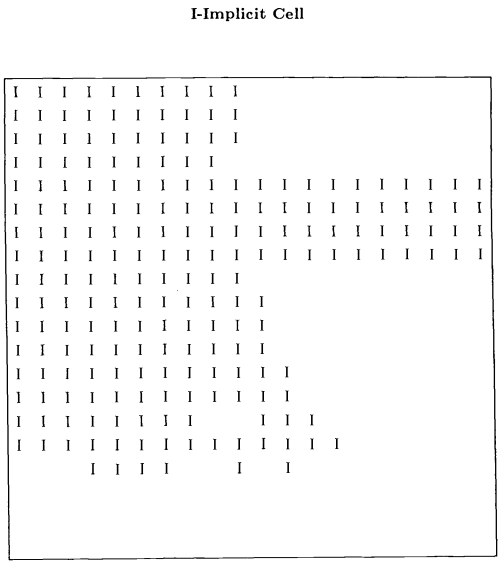


FIG 15. *Implicit map for the coning problem, time* = 1400 *days,* 22×22 *grid.*

Table 5 gives the run statistics for both adaptive implicit and fully implicit solutions of the coning problem. A 35 percent reduction in CPU time was obtained for the adaptive implicit method compared to the fully implicit method, while giving virtually identical results.

TABLE 5
Run statistics for the coning problem, 22×22 *grid.*

	Fully implicit	Adaptive implicit
Average degree of implicitness	1.0	.23
Normalized CPU time	1.0	.65

9. Conclusions. Previous switching criteria for adaptive implicit methods were based on attempting to detect stepwise instability, and ensuring that the solution remained bounded. While this approach can detect extreme instabilities, it is unable to detect a nonmonotone discretization, which can give rise to a bounded but nonphysical solution with new local maxima.

The monotonicity conditions for two-phase, incompressible flow with gravity and capillary pressure were developed in this work. This criteria is rigorously valid for one-dimensional flow, and for multidimensional problems, provided the discrete fractional flow is constant over the set of neighbour cells. For cells that are not on the domain boundaries, the monotonicity condition ensures that nonphysical local maxima and minima cannot occur.

Some approximations were introduced to yield a practical monotonicity test. The test was demonstrated for both one- and two-dimensional example problems. The one-dimensional example demonstrated the advantages of detecting nonmonotone, nonphysical behaviour, while the two-dimensional examples showed the benefits of being able to switch cells from IMPES to fully implicit, and vice versa.

In many situations, an implicit method is required only near sharp fronts. This makes the adaptive implicit method, with the criteria developed here, extremely attractive for two-phase flow problems. Note that phase upstream weighting is used in this study. This gives a monotone discretization in countercurrent (oil phase flowing in opposite direction to water phase) flow situations. Many groundwater-contamination problems can be reasonably approximated by two-phase flow [15], so the methods developed here may be used unchanged for this application.

Since fully implicit methods are expensive in terms of cost per timestep, large timesteps must be taken to produce a method that is more efficient than an IMPES technique. The adaptive implicit solutions obtained in this study used the same timestepping parameters as the fully implicit runs, and hence used virtually the same timestep sequence. However, the flexibility of an adaptive implicit method that allows switching of cells from IMPES to fully implicit and back to IMPES, increases the range of possibilities for producing a simulation with the smallest computational cost. For example, it may be possible that a sequence of timesteps that is intermediate in size between IMPES and fully implicit would produce the lowest computational cost. In this case, just a few cells would have to be implicit, and the smaller timesteps would require fewer Newton iterations per timestep.

The basic method used in the spatial discretization involves single-point phase upstream weighting. This permits use of adaptive implicit criteria based on preserving local monotonicity. Of course, use of upstream weighting causes smearing of fronts. However, at least for the examples presented in this study, the front smearing does not affect the oil rates to a large degree. From a practical point of view, the oil rates are the most important solution parameters. However, for problems where grid-orientation effects are severe, another form of spatial discretization is highly preferable [12]. As long as the basic underlying method enforces local monotonicity, the concept of adaptive implicit timestepping based on monotonicity criteria should be applicable.

The analysis of this study has been restricted to two-phase, incompressible problems. Many practical reservoir-simulation problems involve three-phase, compressible flow with mass transfer between phases. It can be expected that the adaptive implicit criteria will be considerably more complex in these situations.

REFERENCES

[1] K. AZIZ AND A. SETTARI, *Petroleum Reservoir Simulation*, Applied Science, London.

[2] L. M. ABRIOLA AND G. F. PINDER, *A multi-phase approach to modelling of ground-water contamination, 1. Equation development*, Water Resour. Res., 21 (1985), pp. 11–18.

[3] G. W. THOMAS AND D. H. THURNAU, *Reservoir simulation using an adaptive implicit method*, Soc. Pet. Engrg. J., 23 (1983), pp. 750–768.

[4] W. I. BERTIGER AND F. J. KELSEY, *Inexact Newton methods*, in Proc. 8th SPE Symposium on Reservoir Simulation, Dallas, TX, 1985.

[5] P. A. FORSYTH AND P. H. SAMMON, *Practical considerations for adaptive implicit methods in reservoir simulation*, J. Comput. Phys., 62 (1986), pp. 265–281.

[6] T. B. TAN, *Implementation of an improved adaptive implicit method in a thermal compositional simulator*, in Proc. 9th SPE Symposium on Reservoir Simulation, San Antonio, TX, 1987.

[7] J. E. KILLOUGH AND C. A. KOSSACK, *Fifth comparative solution project: evaluation of miscible flood simulators*, in Proc. 9th SPE Symposium on Reservoir Simulation, San Antonio, TX, 1987.

[8] L. S. K. FUNG, D. COLLINS, AND L. NGHIEM, *An adaptive implicit switching criterion based on numerical stability analysis*, in Proc. 9th SPE Symposium on Reservoir Simulation, San Antonio, TX, 1987.

[9] P. QUANDALLE AND J. C. SABATHIER, *Typical features of a new multipurpose reservoir simulator*, in Proc. 9th SPE Symposium on Reservoir Simulation, San Antonio, TX, 1987.

[10] P. A. FORSYTH AND P. H. SAMMON, *Local mesh refinement and modelling of faults and pinchouts*, SPE J. Form. Eval., 1 (1986), pp. 275–286.

[11] J. GLIMM, B. LINGUIST, O. MCBRYAN, B. PHOHR, AND S. YANIV, *Front tracking in reservoir simulation*, paper SPE 12238, 7th SPE Symposium on Reservoir Simulation, San Francisco, CA, 1983.

[12] J. BELL AND G. SHUBIN, *Higher order Godunov methods for reducing numerical dispersion in reservoir simulation*, paper SPE 13514, 8th SPE Symposium on Reservoir Simulation, Dallas, TX, 1985.

[13] P. H. SAMMON, *An analysis of upstream differencing*, Soc. Pet. Engrg. J. Res. Engrg., 3 (1988), pp. 1053–1056.

[14] D. W. PEACEMAN, *Fundamentals of Reservoir Simulation*, Elsevier, Amsterdam, 1977.

[15] C. R. FAUST, *Transport of immiscible fluids within and below the unsaturated zone: a numerical model*, Water Resour. Res., 21 (1985), pp. 587–596.

[16] M. OSBORNE AND J. SYKES, *Numerical modelling of immiscible organic transport at the Hyde Park landfill*, Water Resour. Res., 22 (1986), pp. 25–33.

[17] J. CHAPPELEAR AND J. NOLAN, *Second comparative solution study: a three phase coning problem*, paper SPE 10484, 6th SPE Symposium on Reservoir Simulation, New Orleans, 1982.

SIAM J. SCI STAT. COMPUT.
Vol. 10, No. 2, pp. 253-273, March 1989
© 1989 Society for Industrial and Applied Mathematics
003

A TOTAL VARIATION DIMINISHING-PROJECTION METHOD FOR SOLVING IMPLICIT NUMERICAL SCHEMES FOR SCALAR CONSERVATION LAWS: A NUMERICAL STUDY OF A SIMPLE CASE*

A. BOURGEAT† AND BERNARDO COCKBURN‡

Abstract. The stability of Newton's method applied to implicit numerical schemes for scalar conservation laws requires, in the general case, an upper bound on the size of the timesteps. A simple locally defined projection, a total variation diminishing (TVD)-projection, is introduced, and it is used to obtain always-stable extensions of Newton's method. A numerical study of this method applied to Godunov's implicit scheme is presented.

Key words. scalar conservation laws, implicit schemes, Newton's method

AMS(MOS) subject classifications. 65M10, 65N20, 35L65

1. Introduction.

In this paper we consider the problem of actually solving finite difference implicit schemes for the following boundary value problem:

$$\partial_t u + \partial_x f(u) = 0, \qquad \text{in } (0,T)\times(0,1), \tag{1.1a}$$

$$u(x=0) = b_0, \qquad \text{on } (0,T), \tag{1.1b}$$

$$u(x=1) = b_1, \qquad \text{on } (0,T), \tag{1.1c}$$

$$u(t=0) = u_0, \qquad \text{on } (0,1), \tag{1.1d}$$

where $f \in C^1$, b_0 and $b_1 \in BV(0,T)$, and $u_0 \in BV(0,1)$. The space $BV(\Omega)$ is the space of functions of finite total variation in Ω. For a suitable interpretation of this boundary value problem see the work of Bardos, LeRoux, Nédélec [1]. It is very well known that the main difficulty concerning the resolution of an implicit scheme is solving the nonlinear equation defining the approximate solution at each timestep. One of the most popular methods used for doing this is Newton's method. Its quadratic convergence makes it very attractive; however, its lack of global stability makes it useful only if a suitable initial guess can be obtained. A widely used practice is to take the approximate solution at time t^n, u_h^n as the initial guess for the calculation of u_h^{n+1}:

(1) Set $u_h^{(1)} = u_h^n$;
(2) For $m = 1, \cdots, M^n$ compute $u_h^{(m+1)} = \Phi_h^n(u_h^{(m)})$;
(3) Set $u_h^{n+1} = u_h^{M^n+1}$.

*Received by the editors October 19, 1987; accepted for publication (in revised form) May 31, 1988.

†Université de Saint Etienne, Faculté des Sciences et Technologie, 23, rue Dr. P. Michelon, 42023 St. Etienne Cedex 2, France. This work was done in part while this author was visiting the Applied Mathematics Division, Argonne National Laboratory, Argonne, Illinois, from July to October 1986, and during a visit to the Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, Minnesota, in December 1986.

‡Institute for Mathematics and its Applications, University of Minnesota, 514 Vincent Hall, Minneapolis, Minnesota 55455. Present address, School of Mathematics, University of Minnesota, 127 Vincent Hall, Minneapolis, Minnesota 55455.

But in this case the sufficient conditions for the convergence of this method, given by the now classical theorem of Kantorovich (see the reference in Rall [8]), impose an upper bound on the size of the timestep $\Delta t^n = t^{n+1} - t^n$. In this way, the main advantage of using an implicit scheme is lost.

In practice, the number of inner iterations M^n is usually taken to be equal to 1. This is justified when the timestep is small enough; see, for example, the analysis of these Newtonian iterations given by Douglas [4] for the case of the long wave regularization implicit finite difference scheme for (1.1) developed by Douglas, Kendall, and Wheeler [5]. This choice is also made because doing M^n inner iterations with the timestep $\Delta t = \Delta t^n$ costs as much as doing M^n time iterations of a single inner iteration with $\Delta t = \Delta t^n$; this strategy has been adopted, for example, for the implicit–explicit version of the Piecewise Parabolic Method developed by Fryxell, Woodward, Colella, and Winkler [6]. Other authors prefer to do a few inner iterations to improve the accuracy of the approximate solution; see, for example, the implicit version of the Osher upwind scheme developed by Rai and Chakravarthy [10]. In any case, if the timestep is too big the stability of the method is lost.

For instance, consider a problem of the form (1.1) with

$$
\begin{aligned}
&f(u) = 20 \cdot u^3/(20 \cdot u^3 + (1-u)^3), \qquad u \in [0,1],\\
&b_0(t) = 1, \qquad t \in (0, T = 0.3),\\
&b_1(t) = 0, \qquad t \in (0, T = 0.3),\\
&u_0(x) = 0, \qquad x \in (0,1),
\end{aligned}
$$

and consider the approximate solution obtained by discretizing (1.1) with the implicit Godunov scheme. Then, if we solve it with no limitation on M^n, it becomes unstable as the timestep becomes greater or equal to $0.690\Delta x$; (see Figs. 0.1 and 0.2). Note that the cfl-condition for the explicit Godunov scheme in this case is $\Delta t \leq 0.256\Delta x$!

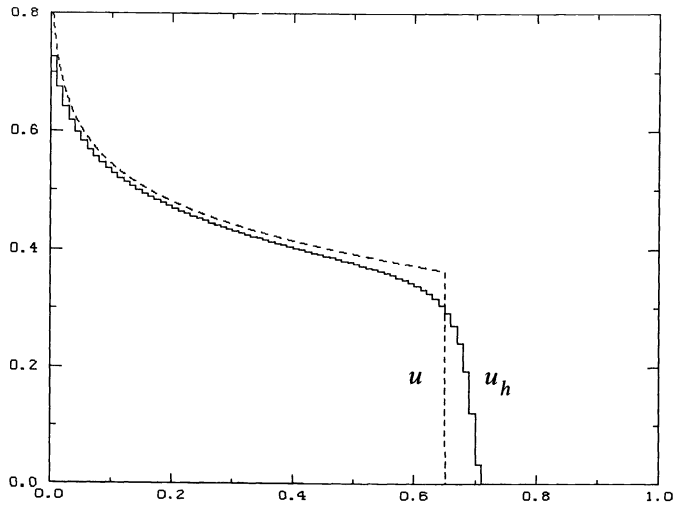


FIG. 0.1. *The exact solution, $x \mapsto u(T = 0.3, x)$, and the approximate solution, u_h, determined by the implicit Godunov scheme with $\Delta x = 0.01$, $\Delta t = 0.00682$.*

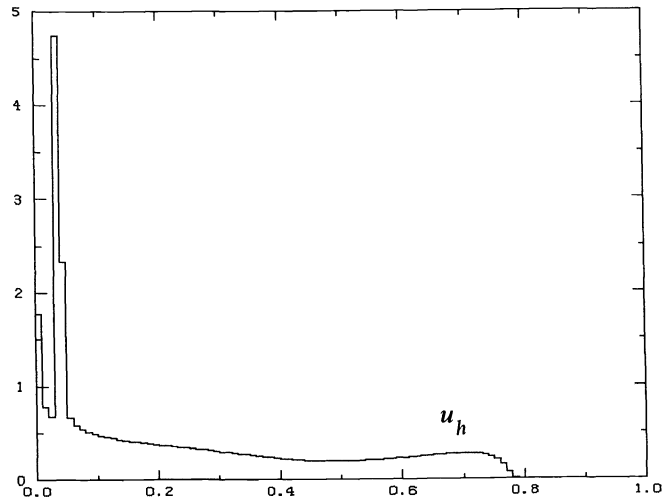


FIG. 0.2. *Example of instability of the implicit Godunov scheme. Unrealistic values obtained with* $\Delta x = 0.01$, $\Delta t = 0.00698$.

In this paper we introduce a new and very simple method for overcoming this difficulty. It consists in replacing the Newtonian iterations by the following modification:

(1) Set $u_h^{(1)} = u_h^n$;
(2) For $m = 1, \cdots, M^n$ compute $u_h^{(m+1)} = \mathbf{P}(u_h^{(m)})\Phi_h^n(u_h^{(m)})$;
(3) Set $u_h^{n+1} = u_h^{M^n+1}$;

where Φ_h^n is the same as before, and the operator $\mathbf{P}(v_h)$ is a locally defined projection verifying the following stability properties:

$$\inf_x \mathbf{P}(v_h)w_h \geq \inf_x v_h,$$
$$\sup_x \mathbf{P}(v_h)w_h \leq \sup_x v_h,$$
$$\|\mathbf{P}(v_h)w_h\|_{BV} \leq \|v_h\|_{BV}.$$

Because of this last property the operator $\mathbf{P}(\cdot)$ is said to be a TVD (total variation diminishing)-projection. With this new procedure the oscillations due to the linearization leading to the definition of Φ_h^n do not appear, and the method is *always* $L^\infty \cap BV$- stable, regardless of the size of the timestep.

In this work we have tested numerically the proposed method in the case of Godunov's implicit scheme, the simplest and best-known implicit finite difference scheme for (1.1).

An outline of the paper follows. In §2 Newton's method for solving the implicit Godunov scheme for (1.1) is given. We then display the sufficient Kantorovich conditions for its convergence in the simple but important case in which $f' \geq 0$. In this situation a nonlinear cfl–type condition is obtained (see Theorem 2.1). In particular, it is shown that if the ratio $\|f'\|_{L^\infty}/\|f''\|_{L^\infty}$ is big enough the Newtonian iterations are stable. Section 3 is devoted to testing the stability of Newton's method in five different test problems. In §4 we introduce our TVD-projection and we show how to modify the Newtonian iterations so that we always have $L^\infty \cap BV$-stability (see our

main result, Theorem 4.3). In §5 we test this method in the same problems in which Newton's method was tested, and we compare the corresponding performances. Some concluding remarks are given in §6.

2. Newton's method for Godunov's implicit scheme. In this section we give Godunov's implicit scheme for (1.1) and its Newtonian iterations. The stability of Newton's iterations seems to depend essentially on the size of $|f''|$. For if $f'' = 0$, then the Newtonian iterations are trivially stable regardless of the size of $|f'|$. Also, it is intuitively clear that the presence of a discontinuity represents a threat to the stability of the method. In the simple, but illuminating, case in which $f' \geq 0$ (and $f \in C^2$) these statements can be made precise. In fact, in this situation the Kantorovich sufficient conditions can be explicitly calculated, and rewritten as a nonlinear cfl–type condition involving only the following quantity:

$$\|f'\|_{L^\infty}/(\mathrm{disc}(u_o)\,\|f''\|_{L^\infty}),$$

where $\mathrm{disc}(u_o)$ represents the maximum size of the discontinuities of u_o.

2.1. The scheme. As usual, let $\{t^n\}_{n=0,\cdots,nt}$ be a partition of $[0,T]$, and let $\{x_{i+1/2}\}_{i=0,\cdots,nx}$ be one of $[0,1]$. Let us set $\Delta t^n = t^{n+1} - t^n$ and $\Delta x_i = x_{i+1/2} - x_{i-1/2}$. To the partition $\{t^n\}_{n=0,\cdots,nt}$ associate the space

$$W_{\Delta t} = \{v_{\Delta t} \in BV(0,T) : v_{\Delta t}(t) = v^n, t \in (t^n, t^{n+1}), n = 0, \cdots, nt-1\}, \tag{2.1a}$$

and to the partition $\{x_{i+1/2}\}_{i=0,\cdots,nx}$, the space

$$\begin{aligned} V_h = \{v_h \in BV(0,1) : v_h(x_{1/2}) = v_0, v_h(x_{nx+1/2}) = v_{nx+1}, \\ v_h(x) = v_i, x \in (x_{i-1/2}, x_{i+1/2}), i = 1, \cdots, nx\}. \end{aligned} \tag{2.1b}$$

Define $b_{0\Delta t}$ and $b_{1\Delta t}$ to be the L^2-projection of b_0 and b_1, respectively, on the space $W_{\Delta t}$, and let u_{0h} be the L^2-projection of u_0 on V_h. Then, Godunov's implicit scheme defines an approximate solution $u_h \in W_{\Delta t} \times V_h$ in the following manner:

(2.2a) Compute the degrees of freedom of u_h^0 as follows:

$$u_i^0 = \int_{x_{i-1/2}}^{x_{i+1/2}} u_0(s)\,ds/\Delta x_i, \quad i = 1, \cdots, nx;$$

(2.2b) For $n = 1, \cdots, nt$ compute the degrees of freedom of u_h^n as the solution of

$$\begin{aligned} &u_0^n = b_0^n, \quad u_{nx+1}^n = b_1^n, \\ &(u_i^n - u_i^{n-1})/\Delta t^n + (f_{i+1/2}^{G,n} - f_{i-1/2}^{G,n})/\Delta x_i = 0, \quad i = 1, \cdots, nx, \end{aligned}$$

where the Godunov flux $f_{i+1/2}^{G,n} = f^G(u_{i-1/2}^n, u_{i+1/2}^n)$ is defined by

$$f^G(v,u) = f(\xi), \tag{2.2c}$$

where ξ is any point of the interval $I(u,v) = [\min\{u,v\}, \max\{u,v\}]$ such that

$$(\mathrm{sign}(u-v)) \cdot (f(\xi) - f(c)) \leq 0, \quad \forall c \in I(u,v). \tag{2.2d}$$

Note that when $f' \geq 0$, then $f_{i+1/2}^{G,n} = f(u_i^n)$, and when $f' \leq 0$ then $f_{i+1/2}^{G,n} = f(u_{i+1}^n)$. Thus, f^G is a generalization of the classical upwinding flux. Also, among the fluxes of the form $\tilde{f}_{i+1/2} = \tilde{f}(u_i, u_{i+1})$ it is the one that produces the smallest amount of viscosity. (See Osher [7] and Brenier and Osher [2] for more details.) We point out that the uniqueness of the approximate solution defined by (2.2) follows easily from the fact that the flux $f^G(v,u)$ is nonincreasing in u and nondecreasing in v.

2.2. The Newtonian iterations. It is clear that when $f' \geq 0$ or $f' \leq 0$ the Godunov flux is C^1, provided, of course, that $f \in C^1$. However, in the general case this is not true. For example, if $f(u) = u \cdot (1-u)$, then for $w > 1 - w \neq \frac{1}{2}$:

$$\partial_1 f^{G+}(1-w, w) = 0, \qquad \partial_1 f^{G-}(1-w, w) = f'(1-w),$$
$$\partial_2 f^{G+}(1-w, w) = f'(w), \quad \partial_2 f^{G-}(1-w, w) = 0.$$

To define our Newtonian iterations we shall use the following definition of $\partial_1 f^G$, $\partial_2 f^G$:

$$\partial_1 f^G(v,u) = \begin{cases} f'(v), & \text{if } v = \xi, \\ 0, & \text{otherwise;} \end{cases} \tag{2.3a}$$

$$\partial_2 f^G(v,u) = \begin{cases} f'(u), & \text{if } u = \xi, \\ 0, & \text{otherwise;} \end{cases} \tag{2.3b}$$

where ξ is the same as before (see (2.2c), (2.2d)). Thus, we define the operator Φ_h^n as follows:

$$\begin{aligned} &\Phi_h^n : V_{\Delta x} \to V_{\Delta x}, \\ &\quad u_h^{(m)} \mapsto u_h^{(m+1)}, \end{aligned} \tag{2.4a}$$

where

$$\begin{aligned} &u_0^{(m+1)} = b_0^n, \quad u_{nx+1}^{(m+1)} = b_1^n, \\ &(u_i^{(m+1)} - u_i^{n-1})/\Delta t^{n-1} + (f_{i+1/2}^{NG,m+1/2} - f_{i-1/2}^{NG,m+1/2})/\Delta x_i = 0, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = 1, \cdots, nx, \end{aligned} \tag{2.4b}$$

and the flux $f_{i+1/2}^{NG,m+1/2}$ is given by

$$\begin{aligned} f_{i+1/2}^{NG,m+1/2} = f^G(u_i^{(m)}, u_{i+1}^{(m)}) &+ [\partial_1 f^G(u_i^{(m)}, u_{i+1}^{(m)})] \cdot (u_i^{(m+1)} - u_i^{(m)}) \\ &+ [\partial_2 f^G(u_i^{(m)}, u_{i+1}^{(m)})] \cdot (u_{i+1}^{(m+1)} - u_{i+1}^{(m)}), \end{aligned} \tag{2.4c}$$

where $\partial_1 f^G$and $\partial_2 f^G$ are defined by (2.3). It is an easy matter to show that the operator Φ_h^n is well defined for every partition $\{t^n\}_{n=0,\cdots,nt} \times \{x_{i+1/2}\}_{i=0,\cdots,nx}$. As we pointed out before, this is again guaranteed by the fact that $f^G(v,u)$ is nonincreasing in u and nondecreasing in v.

Finally, Newton's method can be rewritten as follows:

(2.5a) Initialize the iterations by setting $u_h^{(1)} = u_h^{n-1}$;
(2.5b) For $m = 1, \cdots, M^n$ compute $u_h^{(m+1)} = \Phi_h^n(u_h^{(m)})$;
(2.5c) Set $u_h^n = u_h^{M^n+1}$.

2.3. The Kantorovich sufficient convergence conditions in the case $f' \geq 0$. Let us assume that $f' \geq 0$, and that $f \in C^2$. Note that in this case the flux $f_{i+1/2}^{G,n}$ is simply $f(u_i^n)$. Let us define the residue of $w_h \in V_h$ as follows:

$$\mathrm{res}_h(w_h)_i = w_i - u_i^{n-1} + \frac{\Delta t^{n-1}}{\Delta x_i}(f(w_i) - f(w_{i-1})), \quad i = 1, \cdots, nx.$$

Then, the Kantorovich sufficient conditions for the convergence of the Newtonian iterations (2.5) can be written in the following manner (see [8]):

$$abc \leq \frac{1}{2},$$

where

$$a = \|\Phi_h^n(u_h^{n-1}) - u_h^{n-1}\|,$$
$$b = \|(D\ \mathrm{res}_h(u_h^{n-1}))^{-1}\|,$$
$$c = \sup_{w_h \in \mathcal{B}} \|D^2\ \mathrm{res}_h(u_h^{n-1})\|,$$
$$\mathcal{B} = \{w_h \in V_h : \|w_h - u_h^{n-1}\| < \delta\},$$
$$\delta \geq (1 - (1 - 2 \cdot a \cdot b \cdot c)^{1/2})/bc,$$

and $\|\cdot\|$ has been chosen to be the L^∞-norm (and its subordinate matrix norms!).

In the case under consideration it is easy to obtain sharp estimates a_0, b_0, c_0 for the quantities a, b, c, respectively. This will be done in the next section. At first glance the most difficult part seems to be the estimate of c, for it involves the quantity δ, defined itself in terms of the quantities we want to estimate. However, this is not at all the case, as the following argument shows.

It is well known that the approximate solution u_h given by (2.2) verifies the maximum principle

$$u_h(t,x) \in [\alpha, \beta], \quad (t,x) \in [0,T] \times [0,1],$$

where $\alpha = \inf_{t,x}\{b_0(t), b_1(t), u_0(x)\}$ and $\beta = \sup_{t,x}\{b_0(t), b_1(t), u_0(x)\}$. As a consequence, the only relevant values of f are those attained on the interval $[\alpha, \beta]$. We can then modify f outside this interval without modifying the approximate solution u_h. Let us denote by $\tilde{f}$ any C^2 extension of f outside $[\alpha, \beta]$ verifying

$$\|\tilde{f}'\|_{L^\infty(\mathbf{R})} = \|f'\|_{L^\infty(\alpha,\beta)}, \quad \|\tilde{f}''\|_{L^\infty(\mathbf{R})} = \|f''\|_{L^\infty(\alpha,\beta)}.$$

For the sake of simplicity we shall write f instead of $\tilde{f}$. In this way, we can estimate the quantity c in the following (very crude) manner:

$$c \leq \sup_{w_h \in \mathcal{B}'} \|D^2\ \mathrm{res}_h(w_h)\|,$$
$$\mathcal{B}' = \{w_h \in V_h : w_h(x) \in [\alpha, \beta], \forall x \in [0,1]\},$$

which is now independent of δ. In fact, the most difficult estimate is the estimate of b, for we must deal with the inverse of $D\ \mathrm{res}_h(w_h)$, which is difficult to calculate in the general case.

Once the estimates a_0, b_0, c_0 have been found, we impose the condition $a_0 b_0 c_0 \leq \frac{1}{2}$ and do some algebraic manipulations to obtain the following convergence result.

THEOREM 2.1. *Suppose that $f' \geq 0$, and that $f \in C^2$. Then, the Newtonian iterations* (2.5) converge if

$$\varphi_{nx}(cfl) \leq \frac{1}{2}(\|f'\|_{L^\infty}/(\mathrm{disc}(u_h^{n-1}) \cdot \|f''\|_{L^\infty}))^{1/2} / \sup_{i,j}\left(\frac{\Delta x_j}{\Delta x_i}\right), \tag{2.7a}$$

where

$$cfl = \sup_i \frac{\Delta t^{n-1}}{\Delta x_i}\|f'\|_{L^\infty}, \tag{2.7b}$$

$$\varphi_{nx}(cfl) = cfl \cdot (1 + cfl) \cdot \left(1 - \left(\frac{cfl}{1 + cfl}\right)^{nx}\right), \tag{2.7c}$$

$$\mathrm{disc}(u_h^{n-1}) = \sup_i |u_{i+1}^{n-1} - u_i^{n-1}|. \tag{2.7d}$$

The proof of this result will be given in the next section. Note that no maximum principle of the Newtonian iterations (2.5) is needed (explicitly). The fact that $\varphi_{nx}(cfl) \leq cfl \cdot (1 + cfl)$ leads immediately to the following Corollary.

COROLLARY 2.2. *Under the hypothesis of Theorem* 2.1 *the Newtonian iterations* (2.5) *converge if*

$$(2.8)\quad cfl \leq \left\{ \frac{1}{2}(\|f'\|_{L^\infty}/(disc(u_h^{n-1}) \cdot \|f''\|_{L^\infty}))^{1/2} / \sup_{i,j}\left(\frac{\Delta x_j}{\Delta x_i}\right) + \frac{1}{4} \right\}^{1/2} - \frac{1}{2}.$$

Note that for cfl fixed $\varphi_{nx}(\text{cfl})$ behaves like cfl·(1+cfl) as $nx \to \infty$. Thus, to bound the former function by the latter one is not a crude estimate. Note also that if u_h^{n-1} is smooth in the sense that

$$\text{disc}(u_h^{n-1}) \leq C \cdot h, \text{ as } h \downarrow 0,$$

where $h = \sup_i \Delta x_i$, then (2.8) is verified if $\text{cfl} \leq C \cdot h^{-1/4}$. This shows how the smoothness of the initial data enhances the stability of the method. Also, if the scheme converges in such a way that $\text{disc}(u^{n-1}) \leq C \cdot h^\alpha$ for $n \gg 1$ (which implies that the $L^\infty(L^1)$-order of convergence is equal to $1-\alpha$), then (2.8) is verified if $\text{cfl} \leq C \cdot h^{-\alpha/4}$. This shows that the Kantorovich convergence condition may be restrictive only in the begining of the computation.

To end this section, let us illustrate the convergence condition (2.8). Assume that the meshes are uniform:

(1) If $[\alpha, \beta] = [0, u_\infty]$ and $f(u) = u^2$ then (2.8) reads

$$\begin{aligned} cfl &\leq \{\frac{1}{2}(u_\infty/\text{disc}(u_h^{n-1}))^{1/2} + \frac{1}{4}\}^{1/2} - \frac{1}{2} \\ &\leq 0.366 \text{ in the worst case.} \end{aligned}$$

(2) If $[\alpha, \beta] = [0, 1]$ and $f(u) = 20 \cdot u^3/(20 \cdot u^3 + (1-u)^3)$ then

$$\begin{aligned} cfl &\leq \{0.187(1/\text{disc}(u_h^{n-1}))^{1/2} + \frac{1}{4}\}^{1/2} - \frac{1}{2} \\ &\leq 0.161 \text{ in the worst case.} \end{aligned}$$

This is the example considered in the Introduction. The last condition is equivalent to $\Delta t \leq 0.041 \Delta x$. Compare it with the cfl-condition for the explicit Godunov scheme $\Delta t \leq 0.256 \Delta x$, and with the size of the timestep for which the Newton iterations became unstable $\Delta t = 0.690 \Delta x$!

2.4. Proof of Theorem 2.1. We shall proceed in several steps.

(1) Estimating a. We have:

$$\begin{aligned} a &= \|\Phi_h^n(u_h^{n-1}) - u_h^{n-1}\| \\ &= \|(D\ \text{res}_h(u_h^{n-1}))^{-1}\ \text{res}_h(u_h^{n-1})\| \\ &\leq \|(D\ \text{res}_h(u_h^{n-1}))^{-1}\| \cdot \|\text{res}_h(u_h^{n-1})\| \\ &= b \cdot \|\text{res}_h(u_h^{n-1})\| \\ &= b \cdot \sup_i \frac{\Delta t^{n-1}}{\Delta x_i} |f(u_i^{n-1}) - f(u_{i-1}^{n-1})| \\ &\leq b \cdot cfl \cdot \text{disc}(u_h^{n-1}) \\ &\leq b_0 \cdot cfl \cdot \text{disc}(u_h^{n-1}) \equiv a_0. \end{aligned}$$

(2) Estimating b. We need the following Lemma, the proof of which is left to the reader.

LEMMA 2.3. *The matrix* $\Gamma = (D\,\mathrm{res}_h(u_h^{n-1}))^{-1}$ *is given by:*

$$\Gamma_{ij} = 0, \quad \text{for } i < j,$$
$$\Gamma_{ii} = 1 - \theta_i,$$
$$\Gamma_{ij} = \theta_i \cdots \theta_{i-j}(1-\theta_i)\frac{\Delta x_j}{\Delta x_i}, \quad \text{for } i > j,$$

where $\theta_i = cfl_i/(1+cfl_i)$ *and* $cfl_i = \dfrac{\Delta t^{n-1}}{\Delta x_i} f'(u_i^{n-1})$, *for* $i, j = 1, \cdots, nx$.

Now, set $\theta = \max_i \theta_i$. As a consequence of this result we have the following:

$$\begin{aligned} b &= \|\Gamma\|, \\ &= \max_i \sum_{j=1}^{i} \Gamma_{ij} \\ &\le \max_i (1-\theta_i) \cdot \sum_{j=1}^{i} \theta^{j-1} \frac{\Delta x_j}{\Delta x_i} \\ &\le \max_i (1-\theta_i) \cdot \left(\sum_{j=1}^{i} \theta^{j-1} \right) \cdot \left(\sup_{i,j} \frac{\Delta x_j}{\Delta x_i} \right) \\ &\le \left(\frac{1-\theta^{nx}}{1-\theta} \right) \cdot \left(\sup_{i,j} \frac{\Delta x_j}{\Delta x_i} \right) \equiv b_0. \end{aligned}$$

(3) Estimating c. From the definition of res_h, we have

$$\frac{\partial^2}{\partial w_\ell \partial w_j} \mathrm{res}_k(u_h^{n-1}) = \frac{\Delta t^{n-1}}{\Delta x_k} \{ f''(u_k^{n-1}) \delta_{kj} \delta_{k\ell} - f''(u_{k-1}^{n-1}) \delta_{(k-1)j} \delta_{(k-1)\ell} \},$$

and thus,

$$\begin{aligned} c &\le \sup_{w_h \in B'} \|D^2\, \mathrm{res}_h(u_h^{n-1})\|, \\ &= \sup_{w_h \in B'} \left\{ \max_k \sum_{j,\ell} \left| \frac{\partial^2}{\partial w_\ell \partial w_j} \mathrm{res}_k(u_h^{n-1}) \right| \right\} \\ &\le 2\, cfl\, \|f''\|_{L^\infty} / \|f'\|_{L^\infty} \equiv c_0. \end{aligned}$$

(4) From the preceding steps we have

$$\begin{aligned} a \cdot b \cdot c &\le a_0 \cdot b_0 \cdot c_0 \\ &= 2\, cfl^2 \cdot \left(\frac{1-\theta^{nx}}{1-\theta} \right)^2 \cdot \left(\sup_{i,j} \frac{\Delta x_j}{\Delta x_i} \right)^2 \cdot \mathrm{disc}(u_h^{n-1}) \cdot \frac{\|f''\|_{L^\infty}}{\|f'\|_{L^\infty}}. \end{aligned}$$

As the application $x \mapsto x/(1+x)$ is always increasing for $x \ge 0$, we can take $\theta = cfl/(1+cfl)$. Finally, from the inequality $a_0 b_0 c_0 \le \frac{1}{2}$ it is easy to obtain (2.7). This proves Theorem 2.1.

3. Testing the stability of Newton's method. In this section we shall test numerically the stability of the Newtonian iterations (2.5) for (2.2) in five different problems of increasing difficulty. All of them are Riemann problems that have an easy-to-calculate self-similar solution. The first three have a nondecreasing nonlinearity; i.e., $f' \ge 0$. The last two have a nonconvex f and each of their solutions presents a

sonic point, i.e., a point u^* for which $f'(u^*) = 0$. After displaying the test problems, we define a function that measures the stability of (2.2), (2.5), and we evaluate it numerically. We end with a brief discussion of the numerical results that will serve as a reference for evaluating the performance of our proposed method.

3.1. The test problems. The test problems are of the form (1.1), where

$$f_i(u) = z_i u^3 \,(1 - r_i(1-u)^3)/(z_i u^3 + (1-u)^3)\ ,\ i = 1, \cdots, 5, \tag{3.1}$$

$$\begin{aligned} u_{0i}(x) &= 0, & x &\in (0,1), & i &= 1,2,3,\\ b_{0i}(t) &= 1, & t &\in (0, T = 0.3), & i &= 1,2,3,\\ b_{1i}(t) &= 0, & x &\in (0, T = 0.3), & i &= 1,2,3, \end{aligned}$$

(3.2)

$$\begin{aligned} u_{0i}(x) &= 0, & x &\in (0, 0.6),\\ &= 1, & x &\in (0.6, 1), & i &= 4,5,\\ b_{0i}(t) &= 0, & t &\in (0, T = 0.05), & i &= 4,5,\\ b_{1i}(t) &= 1, & x &\in (0, T = 0.05), & i &= 4,5. \end{aligned}$$

See Figs. 1.1, 1.2, 2.1, and 2.2, and Tables 1 and 2.

3.2. Measuring the stability (and convergence) of the method. To measure the stability properties of (2.2), (2.5) we proceed as follows. First, we fix Δx (all the grids are taken to be uniform) small enough to ensure that the error is already in the region of its asymptotic behavior. Then we define the quantity Λ as follows:

$$\Lambda = \sup\{\Delta t_0 : rc(\Delta t, \Delta x) > 1/2 \quad \forall \quad \Delta t < \Delta t_0\}/\Delta x, \tag{3.3a}$$

where

$$rc(\Delta t, \Delta x) = \left(\log \frac{\|u(T) - u_h(\Delta t, \Delta x;\ T)\|_{L^1(0,1)}}{\|u(T) - u_h(\Delta t/2, \Delta x/2; T)\|_{L^1(0,1)}}\right) / \log\ 2, \tag{3.3b}$$

is the estimated rate of convergence. We denoted by $u_h(\Delta t, \Delta x; T)$ the approximate solution, obtained by (2.2), (2.5) with $\Delta t^n \equiv \Delta t$ and $\Delta x_{i+1/2} \equiv \Delta x$, at time T; u(T) is the exact solution at that time.

We recall the fact that $\frac{1}{2}$ is the worst rate of convergence for the scheme (2.2). (This is also true for three-point implicit monotone schemes; see Sanders [9].) Thus, we shall consider Λ as a measure of the stability of the scheme. The bigger Λ, the more stable the scheme. We shall take $\Delta x = 0.001$ in all the examples.

3.3. The numerical results. For each of our five test problems we have calculated $\Lambda(M^n \equiv 1, \equiv 2, \equiv 3)$ (see Table 3), the corresponding total number of time iterations nt (i.e., M^n times the number of timesteps; see Table 4), the error $\|u(T) - u_h(T)\|_{L^1(0,1)}$ (see Table 5), and finally, the efficiency of the algorithm (Table 6), defined as follows:

$$eff = \{\|u_{(T)} - u_h(T)\|_{L^1(0,1)} \cdot nt\}^{-1}. \tag{3.4}$$

In Tables 3, 4, 5, and 6, the approximate solution $u_h(T)$ is the one obtained with $\Delta t = \Lambda\ \Delta x$. From Table 3 we see that the stability of (2.5) increases when passing from one inner iteration to two, then it decreases when passing to three inner iterations. (For more than three inner iterations Λ can possibly oscillate slightly around $\Lambda(M^n \equiv 3)$ and then decrease slowly to an asymptotic value; we have not shown these results.) It is also interesting to note that for a fixed value of M^n, the value of Λ decreases

monotonically with i, the number of our test problems. This corresponds to the increasing values of $\|f_i'\|_{L^\infty}$ and $\|f_i''\|_L^\infty$ (see Tables 1 and 2). Although it is not easy to find a simple relation between the values of Λ and those of f', f'', Table 7 seems to indicate that the following relationship is verified: $\Lambda_i(M^n \equiv 1) \cdot \max\{s_i^+, s_i^-\} = 1$.

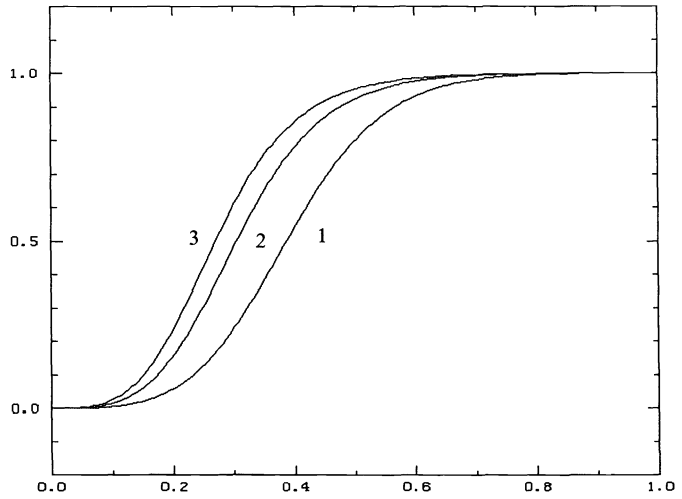


FIG. 1.1. *The functions* $u \mapsto f_i(u)$, $i = 1, 2, 3$.

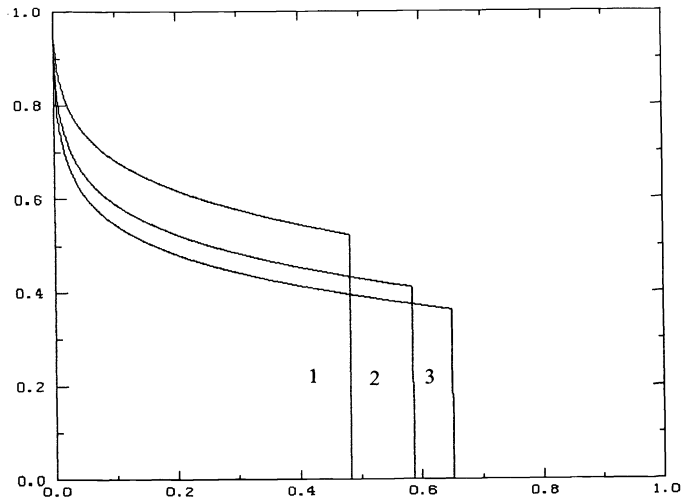


FIG. 1.2. *The solutions* $x \mapsto u_i(T = 0.3, x)$, $i = 1, 2, 3$.

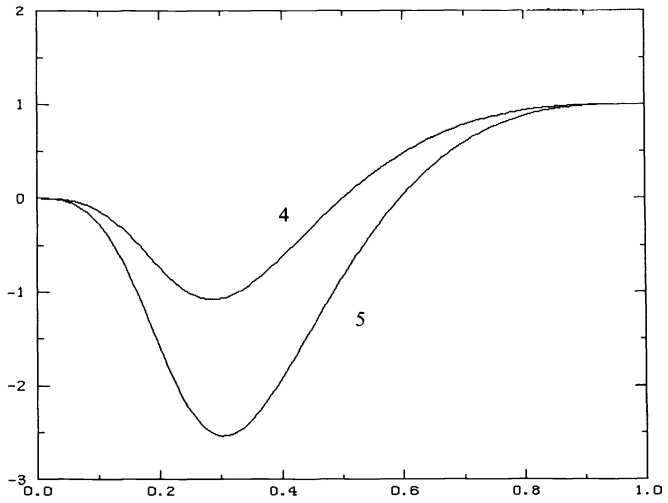


FIG. 2.1. *The functions* $u \mapsto f_i(u)$ $i = 4, 5$.

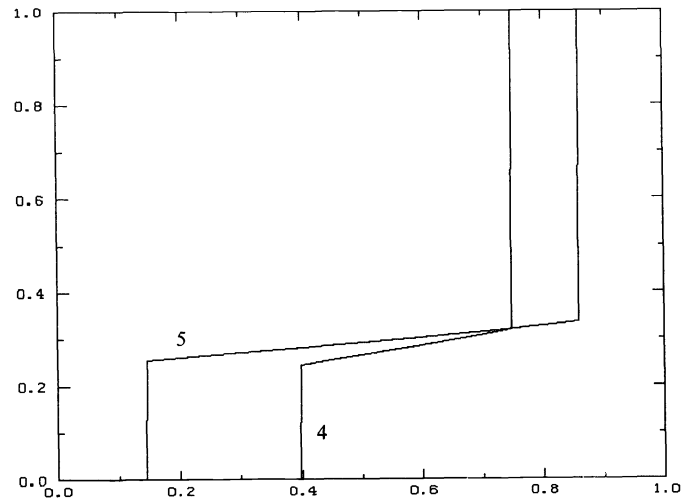


FIG. 2.2. *The solutions* $x \mapsto u_i(T = 0.05, x)$, $i = 4, 5$.

TABLE 1

The index i indicates the number of the test problem, and s_i the speed of the discontinuity of the solution of problem i.

i	z_i r_i	$\|f_i'\|_{L^\infty(0,1)}$	$\|f_i''\|_{L^\infty(0,1)}$	s_i
1	4 0	3.18	16.67	1.61
2	12 0	3.61	23.18	1.95
3	20 0	3.91	28.19	2.17

TABLE 2

As before, i indicates the number of the problem; s_i^+ and s_i^- are the speeds of the discontinuities (of the solution of problem i) traveling to the right and to the left, respectively.

i	z_i r_i	$\|f_i'\|_{L^\infty(0,1)}$	$\|f_i''\|_{L^\infty(0,1)}$	s_i^+ s_i^-
4	20 8	7.17	99.19	2.98 4.05
5	20 15	16.10	194.99	5.19 9.08

TABLE 3

Values of Λ (defined by (3.3)) for each of the test problems "i", $i = 1, \cdots, 5$ and $M^n \equiv 1, 2, 3$ where M^n is the number of total inner iterations in the Newtonian iterations (2.5).

Test problem	$M^n \equiv 1$	$M^n \equiv 2$	$M^n \equiv 3$
1	0.617	1.327	1.240
2	0.507	0.811	0.777
3	0.457	0.694	0.682
4	0.250	0.595	0.342
5	0.110	0.221	0.160

TABLE 4

Values of the total number of time iterations $nt = M^n \times$ *number of timesteps. Note that* $nt = M^n \times T/\Delta t = M^n \times T/(\Delta x \cdot \Lambda)$.

Test problem	$M^n \equiv 1$	$M^n \equiv 2$	$M^n \equiv 3$
1	486	452	726
2	592	740	1158
3	656	864	1320
4	200	168	438
5	452	936	936

TABLE 5

Values of $10^3 \times \|u(T) - u_h(T)\|_{L^1(0,1)}$ *with* $\Delta t = \Lambda \cdot \Delta x$.

Test problem	$M^n \equiv 1$	$M^n \equiv 2$	$M^n \equiv 3$
1	2.18	62.34	30.50
2	2.24	16.05	12.38
3	2.23	137.82	6.30
4	2.51	31.61	8.46
5	1.41	8.66	4.42

TABLE 6

Values of $10^4 \times$ *eff defined by* (3.4).

Test problem	$M^n \equiv 1$	$M^n \equiv 2$	$M^n \equiv 3$
1	9.44	0.35	0.45
2	7.54	0.84	0.70
3	6.84	0.84	1.20
4	19.92	1.88	2.70
5	15.62	2.55	2.42

TABLE 7
The cfl-number $\Lambda_i(M^n \equiv 1) \cdot \max\{s_i^+, s_i^-\}$. *(For* $i = 1$, *2, and* 3 $\max\{s_i^+, s_i^-\}$ *is simply* s_i.)

Test problem	$\Lambda_i(M^n \equiv 1) \cdot \max\{s_i^+, s_i^-\}$
1	0.993
2	0.988
3	0.991
4	1.010
5	0.999

This result indicates that the stability of the Newtonian iterations (2.5) with $M^n \equiv 1$ depends only on the cfl–number. The stability condition is the same as the one for the explicit version of Godunov scheme, which with less computational effort gives better results.

Table 4 shows that the amount of work for $M^n \equiv 1, 2, 3$ is of the same order of magnitude. Tables 5 and 6 show that $M^n \equiv 1$ is not only the most efficient choice of M^n, but that it gives rise to a better approximate solution.

4. The TVD-projection method. In this section we define and study the stability properties of the so-called TVD (total variation diminishing)-projection $\mathbf{P}(w_h)$. Then, we write down our TVD-projection method and study its stability. We end with some remarks on the relationship between the convergence and the conservativity of the method.

4.1. Definition of $\mathbf{P}(w_h)$. Let w_h and v_h be two arbitrary elements of V_h. To define $\mathbf{P}(w_h)v_h$ proceed as follows. First, to each point of the partition $\{x_{i+1/2}\}_{i=0,\cdots,nx}$ associate a value $\eta_{i+1/2}$ defined as follows:

$$\eta_{i+1/2} = \theta_{i+1/2} w_i + (1 - \theta_{i+1/2}) w_{i+1}, \tag{4.1a}$$

where

$$\theta_{i+1/2} = \begin{cases} 1, & \text{if } \operatorname{sgn}(v_{i+1} - w_{i+1}) = \operatorname{sgn}(v_i - w_i) \neq \operatorname{sgn}(w_{i+1} - w_i), \\ 0, & \text{if } \operatorname{sgn}(v_{i+1} - w_{i+1}) = \operatorname{sgn}(v_i - w_i) = \operatorname{sgn}(w_{i+1} - w_i), \\ \frac{v_{i+1} - w_{i+1}}{(v_{i+1} - w_{i+1}) - (v_i - w_i)}, & \text{otherwise.} \end{cases} \tag{4.1b}$$

Also, define $\eta_{-1/2}$ and $\eta_{nx+3/2}$ as follows:

$$\begin{aligned} \eta_{-1/2} &= w_0, \\ \eta_{nx+3/2} &= w_{nx+1}. \end{aligned} \tag{4.1c}$$

The definition of $\eta_{i+1/2}$ is illustrated in Fig. 3.

Next, for $i = 0, \cdots, nx + 1$, define the intervals I_i as follows:

(4.2)
$$I_i = \begin{cases} I(w_i, \eta_{i+1/2}) \cup I(w_i, \eta_{i-1/2}), & \text{if sign } (w_i - \eta_{i+1/2})(w_i - \eta_{i-1/2}) < 0\ , \\ I(w_{i,\eta_i+1/2}) \cap I(w_i, \eta_{i-1/2}), & \text{otherwise.} \end{cases}$$

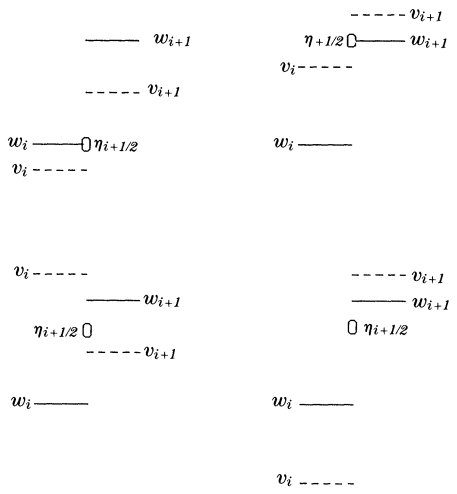


FIG. 3. *Examples of* $\eta_{i+1/2}$.

Now, introduce the L^∞–projection on the interval $[a, b]$:

(4.3a)
$$P^\infty_{[a,b]}(c) = \begin{cases} b, & \text{if } c > b, \\ c, & \text{if } c \in [a, b], \\ a, & \text{if } c < a. \end{cases}$$

Finally, define the projection $\mathbf{P}(w_h)v_h$ as follows:

(4.3b)
$$\mathbf{P}(w_h) : V_h \to V_h,$$
$$v_h \mapsto u_h,$$

where

(4.3c)
$$u_i = P_{I_i}{}^\infty(v_i), \quad i = 0, \cdots, nx + 1,$$

where the $\eta_{i+1/2}$ are defined by (4.1). Note that the operator $\mathbf{P}(w_h)$ is a projection, for it is easy to check that

$$\mathbf{P}(w_h)\ \mathbf{P}(w_h)\ v_h = \mathbf{P}(w_h)\ v_h.$$

4.2. Stability properties of $\mathbf{P}(w_h)$. Let us begin with the following result.

LEMMA 4.1. *Set $u_h = \mathbf{P}(w_h)v_h$. Then for $i = 1, \cdots, nx$:*

$$u_i \in I_i, \tag{4.4a}$$

$$\eta_{i+1/2} \in I(w_i\ ,\ w_{i+1}), \quad (\textit{valid also for } i = 0), \tag{4.4b}$$

$$|\eta_{i+1/2} - u_i| + |u_i - \eta_{i-1/2}| \leq |\eta_{i+1/2} - w_i| + |w_i - \eta_{i-1/2}|. \tag{4.4c}$$

PROOF. Property (4.4a) is a direct consequence of (4.2) and (4.3). Property (4.4b) follows from the definition of $\eta_{i+1/2}$, (4.1). Finally, (4.4c) follows from (4.4a). □

We can state now our key result.

PROPOSITION 4.2. *Set $u_h = \mathbf{P}(w_h)v_h$. Then,*

$$\inf_x u_h \geq \inf_x w_h\ , \tag{4.5a}$$

$$\sup_x u_h \leq \sup_x w_h\ , \tag{4.5b}$$

$$\|u_h\|_{BV} \leq \|w_h\|_{BV}. \tag{4.6}$$

PROOF. Inequalities (4.5) are a corollary of (4.4a),(4.4b), and (4.3). To prove (4.6) we proceed as follows:

$$\begin{aligned}
\|u_h\|_{BV} &= \sum_{i=0}^{nx} |u_{i+1} - u_i| \\
&\leq \sum_{i=0}^{nx} (|u_{i+1} - \eta_{i+1/2}| + |\eta_{i+1/2} - u_i|) \\
&= |\eta_{1/2} - u_0| + \sum_{i=1}^{nx} (|\eta_{i+1/2} - u_i| + |u_i - \eta_{i-1/2}|) + |\eta_{nx+1/2} - u_{nx+1}| \\
&= |\eta_{1/2} - \eta_{-1/2}| + \sum_{i=1}^{nx} (|\eta_{i+1/2} - u_i| + |u_i - \eta_{i-1/2}|) + |\eta_{nx+1/2} - \eta_{nx+3/}
\end{aligned}$$

by (4.1c), and the definition of $\mathbf{P}(w_h)$, (4.3),

$$= |\eta_{1/2} - \eta_{-1/2}| + \sum_{i=1}^{nx} (|\eta_{i+1/2} - w_i| + |w_i - \eta_{i-1/2}|) + |\eta_{nx+1/2} - \eta_{nx+3/}$$

by (4.4c),

$$= |\eta_{1/2} - w_0| + \sum_{i=1}^{nx} (|\eta_{i+1/2} - w_i| + |w_i - \eta_{i-1/2}|) + |\eta_{nx+1/2} - w_{nx+1}|,$$

by (4.1c),

$$\begin{aligned}
&= \sum_{i=0}^{nx} (|\eta_{i+1/2} - w_i| + |w_{i+1} - \eta_{i+1/2}|), \\
&= \sum_{i=0}^{nx} |w_{i+1} - w_i|, \text{ by (4.4b)}\ , \\
&= \|w_h\|_{BV}.
\end{aligned}$$

This completes the proof. □

4.3. The TVD–projection method. With the notation introduced previously, we can write our algorithm for the inner iterations of (2.2) as follows:

(4.7a) Initialize the iterations by setting $u_h^{(1)} = u_h^n$;

(4.7b) For $m = 1, \cdots, M^n$ compute $u_h^{(m+1)} = \mathbf{P}(u_h^{(m)})\Phi_h^n(u_h^{(m)})$;

(4.7c) Set $u_h^n = u_h^{M^n+1}$.

Compare this with the Newtonian iterations (2.5). This method has the advantage of being stable regardless of the characteristics of the partitions $\{t^n\}_{n=0,\cdots,nt}$ and $\{x_{i+1/2}\}_{i=0,\cdots,nx}$. Moreover, it also guarantees the compactness of the sequence $\{u_h\}_{h>0}$ determined by (2.2), (4.7) in the $L^\infty(0,T;L^1(0,1))$–norm. This is a direct consequence of the following result.

THEOREM 4.3. *Let u_h be the approximate solution obtained by the implicit Godunov scheme* (2.2) *and the modified Newtonian iterations* (4.7). *Then,*

$$\inf u_h \geq \inf_{t,x} [b_0(t), b_1(t), u_0(x)],$$

$$\sup u_h \leq \sup_{t,x} [b_0(t), b_1(t), u_0(x)],$$

$$\|u_h\|_{L^\infty(0,T;BV(0,1))} \leq \|b_0\|_{BV(0,T)} + \|b_1\|_{BV(0,T)} + \|u_h\|_{BV(0,1)}.$$

The proof is straightforward. It is a direct consequence of Proposition 4.2 and the definition of $b_{0\Delta t}, b_{1\Delta t}, u_{0h}$. Thus, the sequence $\{u_h\}_{h>0}$ has a subsequence converging strongly in $L^\infty(0,T;L^1(0,1))$ to a limit. The problem of proving that this limit is indeed the entropy solution of (1.1) is very delicate and will not be considered here.

4.4. The price of stability. From the definition of the Newtonian iterations (2.4), (2.5) we easily obtain the following expression for the conservations of mass:

$$\sum_{i=1}^{nx} u_i^{(m+1)} \Delta x_i = \sum_{i=1}^{nx} u_i^{n-1} \Delta x_i - \Delta t^{n-1}(f_{nx+1/2}^{NG,m+1/2} - f_{1/2}^{NG,m+1/2}),$$

where $u_h^{(m+1)} = \Phi_h^n(u_h^{(m)})$. On the other hand, by (2.2) we have for the approximate solution of the Godunov scheme

$$\sum_{i=1}^{nx} u_i^n \Delta x_i = \sum_{i=1}^{nx} u_i^{n-1}\ \Delta x_i - \Delta t^{n-1}(f_{nx+1/2}^{G,n} - f_{1/2}^{G,n}).$$

Thus, the error in the total amount of mass at the timestep (t^{n-1}, t^n) introduced by Newton's method is simply

$$-\Delta t^{n-1}((f_{nx+1/2}^{G,n} - f_{nx+1/2}^{NG,m+1/2}) - (f_{1/2}^{G,n} - f_{1/2}^{NG,m+1/2})), \tag{4.8a}$$

with $m = M^h$.

In our method, the error in the total amount of mass has a different form. Set $w_h = \Phi_h^n(u_h^{(m)})$ and $u_h^{(m+1)} = \mathbf{P}(u_h^{(m)})w_h$; then

$$\begin{aligned}\sum_{i=1}^{nx} u_i^{(m+1)} \Delta x_i &= \sum_{i=1}^{nx} w_i\ \Delta x_i + \sum_{i=1}^{nx} (u_i^{(m+1)} - w_i)\Delta x_i \\ &= \sum_{i=1}^{nx} u_i^{n-1} \Delta x - \Delta t^{n-1}(f_{nx+1/2}^{NG,m+1/2} - f_{1/2}^{NG,m+1/2}) \\ &\quad + \sum_{i=1}^{nx} (u_i^{(m+1)} - w_i)\Delta x_i\end{aligned}$$

where, of course $f^{NG} = f^{NG}(u_h^{(m)}, w_h)$. This implies that the error in the total amount of mass produced by our method is the sum of a term similar to (4.8a) and the following:

$$\sum_{i=1}^{nx}(u_i^{(m+1)} - w_i)\Delta x_i = \int_0^1 (\mathbf{P}(u_h^{(m)}) - Id)w_h(x)\, dx \ . \tag{4.8b}$$

The term (4.8a) is due to the linearization procedure, and the term (4.8b) is only due to the projection $\mathbf{P}(u_h^{(m)})$. In other words, the price that we have to pay for stability is an increase in the loss of mass.

Nevertheless, we must point out that if $w_h = u_h^{(m)}$, then $\mathbf{P}(u_h^{(m)})u_h^{(m)} = u_h^{(m)}$ and the term (4.8b) is equal to zero. Moreover, if $\mathrm{del}(u_h^{(m)}) = \Phi_h^n(u_h^{(m)}) - u_h^{(m)} = w_h - u_h^{(m)}$ is small enough, then $\mathbf{P}(u_h^{(m)})w_h = w_h$ and (4.8b) vanishes. If this is the case, the inner iteration of our method reduces simply to a Newtonian one. In fact, this happens when Δt^{n-1} is small enough. Thus, our method can be considered as an extension of Newton's method. When Δt^{n-1} is big, the term (4.8b) is no longer equal to zero, and indeed it can be very big. If our method converges we expect the term (4.8b) to go to zero as the number of iterations m increases. Our numerical experience indicates that this is always the case, and that once (4.8b) becomes zero it stays equal to zero, so that the method coincides with Newton's method after a certain number of iterations. Once this happens, convergence is achieved very quickly, but it is extremely inefficient to ask that the term (4.8b) be equal to zero in order to stop the iterations. It suffices to stop the inner iterations when (4.8b) is "small enough." We consider this problem in the next section.

5. Testing the stability of the TVD–projection method. In this section we test numerically the stability properties of our method on the same six test problems on which the method (2.2), (2.5) has been tested. Our stopping criterion is based on a control of the amount of mass lost by the use of our projection. After defining it we calculate the quantities Λ defined by (3.3), and we compare them with the Λ obtained for Newton's method; Δx has been taken to be equal to 0.001, as in §3.3.

5.1. The stopping criterion. To define a stopping criterion for (4.7) is to define M^n. We define such a quantity as follows:

$$M^n = \min\{m : \mathrm{LM}(u_h^{(m)}) > \epsilon\} \ , \tag{5.1a}$$

where the loss of mass, LM, is defined by

$$\mathrm{LM}(u_h^{(m)}) = \sum_{i=1}^{nx} |(\mathbf{P}(u_h^{(m)}) - I_d)\Phi_h^n(m_h^{(m)}))_i|\, \Delta x_i \ . \tag{5.1b}$$

In our numerical experiences we shall take ϵ as follows:

$$\epsilon = \Delta t \cdot (\Delta x)^{j/2}, \qquad j = 1, 2, 3, 4. \tag{5.2}$$

Roughly speaking at each timestep we are allowing ourselves to lose an amount of mass equal to ϵ ; thus, in nt iterations we allowed ourselves to lose a total amount of mass equal to $T \cdot (\Delta x)^{j/2}$. Taking into account that the error behaves as does $C_0(\Delta t)^\alpha$ for $\alpha \in (0.5, 1)$, and that $\Delta t \gg \Delta x$, we consider that this is a reasonable criterion.

5.2. The numerical results. Our numerical results are shown in Tables 8–11.

TABLE 8

Values of Λ *for the* TVD*-projection method. The value* j *is associated with the stopping criterion* (5.1) *by* (5.2). (*Compare with values in Table* 3.)

Test problem	$j = 1$	$j = 2$	$j = 3$	$j = 4$
1	75	75	150	150
2	50	75	150	150
3	50	50	75	75
4	25	25	25	25
5	25	25	25	25

TABLE 9

The total number of iterations nt, *i.e., the sum over the time-steps of the number of inner iterations.* (*Compare with Table* 4 .)

Test problem	$j = 1$	$j = 2$	$j = 3$	$j = 4$
1	583	618	727	728
2	648	769	926	927
3	761	780	889	889
4	196	394	645	645
5	365	369	1465	1465

TABLE 10

The values of $10^3 \times \|u_{(T)} - u_h(T)\|_{L^1(0,1)}$, *with* $\Delta t = \Lambda \Delta x$.

Test problem	$j = 1$	$j = 2$	$j = 3$	$j = 4$
1	64.29	71.75	115.69	115.69
2	50.43	73.71	122.14	122.14
3	50.16	48.33	80.06	80.06
4	38.54	55.26	49.69	49.69
5	96.95	96.42	93.00	93.00

TABLE 11
The values of $10^4 \times$ *eff defined by* (3.4).

Test problem	$j = 1$	$j = 2$	$j = 3$	$j = 4$
1	0.27	0.23	0.12	0.12
2	0.31	0.18	0.09	0.09
3	0.26	0.27	0.14	0.14
4	1.32	0.46	0.31	0.31
5	0.28	0.28	0.07	0.07

Note that $T = 0.3$ for problems 1, 2, and 3, and that $T = 0.05$ for problems 4 and 5. The maximum value $\Lambda_{\max}$ for Λ is 150 for problems 1, 2, and 3 and 25 for problems 4 and 5, because, by (3.3):

$$\Lambda_{\max} = \Delta t_{\max}/\Delta x = \frac{T}{2}/\Delta x = 500\,T.$$

From these results we may conclude that the stopping criterion (5.1) works well, and works better for $j = 1$; see the errors and efficiency in Tables 10 and 11, respectively. Note that the results obtained with $j = 3$ and 4 are essentially the same.

By comparing the values of Λ for the Newtonian iterations (2.5), Table 3, and those for our method, Table 8, we see that our Λ are at least two orders of magnitude larger than those of the Newtonian iteration. Thus, we can say that the use of the TVD–projection enormously enhances the stability of Newton's iterations.

6. Conclusion. In this paper we have introduced a very simple and locally defined projection, the TVD (total variation diminishing)–projection, and we have shown how to use it to define an always–stable extension of Newton's method, applied to solve implicit schemes for the scalar conservation law (1.1). After proving that the TVD–projection makes the inner iterations $L^\infty \cap BV$–stable, we tested our method on Godunov's implicit scheme (1.1). Our numerical results show that the timestep sizes allowed by our method are at least two orders of magnitude larger than those allowed by standard Newtonian iterations.

Because first–order schemes are not very efficient, we are planning to apply this technique for solving more accurate implicit schemes. In particular, we are interested in solving the implicit finite element quasimonotone schemes recently introduced by one of the authors [3b], [3c]. These schemes are total variation diminishing in the means (TVDM), and seem to be a suitable application of our TVD–projection method.

REFERENCES

[1] C. BARDOS, A. Y. LEROUX, AND J. C. NÉDÉLEC, *First order quasilinear equations with boundary conditions*, Comm. Partial Differential Equations **4** (1979), 1017–1034.

[2] Y. BRENIER AND S. OSHER, *Approximate Riemann solvers and numerical flux functions*, ICASE-NASA Langley Center Report No. 84-63, Hampton, Virginia, 1984.

[3a] B. COCKBURN, *The Quasi-monotone schemes for scalar conservation laws, Part* I, Institute for Mathematics and its Applications, Preprint Series 263, University of Minnesota, Minneapolis, Minnesota, 1986.

[3b] ———, *The Quasi-monotone schemes for scalar conservation laws, Part* II, Institute for Mathematics and its Applications, Preprint Series 268, University of Minnesota, Minneapolis, Minnesota, 1986.

[3c] ———, *The Quasi-monotone schemes for scalar conservation laws, Part* III, Institute for Mathematics and its Applications, Preprint Series 277, University of Minnesota, Minneapolis, Minnesota, 1986.

[4] J. DOUGLAS, JR. , *Simulation of a linear waterflood*, Proceedings of a Seminar held in Pavia, September–October 1979, Vol. II, Istituto Nazionale de Alta Matematica Franceso Severi, Rome, 1980.

[5] J. DOUGLAS, JR., R. P. KENDALL, AND M. F. WHEELER, *Long wave regularization of one-dimensional, two-phase, immiscible flow in porous media*, Finite Element Methods for Convection Dominated Flows, AMD–Vol. 34, American Society of Mechanical Engineers, New York, 1979, pp. 201-211.

[6] B.A. FRYXELL, P. R. WOODWARD, P. COLELLA, AND K. H. WINKLER, *An implicit–explicit hybrid method for Lagrangian hydrodynamics*, J. Comput. Phys. **63** (1986), 283–310.

[7] S. OSHER, *Riemann solvers, the entropy condition, and difference approximations*, SIAM J. Numer. Anal. **21** (1984), 217–235.

[8] L.B. RALL, Computational Solution of Nonlinear Operator Equations, John Wiley, New York, 1969.

[9] R. SANDERS, *On convergence of monotone finite difference schemes with variable spatial differencing*, Math. Comp. **40** (1983), 91–106.

[10] M. M. RAI AND S. R. CHAKRAVARTHY, *An implicit form for the Osher upwind scheme*, American Institute of Aeronautics and Astronautics **24** (1986), 735–743.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 274–297, March 1989
© 1989 Society for Industrial and Applied Mathematics
004

A STABILIZED GALERKIN METHOD FOR CONVECTION-DIFFUSION PROBLEMS*

P. P. N. DE GROEN† AND M. VAN VELDHUIZEN‡

Abstract. A new method for the solution of convection-dominated diffusion equations is described. The method is based on the idea that the symmetric (or Bubnov-) Galerkin finite-element discretization can be stabilized by a suitable extension of the test space. The resulting overdetermined system is solved in least-squares sense using the normal equations. Although the resulting discretization is derived by a *conforming* method and yields a *symmetric* system of equations, it shows a resemblance to the (nonconforming) streamline-upwind/Petrov-Galerkin method; the results of this new stabilized Galerkin method have a comparable accuracy.

Key words. convection, diffusion

AMS(MOS) subject classification. 65N30

1. Introduction. It is well known that the standard finite-element method for convection-diffusion problems with boundary layers or internal layers gives rise to spurious oscillations and bad results. This also holds true for central finite-difference schemes.

The first remedy against these oscillations is an upwind-type discretization of the convective term. The disadvantage of this technique is twofold. First, the order of accuracy drops to one. Second, the artificial diffusion introduced by the upwinding smooths the solution in all directions. In particular, this artificial diffusion gives too much smoothing in the direction perpendicular to the streamlines.

A natural improvement is the restriction of the upwinding to directions parallel to the streamlines. This so-called streamline upwinding gives better results, but the order of accuracy is still low.

The accuracy in a streamline upwind method can be enhanced by changing it to a streamline upwind/Petrov-Galerkin method. This is a nonconforming finite-element method, which gives a streamline upwinding in the discretization of the operator, plus a consistent treatment of the inhomogeneous term. This method is described in Brooks and Hughes [4]. We refer to this paper for a description of the streamline upwind/Petrov-Galerkin method, as well as for a more extensive introductory survey.

The aim of this paper is the description of another discretization method for convection-diffusion problems. In addition, we want to show that an implementation is feasible, and that it gives good results. To that end we give a detailed introduction to the new method in § 2. Its basis is an abstract observation in a Hilbert space. We explain how this applies to convection-diffusion problems. There is some resemblance to the H^{-1}-Galerkin method (cf. [3]). The new method modifies the standard finite-element discretization. It stabilizes it, and therefore we call it a stabilized Galerkin method. In § 2 we give a general description. However, this description involves solving a rather complicated set of linear equations. In order to show that the new method is feasible, we restrict ourselves to a particular case, namely bilinear finite elements on a rectangular subdivision of a rectangle. This particular case is studied in § 3. In this

* Received by the editors December 28, 1987; accepted for publication (in revised form) May 19, 1988.
† Department of Mathematics and Computer Science, Vrije Universiteit Brussel, Peinlaan 2, B-1050 Brussels, Belgium. Electronic mail address: Z00301@BBRBFU01.BITNET.

‡ Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam, P.O. Box 7161, 1007MC Amsterdam, the Netherlands. Electronic mail address: VELM@SARA.NL.BITNET.

section we describe a linear equation solver for the complicated set of discretized equations. In addition, we describe modifications of the method that lead to sparse (band-structured) sets of discretized equations. Numerical results are given in § 4. In all cases we compare the results obtained by the stabilized Galerkin method with the results from the streamline upwind/Petrov-Galerkin method as described in Brooks and Hughes [4]. The results are not much different, but the stabilized Galerkin method seems somewhat more accurate in the case of Dirichlet boundary conditions.

2. The stabilized Galerkin method. In this section we give a derivation of the stabilized Galerkin method, a new method for convection-diffusion problems. First, we give a derivation for one-dimensional problems. By means of some operator theory we analyze the finite-element method for a simple convection-diffusion problem. As a result of this analysis we are able to indicate a new method. This method, based on a formulation in a Hilbert space, is then described for convection-diffusion problems in two space dimensions. The resulting method is not an upwind-type method.

To begin, we consider a simple one-dimensional problem on $[0, 1]$:

$$-\varepsilon\ddot{u}+\dot{u}=f(x), \qquad u(0)=u(1)=0, \tag{2.1}$$

where $\varepsilon>0$ is a small parameter. A weak form of (2.1) is given by

$$\varepsilon(\dot{u}, \dot{v})+(\dot{u}, v)=(f, v) \quad \forall v\in H_0^1([0, 1]), \tag{2.2}$$

with unknown $u\in H_0^1([0, 1])$. The $L^2([0, 1])$ inner product is denoted by $(\cdot, \cdot)$. In order to explain our idea, we modify this weak form by integration by parts to

$$\varepsilon(\dot{u}, \dot{v})-(u, \dot{v})=-(F, \dot{v}),$$

where F is a primitive of f. In this equation the projection of $\varepsilon\dot{u}-u+F$ onto the space of derivatives of $H_0^1([0, 1])$-functions is set to zero. The space of derivatives of functions in $H_0^1([0, 1])$ is $L^2([0, 1])$ minus the nonvanishing constant functions on $[0, 1]$. The orthogonal L^2-projection onto this space is then the orthogonal L^2-projection $I-M$, where Mv denotes the mean value on $[0, 1]$ of v, i.e., the orthogonal projection of v onto the constant functions on $[0, 1]$. Hence, the boundary value problem (2.1) is now rewritten as

$$\varepsilon\dot{u}-(I-M)u=-(I-M)F. \tag{2.3}$$

Our idea is best explained by considering a finite-element approximation based on the weak form (2.2). We choose the finite-element space of roof functions satisfying the homogeneous boundary conditions at the endpoints, and we consider these functions on an equidistant mesh with mesh-spacing $h=1/N$. Let S^h denote this space. The finite-element equations based on (2.2) then read as follows:

$$\varepsilon(\dot{u}^h, \dot{v}^h)+(\dot{u}^h, v^h)=(f, v^h) \quad \forall v^h\in S^h,$$

with unknown $u^h\in S^h$. Observe that the space of derivatives of functions in S^h is just the space of piecewise constant functions with vanishing mean value. Using this, we easily transform the above finite-element equations to

$$\varepsilon\dot{u}^h-\Pi_h(I-M)u^h=-\Pi_h(I-M)F, \tag{2.4}$$

where Π_h is the orthogonal $L^2([0, 1])$-projection onto the piecewise constant functions (with respect to the mesh).

It is well known that the solution of the model (2.1) usually has a boundary-layer behavior at $x=0$. We might approximate this boundary-layer behavior by asymptotic

methods, as in Flaherty and O'Malley [7]. However, for two-dimensional problems this technique hardly seems promising in view of the complicated asymptotic expansions (cf. Eckhaus [6]), so we stick to finite-element or finite-difference methods.

The roof functions on an equidistant mesh are not capable of approximating a boundary layer, at least not if the width of the boundary layer is small in comparison to the mesh spacing. Rather, a boundary layer is approximated by an oscillatory component. We shall indicate the source of this problem by analyzing the projection method (2.4). Consider an oscillatory element $u^h \in S^h$ given by $u^h(jh) = (-1)^j$, $j = 1, 2, \cdots, N-1$. In the $L^2([0,1])$-norm we find by direct computation $\|(I-M)u^h\| \geqq \frac{1}{6}$. Now we compute the norm of the corresponding element $\Pi_h(I-M)u^h$ in (2.4). We find $\|\Pi_h(I-M)u^h\|^2 \leqq \frac{3}{2}h$. For small $\varepsilon > 0$ this means that the projection Π_h onto the subspace of first derivatives of elements from S^h (almost) annihilates an element of the form $\varepsilon \dot{u}^h - (I-M)u^h$, $u^h \in S^h$. The general convergence theory of projection methods deals with this situation (cf. Krasnosel'skii et al. [12]). It just means that the method is not stable (see [12] for a rigorous statement and proof). This confirms what we already know: symmetric finite elements on an equidistant mesh are bad for convection-diffusion problems.

The analysis shows ways of improving the discretization. We might change the projection Π_h (e.g., by choosing a suitable quadrature rule for the inner product evaluation). The one-dimensional upwind scheme and some generalizations may be derived and analyzed in this way (cf. van Veldhuizen [16]). We might also modify the space of test functions (in the above example the space of test functions is identical to the space in which the solution is sought). An example is the method of Hemker [11], de Groen and Hemker [8]. The method described by Barrett and Morton [2] may be put in this class. Another example is the finite-element upwind method of Heinrich et al. [10]. For the example considered here, this latter method achieves the upwinding by choosing as test functions a subspace of dimension $N-1$ from the piecewise quadratic functions. The quadratic part of the test functions is then used to suppress the unwanted oscillatory component.

The streamline upwind/Petrov-Galerkin method described by Brooks and Hughes [4] does something similar. This method perturbs the finite-element method by a streamline unwinding and a corresponding correction in the right-hand side which preserves the order of accuracy. However, the method has nonconforming aspects; the usual Sobolev spaces are therefore not appropriate for an analysis of these methods. More complicated spaces are necessary.

Let us now see why quadratic functions are able to suppress the oscillatory behavior. Clearly, the oscillatory component shows up in the finite-element method approximation since the projection $\Pi_h(I-M)$ (almost) annihilates such a component. If we project in such a way, or on such a space, that oscillatory components are not annihilated, then such components do not show up (unless the inhomogeneous term contains them). On the subinterval (x_j, x_{j+1}) the oscillatory component appears as a multiple of the linear function $x - \frac{1}{2}(x_j + x_{j+1})$. This function is the derivative of the parabola $\frac{1}{2}(x-x_j)(x-x_{j+1})$. So we should project onto the derivatives of these parabolas. In the finite-element upwinding of Heinrich et al. [10], one achieves it by perturbing the linear test functions by parabolas. We propose a new method in which we *add* the parabolas to the space of roof functions. Effectively, we then use the piecewise quadratic functions in $H_0^1([0,1])$ as test functions. But there is a different problem now: there are more equations (the dimension of the space of piecewise quadratic polynomials in $H_0^1([0,1])$) than unknowns (the dimension of S^h). That is, we have an overdetermined system of equations if we project onto the first derivatives of piecewise quadratic

functions. However, by looking for a least-squares solution we can still take advantage of the better projection properties.

The method outlined in the above paragraph looks rather complicated. However, some simplifications are possible. For instance, the weak form (2.4) is not required in the formation of the problem. We use it in explaining the method only. Also, the projection onto the larger space of piecewise quadratic functions can be described in terms of the more familiar weak formulation (2.2). For one-dimensional problems—but not for problems in two space dimensions—the new method as sketched above turns out to be equivalent to a method described by Barrett and Morton [2] (cf. Odenhoven and van Veldhuizen [14]). However, we shall not pursue the one-dimensional example, but rather turn to two-dimensional problems.

We now consider a typical convection-diffusion equation in two space dimensions

$$-\varepsilon \Delta u + pu_x + qu_y = f \tag{2.5}$$

on a domain $\Omega \in \mathbb{R}^2$ with suitable boundary conditions. From asymptotic results (cf. Eckhaus [6]), it is well known that the problem (2.5) with Dirichlet boundary conditions is well posed uniformly in the small parameter ε. We must be careful with Neumann boundary conditions, in relation to the vector $(p, q)^T$ (cf. Eckhaus [6]). Those parts of the boundary on which we may safely prescribe a Neumann condition will be called outflow boundaries (even if there is no flow). The remaining part will be called an inflow boundary. On the inflow boundary we prescribe function values (Dirichlet boundary condition). In this introductory description we shall assume homogeneous boundary conditions. Also, the velocity vector $\mathbf{v} = (p, q)^T$ is assumed constant. All these simplifications are not essential. Typically, the scalar parameter ε is much smaller than the velocity $\|\mathbf{v}\|$. Taking into account the boundary conditions, we have that a weak formulation of (2.5) is given by

$$\varepsilon[(u_x, v_x) + (u_y, v_y)] + (pu_x, v) + (qu_y, v) = (f, v) \quad \forall v \in H_E(\Omega), \tag{2.6}$$

where $H_E(\Omega)$ is a linear subspace of the Sobolev space $H^1(\Omega)$ and $(\cdot, \cdot)$ denotes the $L^2(\Omega)$ inner product. In fact, $H_E(\Omega)$ consists of the functions in $H^1(\Omega)$ vanishing on the Dirichlet boundary. The bilinear form in the left-hand side of (2.6) will be denoted by $a_\varepsilon(\cdot, \cdot)$.

As in the one-dimensional example above, we want to write this method as an operator equation. To that end we apply Green's formula to the first-order derivative terms. In view of the boundary conditions, we then find $(pu_x, v) = -(pu, v_x)$ and $(qu_y, v) = -(qu, v_y)$. We also want to write the (f, v)-term as an inner product involving v_x and v_y. To that end let G denote the inverse of the Laplacian (with homogeneous boundary conditions as for (2.5)) on Ω. Then, from the weak formulation of the equation $-\Delta u = f$ we obtain

$$(f, v) = ([Gf]_x, v_x) + ([Gf]_y, v_y) \quad \forall v \in H_E(\Omega). \tag{2.7}$$

With $\langle \cdot, \cdot \rangle$ the usual inner product in $L^2(\Omega) \times L^2(\Omega)$, we see that (2.7) is equivalent to $(f, v) = \langle \text{grad } Gf, \text{grad } v \rangle$. Using these formulas in (2.6) we find as weak formulation

$$\langle \varepsilon \text{ grad } u - u\mathbf{v} - \text{grad } Gf, \text{grad } v \rangle = 0 \quad \forall v \in H_E(\Omega), \tag{2.8}$$

where $\mathbf{v} = (p, q)^T$ denotes the velocity field. Let Π_E denote the orthogonal $L^2(\Omega) \times L^2(\Omega)$ projection onto the linear subspace of all gradients of $H_E(\Omega)$-functions. In view of the definition of G and $u \in H_E(\Omega)$ we then find for (2.6) the equivalent formulation

$$\varepsilon \text{ grad } u - \Pi_E u\mathbf{v} = \text{grad } Gf. \tag{2.9}$$

This is the two-dimensional analogue of (2.3).

Let S_E^h denote a (parametrized) family of finite-dimensional linear subspaces of $H_E(\Omega)$. A finite-element approximation $u^h \in S_E^h$ of the solution u of (2.6) is then determined by the equations

$$\varepsilon[(u_x^h, v_x^h)+(u_y^h, v_y^h)]+(pu_x^h, v^h)+(qu_y^h, v^h)=(f, v^h) \quad \forall v^h \in S_E^h. \tag{2.10}$$

This finite-element method may be written as a projection method related to the operator (2.8). By repeating the arguments we find

$$\varepsilon \text{ grad } u^h - \Pi_E^h u\mathbf{v} = \Pi_E^h \text{ grad } Gf,$$

where Π_E^h denotes the orthogonal $L^2(\Omega)\times L^2(\Omega)$ projection onto the linear subspace of gradients of functions in S_E^h. In this section we prefer the equivalent formulation

$$\Pi_E^h[\varepsilon \text{ grad } u^h - \Pi_E u^h\mathbf{v}] = \Pi_E^h \text{ grad } Gf. \tag{2.11}$$

This formula shows how the finite-element method may be written as a projection method in the sense of Krasnosel'skii et al. [12]. The interpretation is not new (cf. Babuška and Aziz [1, pp. 186–189]).

A necessary condition for the convergence of a projection method is its stability. For $\varepsilon \neq 0$ the finite-element method (2.7) is stable in the sense of Krasnosel'skii et al. [12] if

$$\frac{\|\Pi_E^h[\varepsilon \text{ grad } v^h - \Pi_E v^h\mathbf{v}\|}{\|\varepsilon \text{ grad } v^h - \Pi_E v^h\mathbf{v}\|} \geqq \tau_\varepsilon > 0 \quad \forall v^h \in S_E^h \tag{2.12}$$

for all subspaces of sufficiently large dimension from the family S_E^h. For the one-dimensional case considered above $\tau_\varepsilon \to 0$ for $\varepsilon \to 0$ and $h \to 0$. This also happens for two-dimensional problems such as (2.5). We give an example.

Example 2.1. The unit square $[0, 1]\times[0, 1]$ is subdivided into squares by using an equidistant mesh in both directions with stepsize $h = 1/N$. On this mesh we choose the finite-element space of piecewise bilinear functions. We impose homogeneous Dirichlet boundary conditions. We want to compute τ_ε as defined above. For easier presentation we assume $p = 1$ and $q = 0$.

We start with $\Pi_E^h v^h\mathbf{v}$. Let N be odd. In the nodal finite-element basis functions this projection, with respect to the basis of gradients, is given by a skew-symmetric matrix. In fact, it represents just the (pu_x^h, v^h)-term in (2.10). This skew-symmetric matrix has an eigenvalue zero if N is odd. Choose v^h as the corresponding eigenvector normalized in some way. This choice implies $\Pi_E^h v^h\mathbf{v} = 0$.

The definition of the projection Π_E implies that $\Pi_E v^h\mathbf{v}$ is the gradient of the solution z of the Laplace equation $\Delta z = -v_x^h$. Clearly, for the eigenvector v^h chosen above we have $\Pi_E v^h\mathbf{v} \neq 0$, since $v_x^h \neq 0$. Hence, for this choice and $\varepsilon \to 0$ we find

$$\frac{\|\Pi_E^h[\varepsilon \text{ grad } v^h - \Pi_E v^h\mathbf{v}]\|}{\|\varepsilon \text{ grad } v^h - \Pi_E v^h\mathbf{v}\|} \to 0.$$

This implies $\tau_\varepsilon \to 0$ for $\varepsilon \to 0$. If N is even the computations are much more complicated, but then again we find $\tau_\varepsilon \to 0$ for $\varepsilon \to 0$ and $h \to 0$. □

This example shows that the bilinear elements on a regular mesh result in an unstable method for convection-diffusion problems (with ε much smaller than a typical mesh spacing). Clearly, the problem is similar to the problem in one space dimension. The space onto which the projection Π_E^h projects is too small. The remedy we propose is very much as in the one-dimensional case. Just extend the space onto which $v^h\mathbf{v}$ is projected in such a way that the method becomes stable for small $\varepsilon > 0$. Also, the extension should be chosen in such a way that the finite-element formalism (2.10) can still be used rather than (2.11).

In the *new method* we propose, we choose as the solution space some standard finite-element space S_E^h. For testing, we supplement S_E^h to a larger space T_E^h by adding a collection test functions that can suppress unwanted oscillatory solutions. So we must determine the solution $u^h \in S_E^h$ from the overdetermined set of equations (cf. (2.10))

$$\varepsilon[(u_x^h, v_x^h)+(u_y^h, v_y^h)]+(pu_x^h, v^h)+(qu_y^h, v^h)=(f, v^h) \tag{2.13}$$

for all $v^h \in T_E^h$. Similarly to (2.10)-(2.11), this method can be written as a projection method:

$$\Pi_T^h[\varepsilon \operatorname{grad} u^h - \Pi_E u^h p] = \Pi_T^h \operatorname{grad} Gf, \tag{2.14}$$

where Π_T^h is the $L^2(\Omega)\times L^2(\Omega)$ orthogonal projection onto the linear space grad T_E^h. Obviously, since u^h in the left-hand side is chosen from S_E^h, which is of a smaller dimension than grad T_E^h into which the right-hand side is projected, this equation must be understood in a least-squares sense. However, if S_E^h contains a good approximation of the exact solution of problem (2.5), the residual $\mathbf{r}^h$ of (2.14) will become small in the $L^2(\Omega)\times L^2(\Omega)$-norm. This indicates that the desired solution of (2.13) or (2.14) is the one that minimizes the residual $\mathbf{r}^h$ of (2.14) in the $L^2(\Omega)\times L^2(\Omega)$-norm. We shall indicate how to do this without the explicit use of (2.14). The equivalent equations (2.13) suffice.

Choose a basis $\{v_1, \cdots, v_s\}$ in S_E^h. Extend it to a basis $\{v_1, \cdots, v_t\}$ in T_E^h. Here dim $S_E^h = s$ and dim $T_E^h = t$. With respect to these bases, the overdetermined equations (2.13) may be written as follows:

$$A\vec{u} = \vec{b}, \tag{2.15}$$

where $\vec{u} \in \mathbb{R}^s$ is the vector of coordinates of $u^h \in S_E^h$, and $\vec{b} \in \mathbb{R}^t$ is the vector of inner products (f, v_i^h). The entries of the matrix A are given by

$$A_{ij} = a_\varepsilon(v_j, v_i), \qquad i = 1 \cdots t, \quad j = 1 \cdots s. \tag{2.16}$$

Observe that the $L^2(\Omega)\times L^2(\Omega)$ inner product in grad T_E^h induces an inner product in $\mathbb{R}^t$ through the basis elements chosen. Let $\mathbf{M}$ denote the $t \times t$ matrix of inner products of basis elements grad v_i in grad T_E^h. Thus,

$$\mathbf{M}_{ij} = \langle \operatorname{grad} v_i, \operatorname{grad} v_j \rangle.$$

We want to minimize the residual $\mathbf{r}^h$ in the $L^2(\Omega)\times L^2(\Omega)$ inner-product norm. Let us expand $\mathbf{r}^h$ in the (nonorthogonal) basis $\{\operatorname{grad} v_i\}$ in grad T_E^h. We have the formula

$$\langle \operatorname{grad} \mathbf{r}^h, \operatorname{grad} \mathbf{r}^h \rangle = \sum_{1 \le i,j \le t} \langle \operatorname{grad} \mathbf{r}^h, \operatorname{grad} v_i \rangle (\mathbf{M}^{-1})_{i,j} \langle \operatorname{grad} \mathbf{r}^h, \operatorname{grad} v_j \rangle.$$

The inner products in the right-hand side are already familiar. We have

$$\langle \operatorname{grad} \mathbf{r}^h, \operatorname{grad} v_i \rangle = a_\varepsilon(u^h, v_i) - (f, v_i).$$

Since u^h may be written as a linear combination of basis elements v_i, $i = 1 \cdots s$, we find from (2.16) the equality

$$\langle \operatorname{grad} \mathbf{r}^h, \operatorname{grad} \mathbf{r}^h \rangle = (A\vec{u} - \vec{b})^T \mathbf{M}^{-1} (A\vec{u} - \vec{b}).$$

Hence, the least-squares formulation consistent with (2.14) (without explicit reference to the projections) is given by minimization in $\mathbb{R}^s$ of the quadratic form

$$\vec{x} \to (A\vec{x} - \vec{b})^T \mathbf{M}^{-1} (A\vec{x} - \vec{b}) \tag{2.17}$$

or by solving the corresponding normal equations

$$A^T \mathbf{M}^{-1} A\vec{x} = A^T \mathbf{M}^{-1} \vec{b}. \tag{2.18}$$

This is the formulation required for a practical implementation. Observe that the matrix of the normal equations is invariant under multiplication of basis elements by scalar factors. This kind of scaling-invariance will play a part in § 3.5. The above formulation uses only the bilinear form $a_\varepsilon(\cdot,\cdot)$ and not the formula (2.14). Observe that the matrix $\mathbf{M}$ is just the finite-element matrix of the Laplace operator $-\Delta$ with respect to the finite-element space T_E^h on the basis chosen in T_E^h.

A key question is the choice of the extended finite-element space T_E^h. If we choose this space too small it may fail to cure the stability problem. On the other hand, too large a space T_E^h may result in inefficient algorithms. In view of multigrid techniques, a possibility is the choice $T_E^h = S_E^{h'}$, where $h' < h$. In an unpublished report (cf. [17]) it has been shown that this choice gives stable schemes for two-dimensional convection-diffusion problems (in the sense of Krasnosel'skii et al. [12]), provided $h' < \sigma h$ for some $\sigma \in (0, 1)$ depending only on the problem, not on the mesh spacing h. This result proves the feasibility of the algorithm proposed in this paper. However, in practice we extend the space S_E^h by adding higher-degree polynomials. In [9, § 2, p. 154] we give an example, in which we can show, that such a type of stabilization prevents the formation of large spurious oscillations.

A second observation is in order. The quadratic form (2.17) is not a simple one. For instance, the matrix $\mathbf{M}^{-1}$ is usually not sparse. In a later section we shall return to this computational problem, and we shall solve it by replacing $\mathbf{M}$ with a simpler matrix.

3. A case study. In this section we give a detailed description of the new method in case of bilinear elements on rectangles.

3.1. Bilinear elements on rectangles. On the unit square $\Omega := [0, 1] \times [0, 1]$ we consider the convection-diffusion equation

$$-\varepsilon \Delta u + p u_x + q u_y = f. \tag{3.1}$$

The boundary Γ of Ω consists of two parts, Γ_D and Γ_N, with Dirichlet boundary conditions on Γ_D, and homogeneous Neumann boundary conditions on Γ_N,

$$u(x, y) = g(x, y), \qquad (x, y) \in \Gamma_D, \tag{3.2a}$$

$$\frac{\partial}{\partial n} u(x, y) = 0, \qquad (x, y) \in \Gamma_N. \tag{3.2b}$$

We always assume that Γ_N is an outflow boundary. Observe that Γ_N may be empty, i.e., Dirichlet conditions only. The bilinear form associated with this problem is

$$a_\varepsilon(u, v) = \varepsilon[(u_x, v_x) + (u_y, v_y)] + (p u_x, v) + (q u_y, v) \tag{3.3}$$

(cf. (2.6) and (2.10)). Here $u \in H^1(\Omega)$ must satisfy the essential boundary condition (3.2a). The test functions v must belong to $H_E(\Omega)$, the subspace of the Sobolev space $H^1(\Omega)$ of all functions vanishing on Γ_D. The weak formulation of the above boundary value problem is given by

$$a_\varepsilon(u, v) = (f, v) \quad \forall v \in H_E(\Omega). \tag{3.4}$$

The velocity vector $\mathbf{v} = (p, q)^T$ may depend on the position in the rectangle Ω.

We now describe the finite-element discretization (cf. Ciarlet [5]). The subdivision of the unit square Ω in rectangles is obtained by taking the product of the divisions

$$\{x_0 := 0, x_1, \cdots, x_n := 1\}, \qquad \{y_0 := 0, y_1, \cdots, y_m := 1\}$$

with meshsizes $h_j := x_j - x_{j-1}$ and $k_j := y_j - y_{j-1}$ in x- and y-direction, respectively. The finite-element space S^h is chosen as the space of bilinear functions on these rectangles. Let S_E^h denote the linear space $S^h \cap H_E(\Omega)$. In S_E^h we choose the basis of nodal finite elements ϕ_i. Unless stated otherwise, the ϕ_i are numbered in the usual manner, in which the index corresponding to the x-direction increases most rapidly. The total number of nodes, i.e., the dimension of S_E^h is again denoted by s. Hence $s = (n-1)(m-1)$ if Γ_N is empty. The extensions, the basis elements of the difference of S_E^h and T_E^h, are defined on $[x_{i-1}, x_i] \times [y_{j-1}, y_j]$ as the local biparabolic functions

$$\psi_{i,j}(x, y) = \xi(1-\xi)\eta(1-\eta), \quad \xi = \frac{x - x_{i-1}}{h_i}, \quad \eta = \frac{y - y_{j-1}}{k_j}. \tag{3.5}$$

Outside this rectangle we put $\psi_{i,j}(x, y) = 0$. Clearly, the $\psi_{i,j}$ belong to $H_E(\Omega)$. Observe $t = \dim T_E^h = \dim S_E^h + n \times m$. The overdetermined equations (2.13) now read

$$a_\varepsilon(u^h, \phi_i) = (f, \phi_i), \qquad i = 1, \cdots, s, \tag{3.6a}$$

$$a_\varepsilon(u^h, \psi_{i,j}) = (f, \psi_{i,j}), \qquad i = 1, \cdots, n, \quad j = 1, \cdots, m. \tag{3.6b}$$

The finite-element approximation u^h is an element of the manifold $\tilde{u}^h + S_E^h$, where $\tilde{u}^h \in S^h$ interpolates the boundary condition on Γ_D. In the matrix-vector notation of § 2 these equations may be written as

$$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \vec{u} = \begin{bmatrix} \vec{b}_1 \\ \vec{b}_2 \end{bmatrix}, \tag{3.7}$$

where $\vec{u}, \vec{b}_1 \in \mathbb{R}^s$, $\vec{b}_2 \in \mathbb{R}^{t-s}$. Again, the first s equations $A_1\vec{u} = \vec{b}_1$ form the standard finite-element discretization with bilinears for the problem (3.1). Observe that the matrix A_1 of this first part is a banded matrix with nine nonvanishing (co)diagonals, provided the usual ordering on a rectangular grid is being used.

The stabilizing equations corresponding to the extensions are given by (3.6b). The matrix A_2 corresponding to these equations has s columns and $n \times m$ rows. Since each $\psi_{i,j}$ interacts with at most four nodal elements ϕ_k at the corners of the corresponding rectangle, it follows that at most four elements per row are different from zero. Also, due to the subdivision in rectangular elements it is easily seen that for all possible indices

$$([\phi_k]_x, [\psi_{i,j}]_x) + ([\phi_k]_y, [\psi_{i,j}]_y) = 0. \tag{3.8}$$

Consequently, the last $n \times m$ equations (3.6b) only involve the convection terms pu_x and qu_y. The element matrix with constant p and constant q for the matrix A_2 is given by

$$\frac{pk}{72}\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} + \frac{qh}{72}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}. \tag{3.9}$$

Since supp $(\psi_{i,j})$, the support of $\psi_{i,j}$, is just one rectangle in the subdivision, it follows that these element matrices correspond directly to difference stencils for the matrix A_2. Clearly, these stencils connect adjacent nodes. Such stencils do not annihilate oscillatory solutions, and this suggests why the addition of these extended equations will suppress oscillatory components.

The overdetermined equations (3.6a), (3.6b) are to be solved in an $L^2(\Omega) \times L^2(\Omega)$ least-squares sense (cf. § 2). Due to (3.8) the stiffness matrix $\mathbf{M}$, (cf. § 2) reduces to

$$\mathbf{M} = \begin{bmatrix} M_1 & 0 \\ 0 & M_2 \end{bmatrix}, \tag{3.10}$$

where M_1 is the stiffness matrix corresponding to the finite-element space S_E^h and the bilinear form $\langle \text{grad } u, \text{grad } v\rangle$, while M_2 is a diagonal matrix with entries $\langle \text{grad } \psi_{i,j}, \text{grad } \psi_{i,j}\rangle$. The diagonal element corresponding to the extension element $\psi_{i,j}$, i.e., to a rectangle with sides h_i, k_j is given by

$$\frac{1}{90}\left(\frac{h_i}{k_j}+\frac{k_j}{h_i}\right). \tag{3.11}$$

In our implementation the velocity vector $\mathbf{v}$ is replaced on each rectangle of the subdivision by its value in the middle of that rectangle; then we use exact integration in the evaluation of the bilinear form $a_\varepsilon(\cdot,\cdot)$. The right-hand side is evaluated by using product quadrature (Simpson's rule).

The normal equations (2.18) for the bilinear elements reduce to

$$[A_1^T M_1^{-1} A_1 + A_2^T M_2^{-1} A_2]\vec{x} = A_1^T M_1^{-1}\vec{b}_1 + A_2^T M_2^{-1}\vec{b}_2. \tag{3.12}$$

In § 3.4 the linear equation solver for these equations shall be described.

It is interesting to study the contributions $A_2^T M_2^{-1} A_2$ of the extension to the normal equations. On an equidistant mesh and interpreted as a finite-difference operator on the nodes, its stencil is the nine-point rule

$$\frac{5}{288}\frac{hk}{h^2+k^2} \times\left\{p^2k^2\begin{bmatrix}-1 & 2 & -1\\ -2 & 4 & -2\\ -1 & 2 & -1\end{bmatrix}+q^2h^2\begin{bmatrix}-1 & -2 & -1\\ 2 & 4 & 2\\ -1 & -2 & -1\end{bmatrix}+2qphk\begin{bmatrix}1 & 0 & -1\\ 0 & 0 & 0\\ -1 & 0 & 1\end{bmatrix}\right\}. \tag{3.13}$$

Apart from a constant factor this is a second-order symmetric (central) discretization of the partial differential operator

$$u \to \left(p\frac{\partial}{\partial x}+q\frac{\partial}{\partial y}\right)^2 u. \tag{3.14}$$

Clearly, this operator is a diffusion operator along the streamlines. This observation gives a nice interpretation of the method proposed in this paper. The new method consists of a set of normal equations—derived from the standard finite-element discretization—perturbed by an additional diffusion along the streamlines. The mathematical formulation of the new method enables a correct and consistent treatment of the inhomogeneous terms. In some respects the new method is similar to the "streamline upwind method" of Brooks and Hughes [4]. Brooks and Hughes add an artificial diffusion along streamlines to the standard finite-element equations. The artificial diffusion in the "streamline upwind method" is very similar to the contribution of the extension $A_2^T M_2^{-1} A_2$.

3.2. Lumping. Experiments have shown that the extension of the test space stabilizes the discretization as expected from the theoretical considerations mentioned above. In particular, the experiments indicate that the method seems to suppress all oscillations if $k|p| \sim h|q|$. This may be explained as follows. If $kp = hq$, (3.13) results in a stencil (nine-point formula) proportional to

$$\begin{bmatrix}0 & 0 & -1\\ 0 & 2 & 0\\ -1 & 0 & 0\end{bmatrix}. \tag{3.15a}$$

This stencil is a discrete diffusion operator along the diagonals in the rectangular grid. It does not annihilate a discrete oscillation in the horizontal or the vertical direction. Consequently, if the inhomogeneous term does not contain oscillatory components, then neither does the solution itself.

A second extreme situation occurs if the vector field $\mathbf{v}$ is parallel to one of the sides of the rectangle Ω. For instance, let $(p, q) = (1, 0)$. In this case (3.13) reduces to a stencil proportional to

$$\begin{bmatrix} -1 & 2 & -1 \\ -2 & 4 & -2 \\ -1 & 2 & -1 \end{bmatrix}. \tag{3.15b}$$

Its kernel contains oscillations in the vertical direction, e.g., $u(x_i, y_j) = (-1)^j$ is annihilated by this stencil. Hence these oscillations may show up in the solution, making this stencil useless. However, a small change in it cures the problem. Consider the striking difference between (3.15a) and (3.15b). The oblique difference (3.15a) uses a minimum of nonzero weights for the representation of a second derivative along a streamline; it has good properties, because it is so compact and does not work in a transversal direction. The other, (3.15b), uses three rows of weights for a second derivative in the horizontal direction, and these average out unwanted oscillations before they can be detected by the scheme. Since one row of three weights suffices for a horizontal second derivative, the cure is to lump the three rows onto the middle one, replacing (3.15b) with

$$\begin{bmatrix} 0 & 0 & 0 \\ -4 & 8 & -4 \\ 0 & 0 & 0 \end{bmatrix}. \tag{3.16}$$

This lumped scheme fully retains its stabilizing influence in the direction of the stream. In addition, it suppresses oscillatory components in the vertical direction.

The example shows that it may be necessary to improve the transversal properties of the scheme originating from $A_2^T M_2^{-1} A_2$ by lumping. In simple cases, regular mesh spacing, constant vector field $\mathbf{v}$, the lumping process can be done by hand. The matrix corresponding to the lumped scheme is again symmetric. However, in more general cases it is not so obvious how to do the lumping. Clearly, it seems very hard, if not impossible, to base the lumping process only on the matrix A_2. A more feasible way is to do it for the product $A_2^T M_2^{-1} A_2$. To get an idea how this can be done, we consider first the bilinear form $\langle \text{grad } u, \text{grad } v \rangle$.

The use of bilinear trial and test functions on an equidistant square mesh yields the nine-point rule for the Laplacian, while the more compact five-point rule is obtained if exact integration is replaced by the product trapezoidal rule on the elementary rectangles. Since this way of lumping also works for the first-order terms $pu_x + qu_y$, we can lump the matrix A_1 from a nine-diagonal matrix to a more compact five-diagonal one. This lumping not only diminishes the diffusive properties (which is less important for this component in the equation), but it also considerably diminishes the work to be done in a matrix-vector multiplication. This lumping process enhances an iterative solution procedure. *Henceforth, we shall assume that the matrices A_1 and M_1 have been lumped to a five-diagonal matrix,* unless explicitly stated otherwise.

A satisfactory lumping of $A_2^T M_2^{-1} A_2$ requires more than only a change of integration method. First we remark that there is no point in trying to lump A_2, since the underlying difference scheme is supported by only four nodes. Those nodes enter the formulae in a completely symmetric way. There is no possibility of shifting a part of

a weight from one node to another without seriously damaging the difference formula. So we turn to $A_2^T M_2^{-1} A_2$. For a general lumping strategy we consider an element of this matrix:

$$B_{i,j} := (A_2^T M_2^{-1} A_2)_{i,j} = \sum_{1 \leqq \alpha,\beta \leqq t} \frac{a_\varepsilon(\phi_i, \psi_{\alpha,\beta}) a_\varepsilon(\phi_j, \psi_{\alpha,\beta})}{(\text{grad } \psi_{\alpha,\beta}, \text{grad } \psi_{\alpha,\beta})}. \tag{3.17}$$

Since $\langle \text{grad } \phi_i, \text{grad } \psi_{\alpha,\beta} \rangle = 0$, we have

$$a_\varepsilon(\phi_i, \psi_{\alpha,\beta}) = (\mathbf{v} \cdot \text{grad } \phi_i, \psi_{\alpha,\beta}).$$

Because $\psi_{\alpha,\beta} - 1/36$ is orthogonal to all linear polynomials, we can write

$$\begin{aligned}(\mathbf{v} \cdot \text{grad } \phi_i, \psi_{\alpha,\beta})(\mathbf{v} \cdot \text{grad } \phi_j, \psi_{\alpha,\beta}) \\ = \frac{h_\alpha k_\beta}{36^2} (\mathbf{v} \cdot \text{grad } \phi_i)(\mathbf{v} \cdot \text{grad } \phi_j)|_{m_{\alpha,\beta}} + O(h_\alpha k_\beta (h_\alpha^2 + k_\beta^2)),\end{aligned} \tag{3.18}$$

where $m_{\alpha,\beta}$ is the center of the rectangle $(x_{\alpha-1}, x_\alpha) \times (y_{\beta-1}, y_\beta)$. The order term vanishes if $\mathbf{v}$ is constant or linear in x and y. Observe that up to the same order in h and k the formula may be interpreted as the evaluation of the inner product $(\mathbf{v} \cdot \text{grad } \phi_i, \mathbf{v} \cdot \text{grad } \phi_j)$ on the elementary rectangle by the midpoint rule. Hence, $B_{i,j}$ may be interpreted as the same inner product integrated over the whole domain (up to a small error). This offers a nice opportunity for lumping unwanted terms in $A_2^T M_2^{-1} A_2$ by changing the local integration method for this inner product, while symmetry of the resulting lumped matrix is guaranteed.

In order to determine which of the nine weights in the computational molecule resulting from $A_2^T M_2^{-1} A_2$ can be considered for lumping, we remark that in the generic case a second-order central difference formula for the operator (3.14) on a square mesh requires at least seven points arrayed in either of the two patterns:

$$\begin{array}{ccc} & * & * \\ * & * & * \\ * & * & \end{array} \quad \text{or} \quad \begin{array}{ccc} & * & * \\ * & * & * \\ & * & * \end{array}.$$

The first alternative is best when the vector (p, q) points into the North-East or South-West quadrant ($pq > 0$), and the second otherwise. Guided by this idea, we choose the terms in $A_2^T M_2^{-1} A_2$ to be lumped accordingly, depending on the sign of pq: if $pq > 0$, we lump the North-East and South-West elements. Otherwise we lump the North-West and South-East elements. Since these elements result from the inner product of two basis functions whose central nodes are diagonally opposite on a rectangle and whose supports have only one elementary rectangle in common, we can do the lumping completely on the level of the element matrix.

Let us consider a generic rectangle and order its elements counterclockwise, beginning at the South-West corner. The four basis functions (nodal finite elements) in this rectangle are numbered accordingly. The element matrix for the matrix (3.17), integrated approximately by the midpoint rule, reads

$$\frac{5}{288} \frac{hk}{h^2 + k^2} \begin{bmatrix} S^2 & -ST & -S^2 & ST \\ -ST & T^2 & ST & -T^2 \\ -S^2 & ST & S^2 & -ST \\ ST & -T^2 & -ST & T^2 \end{bmatrix}, \tag{3.19}$$

where $S = pk + qh$ and $T = pk - qh$. If we use the trapezoidal rule for the integration of the inner product $(\mathbf{v} \cdot \operatorname{grad} \phi_i, \mathbf{v} \cdot \operatorname{grad} \phi_j)$ in the numerator, we obtain

$$\frac{5}{144}\frac{hk}{h^2+k^2} \times \begin{bmatrix} P^2+Q^2+PQ & -P^2 & -PQ & -Q^2 \\ -P^2 & P^2+Q^2-PQ & -Q^2 & PQ \\ -PQ & -Q^2 & P^2+Q^2+PQ & -P^2 \\ -Q^2 & PQ & -P^2 & P^2+Q^2-PQ \end{bmatrix}, \tag{3.20}$$

where $P = pk$ and $Q = qh$. If $pq > 0$ the $(2,4)$- and $(4,2)$-elements must be zero. Since those elements in (3.19) and (3.20) have opposite signs, we can make them zero by a convex linear combination of both formula. Analogously, if $pq < 0$, the $(1,3)$- and $(3,1)$-elements to be eliminated have opposite signs and can be made zero by a convex combination of both formulae.

3.3. Accuracy. It is not so clear how to describe the accuracy of a discretization method for a singularly perturbed problem. In general we use the order of accuracy of the truncation error of the scheme. This works fine if the method is given by a stencil with a small number of nonvanishing weights. The new method is not described by a simple stencil with five or nine weights. On the contrary, since the matrix $\mathbf{M}^{-1}$ (cf. (3.10)) is not sparse, the new scheme corresponds to a full matrix. So it is almost impossible to compute the truncation error of the new method.

A theoretically satisfactory description is given by an estimate for the global error of the method. For a model problem in one space dimension global error estimates may be obtained. (See Odenhoven and van Veldhuizen [14].) For problems in two space dimensions an estimate of the global error has not been obtained as yet.

Here we choose a third method for describing the accuracy of a scheme. Apart from some technical restrictions, we shall show that the new method is nodally exact for all solutions which are polynomials of degree at most two. Many second-order schemes have this property. We shall consider a regular mesh with $h_i = h$ and $k_j = k$ for all i and j. In addition, we let the vector field $\mathbf{v}$ be constant.

PROPOSITION 3.1. *Let the above assumptions hold. Let H be any $s \times s$ matrix. Let the exact solution of* (3.1) *be a quadratic polynomial, and let its interpolant $u^h \in S_E^h$ correspond to the coordinate vector $\vec{u}$. Then*

$$[A_1^T H A_1 + A_2^T M_2^{-1} A_2]\vec{u} = A_1^T H \vec{b}_1 + A_2^T M_2^{-1} \vec{b}_2.$$

Proof. On the regular grid the local discretization error of the standard finite-element equations vanishes for a quadratic polynomial. This means $A_1 \vec{u} = b_1$. Thus the terms involving the matrix H are identical in the above formula. They may be dropped. Consider the residual $A_2^T M_2^{-1} A_2 \vec{u} - A_2^T M_2^{-1} \vec{b}_2$. Due to the regular mesh, M_2 is a multiple of the identity matrix. Hence, we may replace M_2 in the residual by the identity matrix without loss of generality. In the modified residual the s-vector $A_2^T A_2 \vec{u}$ describes a symmetric discretization of the second-order diffusion operator (3.14) applied to an interpolant of a quadratic polynomial. Since the vector field $\mathbf{v}$ is constant, it follows that all coordinates of this vector are equal. The inhomogeneous term $A_2^T \vec{b}_2$ describes a consistent discretization of the derivative along the streamlines of f (see (3.1)). The assumptions imply that f is a linear polynomial in x and y (possibly a constant). Hence, all coordinates of the s-vector $A_2^T \vec{b}_2$ are equal. There are no problems

at the Dirichlet boundary due to the nice interplay of nodal values and values at the center of the elements. Hence, the modified residual describes a nice discretization of the problem

$$\|\mathbf{v}\| D_{\mathbf{v}}^2 u = D_{\mathbf{v}} f,$$

where

$$D_{\mathbf{v}} u = \frac{1}{\|\mathbf{v}\|}\left[p\frac{\partial u}{\partial x} + q\frac{\partial u}{\partial y}\right]$$

is a derivative along the streamlines. This discretization is again exact for quadratic polynomials. Since u is a quadratic polynomial, this proves the proposition. □

In the above proposition it is essential that the vector field $\mathbf{v}$ be constant. So is the assumption about the regularity of the mesh. Therefore the assertion of the proposition is valid in rather restricted instances. Nevertheless, this result distinguishes the method proposed in this paper from artificial viscosity methods. Indeed, if a method adds only a diffusion term to the scheme as most upwind-type methods do, then the method is no longer exact for quadratic polynomials. Therefore, the proposition suggests that the new method is of improved accuracy for smooth solutions. In many instances the solution of a singularly perturbed problem is not smooth everywhere. Boundary layers and internal layers are very common in these problems. But apart from these layers the solution may be very smooth. If the layer error does not pollute the solution too much, we might hope that the new scheme is more accurate outside the (boundary) layers. Since boundary layers are thin this should show up in the mean-square norm of the error. There is some numerical evidence for this (cf. § 4).

Observe that the streamline upwind/Petrov–Galerkin method described in Brooks and Hughes [4] is also exact for quadratic polynomials. As in the above proposition, the word "exact" means nodally exact. The method cannot be exact in the H_E sense, since the finite-element space S_E^h does not contain all quadratic polynomials.

The assertion of the above proposition is also valid if lumped matrices are being used. Indeed, if we lump A_1 to a five-diagonal matrix, the accuracy argument remains valid. Also, if we lump $A_2^T M_2^{-1} A_2$ as described, we do not introduce any error if the solution is a quadratic polynomial. Hence, all arguments in the proof of Proposition 3.1 remain valid, and so does the conclusion.

At this moment it seems quite difficult to obtain global error estimates for the new method. For a global error estimate we need some form of a local error estimate plus a kind of stability result. Only one stability result is available, namely the one in [17]. This result is related to a multigrid approach, with the space T_E^h and the space S_E^h belonging to the same family of finite-element spaces. As yet, no estimate for a local error estimate have been obtained.

3.4. Solving the normal equations. The new method results in a set of linear equations for the unknown coordinate vector $\vec{u}$. These equations (cf. (3.12)) are given by

$$D\vec{x} = (A_1^T M_1^{-1} A_1 + A_2^T M_2^{-1} A_2)\vec{x} = (A_1^T M_1^{-1}\vec{b}_1 + A_2^T M_2^{-1}\vec{b}_2), \tag{3.21}$$

where A_1 and M_1 have been lumped as described in § 3.2. A direct method (e.g., Gaussian elimination) is not suitable for solving these equations. Since M_1^{-1} is a full matrix, a direct method would require too much storage and too much time. An iterative solution method is used instead. We have chosen a Preconditioned Conjugate Gradient Method (PCGM). We remark that the least-squares problem is of full rank, since otherwise we have chosen the wrong stabilizing extension. In view of the adopted

numbering of the nodes, and taking into account the lumping process described in § 3.2, we have the following sparsity patterns:

A_1, M_1 block-tridiagonal with tridiagonal diagonal blocks and diagonal off-diagonal blocks, M_1 being symmetric,
A_2 block bidiagonal with bidiagonal blocks,
M_2 diagonal,
$A_2^T M_2^{-1} A_2$ symmetric block-tridiagonal with tridiagonal blocks; if $\mathbf{v}$ points into the same quadrant everywhere, there are only seven nontrivial diagonals.

Since a PCGM requires only the matrix-vector product, it suffices to store the 17 nontrivial (co-)diagonals of A_1, M_1, and $A_2^T M_2^{-1} A_2$. This suffices for the computation of the necessary matrix-vector products (as far as the matrices are concerned; we also need storage for vectors). We should not compute the full matrix M_1^{-1}. Since we only need the matrix-vector product involving M_1^{-1}, it is more efficient to solve the corresponding set of equations. We do this by an iterative process. We use an ICCG (0) strategy of Meijerink and van der Vorst [13], in which we compute a preconditioner by an incomplete Cholesky factorization with the same sparsity pattern as in the original matrix. This process works well (5–20 CG-iterations for 100–5000 unknowns).

The conjugate gradient process applied to (3.21) does not work well unless we use a good preconditioner. We obtain one by the following heuristics. A good preconditioner C, which changes the system $D\vec{x} = \vec{b}$ into $(C^{-T} D C^{-1}) C\vec{x} = C^{-1}\vec{b}$, should look like the Cholesky factor of D as much as possible. Let

$$B = \begin{bmatrix} L^{-1} A_1 \\ M_2^{-1/2} A_2 \end{bmatrix},$$

where $L^T L$ is the Cholesky factorization of M_1. The left Cholesky factor of the positive definite matrix D can be obtained by a QR factorization of B. Thus we avoid the explicit computation of $D = B^T B$ (cf. (3.21)). Since we are only interested in an approximate Cholesky factor of D, we make some simplifications. Instead of the Cholesky factor L of M_1 we use an incomplete Cholesky factor $\tilde{L}$, and instead of the product $L^{-1} A_1$ we use an incomplete product of $\tilde{L}^{-1}$ and A_1. To be precise, we perform the following steps:

(1) Form the *incomplete* Cholesky factor $\tilde{L}$ of M_1, allowing no fill-in. This matrix will have a nontrivial diagonal and two nontrivial co-diagonals. We need this matrix anyway as a preconditioner in solving the Laplace equation $M_1 \vec{y} = \vec{z}$.
(2) Form the *incomplete* product $\tilde{L}^{-1} \tilde{\times} A_1$ by setting to zero in each sweep of the back-substitution all elements outside the nontrivial diagonals.
(3) Form an *incomplete* QR-factorization of the matrix

$$\begin{pmatrix} \tilde{L}^{-1} \tilde{\times} A_1 \\ M_2^{-1/2} A_2 \end{pmatrix}.$$

In each Householder sweep we set to zero all elements that do not correspond to a difference stencil as shown in Fig. 1. Except for the (co-)diagonals corresponding to the usual nine-point stencil, this incomplete factorization allows for five additional co-diagonals of fill-in. Without this fill-in the convergence of the process is rather poor. The computed incomplete Cholesky factor with 10 (co-)diagonals requires about the same amount of storage as (A_1, A_2). The Householder reflections need not be stored since they are not needed for the rest of the computations.

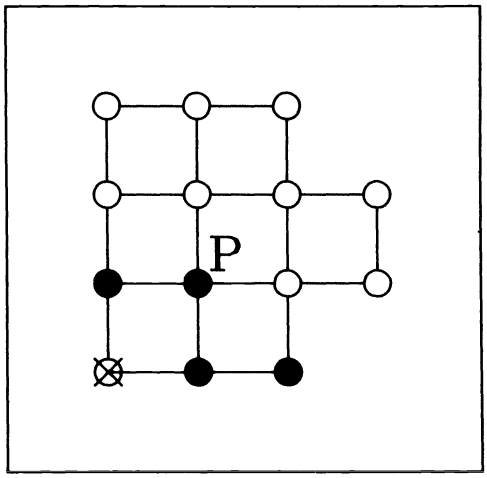


FIG. 1. *Stencil of nodes around the node P used in the incomplete* QR*-decomposition described in step* (3) *of* § 3.4. *The black nodes belong to the diagonal and the lower diagonal part of the matrix* A_1. *The node in the* SW *corner is not involved in the process.*

The incomplete upper triangular factor, thus obtained, works well as a preconditioner. Many variants for the computation of a preconditioner can be conceived. The following one gives a clear improvement. Since $\tilde{L}$ is *lower triangular*, the factor $\tilde{L}^{-1}$ in the incomplete product $\tilde{L}^{-1} \tilde{\times} A_1$ moves "mass" in the matrix downward into the lower triangular part. In the next step, the Householder reflections must return this mass to the upper triangular part. It is more natural to leave that mass in the upper triangular part by the use of an *upper triangular* matrix $\tilde{U}$, and hence to perform an incomplete Cholesky-like factorization that yields

$$M_1 \sim \tilde{U}\tilde{U}^T$$

with an (incomplete) upper triangular factor $\tilde{U}$.

On test problems this solution method works well and gives accurate results, provided the stopping criterion in the Laplace solver guarantees sufficiently accurate solutions of $M_1\vec{y} = \vec{z}$. However, each iteration takes quite a long time because of this Laplace solver.

3.5. Simplified least squares. Because of the many lengthy iterations required by the solution process of the new method, it is worthwhile investigating what might be done to speed up the solution process. A big gain can be expected from a different choice of the inner product, although this would change the method too. Nevertheless, it would be a real gain if we could eliminate the need for a Laplace solver in each iteration. Another reduction of the cost of solving the equations can be expected from a better and cheaper preconditioner.

In a crude approach we replace the matrix **M** by the identity matrix, i.e., we solve the normal equations

$$[A_1^T A_1 + A_2^T A_2]\vec{x} = \vec{b}.$$

In this case the matrix between brackets can easily be computed. The matrix is symmetric with 13 nonvanishing (co-)diagonals. Because of symmetry we have to store only 7 (co-)diagonals. In many instances a preconditioner (incomplete Cholesky factor) is easily obtained, although the matrix need not be an M-matrix. But even if a reasonable

preconditioner has been found, the method behaves not so well on irregular meshes. Obviously, the matrix **M** provides a natural scaling of the basis functions, which is lost if **M** is replaced by the identity matrix.

In attempting a more subtle approach, we first remark that there is no need to change the diagonal matrix M_2. The matrix $A_2^T M_2^{-1} A_2$ is already banded with at most nine nonvanishing (co-)diagonals. However, in our final simplification we do change M_2 and M_1 simultaneously, as shown below. As for the part involving M_1, we might use an incomplete factorization instead of M_1 itself. This eliminates the need for a Laplace solver. Nevertheless, the solution process by means of a preconditioned conjugate gradient algorithm is still quite complex. Three incomplete operations on top of each other are required, making this choice unattractive. There is a more serious objection. Any change in M_1 is a change in the discretization method. By replacing M_1 by an incomplete Cholesky factorization, we induce in the system some sense of direction. This sense of direction depends on the numbering of the nodes, which is highly undesirable. For these reasons this choice has been abandoned.

For reasons of efficiency we replace M_1 by a diagonal matrix. The advantage is obvious. Replacement of M_1 by any invertible diagonal matrix leads to a banded matrix of the normal equations. No Laplace solver is needed. Also by choosing a suitable diagonal matrix $\tilde{M}_1$ instead of M_1 the method remains independent of the numbering of the nodes. On a regular mesh and for constant coefficient problems, it turns out that the choice

$$\tilde{M}_1 = \mu^{-1} \text{ diagonal } \{M_1\}$$

gives good results, where μ is a scalar depending only on h_i/ε and k_j/ε. The choice of the scalar function μ is important. Observe that $\tilde{M}_1$ should be as close as possible to M_1 (in an intuitive sense). Now consider a regular mesh where the meshsize is large in relation to ε. For such a mesh the smooth solution components are represented through the extension part of the stabilized Galerkin method. Here the components correspond to eigenvectors of the discretized Laplacian on the mesh. For the other components, the higher oscillatory ones, the matrix M_1 behaves roughly as two to four times its diagonal. Hence, for an extremely stiff situation we choose $\mu = 4$. For mildly stiff problems we decrease the value of μ, and finally in the limit (mesh spacing $\to 0$) we let $\mu \to 0$.

Observe that this modification of the original stabilized Galerkin method still satisfies the assumptions of Proposition 3.1. Thus, this method with $\tilde{M}_1$ instead of M_1 is nodally exact for quadratic polynomials.

In our final simplification we place a diagonal matrix **D** in the extension part. That is, we replace (3.21) with

$$[A_1^T \tilde{M}_1^{-1} A_1 + A_2^T \mathbf{D} M_2^{-1} A_2]\vec{x} = A_1^T \tilde{M}_1^{-1} \vec{b}_1 + A_2^T \mathbf{D} M_2^{-1} \vec{b}_2. \tag{3.22}$$

The diagonal elements of the diagonal matrix **D** correspond to elements in the subdivision of the domain. We let these diagonal elements be functions of ε, the vector field $\mathbf{v}$, and the step sizes h_i, k_j such that $\mathbf{D} = \mu I$ on a regular mesh and a constant coefficient problem. Thus, for regular meshes the new formulation and the previous one are equivalent. The multiplication by the diagonal matrix **D** is done in assembling the matrix $A_2^T M_2^{-1} A_2$ in its lumped finite-element form. On each element the contribution is multiplied by a factor $\sigma(ph_i/\varepsilon, qk_j/\varepsilon)$,

$$\sigma(x, y) = 4 \tanh (\alpha \max \{|x|, |y|\})^{\beta}$$

where α, β are constants to be determined. This function σ establishes the smooth transition from the extremely stiff case ($\sigma \sim 4$) to the classical nonstiff case ($\sigma \sim 0$).

With this simplification of the scheme the matrix of (3.22) is independent of the length of the finite-element basis functions. Hence, the simplified scheme now has the same scaling invariance as the scheme (2.18).

The two parameters α and β are determined as follows. Choose a simple constant coefficient problem with an explicit formula for the exact solution. Solve the problem by the simplified method described by (3.22). Since the mesh is regular, we have $\mathbf{D} = \mu I$. Now determine an optimal μ by requiring that the global approximation error of the method (3.22) be minimized. Do this for ε in the range $1 - 10^{-4}$. Then determine the constants α, β by a least squares fit. It turns out that the values

$$\alpha = 0.0147, \qquad \beta = 1.26$$

work well in many examples. Of course, all this is reminiscent of the determination of the upwind factor in the streamline upwind/Petrov-Galerkin method described by Brooks and Hughes [4].

We should be aware that the matrix of (3.22) is not an M-matrix in general. The existence of an incomplete Cholesky factorization is not assured. The process of forming the incomplete factors may break down because of too small a pivot. Of course, we should always implement it as an LDL^T decomposition, because negative pivots are harmless then. In our process we test on small pivots. If too small a pivot is detected, then the diagonal of the matrix is enlarged by 10 percent of the largest diagonal element (of the original diagonal), and the process of forming an incomplete factorization is started all over again.

4. Numerical illustration. In this section we give numerical results for a few implementations of the stabilized Galerkin (SG) method and the Brooks-Hughes (BH) streamline upwind/Petrov-Galerkin method (cf. [4]). We use the following implementations:

SG0: This is the stabilized Galerkin method as described in §§ 3.1 and 3.2. We lump the matrices A_1 and M_1 into a five-diagonal matrix. We do not lump the $A_2^T M_2^{-1} A_2$-term. The right-hand side is integrated by the product Simpson rule on each of the rectangular elements. A varying vector field $\mathbf{v}$ is replaced by its value at the element center. The integrations are then performed exactly using this value.

SG1: This is the stabilized Galerkin method as described in §§ 3.1 and 3.2. It is obtained from the method SG0 by lumping the $A_2^T M_2^{-1} A_2$-term.

SG2: This is a modification of SG1 according to § 3.5. That is, we replace the lumped matrix M_1 by a suitable scalar factor of its diagonal as described in § 3.5.

BH1: This is the streamline upwind/Petrov-Galerkin method as described in Brooks and Hughes [4]. We use the product Simpson rule for the right-hand side. The vector field $\mathbf{v}$ is replaced by its value at the element center on each of the elements. All remaining quadratures are done exactly. The upwind parameter is determined as in Brooks and Hughes [4].

All computations have been done on a Cyber 750 or a Cyber 205 (large memory). The arithmetic of these two machines is slightly different, but on both of them a floating point number has a 48-bit mantissa. We feel it is possible to mix the results from these two machines. In our examples the influence of rounding errors is not really significant.

In Tables 1 and 2 below we list errors as follows. In all instances we consider the error in the nodes only. In columns indicated by E_∞ we list the supremum norm of this vector of errors, i.e., the largest error in modulus. In a column indicated by E_2

we list the weighted Euclidean norm of this error. In many instances we compare the obtained numerical approximation with an asymptotic approximation. In these instances the errors are indicated by $E_{2,a}$ and $E_{\infty,a}$, respectively. On a mesh with $N \times M$ nodes in a rectangular pattern the weighted Euclidean norm is just the Euclidean norm divided by $\sqrt{NM}$.

The workload of both the SG2 and the BH1 methods depends heavily on the solution method for the linear system of equations. In SG2, with a symmetric matrix, a preconditioned conjugate gradient method works well with modest memory requirements. For the unsymmetric linear system, arising in the BH1-method, a direct sparse matrix solver (rather demanding in memory), or an iterative method may be used. However, for symmetric matrices the convergence properties of iterative methods are much better known than for nonsymmetric ones. In our examples we compute the BH1-approximation by a direct method (band solver).

Example 1. Consider the differential equation

$$-\varepsilon \Delta u + \cos\phi\, u_x + \sin\phi\, u_y = f \tag{4.1}$$

on the unit square $[0, 1] \times [0, 1] \subset \mathbb{R}^2$. We prescribe homogeneous Dirichlet boundary conditions on the boundary $x = 0$, $y = 0$, and homogeneous Neumann boundary conditions on the two remaining sides. An equidistant mesh is used with stepsize $h_i = k_j = h = 1/N$. We choose $\varepsilon = 10^{-6}$, $\phi = \frac{1}{8}\pi$. The right-hand side is determined by the exact solution chosen as

$$u(x, y) = \sin\left(\frac{\pi}{2}x\right) \sin\left(\frac{\pi}{2}y\right).$$

In Table 1 we list some results for the method SG2 and the method BH1. The results for the SG0-method are identical to the corresponding ones of SG2. However, solving the linear system takes much longer. The results agree with a second-order behavior of the global error. It is rather remarkable that the methods SG0, SG2, and BH1 give results so similar, although there is quite a difference between (3.22) and the Brooks-Hughes method. □

TABLE 1
Results for the problem described in Example 1.

	SG2		BH1	
N	E_∞	E_2	E_∞	E_2
10	2.45E−03	5.07E−04	4.26E−03	1.92E−03
20	6.13E−04	1.12E−04	1.15E−03	4.95E−04
30	2.74E−04	4.74E−05	5.18E−04	2.22E−04
40	1.56E−04	2.58E−05	2.91E−04	1.25E−04
60	7.11E−05	1.08E−05	1.26E−04	5.46E−05

Example 2. Consider the differential equation, the unit square, and its regular subdivision as in Example 1. Again we prescribe homogeneous boundary conditions, Dirichlet conditions for $x = 0$, $y = 0$, and Neumann conditions for $x = 1$, $y = 1$. We choose $\varepsilon = 10^{-6}$, $\phi = \frac{1}{8}\pi$, and $f(x, y) = \cos(x + y)$. For this problem the exact solution is not known. However, we expect an asymptotic approximation to be very close to the exact solution. The asymptotic approximation of order zero u_{asymp} is obtained by integration of the hyperbolic problem

$$\cos\phi\, u_x + \sin\phi\, u_y = f$$

with homogeneous boundary conditions on the Dirichlet boundaries $x=0$, $y=0$. The outflow boundary plays no part in this asymptotic approximation. See Eckhaus [6] for an introduction to this kind of asymptotic results.

In Table 2 we list the results for SG2 and BH1 applied to this problem. As in the previous example, the results of the methods SG0 and SG2 are almost identical. The global error is estimated by comparing the obtained numerical solution with the asymptotic approximation just described.

TABLE 2
Results for the problem described in Example 2.

	SG2		BH1	
N	$E_{\infty,a}$	$E_{2,a}$	$E_{\infty,a}$	$E_{2,a}$
10	7.24E−02	1.88E−02	4.14E−02	1.37E−02
15	6.24E−02	1.27E−02	3.67E−02	8.57E−03
20	5.04E−02	9.61E−03	2.92E−02	6.13E−03
30	4.10E−02	6.49E−03	2.33E−02	3.83E−03
40	3.25E−02	4.90E−03	1.83E−02	2.74E−03
60	2.63E−02	3.32E−03	1.37E−02	1.74E−03

In the previous example the solution is smooth, and the methods SG2 and BH1 show a second-order accuracy behavior. In this example the exact solution is smooth in the sense that it does not have steep slopes, but it does have a *fold* along the line $y=x$ due to the corner $(0, 0)$ at the inflow boundary. The boundary layer at the outflow boundary is $O(\varepsilon)$. Along the line $y=x$ the asymptotic (outer) approximation does not have a continuous normal derivative in general, and an $O(\varepsilon)$ transition layer arises there. For all practical purposes it looks as if the exact solution has a discontinuous gradient at $x=y$. This discontinuity must be the cause of the first-order accuracy exhibited by the results of both methods in Table 2. For both methods the weighted Euclidean norm of the error is much smaller than the supremum norm. This is to be expected for problems of which the solution is smooth on large parts of the domain Ω. It is an indication that the errors in the boundary layers do not pollute the smooth part of the solution too much. □

Example 3. In this example we again consider the differential equation (4.1) on the unit square in $\mathbb{R}^2$. Also, the mesh is regular as in the previous examples. We choose $f=0$, $\varepsilon=10^{-6}$, and $0<\phi<\frac{1}{2}\pi$. At the outflow boundary, $x=1$, $y=1$ we prescribe a homogeneous Neumann boundary condition. At the Dirichlet boundary $y=0$ we prescribe $u=0$, while at the Dirichlet boundary $x=0$ we have the discontinuous boundary condition $u=0$ for $y<\frac{1}{4}$, $u=1$ for $y>\frac{1}{4}$. At $x=0$, $y=\frac{1}{4}$ we prescribe $u=\frac{1}{2}$. Again, the exact solution is not known explicitly. Instead, we use an asymptotic approximation as in Example 2. In this instance the asymptotic approximation u_{asymp} is given by

$$u_{\text{asymp}}(x, y)=\operatorname{sgn}(\tan(\phi)x-y-\tfrac{1}{4}),$$

where the sign-function sgn is given by

$$\operatorname{sgn}(x)=\begin{cases}1, & x>0,\\ \frac{1}{2}, & x=0,\\ 0, & x<0.\end{cases}$$

In contrast to the behavior in the two preceding examples, the global error in this example hardly changes as long as the quotient of the meshsize and ε is large (greater than 20, say). This is caused by the steep slopes in the exact solution, corresponding to the discontinuity in the asymptotic approximation. Therefore we present the results in a graphical way in Fig. 2. The angle ϕ and the method are indicated in Fig. 2. All results are shown from a viewpoint at infinity in the direction of the vector $(3, 20, 8)$ in the (x, y, z)-space, where the z-direction is positive upward. The asymptotic approximation is indicated by the capital letter "A." In all cases we have $N = 12$. That is, the discontinuity in the boundary condition takes place at a node of the mesh. This improves the accuracy of the solution somewhat. The spurious oscillations are larger if the point of discontinuity does not coincide with a node. The method without lumping of the $A_2^T M_2^{-1} A_2$-part, i.e., the method SG0 has larger oscillations than the lumped method SG2. This is very apparent for $\phi = \frac{1}{8}\pi$. Both the methods SG2 and BH1 behave rather similarly. □

Example 4. Again we consider the partial differential equation (4.1) with $\phi = \frac{1}{8}\pi$. The exact solution is given by the polynomial

$$u(x, y) = x(2-x)y(2-y)$$

with boundary conditions as in Example 1. In this case we show some results for ε varying between 1 and 10^{-6}. For large ε the stabilized Galerkin method should behave like the ordinary Galerkin method. The previous examples indicate the behavior of the stabilized Galerkin method for small ε. Here we illustrate the behavior for intermediate values of ε. To that end we give in a graphical form the error in the weighted Euclidean norm for implementations SG0, SG2, and BH1. We use the regular grid of Example 1 with $N = 1/12$. The result for the implementation SG0 hardly depends on the value of ε. The results for the implementations SG2 and BH1 show a stronger dependence on ε. For the method SG2 this dependence is governed by the choice of

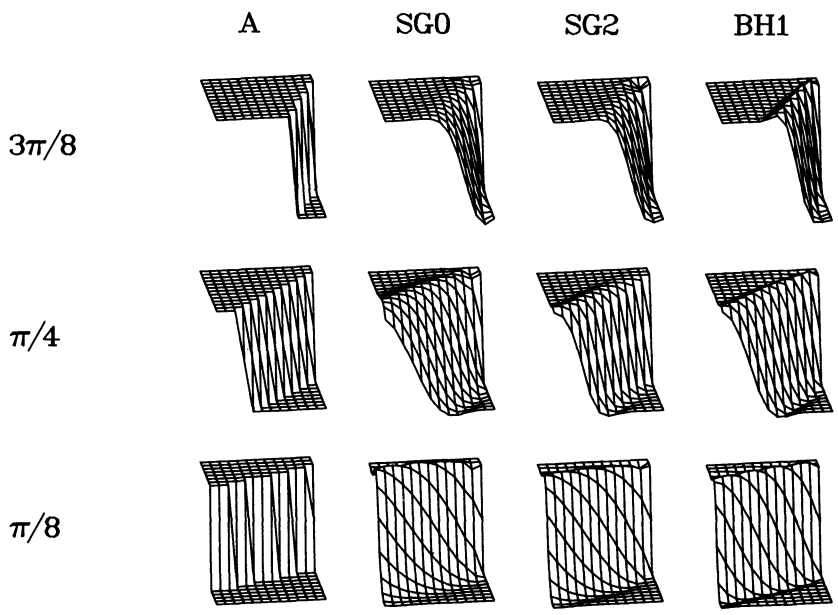


FIG. 2. *Results for the problems described in Example* 3. *The angle* ϕ (*cf.* (4.1)) *is shown at the left-hand side. The asymptotic approximation is indicated by* "A." *The viewpoint is at infinity in the direction of the vector* $(3, 20, 8)$ *in* (x, y, z)*-space.*

the function σ, for the method BH1 by the choice of the upwind parameter. The results shown in Fig. 3 indicate that these choices lead to an acceptable behavior. □

Example 5. Again we consider the partial differential equation (4.1). In this example we consider homogeneous Dirichlet boundary conditions on all of the boundary. We choose the inhomogeneous term as $f(x, y) = 1$ on the unit square. Let $\phi = 0$. This implies that the exact solution has a parabolic boundary layer of width $O(\sqrt{\varepsilon})$ along the boundaries $y = 0$ and $y = 1$. In addition, there is a boundary layer of width $O(\varepsilon)$ along the boundary $x = 1$. See Eckhaus [6] for an introduction to these kind of asymptotic results.

In § 3.2 it has been shown that the original stabilized Galerkin method without lumping gives spurious oscillations for this kind of problem. But the situation is even worse. On a regular mesh, as used in the previous examples, the methods SG0, SG1, and SG2 fail completely. The obtained approximation on a regular mesh has an error of 100 percent almost everywhere on the unit square. This is a consequence of the symmetry of the method. The method itself, or at least the matrix of the discretization, i.e., (3.12) for the original stabilized Galerkin method, is symmetric. Loosely speaking, the matrix contains the product of the operator to be approximated and its adjoint. Hence, the matrix has no good indication where to put the boundary layers. Neither does the right-hand side of (3.12). The discretization is just too symmetric. As a consequence, a boundary-layer term is approximated by a low-frequency oscillation, and this oscillation spoils the approximation.

A remedy is the refinement of the mesh in the expected boundary layers, as proposed by Schatz and Wahlbin [15] (pollution-absorbent mesh). We consider a mesh obtained as the product of a subdivision in the x-direction and the y-direction. In the x-direction the successive steps are given by

$$h, h, h, \cdots, h, \varepsilon, \tfrac{1}{3}\varepsilon, \tfrac{1}{5}\varepsilon$$

such that

$$(N-2)h + \tfrac{23}{15}\varepsilon = 1.$$

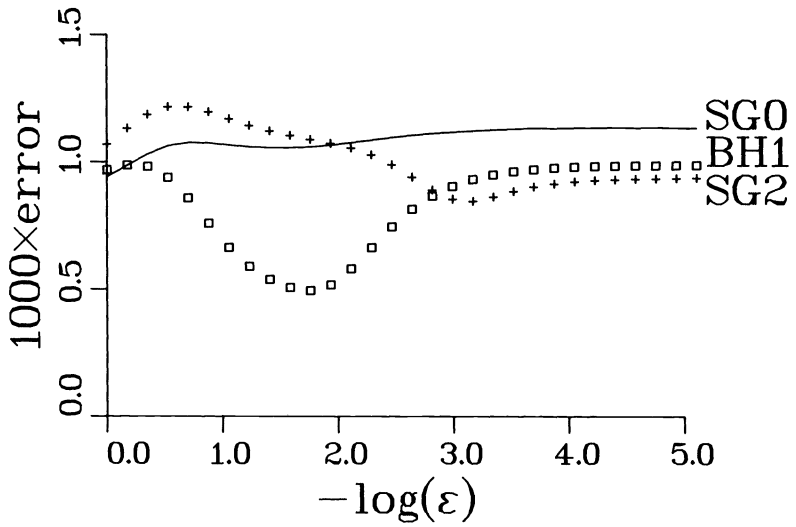


FIG. 3. *The weighted Euclidean norm of the error is shown as a function of the parameter* ε (*cf.* (4.1)) *for the problem of Example* 4. *The results are shown for the implementations* SG0, SG2, *and* BH1.

In the y-direction we use the steps

$$\tfrac{1}{5}\sqrt{\varepsilon}, \tfrac{1}{3}\sqrt{\varepsilon}, \sqrt{\varepsilon}, k, k, \cdots, k, \sqrt{\varepsilon}, \tfrac{1}{3}\sqrt{\varepsilon}, \tfrac{1}{5}\sqrt{\varepsilon}$$

such that

$$(N-5)k+\tfrac{46}{15}\sqrt{\varepsilon}=1.$$

On the mesh obtained by taking the product of these two subdivisions the methods SG0, SG1, and SG2 perform much better. Because of the scaling of the matrices there are no problems in using these extremely small rectangles. In Fig. 4 we show the obtained approximations. Note that the mesh is highly nonuniform. This nonuniform mesh as described above has been used in the plots.

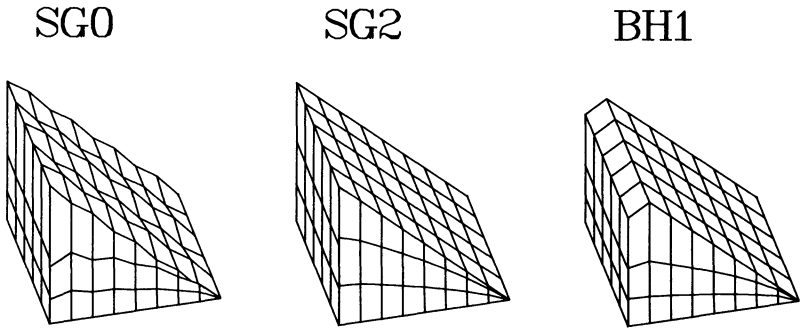


FIG. 4. *Results for the problem of Example* 5. *The results are shown on the refined mesh as described with* 10×10 *internal nodes. The viewpoint is at infinity in the direction of the vector* $(5, 20, 16)$ *in* (x, y, z)*-space.*

It is quite clear that there are still some oscillations left in the SG0-approximation. The SG1-approximation is much better, and so is the SG2-approximation. The result for the Brooks–Hughes method is not so good. On the regular mesh the BH1-approximation attains the maximal value 1.15 (it should be 1.0), and on the refined mesh the maximal value is 0.88. All SG-approximations reach the maximal value 1.00 (rounded to two digits after the decimal point). □

Example 6. We consider (4.1) as before with $\phi=\frac{1}{8}\pi$, with the inhomogeneous term $f(x, y)=1$ on the unit square and with homogeneous boundary conditions. The solution has boundary layers at $x=1$ and $y=1$ and an $O(\varepsilon)$ transition layer along the line $y=x\tan\frac{1}{8}\pi$. Moreover, it can be shown that the exact solution is bounded from above by $x/\cos\frac{1}{8}\pi$ if $x\tan\frac{1}{8}\pi \geqq y$, and by $y/\sin\frac{1}{8}\pi$ elsewhere. So the solution has a maximum with value at most $1/\cos\frac{1}{8}\pi \sim 1.082$. On a regular mesh the stabilized Galerkin approximation does not work, as in the previous example. If the mesh is refined in the boundary layers with the mesh spacings

$$h, h, h, \cdot, \cdot, \cdot, \cdot, h, \varepsilon, \tfrac{1}{3}\varepsilon, \tfrac{1}{5}\varepsilon, \qquad (N-2)h+\tfrac{23}{15}\varepsilon=1$$

in both the x- and the y-direction, then SG2 gives an accurate approximation. This approximation overshoots the maximum by at most 1 percent. On the other hand the BH1 approximations develop instability in the boundary layers (both on the regular and the refined mesh) and show a strong overshooting of the maximum (see Fig. 5). □

5. Concluding remarks. In this paper we have described a new type of discretization for convection-dominated diffusion problems. This new "stabilized Galerkin" method is based on some abstract ideas related to the finite-element method for second-order elliptic partial differential equations. This paper shows that the abstract

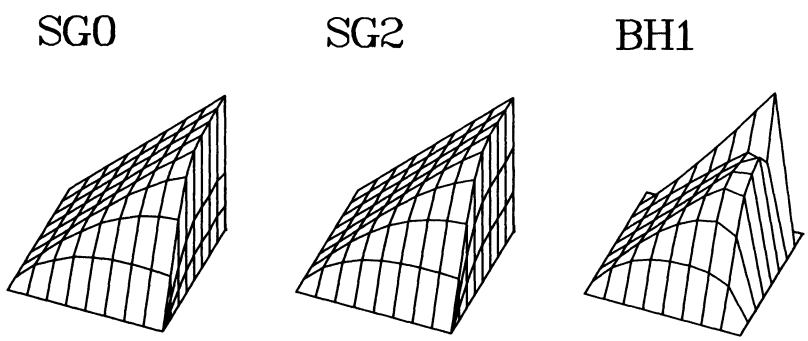


FIG. 5. *Results for the problem of Example 6. The results are shown on the refined mesh as described with* 10×10 *internal nodes. The viewpoint is at infinity in the direction of the vector* $(8, -20, 19)$ *in* (x, y, z)-space.

ideas can be turned into an efficient algorithm. A straightforward implementation leads to quite complicated systems of linear equations, i.e., to an inefficient algorithm. However, by suitable modifications and simplifications we tuned it to a rather efficient one.

The stabilized Galerkin method, as described here, is essentially a symmetric method. In the normal equations to be solved, all sense of direction of the flow has been eliminated. In the case where the boundary condition at the outflow boundary is of Neumann type, the (free) boundary condition provides enough help to find the correct location of the boundary layer. If the boundary condition is of Dirichlet type, the help must be given by a spatial discretization with a few small steps in the region of the outflow boundary. With such help the stabilized Galerkin finds the correct position of the boundary layer and provides an accurate approximation of the solution. We remark that such a local mesh refinement in the boundary layer only is not sufficient for the stabilization of the standard finite-element discretization.

The subject of this paper and this new method in particular deserve further research. Many questions and problems remain unanswered. It would be worthwhile to have a more general basis for the lumping applied to the extension part of the normal equations (cf. § 3.2.). The present description is rather ad hoc and applies to bilinear elements only; e.g., all work has to be done again for an implementation using linear basis functions on triangles.

We have not discussed the problems caused by an application of conforming bilinear elements, in which the elements are quadrilaterals. In this case the matrix $\mathbf{M}$ (cf. (3.10)) does not reduce to two diagonal blocks, decoupling the standard part from the extension part in the normal equations. This corresponds to the nonvanishing third derivative terms in the streamline-upwind/Petrov-Galerkin method. Brooks and Hughes suggest neglecting these terms. It would be in the same spirit to neglect those off-diagonal blocks in $\mathbf{M}$. We have no experience on this point and note that it deserves further investigation.

Finally we remark that we have not yet studied transient problems. This will be a topic for future research.

REFERENCES

[1] I. BABUŠKA AND A. K. AZIZ, *Survey lectures on the mathematical foundations of the finite element method*, in The Mathematical Foundations of the Finite Element Method with Applications to Partial Differential Equations, A. K. Aziz, ed., Academic Press, New York, 1972, pp. 3-363.

[2] J. W. BARRETT AND K. W. MORTON, *Optimal finite element approximation for diffusion-convection problems*, in The Mathematics of Finite Elements and Applications, Proc. MAFELAP 1981, J. R. Whiteman, ed., Academic Press, London, 1982, pp. 403–410.

[3] M. O. BRISTEAU, O. PIRONNEAU, R. GLOWINSKI, J. PÉRIAUX, P. PERRIER, AND G. POIRIER, *Application of optimal control and finite element methods to the calculation of transonic flows and incompressible viscous flows*, IMA Conference on Numerical Mathematics in Applied Fluid Dynamics, B. Hunt, ed., Academic Press, London, 1980, pp. 203–312.

[4] A. N. BROOKS AND TH. J. R. HUGHES, *Streamline upwind / Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations*, Comput. Meth. Appl. Mech. Engrg., 32 (1982), pp. 199–259.

[5] P. G. CIARLET, *The Finite Element Method for Elliptic Problems*, North-Holland- Amsterdam, 1979.

[6] W. ECKHAUS, *Asymptotic Analysis of Singular Perturbations*, North-Holland, Amsterdam, 1979.

[7] J. E. FLAHERTY AND R. E. O'MALLEY, JR., *Numerical methods for stiff systems of two-point boundary value problems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 865–886.

[8] P. P. N. DE GROEN AND P. W. HEMKER, *Error bounds for exponentially fitted Galerkin methods applied to stiff two-point boundary value problems*, in Numerical Analysis of Singular Perturbation Problems, P. W. Hemker and J. J. H. Miller, ed., Academic Press, New York, 1979, pp. 217–249.

[9] P. P. N. DE GROEN AND M. VAN VELDHUIZEN, *An introduction to the stabilized Galerkin method*, in Proc. ICIAM 1987, Contributions from the Netherlands, A. H. P. van der Burgh and R. M. M. Mattheij, eds., Amsterdam, 1987, pp. 151–157.

[10] J. C. HEINRICH, P. S. HUYAKORN, O. C. ZIENKIEWICZ, AND A. R. MITCHELL, *An "upwind" finite element scheme for two-dimensional convective transport equations*, Internat. J. Numer. Methods Engrg., 11 (1977), pp. 134–143.

[11] P. W. HEMKER, *Numerical study of stiff two-point boundary problems*, Ph.D. thesis, University of Amsterdam, 1977; MC-tract 80, Mathematical Centre, Amsterdam, the Netherlands, 1977.

[12] M. A. KRASNOSEL'SKII, G. M. VAINIKKO, AND P. P. ZABREIKO, *Approximate Solutions of Operator Equations*, Wolters-Noordhoff, Groningen, the Netherlands, 1972.

[13] 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*, Math. Comput., (1977), pp. 148–162.

[14] P. J. ODENHOVEN AND M. VAN VELDHUIZEN, *On the 1-D stabilized Galerkin method*, unpublished manuscript, 1985.

[15] A. H. SCHATZ AND L. B. WAHLBIN, *On the finite element method for singularly perturbed reaction diffusion problems in two and one dimensions*, Math. Comput., 40 (1983), pp. 47–89.

[16] M. VAN VELDHUIZEN, *Higher order schemes for a singularly perturbed problem*, Numer. Math., 30 (1978), pp. 267–279.

[17] M. VAN VELDHUIZEN AND P. J. ODENHOVEN, *The stabilized Galerkin method*, Report 277, Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, the Netherlands, 1985.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 298-327, March 1989
© 1989 Society for Industrial and Applied Mathematics
005

A STUDY OF THE VORTEX SHEET METHOD AND ITS RATE OF CONVERGENCE*

ELBRIDGE GERRY PUCKETT†

Abstract. The subject of this study is Chorin's vortex sheet method, which is used to solve the Prandtl boundary layer equations and to impose the no-slip boundary condition in the random vortex method solution of the Navier-Stokes equations. This is a particle method in which the particles carry concentrations of vorticity and undergo a random walk to approximate the diffusion of vorticity in the boundary layer. During the random walk, particles are created at the boundary in order to satisfy the no-slip boundary condition. It is proved that in each of the L^1, L^2, and L^∞ norms the random walk and particle creation, taken together, provide a consistent approximation to the heat equation, subject to the no-slip boundary condition. Furthermore, it is shown that the truncation error is entirely due to the failure to satisfy the no-slip boundary condition exactly. It is demonstrated numerically that the method converges when it is used to model Blasius flow, and rates of convergence are established in terms of the computational parameters. The numerical study reveals that errors grow when the sheet length tends to zero much faster than the maximum sheet strength. The effectiveness of second-order time discretization, sheet tagging, and an alternative particle-creation algorithm are also examined.

Key words. boundary layer equations, particle method, random vortex method, random walk, vorticity

AMS(MOS) subject classifications. 35, 60, 65, 76

1. Introduction. The vortex sheet method is a numerical method for approximating solutions of the Prandtl boundary layer equations. It was developed by Chorin [9] for use with the random vortex method [8] when approximating solutions of the Navier-Stokes equations in domains with solid boundaries. The vortex sheet method is used in regions adjacent to the boundary, while the random vortex method is used in the interior of the flow. Both methods are particle methods. The particles carry concentrations of vorticity; hence, the velocity field is determined by the particle positions. The particles are advected in this velocity field and then undergo a random walk to model the effects of diffusion. In the vortex sheet method, particles are created on the boundary during the random walk in order to approximately satisfy the no-slip boundary condition. The two methods are coupled by requiring that the circulation around a particle remain the same when it passes from one region to the other and that the velocity imposed on the boundary by the random vortex method is the velocity at infinity in the Prandtl equations. Hybrid vortex sheet/random vortex methods have been successfully used to model such problems as flow past a circular cylinder [5], [6], [36]; driven cavity flow [7], [27]; boundary layer instability [11]; flow past a backward-facing step [32]; turbulent combustion [15], [33], [34]; and wind flow over a building [35].

Much theoretical work has been devoted to understanding the accuracy of the vortex method (i.e., the random vortex method without the random walk used to solve

* Received by the editors June 8, 1987; accepted for publication (in revised form) May 25, 1988. This work was part of the author's Ph.D. dissertation in the Mathematics Department, University of California, Berkeley, California. Portions of this work were performed under the auspices of the Office of Naval Research under contract N00014-76-C-0316 and the U.S. Department of Energy at the Lawrence Berkeley Laboratory under contract DE-AC03-76SF00098 and the Lawrence Livermore National Laboratory under contract W-7405-ENG-48. Portions of this work have appeared, in preliminary form, in *Proc. Seventh GAMM Conference on Numerical Methods in Fluid Mechanics*, Louvain-la-Neuve, Belgium 1987 and in *Proc. First National Fluid Dynamics Congress*, Cincinnati, OH, 1988.

† Lawrence Livermore National Laboratory, Livermore, California 94550.

Euler's equations) [1]–[3], [13], [17]–[19]. Although there are fewer theoretical results for the random vortex method, it has been shown that the method converges in a two-dimensional domain with a free-space boundary condition [16], [24]. There is also a numerical study of the random vortex method [28] and a detailed numerical investigation of a hybrid vortex sheet/random vortex method applied to flow past a rearward-facing step [32]. However, to date there has been no theoretical or numerical work to investigate the effectiveness of the vortex sheet method in approximating solutions of the Prandtl equations. This is the goal of the present work.

We begin by reviewing the Prandtl equations. Then in § 2 we describe the method and discuss two conditions, based on a priori estimates, that the computational parameters must satisfy. In § 3 we show that the sheet-creation algorithm is equivalent to interpolating the tangential velocity on the wall induced by the current positions of the sheets. In § 4 we examine the error after one timestep of the random walk and sheet-creation process. We prove that this process is a consistent approximation to the exact solution of the diffusion equation with the no-slip boundary condition in the sense that the expected value of the computed solution is the exact solution plus some error which goes to zero as the method is refined. We estimate the consistency error in the L^1, L^2, and L^∞ norms and show that this error is entirely due to our failure to satisfy the no-slip boundary condition exactly. We also derive bounds on the variance of the computed solution and the probability distribution of the error after one timestep of the random walk and sheet-creation process. Finally, in § 5 we present the results of extensive computations using the method to approximate Blasius flow. Our numerical results demonstrate that the computed solution converges to the exact solution, provided the parameters are chosen correctly. We endeavor to develop guidelines for choosing these parameters so that the resulting computation is optimally accurate and efficient. We also examine the merits of using a second-order time-integration scheme, a different sheet-creation algorithm, and a variance-reduction technique known as sheet tagging.

1.1. The Prandtl equations. In two dimensions the Prandtl equations [10], [23], [30], [37] are

$$u_t + uu_x + vu_y = -p_x + \nu u_{yy}, \tag{1.1a}$$

$$u_x + v_y = 0, \tag{1.1b}$$

with boundary conditions

$$u(x, 0, t) = 0, \tag{1.1c}$$

$$v(x, 0, t) = 0, \tag{1.1d}$$

$$\lim_{y\to\infty} u(x, y, t) = U(x, t), \tag{1.1e}$$

where x, y are orthogonal coordinates, u is the velocity component in the x direction, v is the component in the y direction, p is the pressure, ν is the kinematic viscosity, and $U(x, t)$ is the free-stream velocity. The free-stream velocity or "velocity at infinity" is independent of y and assumed to be known. We also assume the flow has constant density $\rho = 1$.

These equations are derived from the Navier–Stokes equations under the assumption that the velocity component perpendicular to the boundary is small relative to the tangential component. They are valid for general laminar flows along curved walls, as long as the boundary layer thickness is small compared to the wall's radius of curvature (White [37, p. 256]). Both the boundary layer equations [30] and the vortex

sheet method [11] may be generalized to three dimensions. For simplicity we work in two dimensions and assume that the boundary is a flat wall of length L, beginning at $(0, 0)$ and extending to $(L, 0)$.

It follows from $p_y = 0$ [30, p. 121] that the pressure is imposed on the flow from outside the boundary layer and hence, is a known quantity. Furthermore, by (1.1b) and (1.1d),

$$v(x, y) = -\int_0^y u_x(x, y')\, dy', \tag{1.2}$$

and we see that equations (1.1a)-(1.1e) are in fact an evolution equation for one unknown u.

It will be useful to consider the Prandtl equations in vorticity formulation:

$$\frac{D\omega}{Dt} = \nu\omega_{yy}, \tag{1.3a}$$

$$\omega = -u_y, \tag{1.3b}$$

$$u_x + v_y = 0, \tag{1.3c}$$

where ω is the vorticity, $D/Dt = \partial_t + u\partial_x + v\partial_y$ is the material derivative, and the boundary conditions remain (1.1c)-(1.1e). By (1.3b) and (1.1e),

$$u(x, y, t) = U(x, t) + \int_y^\infty \omega(x, y', t)\, dy', \tag{1.4}$$

and it follows that all unknown quantities may be recovered from the vorticity. Thus, the Prandtl equations may also be viewed as an evolution equation for the vorticity ω. This evolution is governed by equation (1.3a). In the vortex sheet method we split this equation into two parts:

$$\frac{D\omega}{Dt} = 0, \tag{1.5}$$

$$\omega_t = \nu\omega_{yy}, \tag{1.6}$$

and solve each part separately.

2. The vortex sheet method. The vortex sheet method is a fractional step method in which one step consists of solving (1.5) and the other of solving (1.6) subject to the boundary condition (1.1c). The particles, called "sheets," carry piecewise linear concentrations of vorticity and the velocity field is reconstructed from the particle positions using (1.4). (We use the words "sheets" and "particles" interchangeably.) The solution of (1.5) is obtained by passively advecting the sheets in this velocity field. The solution of (1.6) is obtained by creating sheets at the wall in order to (approximately) satisfy (1.1c) and then giving all sheets a random displacement in the y direction, reflecting those which go below the wall. This creation of sheets mimics the physical creation of vorticity at the wall.

2.1. Notation. Let $P_{\Delta t}u^0$ denote the solution at time Δt of the Prandtl equations (1.1a)-(1.1e) with initial data u^0. Similarly, let $A_{\Delta t}u^0$ denote the solution at time Δt of the inviscid equation

$$\frac{Du}{Dt} = -p_x, \tag{2.1a}$$

(2.1b) $$u_x + v_y = 0,$$

(2.1c) $$u(x, \infty, t) = U(x), \qquad t > 0,$$

(2.1d) $$v(x, 0, t) = 0, \qquad t > 0,$$

with initial data u^0, and let $D_{\Delta t}u^0$ denote the solution of the diffusion equation

(2.2a) $$u_t = \nu u_{yy}$$

(2.2b) $$u(x, 0, t) = 0, \qquad t > 0$$

with initial data u^0. (Here and in what follows we suppress explicit mention of the dependence of the free-steam velocity U on the time t for notational convenience.) The vortex sheet method consists of finding approximations $\tilde{A}_{\Delta t}$, $\tilde{D}_{\Delta t}$ to $A_{\Delta t}$, $D_{\Delta t}$ and then approximating $P_{\Delta t}u^0$ with $\tilde{D}_{\Delta t}\tilde{A}_{\Delta t}u^0$. We write $u^k = (P_{\Delta t})^k u^0$ and $\tilde{u}^k = (\tilde{D}_{\Delta t}\tilde{A}_{\Delta t})^k u^0$. We always use the tilde to denote an approximation. Thus, $\tilde{u}^k$ is an approximation to u^k, $\tilde{\omega}^k$ is an approximation to ω^k, etc.

2.2. The approximate velocity field $\tilde{u}^k$. For now we omit all reference to time and consider the vortex sheet method approximation to the velocity field at a fixed instant in time. The principle common to all vortex methods is that the velocity field is completely determined by the vorticity field and boundary conditions. In the vortex sheet method we replace the integral on the right-hand side of (1.4) by the sum

(2.3) $$\tilde{u}(x, y) = U(x) + \sum_j \omega_j b_h(x - x_j) H(y_j - y),$$

where $H(y)$ is the Heaviside function,

$$H(y) = \begin{cases} 0, & x < 0, \\ 1, & x \geqq 0. \end{cases}$$

The function b_h, called the *cutoff* or *smoothing* function, is defined by $b_h(x) = b(x/h)$, where b can be chosen in many ways. All cutoffs that we consider will be normalized so that $b(0) = 1$ and have compact support,

(2.4) $$b(x) = 0 \quad \text{for } |x| \geqq R.$$

Typically, $R = 1$ or $R = 2$. We discuss cutoffs at length in § 3.

Each term of the sum in (2.3) is referred to as a *vortex sheet.* The jth sheet has center (x_j, y_j), *strength* or *weight* ω_j, and length $2Rh$, where R depends on the particular smoothing function chosen. The parameter h is frequently referred to as the "sheet length," even though the actual length is usually nh for some integer $n > 1$. From (2.3) we see that the jump in $\tilde{u}$ along the jth sheet is $\omega_j b_h(x - x_j)$. This is the motivation for referring to the computational elements as "vortex sheets."

We can rewrite (2.3) as

$$\tilde{u}(x, y) = U(x) + \int_y^\infty \sum_j \omega_j b_h(x - x_j)\delta(y_j - y')\, dy'$$

where δ is the Dirac delta function. Thus, our approximation to the vorticity is

$$\tilde{\omega}(x, y) = \sum_j \omega_j b_h(x - x_j)\delta(y_j - y)$$

and we see that each sheet carries a linear concentration of vorticity that varies like $b_h(x - x_j)$ as we move along the line segment $(-Rh \leqq x - x_j \leqq Rh, y = y_j)$.

Using (1.2) we find that our approximation to v is given by

$$\tilde{v}(x, y) = -U_x(x)y - \sum \omega_j \partial_x b_h(x - x_j) \min(y, y_j).$$

For cutoffs that are not differentiable everywhere (this is the case for the most commonly used cutoff) we can use a divided difference approximation to $\partial_x b_h$:

$$\tilde{v}(x, y) = -U_x(x)y - \sum \omega_j h^{-1}\left[b_h\left(x + \frac{h}{2} - x_j\right) - b_h\left(x - \frac{h}{2} - x_j\right)\right] \min(y, y_j). \tag{2.5}$$

2.3. Integration of the equations in time. Our approximation $(\tilde{u}, \tilde{v})$ to the velocity field (u, v) is completely determined by the positions of the particles (x_j, y_j) and their weights ω_j. Denote the timestep by Δt, the position of the jth particle at time $k\Delta t$ by (x_j^k, y_j^k) and the velocity field derived from these positions by $(\tilde{u}^k, \tilde{v}^k)$. Given $(\tilde{u}^k, \tilde{v}^k)$ the velocity field at the next timestep is determined as follows.

2.3.1. The advection step. The solution of the advection equation (2.1a)-(2.1d) with initial data $(\tilde{u}^k, \tilde{v}^k)$ is found by moving the sheets according to

$$x_j^{k+1/2} = x_j^k + \Delta t \tilde{u}^k(x_j^k, y_j^k), \tag{2.6a}$$

$$y_j^{k+1/2} = y_j^k + \Delta t \tilde{v}^k(x_j^k, y_j^k). \tag{2.6b}$$

Thus, the velocity field after the first of the two fractional steps, denoted by $(\tilde{u}^{k+1/2}, \tilde{v}^{k+1/2})$, is

$$\tilde{u}^{k+1/2}(x, y) = U(x) + \sum_j \omega_j b_h(x - x_j^{k+1/2}) H(y_j^{k+1/2} - y),$$

$$\tilde{v}^{k+1/2}(x, y) = -U_x(x)y - \sum_j \omega_j \partial_x b_h(x - x_j^{k+1/2}) \min(y, y_j^{k+1/2}).$$

Note that since the sheet strengths remain constant in the advection step, $\tilde{\omega}^{k+1/2} \equiv -\partial_y \tilde{u}^{k+1/2}$ approximates the solution of (1.5) with initial data $\tilde{\omega}^k$ in the sense that vorticity is constant on approximate particle paths $(x_j^k, y_j^k) \to (x_j^{k+1/2}, y_j^{k+1/2})$.

In (2.6a), (2.6b) we have used Euler's method to solve the system of ordinary differential equations (ODEs):

$$\frac{dx_j}{dt} = u(x_j, y_j, t), \tag{2.7a}$$

$$\frac{dy_j}{dt} = v(x_j, y_j, t), \tag{2.7b}$$

with the right-hand side replaced by $\tilde{u}^k$ and $\tilde{v}^k$. We can choose to solve the system (2.7a), (2.7b) with a higher-order ODE solver. Extensive theoretical work (e.g., [1], [21]) has demonstrated the effectiveness of using a higher-order ODE solver in the vortex method solution of the Euler equations. However, to date there has been no such work on the use of a higher-order solution of (2.7a), (2.7b) for the vortex sheet method. In § 5.4 we present the results of a numerical experiment, comparing a second-order solution of (2.7a), (2.7b) with the first-order solution given by (2.6a), (2.6b).

2.3.2. The particle-creation and diffusion step. The second part of the splitting procedure is the random walk solution of (2.2a), (2.2b) with initial data $\tilde{u}^{k+1/2}$. This consists of creating sheets at gridpoints on the wall to approximately satisfy the no-slip boundary condition (1.1c), and then letting all sheets undergo a random walk in the y direction, reflecting those sheets that go below the wall.

Choose h so that the wall length L is an integral multiple of h, $L = rh$. Choose equally spaced points $a_1, \cdots, a_r$ on the wall with $a_1 = h/2$, $a_i - a_{i-1} = h$, and $a_r = L - h/2$. We approximately satisfy (1.1c) by requiring that at the end of every timestep, $\tilde{u}^{k+1}(a_i, 0) \approx 0$. This is accomplished as follows. Let $\omega_{\max}$ denote a computational parameter called the *maximum sheet strength.* All ω_j will be chosen so that $|\omega_j| \leqq \omega_{\max}$.

This is the parameter that most directly influences the accuracy of the random walk. Let $u_i = \tilde{u}^{k+1/2}(a_i, 0)$. At each a_i we create $q_i \geqq 0$ sheets with centers $(x_i, y_i) = (a_i, 0)$ and strengths ω_{il} so that

$$\left| \tilde{u}^{k+1/2}(a_j, 0) + \sum_{i=1}^{r} \sum_{l=1}^{q_i} \omega_{il} b_h(a_j - x_i) H(y_i - 0) \right| = \left| u_j + \sum_{i=1}^{r} \sum_{l=1}^{q_i} \omega_{il} b_h(a_j - a_i) \right| \leqq \omega_{\max}$$

for each j. We assume that for a given i the ω_{il} are all equal and sometimes write ω_i rather than ω_{il}.

We have considerable leeway when choosing q_i and ω_i. We therefore discuss this aspect of the algorithm in greater detail. We begin by describing the version originally proposed by Chorin in [9]. We then present a variation on his idea which leads to fewer sheets at the expense of approximating the no-slip boundary condition less accurately. Assume that $R = 1$ in (2.4). Hence, $b_h(a_i - a_j) = \delta_{ij}$, where δ_{ij} is the Kronecker delta function. This is the case for the most commonly used cutoff. In § 3.3 we will remove this assumption and describe a different creation algorithm for which $R = 2$.

PARTICLE-CREATION ALGORITHM A. In this version a sheet is created at a_i if $|u_i| \geqq \varepsilon$ where, for example, ε might be chosen to be on the order of the computer's roundoff error. Let $[x]$ denote the greatest integer less than or equal to x. If $|u_i| \geqq \varepsilon$, then we create

$$q_i = \begin{cases} |u_i|/\omega_{\max} & \text{if } \omega_{\max} \text{ divides } u_i \text{ evenly,} \\ [|u_i|/\omega_{\max} + 1] & \text{otherwise,} \end{cases}$$

sheets at $(a_i, 0)$, each of strength $\omega_i = -u_i/q_i$. Otherwise we set $q_i = 0$ and create no new sheets at $(a_i, 0)$. As we shall see presently, the random walk does not alter the tangential velocity at the wall. Hence, the tangential velocity at time $(k+1)\Delta t$ satisfies

$$|\tilde{u}^{k+1}(a_i, 0)| = |u_i + q_i \omega_i| < \begin{cases} 0, & \text{if } q_i > 0, \\ \varepsilon, & \text{if } q_i = 0. \end{cases} \tag{2.8}$$

PARTICLE-CREATION ALGORITHM B. An alternate version of this particle-creation algorithm was used in [32], as well as in the present work. At the ith gridpoint we create $q_i = [|u_i|/\omega_{\max}]$ sheets, each with strength $\omega_i = -\text{sign}\,(u_i)\omega_{\max}$ and center $(a_i, 0)$. Our approximation to the tangential velocity at time $(k+1)\Delta t$ now satisfies

$$|\tilde{u}^{k+1}(a_i, 0)| = |u_i - q_i \,\text{sign}\,(u_i)\omega_{\max}| < \omega_{\max}. \tag{2.9}$$

Note that Algorithm B is not Algorithm A with $\varepsilon = \omega_{\max}$, since in Algorithm B all sheet strengths have the same magnitude $\omega_{\max}$. In § 5.6 we will show that this diminishes the number of sheets in the flow without adversely affecting the overall accuracy of the method.

Now let every sheet (including those just created) take a random walk in the y direction, reflecting those sheets that go below the wall. The new particle positions are given by

$$x_j^{k+1} = x_j^{k+1/2}, \qquad y_j^{k+1} = |y_j^{k+1/2} + \zeta_j|$$

where ζ_j is a Gaussian-distributed random number with mean 0 and variance $2\nu\Delta t$. The tangential velocity at time $(k+1)\Delta t$ is therefore

$$\begin{aligned} \tilde{u}^{k+1}(x, y) &= U(x) + \sum_{j=1}^{N_k} \omega_j b_h(x - x_j^{k+1/2}) H(|y_j^{k+1/2} + \zeta_j| - y) \\ &\quad + \sum_{i=1}^{r} \sum_{l=1}^{q_i} \omega_i b_h(x - a_i) H(|\zeta_{il}| - y) \\ &= U(x) + \sum_{j=1}^{N_{k+1}} \omega_j b_h(x - x_j^{k+1}) H(y_j^{k+1} - y) \end{aligned} \tag{2.10}$$

where N_k (respectively, N_{k+1}) is the number of sheets in the flow at time $k\Delta t$ (respectively, time $(k+1)\Delta t$). Using the notation introduced in § 2.1 we write $\tilde{u}^{k+1} = \tilde{D}_{\Delta t}\tilde{u}^{k+1/2}$.

It is appropriate to make several remarks here concerning the random walk and sheet-creation algorithm. Let E denote the expected value with respect to the ζ_j and ζ_{il} in (2.10). A consequence of Theorem 4.1 is that for each $x \in [0, L]$,

$$\tilde{u}^{k+1}(x, 0) = 0 \Rightarrow D_{\Delta t}\tilde{u}^{k+1/2}(x, y) = E[\tilde{D}_{\Delta t}\tilde{u}^{k+1/2}(x, y)]. \tag{2.11}$$

We then show that for either of the creation algorithms described above

$$\|D_{\Delta t}\tilde{u}^{k+1/2}(x, y) - E[\tilde{D}_{\Delta t}\tilde{u}^{k+1/2}(x, y)]\|_s = O((h + \omega_{\max})(\nu\Delta t)^{1/2s}) \tag{2.12}$$

for $s = 1, 2, \infty$. Thus, $\tilde{D}_{\Delta t}\tilde{u}^{k+1/2}$ is a consistent approximation to $D_{\Delta t}\tilde{u}^{k+1/2}$ with truncation error $O((h + \omega_{\max})(\nu\Delta t)^{1/2s})$. It follows from (2.11) and (2.12) that this error is entirely due to our failure to satisfy the no-slip boundary condition exactly. In § 3 we will show that the cutoff function determines how well $\tilde{u}^{k+1}(x, 0)$ satisfies the no-slip boundary condition at each point x on the wall.

In the original version of the vortex sheet method, Chorin created twice as many sheets as needed, the total of which was twice that needed to cancel the velocity at the wall [9, p. 423]. He then let these new sheets random walk *without reflection*, throwing away those sheets that went below the wall. (Of course the other sheets underwent a random walk, with reflection, as above.) In a more recent version [11, p. 6], Chorin again created twice as many sheets as needed, but this time he employed a rejection technique to ensure that *exactly half* of these sheets took their random walk in the positive y direction. However, (2.11) and (2.12) show us that these special procedures are unnecessary. It is sufficient to create sheets as in Algorithms A or B above and allow them to undergo a random walk with reflection. These results also demonstrate that the sheet creation and random walk are part of the same process. Together they comprise one step of a two-step fractional-step method, rather than two separate steps of a three-step method. We emphasize that to obtain (2.11) and (2.12) the new sheets must undergo a random walk immediately after being created, without waiting a timestep before diffusing them from the wall.

2.4. The CFL condition. The most well known and perhaps the only universally acknowledged condition on the parameters in the vortex sheet method is the so-called "CFL" condition. For $U_{\max} \equiv \max U(x)$ this condition states that Δt and h must be chosen so that

$$\Delta t U_{\max} \leqq h. \tag{2.13}$$

This requirement was originally proposed by Chorin in [9]. The justification usually given for (2.13) is that we want to ensure that the distance a sheet travels downstream in one timestep is no more than one sheet length. Hence, sheets created at the ith gridpoint must influence the $(i+1)$st gridpoint before moving on downstream. This is essentially an accuracy criterion (as opposed to a stability criterion). It ensures that information propagating in the streamwise direction influences all features of the flow that are at least $O(h)$.

We now offer another justification for (2.13) which takes into account the rate at which vorticity diffuses across the boundary layer in the vortex sheet method. Consider sheets created at the ith gridpoint a_i at a given timestep. Since the velocity is 0 at the wall, the only movement imparted to these sheets at the next timestep will be due to

the random walk. We seek a condition to ensure that these sheets remain in the boundary layer after this random walk. For the flat plate problem, the boundary layer thickness above a point x is

$$\delta_x \approx 5\sqrt{\frac{\nu x}{U(x)}}$$

(Schlichting [30, p. 140]). The random walks are drawn from a Gaussian distribution with mean 0 and standard deviation $\sigma = \sqrt{2\nu\Delta t}$. Thus, if (2.13) holds, then

$$\frac{5}{2}\sigma \leqq 5\sqrt{\frac{\nu h}{2U(x)}} = \delta_{a_1},$$

where $a_1 = h/2$ is the first gridpoint on the wall. This implies that, on the average, more than 98 percent of the sheets created at a_1 will remain in the boundary layer after taking one timestep. Since the boundary layer increases with increasing x for the flat plate problem, this will also be true for the sheets created at the other gridpoints. Similar conditions may be found for other flows; all that is required is an estimate of the boundary layer thickness.

We now propose another constraint which relates $\omega_{\max}$ to h and Δt. Consider the flat plate problem with initial condition

$$u^0(x, y) = \begin{cases} U_{\max}, & y > 0, \\ 0, & y = 0. \end{cases} \tag{2.14}$$

At the first timestep we need to create K sheets of strength $\omega_{\max}$ at each a_i in order to cancel the velocity $U_{\max}$ just above the wall. Thus, after the random walk, we will have approximated $u(a_i, y)$ by a step function $\tilde{u}(a_i, y)$ with K jumps, each of strength $\omega_{\max}$, such that $\tilde{u}(a_i, 0) = 0$ and $\tilde{u}(a_i, \infty) = U_{\max}$. The task is to choose $\omega_{\max}$ (or equivalently K) in a manner consistent with our choice of Δt and h. It is reasonable to require that we resolve u to the same extent that we are resolving features in the streamwise direction. Therefore, let $K = CL/h$ for some constant C, and let $\omega_{\max} = U_{\max}/K$. Using (2.13) we obtain

$$\omega_{\max} \leqq C_0 \frac{h^2}{\Delta t}, \tag{2.15}$$

where $C_0 = 1/CL$ is a constant with dimensions $1/L$. Our experience with the test problem described in § 5 has been that the best results were obtained with $C_0 < 1/L$. The argument presented here can be easily generalized for flows other than the flat plate problem with initial data (2.14). Since

$$\partial_x \omega_j b_h(x - x_j) = O(\omega_{\max}/h),$$

sheets induce local (nonphysical) streamwise gradients in $\tilde{u}$ and $\tilde{\omega}$ that are $O(\omega_{\max}/h)$. Condition (2.15) relates the size of these gradients to the timestep. In § 5 we present numerical results that demonstrate the importance of (2.15).

2.5. Sheet tagging. In [9] Chorin proposed the following "variance-reduction technique." During the sheet-creation process each sheet is assigned a positive integer, called a *tag*, as follows. Let T_k be the last tag assigned at the kth timestep. Then, during the $(k+1)$st timestep, the first sheet created at every gridpoint is assigned the number $T_k + 1$, the second sheet is assigned $T_k + 2$, and so on until all sheets have a tag. Thus, no two sheets created at the same gridpoint will have the same tag, while one sheet at each gridpoint will have the same tag (except, of course, when more sheets

need to be created at one gridpoint than at another). When the sheets undergo the random walk all sheets with the same tag are assigned the same random walk.

The motivation behind this procedure is twofold. First, if we use the vortex sheet method to model flow past an infinite plate with constant free-stream velocity [10, pp. 92–95], then, with sheet tagging, the vortex sheet method reduces to the random walk solution of the heat equation. This eliminates all error due to the advection step. Second, on heuristic grounds it is believed that vorticity leaving the wall diffuses at the same rate everywhere along the wall, and it was thought that the sheet tagging mimicked this process. Our experience has been that, with the exception of the infinite flat plate problem, sheet tagging does not improve the accuracy of the vortex sheet method. In the remarks after the proof of Theorem 4.7, we show how the variance of the solution can actually *increase* when different sheets with the same tag overlap. In § 5.7 we present the results of numerical experiments that substantiate our contention that the sheet-tagging procedure does not improve the accuracy of the method.

3. Smoothing. It is instructive to examine the smoothing process from the following perspective. Given $\tilde{u}^{k+1/2}$ we would ideally like to create sheets at the wall so that $\tilde{u}^{k+1}$ satisfies (1.1c) for all $x \in [0, L]$. Let $g(x) \equiv \tilde{u}^{k+1/2}(x, 0)$ and let ζ and ζ_j be independent, Gaussian-distributed random variables with mean 0 and variance $2\nu\Delta t$. Then

$$\tilde{u}_{\text{opt}}^{k+1}(x, y) \equiv U(x) + \sum \omega_j b_h(x - x_j^{k+1}) H(|y_j^{k+1/2} + \zeta_j| - y) - g(x) H(|\zeta| - y)$$

is the optimal choice for $\tilde{u}^{k+1}$. For $\tilde{u}_{\text{opt}}^{k+1}$ satisfies the no-slip boundary condition exactly at every x and hence, by (2.11),

$$E[\tilde{u}_{\text{opt}}^{k+1}(x, y)] = D_{\Delta t} \tilde{u}^{k+1/2}(x, y)$$

for all $x \in [0, L]$. However, we need to write $\tilde{u}^{k+1}$ as the sum of vortex sheets. Therefore, our task is to find q_i and ω_{il} such that

$$\tilde{g}(x) \equiv -\sum_{i=1}^{r} \sum_{l=1}^{q_i} \omega_{il} b_h(x - a_i) = \sum_{i=1}^{r} u_i' b_h(x - a_i)$$

is a good approximation to g. This amounts to finding a $\tilde{g}$ that interpolates g at the gridpoints a_i. We will now examine several ways in which this can be done.

3.1. Piecewise constant cutoffs. It is natural to consider choices for the cutoff b that result in piecewise constant $\tilde{g}$. For example, let $b(x) = s(x)$, where s is defined by

$$s(x) = H(\tfrac{1}{2} - |x|). \tag{3.1}$$

It has been observed, however, that the vortex sheet method performs poorly when this cutoff is used. We offer the following possible explanation. For this choice of b the tangential velocity is piecewise constant as a function of x. Consequently, our approximation to the derivative $\partial_x \tilde{u}$ in (2.5) is poor. This in turn leads to large errors in the particle paths through (2.6b).

3.2. Piecewise linear smoothing. An obvious improvement over (3.1) is to choose b so that the resulting $\tilde{g}$ is a piecewise linear approximation to g. Thus, let $b(x) = l(x)$, where

$$l(x) \equiv \begin{cases} 1 - |x| & \text{if } |x| \leqq 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

With this choice of smoothing the functions $b_i(x) \equiv b_h(x - a_i)$, $i = 1, \cdots, r$ are a basis for the space of piecewise linear polynomials on $[0, L]$ with breaks at the a_i (see

Schultz [31, § 2.1]). These functions are frequently referred to as "hat" or "tent" functions. This cutoff was first used by Chorin in [9] and is currently the most widely used cutoff. It has been used successfully for a wide variety of problems. For example, see Cheer [5], [6], Ghoniem, Chorin, and Oppenheim [15], Sethian and Ghoniem [32], Sethian [33], [34], and Tiemroth [36].

3.3. Spline smoothing. Finally, we present an algorithm in which $\tilde{g}$ is a piecewise cubic interpolant of g. This may be accomplished in more than one way. For example, we can choose b so that $\tilde{g}$ is only C^1 at the breakpoints a_i, or C^2 at the a_i, etc. See de Boor [4] or Schultz [31] for details. Here we suggest a cutoff $B(x)$ such that $\tilde{g}$ is C^2 at the breakpoints. Define

$$B(x) \equiv \begin{cases} \frac{1}{4}(2-|x|)^3 - (1-|x|)^3, & \text{if } 0 \leqq |x| \leqq 1, \\ \frac{1}{4}(2-|x|)^3, & \text{if } 1 \leqq |x| \leqq 2, \\ 0, & \text{if } |x| > 2. \end{cases} \tag{3.3}$$

For $b(x) = B(x)$ the b_h are known as B-splines [4], [31].

Note that with this choice of b the support of b_h is now $4h$, rather than $2h$. Thus, the sheets overlap each other more than with the piecewise linear cutoff defined by (3.2); each sheet centered at a_i now influences three gridpoints, rather than just one. Thus, in the sheet-creation algorithm we can no longer simply choose ω_i so that $|u_i + q_i\omega_i| < \omega_{\max}$. Instead, we must now solve a system of linear equations

$$\tfrac{1}{4}\alpha_{i-1} + \alpha_i + \tfrac{1}{4}\alpha_{i+1} = u_i, \qquad i = 2, \cdots, r-1$$

and choose q_i, ω_i so that $|\alpha_i + q_i\omega_i| < \omega_{\max}$. We also must decide what to do at the endpoints, $i = 1$ and $i = r$. Endpoint conditions have been extensively studied in the context of the spline interpolation of a function (for example, see de Boor [4]).

For the cubic spline cutoff defined by (3.3) the sheet-creation algorithm should result in fewer sheets. For example, let $L = 1$ and consider the periodic problem in which $x = L$ is identified with $x = 0$ so we have $a_1 = a_{r+1}$, etc. Let $h = 0.2$, $\omega_{\max} = 3^{-1}$, and $u_i = 1.0$, $i = 1, \cdots, r$. With the piecewise linear cutoff we will create three sheets at every gridpoint, each of strength $\omega_i = 3^{-1}$. On the other hand, with the B-spline cutoff we will create two sheets at every gridpoint, each of strength $\omega_i = 3^{-1}$.

Extensive computations with the B-spline cutoff show that, provided (2.15) is satisfied, we do indeed produce fewer sheets. For appropriate choices of the parameters the B-spline cutoff leads to 20–25 percent fewer sheets. However, when the vortex sheet method is used as a stand-alone method for solving the Prandtl equations (as it is in § 5) the added cost of evaluating (3.3) as compared to evaluating (3.2) seems to offset the savings due to fewer sheets. On the other hand, when the vortex sheet method is used in conjunction with the random vortex method, with the sheets eventually becoming vortices, the B-spline cutoff also results in fewer vortices. However, it is not clear whether this will lead to a more economical computation since the B-spline sheets have $\frac{1}{2}$ times more circulation per unit strength than the piecewise linear sheets. The B-spline cutoff does not seem to improve the accuracy of the vortex sheet method or its rate of convergence, most likely because the dominant source of error comes from the use of the random walk to solve (2.2a).

3.4. Smoothing perpendicular to the wall. Let us now consider the smoothness of our approximation as a function of y. From (2.3) it is apparent that $\tilde{u}^k(x, y)$ is a step function in y for each fixed $x \in [0, L]$. This is analogous to the random gradient method described in [25]. In both cases it is natural to inquire what happens if we use something smoother than a step function. In unpublished work, the author replaced the step

function approximation described in § 2 of [25] with a piecewise linear function and used it to compute the traveling wave solution described in § 9.1 of [25]. The piecewise linear version produced a significantly better approximation to the initial data at time $t=0$. After several timesteps, however, the error was comparable to that incurred by the original method. This is because the accuracy in approximating the initial data by a step function is $O(N^{-1})$, where N is the number of particles (jumps in the function). When we use a piecewise linear approximation this accuracy improves. This accounts for the observed decrease in the initial error. However, the error due to solving the diffusion equation by random walking the particles is $O(\sqrt{N}^{-1})$ (see [20], [25], [29], and the results in § 4 below). This rapidly overwhelms the error due to our interpolation of the solution. Similarly, with the vortex sheet method any improvement in accuracy obtained by using a higher-order approximation to $u^k(x, y)$ as a function of y is quickly lost by the random walk.

4. The error due to the random walk operator $\tilde{D}_{\Delta t}$. In this section we prove several bounds on the error that result from using the random walk and sheet-creation algorithm to approximate the exact solution of (2.2a)-(2.2b). The purpose of these estimates is to examine the relationship between the parameters $\nu, \Delta t, \omega_{\max}$, and h and the error made by one timestep of the random walk and sheet-creation process. This analysis also clarifies the relationship between the accuracy with which the no-slip boundary condition is satisfied and the error due to the random walk.

We begin by proving that the random walk and sheet-creation algorithm is consistent in the L^1, L^2, and L^∞ norms:

$$\|E\tilde{D}_{\Delta t}u^0 - D_{\Delta t}u^0\|_s = O((\omega_{\max}+h)(\nu\Delta t)^{1/2s}) \tag{4.1}$$

for $s=1, 2, \infty$. We define u^0, $\tilde{D}_{\Delta t}u^0$ and what we mean by $E\tilde{D}_{\Delta t}u^0$ below. We then derive bounds on the variance of $\tilde{D}_{\Delta t}u^0$ and on the probability distribution of the L^2 error $\|D_{\Delta t}u^0 - \tilde{D}_{\Delta t}u^0\|_2$. These latter bounds are essentially a generalization of Hald's work in [20]. To establish them we first prove

$$\int_0^\infty \operatorname{var} H(|y_j+\zeta_j|-y)\,dy = F(y_j)\sqrt{2\nu\Delta t} \tag{4.2}$$

for some function F, which is bounded between $(2-\sqrt{2})/\sqrt{\pi}$ and $4/\sqrt{\pi}$. (For $y_j=0$ this is Hald's result.) Then, using the fact that the random walks ζ_j taken at a given timestep are independent, we use (4.2) to show

$$\|\operatorname{var}(\tilde{D}_{\Delta t}u^0)\|_1 \leqq C_\nu\sqrt{2\nu\Delta t}\omega_{\max}. \tag{4.3}$$

It follows from (4.1) and (4.3) that the error in the diffusion step decreases as $\nu\to 0$. Thus, we get better results from the random walk at small viscosities for no additional work.

It will be apparent from the proof that (4.3) exhibits the correct dependence on the parameters ν, Δt, and $\omega_{\max}$. In this regard we note that Hald has shown the following:

$$\int_0^\infty \operatorname{var}\Theta(y)\,dy = \frac{(2-\sqrt{2})}{\sqrt{\pi}}\sqrt{2\nu\Delta t}\omega_{\max},$$

where Θ is the random walk solution after one timestep of length Δt to

$$\begin{aligned}
&\Theta_t = \nu\Theta_{yy}, && 0\leqq y<\infty,\\
&\Theta(0, t)=1, && 0<t,\\
&\Theta(y, 0)=0,
\end{aligned}$$

and we have written $\omega_{\max}$ instead of N^{-1} since all of Hald's particles have strength N^{-1}. (This is the second equation in the proof of Theorem 2 in [20].) In other words, the L^1 norm of the variance of Θ is *exactly* $O(\sqrt{\nu \Delta t}\,\omega_{\max})$. In the case of (4.3) equality, in general, fails to hold, but the amount by which the right- and left-hand sides differ does not depend on ν, Δt, or $\omega_{\max}$.

4.1. Notation. In what follows we let u^0 be an arbitrary function of the form

$$u^0(x, y) = U(x) + \sum_{j=1}^{N} \omega_j b_h(x - x_j) H(y_j - y), \tag{4.4}$$

with $|\omega_j| \leqq \omega_{\max}$ for all j. We assume the use of piecewise linear smoothing: $b_h(x) \equiv b(x/h)$ with b defined by (3.2). However, all of the results in this section remain valid with minor modifications if B-spline smoothing is used. We also assume that the y_j in (4.4) have been chosen so that

$$y_i \neq y_j, \qquad i \neq j. \tag{4.5}$$

Therefore, the vorticity field ω^0 corresponding to (4.4) is a measure whose mass is

$$\|\omega^0\| = \int_0^L \int_0^\infty \left| \sum_j \omega_j b_h(x - x_j) \delta(y_j - y) \right| dx\, dy = \sum_j |\omega_j| \int_0^L b_h(x - x_j)\, dx, \tag{4.6}$$

where the sum is over all sheets in the flow. We remark that in the vortex sheet method (without sheet tagging) the probability that $y_i^k = y_j^k$ or $y_i^{k+1/2} = y_j^{k+1/2}$ for some $i \neq j$ is 0. Hence, assumption (4.5) is reasonable.

We assume that the a_i have been chosen as described in § 2.3.2. Let ζ_j and ζ_{il} be independent, Gaussian-distributed random variables with mean 0 and variance $2\nu\Delta t$. Define $\tilde{D}_{\Delta t} u^0$ by

$$\begin{aligned} \tilde{D}_{\Delta t} u^0(x, y) = U(x) &+ \sum_{j=1}^{N} \omega_j b_h(x - x_j) H(|y_j + \zeta_j| - y) \\ &+ \sum_{i=1}^{r} \sum_{l=1}^{q_i} \omega_{il} b_h(x - a_i) H(|\zeta_{il}| - y), \end{aligned} \tag{4.7}$$

where q_i and ω_{il} are chosen as in Algorithm A, Algorithm B, or by solving the no-slip boundary condition exactly at each a_i,

$$\sum_{l=1}^{q_i} \omega_{il} = -u^0(a_i, 0), \qquad i = 1, \cdots, r. \tag{4.8}$$

We let $E\tilde{D}_{\Delta t} u^0$ denote the expectation of $\tilde{D}_{\Delta t} u^0$ taken with respect to the random walks ζ_j, ζ_{il}, and

$$\operatorname{var}(\tilde{D}_{\Delta t} u^0) = E[(\tilde{D}_{\Delta t} u^0 - E\tilde{D}_{\Delta t} u^0)^2]$$

denote the variance of $\tilde{D}_{\Delta t} u^0$ with respect to these random walks.

We will obtain bounds in the $L^1(\Omega)$, $L^2(\Omega)$, and $L^\infty(\Omega)$ norms, where $\Omega = [0, L] \times [0, \infty)$ is our computational domain. For a function $f(x, y)$ defined on Ω we will sometimes need to consider the sup norm of f as a function defined on the wall alone,

$$\|f(\cdot, 0)\|_\infty \equiv \sup_{x \in [0, L]} |f(x, 0)|.$$

Furthermore, if f is piecewise C^1 on $[0, L]$ and C^1 on the open intervals (b_{i-1}, b_i), where $0 = b_1 \leqq \cdots \leqq b_m = L$ then, following Schultz [31, p. 2], we define

$$\|\partial_x f(\cdot, 0)\|_\infty \equiv \max_i \sup_{b_{i-1} \leqq x \leqq b_i} |\partial_x f(x, 0)|.$$

The L^2 norm of $f(x, 0)$ and $\partial_x f(x, 0)$ are defined similarly.

4.2. The analysis. We begin by showing that the exact solution $D_{\Delta t}u^0$ is the expected value of our random walk process when the no-slip boundary condition is solved exactly at each point on the wall.

THEOREM 4.1. *Let u^0 be given by* (4.4) *and let $D_{\Delta t}u^0$ denote the exact solution of* (2.2a)-(2.2b) *with initial data u^0. Let ζ, ζ_j be Gaussian-distributed random variables with mean* 0 *and variance $2\nu\Delta t$ and let E denote the expected value with respect to these random variables. Then*

$$D_{\Delta t}u^0(x,y) = E\left[U(x) + \sum_j \omega_j b_h(x-x_j) H(|y_j+\zeta_j|-y) - u^0(x,0) H(|\zeta|-y) \right]. \tag{4.9}$$

Proof. The exact solution to (2.2a)-(2.2b) on the half plane $y \geqq 0$ can be found by extending the initial data antisymmetrically about $x=0$,

$$u^0(x,y) \equiv -u^0(x,-y), \qquad y<0,$$

and solving the diffusion equation on the *entire* real line $-\infty < y < \infty$ with initial data u^0. We can write down the solution of this latter problem by using the fundamental solution of the heat equation on $\mathbb{R}$,

$$u(x,y,t) = \frac{1}{\sqrt{4\pi\nu t}} \int_{-\infty}^{\infty} \exp\left(\frac{-(y-s)^2}{4\nu t}\right) u^0(x,s)\, ds.$$

Now let $c_j = \omega_j b_h(x-x_j)$, where we have suppressed the dependence on x for notational convenience. Then

$$\begin{aligned}
D_{\Delta t}u^0(x,y) &= \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_0^{\infty} \exp\left(\frac{-(y-s)^2}{4\nu\Delta t}\right)\left(U(x) + \sum_j c_j H(y_j - s)\right) ds \\
&\quad - \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-\infty}^{0} \exp\left(\frac{-(y-s)^2}{4\nu\Delta t}\right)\left(U(x) + \sum_j c_j H(y_j + s)\right) ds \\
&= \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-y}^{\infty} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) U(x)\, d\zeta \\
&\quad - \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-\infty}^{-y} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) U(x)\, d\zeta \\
&\quad + \sum_j c_j \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{y-y_j}^{y} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) d\zeta \\
&\quad + \sum_j c_j \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-y}^{-y-y_j} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) d\zeta \\
&= \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-\infty}^{\infty} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) U(x)\, d\zeta \\
&\quad - \frac{2}{\sqrt{4\pi\nu\Delta t}} \int_{-\infty}^{-y} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) U(x)\, d\zeta \\
&\quad + \sum_j c_j \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{y-y_j}^{\infty} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) d\zeta \\
&\quad + \sum_j c_j \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-\infty}^{-y-y_j} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) d\zeta \\
&\quad - \sum_j c_j \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{y}^{\infty} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) d\zeta - \sum_j c_j \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-\infty}^{-y} \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) d\zeta \\
&= U(x) + \sum_j c_j E[H(|y_j+\zeta|-y)] - u^0(x,0) E[H(|\zeta|-y)].
\end{aligned}$$

It is apparent from (4.9) that the amount by which $\tilde{D}_{\Delta t}u^0$ differs from $D_{\Delta t}u^0$ depends on how well we approximate $u^0(x,0)$ by creating new sheets at the wall. We now show that the maximum value of this difference is $O(h+\omega_{\max})$.

THEOREM 4.2. *Let u^0 and $\tilde{D}_{\Delta t}u^0$ be defined by* (4.4) *and* (4.7), *respectively. Then*

$$\|E\tilde{D}_{\Delta t}u^0 - D_{\Delta t}u^0\|_\infty \leqq \tfrac{1}{2}\|\partial_x u^0(\cdot,0)\|_\infty h + \omega_{\max}. \tag{4.10}$$

Proof. Let $u_i' = -\sum_{l=1}^{q_i}\omega_{il}$. Then, since ζ and the ζ_{il} are identically distributed, (4.7) and (4.9) imply that

$$E\tilde{D}_{\Delta t}u^0(x,y) - D_{\Delta t}u^0(x,y) = -\left[\sum_{i=1}^{r} u_i' b_h(x-a_i) - u^0(x,0)\right] EH(|\zeta|-y). \tag{4.11}$$

For all y, $|EH(|\zeta|-y)| \leqq 1$, and hence

$$\|E\tilde{D}_{\Delta t}u^0 - D_{\Delta t}u^0\|_\infty \leqq \left|\sum_{i=1}^{r} u_i' b_h(x-a_i) - u^0(x,0)\right|.$$

Define $u_i = u^0(a_i,0)$. By definition of the u_i' and one of (2.8), (2.9), or (4.8) we have

$$|u_i' - u_i| = |q_i\omega_i + u_i| < \omega_{\max}. \tag{4.12}$$

Note that for all x, $\sum_{i=1}^{r}|b_h(x-a_i)| = 1$. Therefore,

$$\begin{aligned}\left|\sum_{i=1}^{r} u_i' b_h(x-a_i) - u^0(x,0)\right| &\leqq \left|\sum_{i=1}^{r} u_i' b_h(x-a_i) - \sum_{i=1}^{r} u_i b_h(x-a_i)\right| \\ &\quad + \left|\sum_{i=1}^{r} u_i b_h(x-a_i) - u^0(x,0)\right| \\ &\leqq \omega_{\max} + \tfrac{1}{2}\|\partial_x u^0(\cdot,0)\|_\infty h,\end{aligned}$$

where we have used the fact that $\sum u_i b_h(x-a_i)$ is the piecewise linear interpolant of $u^0(x,0)$ with breaks at the a_i and Exercise 2.3 of Schultz [31] to obtain

$$\left|\sum_{i=1}^{r} u_i b_h(x-a_i) - u^0(\cdot,0)\right| \leqq \frac{1}{2}\|\partial_x u^0(\cdot,0)\|_\infty h. \tag{4.13}$$

We now prove that $\tilde{D}_{\Delta t}u^0$ is a consistent approximation to $D_{\Delta t}u^0$ in the L^1 norm.

THEOREM 4.3. *Let u^0 and $\tilde{D}_{\Delta t}u^0$ be defined by* (4.4) *and* (4.7), *respectively. Then*

$$\|E\tilde{D}_{\Delta t}u^0 - D_{\Delta t}u^0\|_1 \leqq \frac{L}{\sqrt{\pi}}\left\{\frac{1}{2}\|\partial_x u^0(\cdot,0)\|_\infty h + \omega_{\max}\right\}\sqrt{4\nu\Delta t}.$$

Proof. Let u_i and u_i' be as in the proof of Theorem 4.2. From (4.11) we obtain

$$\|E\tilde{D}_{\Delta t}u^0 - D_{\Delta t}u^0\|_1 = \int_0^L \left|\sum_{i=1}^{r} u_i' b_h(x-a_i) - u^0(x,0)\right| dx \int_0^\infty EH(|\zeta|-y)\, dy. \tag{4.14}$$

We use (4.12), (4.13), $rh = L$, and $\int_0^L |b_h(x-a_i)|dx = h$ to estimate the first integral on the right in (4.14):

$$\begin{aligned}\int_0^L \left|\sum_{i=1}^{r} u_i' b_h(x-a_i) - u^0(x,0)\right| dx &\leqq \int_0^L \left|\sum_{i=1}^{r} u_i' b_h(x-a_i) - \sum_{i=1}^{r} u_i b_h(x-a_i)\right| dx \\ &\quad + \int_0^L \left|\sum_{i=1}^{r} u_i b_h(x-a_i) - u^0(x,0)\right| dx \\ &\leqq L\omega_{\max} + \frac{L}{2}\|\partial_x u^0(\cdot,0)\|_\infty h.\end{aligned}$$

In order to estimate the second integral, we write

$$EH(|\zeta|-y)=\frac{1}{\sqrt{4\pi\nu\Delta t}}\int_{-\infty}^{\infty}H(|\zeta|-y)\exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right)d\zeta=\int_{y/\sqrt{2\nu\Delta t}}^{\infty}\frac{2e^{-\zeta^2/2}}{\sqrt{2\pi}}\,d\zeta, \tag{4.15}$$

whereby

$$\int_0^{\infty}EH(|\zeta|-y)\,dy=\int_0^{\infty}\int_{y/\sqrt{2\nu\Delta t}}^{\infty}\frac{2e^{-\zeta^2/2}}{\sqrt{2\pi}}\,d\zeta\,dy=\frac{\sqrt{4\nu\Delta t}}{\sqrt{\pi}}\int_0^{\infty}\int_y^{\infty}e^{-\zeta^2/2}\,d\zeta\,dy=\frac{\sqrt{4\nu\Delta t}}{\sqrt{\pi}}.$$

The last equality follows from integration by parts with respect to y.

Next we prove the consistency of $\tilde{D}_{\Delta t}$ in the L^2 norm. This theorem will also help us establish a bound on the probability distribution of the error in the L^2 norm (Theorem 4.8).

THEOREM 4.4. *Let u^0 and $\tilde{D}_{\Delta t}u^0$ be defined by* (4.4) *and* (4.7), *respectively. Then*

$$\|E\tilde{D}_{\Delta t}u^0-D_{\Delta t}u^0\|_2\leqq\left(\frac{2(\sqrt{2}-1)}{\sqrt{\pi}}\right)^{1/2}\left\{\sqrt{L}\omega_{\max}+\frac{1}{\pi}\|\partial_x u^0(\cdot,0)\|_2 h\right\}(2\nu\Delta t)^{1/4}. \tag{4.16}$$

Proof. Again let u_i and u_i' be defined as in the proof of Theorem 4.2. From (4.11) we find

$$\|E\tilde{D}_{\Delta t}u^0-D_{\Delta t}u^0\|_2=\left(\int_0^L\left(\sum_{i=1}^r u_i'b_h(x-a_i)-u^0(x,0)\right)^2dx\right)^{1/2}\times\left(\int_0^{\infty}(EH(|\zeta|-y))^2\,dx\right)^{1/2}.$$

The right-hand side of (4.16) is the product of two one-dimensional norms. The first of these is

$$\left\|\sum_{i=1}^r u_i'b_h(\cdot-a_i)-u^0(\cdot,0)\right\|_2\leqq\left\|\sum_{i=1}^r u_i'b_h(\cdot-a_i)-\sum_{i=1}^r u_ib_h(\cdot-a_i)\right\|_2+\left\|\sum_{i=1}^r u_ib_h(\cdot-a_i)-u(\cdot,0)\right\|_2.$$

We can evaluate the first term on the right by using (4.12) and the fact that $\sum b_h(x-a_i)\leqq 1$ for all x:

$$\left\|\sum_{i=1}^r u_i'b_h(\cdot-a_i)-\sum_{i=1}^r u_ib_h(\cdot-a_i)\right\|_2\leqq\sqrt{L}\omega_{\max}.$$

To evaluate the second term we use Theorem 2.4 of Schultz [31] to bound the L^2 norm of the difference between $u^0(x,0)$ and its piecewise linear interpolant $\sum u_ib_h(x-a_i)$,

$$\left\|\sum_{i=1}^r u_ib_h(\cdot-a_i)-u^0(\cdot,0)\right\|_2\leqq\frac{1}{\pi}\|\partial_x u^0(\cdot,0)\|_2 h.$$

We evaluate the other expression on the right-hand side of (4.16) by using (4.15) and Lemma 1 of Hald [20] with $a=b=2\nu\Delta t$,

$$\left(\int_0^{\infty}(EH(|\zeta|-y))^2\,dy\right)^{1/2}=\left(\int_0^{\infty}\int_{y/\sqrt{2\nu\Delta t}}^{\infty}\frac{2e^{-\zeta^2/2}}{\sqrt{2\pi}}\,d\zeta\int_{y/\sqrt{2\nu\Delta t}}^{\infty}\frac{2e^{-\zeta^2/2}}{\sqrt{2\pi}}\,d\zeta\,dy\right)^{1/2}=\left(\frac{2(\sqrt{2}-1)}{\sqrt{\pi}}\right)^{1/2}(2\nu\Delta t)^{1/4}.$$

In order to establish a bound on $\operatorname{var} \tilde{D}_{\Delta t} u^0$ in the L^1 norm we will need the following two lemmas, both of which are generalizations of previous work by Hald. The first lemma is a straightforward extension of Lemma 1 in [20] and we refer the reader to it for details of the proof.

LEMMA 4.5. *For $y \geqq 0$ define*

$$F(y) = \frac{2}{\pi}(\sqrt{2} - e^{y^2/2})\left(\frac{\sqrt{\pi}}{\sqrt{2}} + \int_0^y e^{-\zeta^2/2}\, d\zeta\right) + \frac{2\sqrt{2}}{\pi}\int_y^{\sqrt{2}y} e^{\zeta^2/2}\, d\zeta. \tag{4.17}$$

Then $F(0) = (2-\sqrt{2})/\sqrt{\pi}$, $F(\infty) = 4/\sqrt{\pi}$, and, for all $y \in [0, \infty)$,

$$\frac{(2-\sqrt{2})}{\sqrt{\pi}} \leqq F(y) \leqq \frac{4}{\sqrt{\pi}}. \tag{4.18}$$

Furthermore, for all $y_j \geqq 0$ we have

$$F(y_j) = \int_0^\infty \int_y^\infty \frac{2e^{-(\zeta - y_j)^2/2}}{\sqrt{2\pi}}\, d\zeta \int_0^y \frac{2e^{-(\zeta - y_j)^2/2}}{\sqrt{2\pi}}\, d\zeta\, dy.$$

We also need to establish a bound on the L^1 norm of the variance of the independent summands $H(|y_j + \zeta_j| - y)$. Hald proved this lemma in [20] for the case when $y_j = 0$.

LEMMA 4.6. *Let $y_j \geqq 0$, $F(y_j)$ be defined by* (4.17) *and ζ be a Gaussian random variable with mean* 0 *and variance $2\nu\Delta t$. Then*

$$\int_0^\infty \operatorname{var} H(|y_j + \zeta| - y)\, dy \leqq F(y_j/\sqrt{2\nu\Delta t})\sqrt{2\nu\Delta t}.$$

Proof. First note that

$$EH(|y_j + \zeta| - y) = \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-\infty}^\infty H(|y_j + \zeta| - y) \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) d\zeta = P(|y_j + \zeta| \geqq y).$$

Hence, since $H(y)^2 = H(y)$ and $1 - P(a \geqq b) = P(a < b)$, we have

$$\operatorname{var} H(|y_j + \zeta| - y) = P(|y_j + \zeta| \geqq y) P(|y_j + \zeta| < y).$$

Furthermore,

$$\begin{aligned} P(|y_j + \zeta| \geqq y) &= \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-\infty}^\infty H(|y_j + \zeta| - y) \exp\left(\frac{-\zeta^2}{4\nu\Delta t}\right) d\zeta \\ &= \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_y^\infty \exp\left(\frac{-(\zeta - y_j)^2}{4\nu\Delta t}\right) d\zeta \\ &\quad + \frac{1}{\sqrt{4\pi\nu\Delta t}} \int_{-\infty}^{-y} \exp\left(\frac{-(\zeta - y_j)^2}{4\nu\Delta t}\right) d\zeta \\ &\leqq \frac{2}{\sqrt{4\pi\nu\Delta t}} \int_y^\infty \exp\left(\frac{-(\zeta - y_j)^2}{4\nu\Delta t}\right) d\zeta, \end{aligned}$$

since the Gaussian curve is centered at $\zeta = y_j$. Similarly,

$$P(|y_j + \zeta| < y) \leqq \frac{2}{\sqrt{4\pi\nu\Delta t}} \int_0^y \exp\left(\frac{-(\zeta - y_j)^2}{4\nu\Delta t}\right) d\zeta.$$

Now let $z_j = y_j/\sqrt{2\nu\Delta t}$. Then, using Lemma 4.5, we find

$$\begin{aligned}\int_0^\infty \operatorname{var} H(|y_j+\zeta|-y)\,dy &= \int_0^\infty P(|y_j+\zeta|\geqq y)P(|y_j+\zeta|<y)\,dy\\ &\leqq \int_0^\infty \left\{\frac{2}{\sqrt{4\pi\nu\Delta t}}\int_y^\infty \exp\left(\frac{-(\zeta-y_j)^2}{4\nu\Delta t}\right)d\zeta\right.\\ &\qquad \left.\times\frac{2}{\sqrt{4\pi\nu\Delta t}}\int_0^y \exp\left(\frac{-(\zeta-y_j)^2}{4\nu\Delta t}\right)d\zeta\right\}dy\\ &= \int_0^\infty\int_{y/\sqrt{2\nu\Delta t}}^\infty \frac{2e^{-(\zeta-z_j)^2/2}}{\sqrt{2\pi}}\,d\zeta\int_0^{y/\sqrt{2\nu\Delta t}}\frac{2e^{-(\zeta-z_j)^2/2}}{\sqrt{2\pi}}\,d\zeta\,dy\\ &= \sqrt{2\nu\Delta t}F(z_j).\end{aligned}$$

Our bound on the L^1 norm of the variance of $\tilde{D}_{\Delta t}u^0$ now follows easily.

THEOREM 4.7. *Let u^0 and $\tilde{D}_{\Delta t}u^0$ be given by* (4.4) *and* (4.7), *respectively. Then the inequality in* (4.3) *holds with*

$$C_\nu = \frac{4}{\sqrt{\pi}}\left\{\|\omega^0\| + \frac{2}{3}L\|u^0(\cdot,0)\|_\infty\right\}.$$

Proof. Since the ζ_j and ζ_{il} are independent, identically distributed random variables, we have

$$\begin{aligned}\|\operatorname{var}(\tilde{D}_{\Delta t}u^0)\|_1 &= \sum_j \omega_j^2\int_0^L b_h^2(x-x_j)\,dx\int_0^\infty \operatorname{var} H(|y_j+\zeta|-y)\,dy\\ &\quad + \sum_{i=1}^r\sum_{l=1}^{q_i}\omega_{il}^2\int_0^L b_h^2(x-a_i)\,dx\int_0^\infty \operatorname{var} H(|\zeta|-y)\,dy.\end{aligned}\tag{4.19}$$

Since $0\leqq b_h(x)\leqq 1$ for all x, we have

$$\sum_j \omega_j^2\int_0^L b_h^2(x-x_j)\,dx \leqq \omega_{\max}\sum_j|\omega_j|\int_0^L b_h(x-x_j)\,dx = \omega_{\max}\|\omega^0\|.$$

For the piecewise linear b_h we have

$$\int_0^L b_h^2(x-a_i)\,dx = h\int_{-1}^1 l^2(x)\,dx \leqq \frac{2}{3}h,$$

where l is defined by (3.2). By definition of the q_i and ω_i we find

$$\sum_{l=1}^{q_i}\omega_{il}^2 \leqq \omega_{\max}\sum_{l=1}^{q_i}|\omega_{il}| \leqq \omega_{\max}|u^0(a_i,0)| \leqq \omega_{\max}\|u^0(\cdot,0)\|_\infty.$$

Using Lemma 4.6, $rh = L$, and (4.18), we now obtain (4.3),

$$\|\operatorname{var}(\tilde{D}_{\Delta t}u^0)\|_1 \leqq F(\infty)\{\|\omega^0\| + \tfrac{2}{3}L\|u^0(\cdot,0)\|_\infty\}\sqrt{2\nu\Delta t}\,\omega_{\max} \leqq C_\nu\sqrt{2\nu\Delta t}\,\omega_{\max}.$$

It is apparent from (4.3) that the L^1 norm of var $\tilde{D}_{\Delta t}u^0$ goes to zero like $\sqrt{\nu}$. In other words, the statistical fluctuations in our random walk approximation to $D_{\Delta t}u^0$ diminish with diminishing ν. Furthermore, a careful inspection of the proof will reveal that the bound in (4.3) exhibits the correct dependence on ν, Δt, and $\omega_{\max}$. In other words, the bound is sharp. Also note that this proof relies on the independence of the

random walks. When sheet tagging is used this independence is lost since sheets with the same tag overlap. This loss of independence results in a term of the form

$$\int_0^L \int_0^\infty \sum_{j\neq l} \omega_j \omega_l b_h(x-x_j) b_h(x-x_l) E[H(|y_j+\zeta_j|-y) H(|y_l+\zeta_l|-y)]\, dy\, dx$$
$$-\int_0^L \int_0^\infty \sum_{j\neq l} \omega_j \omega_l b_h(x-x_j) b_h(x-x_l) E[H(|y_j+\zeta_j|-y)] E[H(|y_l+\zeta_l|-y)]\, dy\, dx,$$

being added to the right-hand side of (4.19). (Here the sum is over all sheets, new and old.) Since this term will, in general, not be negative it follows that the sheet tagging can in fact *increase* the variance. This observation is corroborated by the numerical results presented in § 5.7.

Finally, we derive a bound on the probability distribution of the error in the L^2 norm. Our proof is based on ideas found in the proof of Theorem 2 of Hald [20].

THEOREM 4.8. *Assume that* $h^2 \leqq \omega_{\max} \leqq 1$, *let* u^0 *and* $\tilde{D}_{\Delta t} u^0$ *be defined by* (4.4) *and* (4.7), *respectively, and define*

$$C_D = C_\nu + \frac{2(\sqrt{2}-1)}{\sqrt{\pi}} (\sqrt{L} + \pi^{-1} \|\partial_x u^0(\cdot, 0)\|_\infty)^2.$$

Then for all $\gamma > 0$,

$$P(\|\tilde{D}_{\Delta t} u^0 - D_{\Delta t} u^0\|_2 \geqq \gamma \sqrt{\omega_{\max}}) \leqq C_D \sqrt{2\nu \Delta t}\, \gamma^{-2}. \tag{4.20}$$

Remark. By using the well-known inequality

$$E[Z] \leqq a\left(1 + \sum_{r=1}^{\infty} P(Z \geqq ra)\right)$$

for all random variables $Z \geqq 0$ and real numbers $a \geqq 0$ (see [12]) we can use (4.20) to establish a bound on the expected value of the error in the L^2 norm,

$$E\|\tilde{D}_{\Delta t} u^0 - D_{\Delta t} u^0\|_2 \leqq (1+2C_D)\sqrt{\omega_{\max}} (2\nu \Delta t)^{1/4}.$$

Proof. Chebyshev's inequality (Feller [14, p. 151]) implies

$$P(\|\tilde{D}_{\Delta t} u^0 - D_{\Delta t} u^0\|_2 \geqq \gamma \sqrt{\omega_{\max}}) \leqq E[\|\tilde{D}_{\Delta t} u^0 - D_{\Delta t} u^0\|_2^2] (\omega_{\max} \gamma^2)^{-1}.$$

Since $E(\tilde{D}_{\Delta t} u^0 - D_{\Delta t} u^0)^2 = \operatorname{var}(\tilde{D}_{\Delta t} u^0) + (E\tilde{D}_{\Delta t} u^0 - D_{\Delta t} u^0)^2$ we can use Fubini's theorem to obtain

$$E[\|\tilde{D}_{\Delta t} u^0 - D_{\Delta t} u^0\|_2^2] = \|\operatorname{var} \tilde{D}_{\Delta t}\|_1 + \|E\tilde{D}_{\Delta t} u^0 - D_{\Delta t} u^0\|_2^2.$$

The inequality in (4.20) now follows from Theorems 4.4 and 4.7, and our assumption that $h^2 \leqq \omega_{\max}$.

4.3. Remarks. Theorems 4.2, 4.3, and 4.4 state that our random walk approximation $\tilde{D}_{\Delta t} u^0$ to the solution $D_{\Delta t} u^0$ is consistent in the L^1, L^2, and L^∞ norms, respectively. The term in (4.10) that is $O(\omega_{\max})$ is due to our failure to satisfy the no-slip boundary condition exactly at the gridpoints a_i. The term that is $O(h)$ is due to using a piecewise linear interpolation to the tangential velocity $u^0(x, 0)$ at the wall. If we use a higher-order interpolation, the order of this last error increases. For example, with B-spline smoothing it is $O(h^2)$. However, numerical results [26] have demonstrated that B-spline smoothing does not lead to an appreciable improvement in the overall accuracy of the method (although it may reduce the total number of sheets in the flow). This is because, as with all Monte Carlo methods, the dominant source of error is that due to the random walk.

Assume for the moment that we have the necessary estimates for the splitting error, the error due to approximating $A_{\Delta t}$ by $\tilde{A}_{\Delta t}$, and the stability of $A_{\Delta t}$. Then, by individually considering the errors made at each timestep, we can use (4.20) and the assumption that $\Delta t = O(\omega_{\max}^{2/3})$ to prove that the expected value of the error in the L^2 norm after k time steps is $O(\Delta t) + O(\omega_{\max}^{1/3}) + O(h^q)$, where $q > 0$ would depend on our estimate of the error due to approximating $A_{\Delta t}$ by $\tilde{A}_{\Delta t}$. (For an example of this type of argument see § 3 of Puckett [25].) It has recently been shown that, for the free-space problem, the random vortex method converges like $O(\log N/\sqrt{N})$ in the L^2 norm (Long [24]). This corresponds to a rate of $\log(\omega_{\max})^{-1}\sqrt{\omega_{\max}}$ for the vortex sheet method. (The $\log N$ is most likely a spurious term introduced by the analysis.) Based on the above observations we believe that the dependence of the vortex sheet method on $\omega_{\max}$ is $O(\omega_{\max}^p)$ for some p, with $\frac{1}{3} \leqq p \leqq \frac{1}{2}$, and that it is very likely that $p = \frac{1}{2}$.

5. Numerical results.

5.1. The test problem. We model two-dimensional flow past a flat plate with a constant free-stream velocity, $U(x) \equiv U_0$. This is known as Blasius flow [30], [37]. Here we take $U_0 = 1$. This is a stationary flow that has the well-known similarity solution

$$u(x, y) = f'(\eta), \tag{5.1}$$

$$\eta = y/\sqrt{\nu x}, \tag{5.2}$$

where f satisfies the ODE

$$ff'' + 2f''' = 0, \tag{5.3a}$$

$$f(0) = 0, \quad f'(0) = 0, \quad f'(\infty) = 1. \tag{5.3b}$$

While we cannot write down f exactly, it is a simple matter to solve (5.3a)-(5.3b) numerically with great accuracy (White [37, p. 262]), thus obtaining an effectively exact solution.

Blasius flow is a solution of equations (1.1a)-(1.1e) over the semi-infinite flat plate, $0 < x < \infty$, $0 \leqq y < \infty$. There is a small neighborhood of the leading edge of the plate in which the transverse velocity component is of the same order of magnitude as the tangential velocity component; hence the Prandtl equations are not valid in this region. In order to ameliorate the effects of this leading edge singularity and to conveniently handle the downstream boundary condition, we consider the following periodic problem.

We compute over the portion of the plate from $a = 3h$ to $b = 1 + 3h$. We map physical space, $\Omega \equiv \{(x, y): x > 0, \text{ and } y \geqq 0\}$, onto the periodic domain $\Omega_C \equiv \{(x, y): 3h \leqq x \leqq 1 + 3h, \text{ and } y \geqq 0\}$ by the transformation $(x, y) \in \Omega \to (x^*, y^*) \in \Omega_C$, where

$$x^* = x \bmod 1, \tag{5.4a}$$

$$y^* = y\sqrt{x^*/x}. \tag{5.4b}$$

Sheets whose centers move to the right of $x = b$ have their centers transformed according to (5.4a)-(5.4b) so that they now appear to lie near the beginning of Ω_C. Similarly, sheets that move backward, to the left of $x = a$, are rescaled and placed at the end of Ω_C. Furthermore, when calculating the velocity of a point that lies within one sheet length of the edge of our computational domain we take care to include the influence of sheets that lie near the other end. Thus we have eliminated the effects of the leading edge singularity by imposing as an upstream boundary condition the computed velocity profile that results from identifying $x = a$ and $x = b$ with appropriately scaled y

coordinates. This also eliminates spurious effects due to throwing sheets away after they pass $x = b$ (e.g., [9, p. 433]).

One drawback of the periodic formulation of this problem is that the total number of vortex elements in the flow increases with time. Once a sheet is created, it exists for the remainder of the computer calculation. Therefore the amount of vorticity above a given portion of the plate also increases with time. For Blasius flow this is nonphysical. It is our contention, however, that this is a good test of the algorithm's effectiveness. Vorticity creation is an important phenomenon and we would like to approximate it as accurately as possible. Successful strategies for minimizing the error here will lead to a better overall algorithm.

All results shown here are at time $T = 2$ with $\nu = 10^{-4}$. We used the piecewise linear cutoff (3.2) and initial data (2.14) with $U_{\max} = 1$.

5.2. Measurement of the error. In order to eliminate the dependence of the L^p norm of the error on the viscosity for $1 \leqq p < \infty$, we measure all such norms in the transformed variables (x, η),

$$\|u - \tilde{u}\|_p = \left(\int_a^b \int_0^\infty |u(x, \eta) - \tilde{u}(x, \eta)|^p \, d\eta \, dx \right)^{1/p}, \tag{5.5}$$

where η is given by (5.2). Furthermore, for $p = 1$ we divide the error in the L^1 norm by

$$\|1 - u\|_1 = \int_a^b \int_0^\infty (1 - f'(\eta)) \, d\eta \, dx = (\eta - f(\eta))\Big|_{\eta=0}^{\eta=\infty} \approx 1.7208$$

(see [30], p. 130) and report the "normalized" error:

$$\|u - \tilde{u}\|_1 / \|1 - u\|_1 .$$

To estimate the integral norm in (5.5) we used

$$\begin{aligned} \|u - \tilde{u}\|_p &\approx \left(\sum_{i=1}^r h \int_0^\infty |u(a_i, \eta) - \tilde{u}(a_i, \eta)|^p \, d\eta \right)^{1/p} \\ &= h^{1/p} \left(\sum_{i=1}^r (\sqrt{\nu a_i})^{-1} \int_0^\infty |u(a_i, y) - \tilde{u}(a_i, y)|^p \, dy \, dx \right)^{1/p}. \end{aligned}$$

The one-dimensional error above a_i was calculated using the trapezoid rule

$$\int_0^\infty |u(a_i, y) - \tilde{u}(a_i, y)|^p \, dy \approx \sum_j ((\mathrm{err}_j)^p + (\mathrm{err}_{j+1})^p) \frac{(y_{j+1} - y_j)}{2}, \tag{5.6}$$

where $\mathrm{err}_j = |u(a_i, y_j) - \tilde{u}(a_i, y_j)|$, the sum is only over those y_j such that

$$b_h(a_i - x_j) \neq 0, \tag{5.7}$$

and we have ordered the y_j so that $y_j \leqq y_{j+1}$ for all j. In other words, we use a grid that corresponds to the location of the sheets above a_i. To ensure that the sum in (5.6) starts at $y = 0$, we place a sheet with no weight at $(a_i, 0)$. The value of $u(a_i, y_j) = f'(y_j/\sqrt{\nu a_i})$ was determined by linear interpolation from an array containing values of f' at equally spaced points $\eta = 0.0, 0.01, \cdots, 8.0$. We estimate the sup norm of the error similarly,

$$\|u - \tilde{u}\|_\infty \approx \max \{|u(a_i, y_j) - \tilde{u}(a_i, y_j)|\},$$

where the max is taken over all i, j satisfying (5.7).

The velocity, vorticity, drag, and each of the various norms of the error are random variables. We therefore make several different runs, which we refer to as *trials*. Each

trial has a different starting seed for the random number generator. We estimate the expected value of a randomly varying quantity Θ by $\bar{\Theta}$, where

$$\bar{\Theta} = \frac{1}{n} \sum_{i=1}^{n} \Theta_i \tag{5.8}$$

and Θ_i is the value of Θ calculated from the ith trial. We estimate the standard deviation of Θ by

$$\tilde{\sigma} = \frac{1}{\sqrt{n-1}} \left(\sum_{i=1}^{n} (\Theta_i - \bar{\Theta})^2 \right)^{1/2}. \tag{5.9}$$

We will also have occasion to refer to the *standard error* σ_Θ of the estimate in (5.8) defined by

$$\sigma_\Theta = \frac{\sigma}{\sqrt{n}} \tag{5.10}$$

([22, p. 21]), where σ is the true standard deviation for the parent distribution of the Θ_i. The quantity σ_Θ is simply the standard deviation of the distribution of the $\bar{\Theta}$'s and is therefore a measure of how good an estimate (5.8) is of the true mean. Since we do not know σ we will use (5.10) with σ replaced by $\tilde{\sigma}$ from (5.9). Unless noted otherwise, all of the data below are estimates based on $n = 25$ trials.

5.3. An estimate of the rate of convergence. We begin by presenting numerical results which demonstrate that the vortex sheet method converges when it is used to model the Blasius flow. Table 1 contains $\bar{\Theta}$ and $\tilde{\sigma}$ for the (normalized) L^1 norm of the error. In all runs displayed here $\Delta t = h/U_{\max}$. Note that as we move down a column the error eventually levels out. Since Δt and h are constant along columns it follows that the error which decreases along the columns is due to $\omega_{\max}$ alone. The "plateau" at the bottom of the column is due to those sources of error that depend on Δt and h. Similarly, it is apparent that for fixed values of $\omega_{\max}$ the error decreases with decreasing h, provided $\omega_{\max} < h^2/2L\Delta t$. This corresponds to $C_0 < 1/2L$ in (2.15). However, after leveling off the error *begins to increase* when this condition is violated. We observed this phenomenon consistently in all of our runs; it occurred with both the piecewise linear and B-spline cutoffs. We will investigate this behavior more thoroughly in § 5.5 below.

TABLE 1
Discrete L^1 norm of the error. Estimated mean and standard deviation of the error.

	h ($\Delta t = h/U_{\max}$)			
$\omega_{\max}$	5^{-1}	10^{-1}	20^{-1}	40^{-1}
5^{-1}	0.4002 ± 0.0734	—	—	—
10^{-1}	0.2989 ± 0.0638	0.2983 ± 0.0521	—	—
20^{-1}	0.2580 ± 0.0475	0.2239 ± 0.0399	0.2230 ± 0.0279	—
40^{-1}	0.2663 ± 0.0473	0.1773 ± 0.0239	0.1657 ± 0.0112	0.1903 ± 0.0165
80^{-1}	0.2483 ± 0.0243	0.1636 ± 0.0202	0.1267 ± 0.0126	0.1346 ± 0.0101
160^{-1}	0.2529 ± 0.0252	0.1594 ± 0.0104	0.1088 ± 0.0103	0.0990 ± 0.0075
320^{-1}	0.2473 ± 0.0146	0.1528 ± 0.0093	0.1007 ± 0.0069	0.0778 ± 0.0077[1]
640^{-1}	0.2511 ± 0.0088	0.1534 ± 0.0068	0.0806 ± 0.0075	0.0627 ± 0.0060[2]

[1] Here $n = 16$, where n is the number of trials.
[2] $n = 2$.

In Fig. 1 we plot the log of the errors in Table 1 versus $\log h^{-1}$ for various relationships between $\omega_{\max}$ and h. The first data point in each sequence is $h = 5^{-1}$ and $\omega_{\max} = 10^{-1}$. For h fixed the abscissa corresponds to $\omega_{\max} = 10^{-1}, \cdots, 80^{-1}$. (For $\omega_{\max}$ fixed the last two data points are not displayed in Table 1.) When $\omega_{\max} = O(h^2)$ the resulting curve is nearly linear. This indicates that the error due to $\omega_{\max}$ is decreasing at least as fast as that due to h. We can attempt to discern a dependence of the error on h of the form error $= O(h^q)$ from this data. The slope of the line connecting the first and last points of the bottommost curve is $-0.7510 \approx -\frac{3}{4}$, implying $q \approx \frac{3}{4}$. However, if instead we choose the data from the last row of Table 1 (the error due to $\omega_{\max}$ should be very small here), then we find $q \approx \frac{2}{3}$. There is not enough data here to determine q beyond all doubt. We conjecture $q \geqq \frac{2}{3}$ and wonder why there is no evidence for $q = 1$. In Fig. 2 we plot the standard deviations that correspond to the data in Fig. 1. The

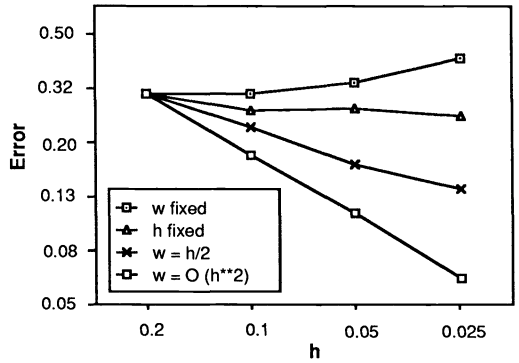


FIG. 1. Log $\bar{\Theta}$ *versus* log $1/h$.

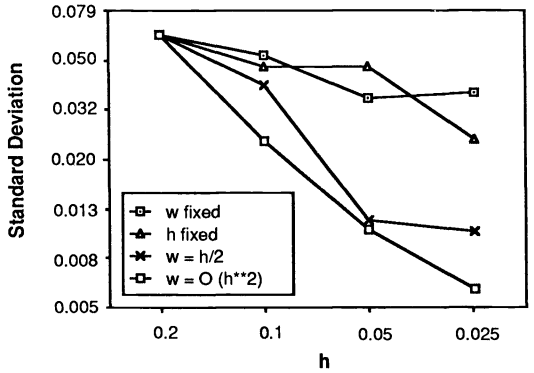


FIG. 2. Log $\tilde{\sigma}$ *versus* log $1/h$.

nearly linear decay of the lowest curve again indicates that when $\omega_{\max} = O(h^2)$ the error due to $\omega_{\max}$ decreases at least as rapidly as that due to h.

We repeated this experiment with the timestep halved, $\Delta t = h/2U_{\max}$, in an effort to determine if any of the errors would decrease when Δt alone was decreased. These errors, presented in Table 2, were remarkably close to those in Table 1, sometimes agreeing to several decimal places. Furthermore, many of these new errors lie within three standard errors of the values in Table 1 (using $\tilde{\sigma}$ from either table), making the two quantities statistically indistinguishable from each other. This was also true for the errors in the L^2 norm. We conclude that the errors presented in Table 1 do not diminish when h and $\omega_{\max}$ are fixed and Δt is decreased. This indicates that the errors due to temporal discretization are much smaller than the other sources of error. This may be because we are computing a stationary flow, and we caution the reader against drawing conclusions for more general flows.

We made a sequence of runs under the assumption that the error is $O(\sqrt{\omega_{\max}}) + O(h^{2/3})$. We set $h = O(\omega_{\max}^{3/4})$, $\Delta t = h/U_{\max}$, and, starting with $\omega_{\max} = 0.025$ and $h = 0.2$, made five runs decreasing $\omega_{\max}$ by two on each run. The results appear in Table 3. Note that the errors here are the results after only one trial, rather than being our estimate of the expected value $\bar{\Theta}$.

The column labeled "time" contains the time in CPU minutes it took for one trial on a CRAY X-MP. The FORTRAN program was designed to be as fast as possible. In particular, we vectorized every loop that would admit vectorization. We also employed a "bin" data structure in which all sheets lying in $a_i - h/2 \leqq x < a_i + h/2$ are kept in the same bin. Thus, to compute the velocity of a sheet in the ith bin we need only loop over those sheets in adjacent bins. Hence, instead of every sheet interacting

TABLE 2

Discrete L^1 norm of the error. The errors in Table 1 do not decrease as Δt is decreased with h and $\omega_{\max}$ left fixed.

	h $(\Delta t = h/2U_{\max})$			
$\omega_{\max}$	5^{-1}	10^{-1}	20^{-1}	40^{-1}
5^{-1}	0.4318 ± 0.0733	—	—	—
10^{-1}	0.3215 ± 0.0762	0.2967 ± 0.0469	—	
20^{-1}	0.2762 ± 0.0670	0.2336 ± 0.0310	0.2463 ± 0.0331	—
40^{-1}	0.2721 ± 0.0502	0.1787 ± 0.0256	0.1778 ± 0.0134	0.2170 ± 0.0167
80^{-1}	0.2402 ± 0.0351	0.1586 ± 0.0232	0.1276 ± 0.0158	0.1498 ± 0.0096

TABLE 3

(One trial per row.) A convergence study with $h = O(\omega_{\max}^{3/4})$ and $\Delta t = h/U_{\max}$.

$\omega_{\max}$	L^1 norm	L^2 norm	L^∞ norm	Sheets	Time	δ_1 av	δ_2 av
40^{-1}	0.2817	0.2489	0.2708	226	0.002	0.0044	0.00161
80^{-1}	0.1617	0.1756	0.2677	939	0.022	0.0024	0.00070
160^{-1}	0.1044	0.1070	0.1536	2954	0.218	0.0013	0.00037
320^{-1}	0.0823	0.0915	0.2719	13755	3.638	0.0009	0.00029
640^{-1}	0.0592	0.0631	0.1141	37206	31.131	0.0007	0.00019

with every other sheet in the advection step, for a total of N^2 interactions, each sheet has approximately $3hN$ interactions. This results in an algorithm that is approximately $O(3hN^2)$. (The actual amount of work depends on how uniformly the sheets are distributed along the plate.)

We also measured the error in approximating the *displacement thickness* and *momentum thickness*. The displacement thickness above x, $\delta_1(x)$, defined by

$$\delta_1(x) = \int_0^\infty \left(1 - \frac{u(x, y)}{U(x)}\right) dy$$

is the amount by which a streamline has been displaced by the thickening of the boundary layer from the beginning of the plate to x. The momentum thickness above x, $\delta_2(x)$, defined by

$$\delta_2(x) = \int_0^\infty \frac{u(x, y)}{U(x)} \left(1 - \frac{u(x, y)}{U(x)}\right) dy$$

is a measure of the amount of momentum that has been lost due to the boundary layer. See Schlichting [30] or White [37] for more details.

We measured the average error in these quantities along the wall as follows. The average error in the *displacement thickness* $\delta_1 av$ is defined by

$$\delta_1 av = \frac{1}{r} \sum_{i=1}^{r} |\delta_1(a_i) - \tilde{\delta}_1(a_i)|, \tag{5.11}$$

where $\tilde{\delta}_1$ is the trapezoid rule approximation to δ_1 given by (5.6) with $u(a_i, y)$ replaced with 1. Similarly, $\delta_2 av$ is the average error in the momentum thickness defined by (5.11) with δ_1 and $\tilde{\delta}_1$ replaced by δ_2 and $\tilde{\delta}_2$, where $\tilde{\delta}_2$ is the trapezoid rule approximation to δ_2. Note that we have not scaled out the effect of ν in our calculation of $\tilde{\delta}_1$ and $\tilde{\delta}_2$ and hence, $\delta_1 av$ and $\delta_2 av$ are $O(\sqrt{\nu}) = O(10^{-2})$.

In Fig. 3 we compare the L^1 error from Table 3 with the conjectured rate of convergence. It is apparent that for this choice of parameters the L^1 error *on the average*

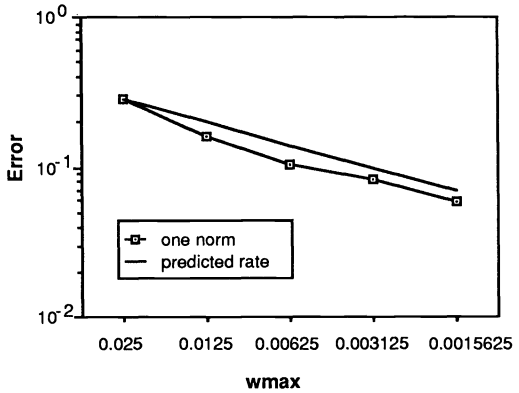


FIG. 3. *Actual versus predicted rate.*

decreases at a rate close to the anticipated one. In other words, the error may decrease by more or less than $\sqrt{2}$ from one row to the next, but from the first row to the last row the error has decreased by approximately a factor of 4. These remarks also apply to the L^2 error and the average error in the momentum and displacement thicknesses. The L^∞ norm was not as well behaved. This is not surprising since for random walk methods there can be small sets (in (x, y)-space) on which the error is large (e.g., Hald [20, § 5]). Of course the integral norms ameliorate the effects of large errors on small sets.

Our goal here has been to demonstrate that we need not average to use effectively these random walk methods. The fact that most of the results reported on in this paper are averages should not be construed as suggesting that we must always "ensemble average" to get reasonable results. It is apparent from Table 3 that we can make one run as long as it is in a regime of small variance. Based on our experience this can best be accomplished by ensuring that $\omega_{\max}$ is always small compared to $h^2/L\Delta t$ and by decreasing $\omega_{\max}$ faster than h and Δt. Here we have chosen $\omega_{\max} \leqq h^2/8L\Delta t$, which corresponds to $C_0 = L/8$ in (2.15) and let $\omega_{\max} = O(h^{4/3})$. In general we recommend taking $\omega_{\max} = O(h^{2q})$ with $\frac{2}{3} \leqq q \leqq 1$. Our experience with $q > \frac{2}{3}$ has not produced as consistently good results as those shown here, but we do not have enough evidence to unequivocally state that the optimal choice of q is $\frac{2}{3}$.

5.4. Second-order integration in time. Next we investigate the effect of solving (2.7a), (2.7b) with a second-order ODE solver. We conduct the same experiment as in Table 1 but with (2.6a), (2.6b) replaced by

$$x_j^{k+1/2} = x_j^k + \Delta t \tilde{u}_{1/2}(x_j', y_j'), \qquad y_j^{k+1/2} = y_j^k + \Delta t \tilde{v}_{1/2}(x_j', y_j')$$

where

$$x_j' = x_j + \frac{\Delta t}{2}\tilde{u}^k(x_j^k, y_j^k), \qquad y_j' = y_j + \frac{\Delta t}{2}\tilde{v}^k(x_j^k, y_j^k)$$

is the position of the jth sheet after one half of a timestep and $(\tilde{u}_{1/2}, \tilde{v}_{1/2})$ is the velocity field induced by the sheets when their centers are at these positions. Comparing these results with Table 1, we observe no increase in accuracy over Euler's method solution of (2.7a), (2.7b). This is consistent with the results displayed in Table 2. Of course we did not use second-order operator splitting (i.e., Strang splitting), and hence do not expect to see all sources of error that depend on Δt decrease. Nonetheless, there appears to be no improvement in the error due to second-order time discretization. It is instructive to compare these results with the effect of using second-order time integration in a one-dimensional random particle method as shown in Tables 3 and 4 of [25].

TABLE 4
Discrete L^1 norm of the error. Second-order integration in time.

	h $(\Delta t = h/U_{\max})$			
$\omega_{\max}$	5^{-1}	10^{-1}	20^{-1}	40^{-1}
5^{-1}	0.4224 ± 0.0995	—	—	—
10^{-1}	0.3290 ± 0.1044	0.3183 ± 0.0494	—	—
20^{-1}	0.2736 ± 0.0551	0.2316 ± 0.0276	0.2547 ± 0.0285	—
40^{-1}	0.2261 ± 0.0425	0.1701 ± 0.0227	0.1759 ± 0.0201	0.2271 ± 0.0204
80^{-1}	0.2325 ± 0.0285	0.1552 ± 0.0225	0.1323 ± 0.0128	0.1537 ± 0.0114

It is unclear to what extent this lack of dependence on the timestep is due to the stationary character of the test problem. Nevertheless, we feel compelled to make several comments regarding second-order time integration. There seems to be a prevalence of opinion among users of hybrid vortex sheet/vortex blob methods that the use of a higher-order time integration will result in a better solution. Presumably this opinion is based on theoretical results for the vortex method solution of Euler's equations with higher-order time integration (e.g., [1], [21]). However, the following points should be made in regard to the vortex sheet method: (i) A second-order scheme takes in twice as much work. (ii) We are constrained by (2.13) to decrease Δt as rapidly as h. (iii) Our results indicate that for the Blasius flow problem the dominant sources of error are those that depend on h and $\omega_{\max}$. Hence, we may be doing twice as much work for a negligible gain in accuracy. The first two considerations also apply when the vortex sheet method is used to model other flows. We note that, for somewhat different reasons, Sethian [33] has also questioned the effectiveness of using a second-order time-integration scheme with the vortex sheet method.

5.5. The behavior of the algorithm with decreasing h. Let us consider the dependence of N on $\omega_{\max}$. For example, assume that $\omega_{\max}$ divides $U_{\max}=1$ evenly and let the initial data be given by (2.14). Let N_x denote the number of sheets above x. Then at the first timestep we have $N_{a_i}=\omega_{\max}^{-1}$ for each gridpoint a_i on the wall. In general, at least for reasonable pressure gradients (UU_x small), we expect that $N_x=O(\omega_{\max}^{-1})$ for any x. In Table 5 we show the average number of sheets in the flow at time $T=2.0$ as a function of $\omega_{\max}$, h, and Δt.[1] It is apparent that if $\omega_{\max}$ is decreased by 2 with h and Δt fixed, then the average number of sheets does not increase by significantly more than 2.

TABLE 5

Discrete L^1 norm of the error. Gradual increase of the L^1 error for decreasing h, Δt, and fixed $\omega_{\max}$.

	h ($\Delta t = h/U_{\max}$)							
$\omega_{\max}$	0.2		0.1		0.05		0.025	
	Sheets	Error	Sheets	Error	Sheets	Error	Sheets	Error
0.2	26	0.4317	63	0.4449	179	0.5434	880	0.6860
0.1	54	0.3106	125	0.2915	328	0.3331	1074	0.4169
0.05	109	0.2724	253	0.2054	641	0.2317	1986	0.2858
0.025	222	0.2524	498	0.1681	1248	0.1630	3702	0.1989

We would also expect that letting h go to $h/2$ with $\omega_{\max}$ fixed would produce similar results. This turns out not to be the case. It is apparent from Table 5 that under these circumstances the average number of sheets in the flow always increases by *more* than 2, often by much more. Since $\Delta t = h/U_{\max}$ in all these runs, we can inquire if this unexpected increase is due to the increased number of timesteps as we decrease Δt. To answer this question we made a sequence of runs fixing $\Delta t = 0.0125$ and reducing h as before. The results appear in Table 6. It is apparent that the average number of

[1] Some of the numbers here differ from those in Table 1 even though the choice of parameters is the same. This is because these trials were made on a different computer with a different random number generator.

TABLE 6
Discrete L^1 norm of the error. Gradual increase of the L^1 error for decreasing h and fixed Δt, $\omega_{\max}$.

	h ($\Delta t = 0.0125$)							
$\omega_{\max}$	0.2		0.1		0.05		0.025	
	Sheets	Error	Sheets	Error	Sheets	Error	Sheets	Error
0.2	31	0.4380	79	0.4489	242	0.5339	1056	0.7514
0.1	64	0.3640	158	0.3110	451	0.3647	1427	0.4549
0.05	128	0.2849	320	0.2310	909	0.2549	2745	0.3228
0.025	246	0.2733	612	0.1789	1712	0.1839	4978	0.2167

sheets continues to grow by significantly more than 2 as we move to the right along a row. Therefore, this phenomenon is not due to decreasing Δt.

Note that the factor by which the number of sheets grows is largest in the upper right-hand corner in both Tables 5 and 6. These regions also display an increase in the error as h is decreased with $\omega_{\max}$ left fixed. Both phenomena may be due to the fact that (2.15) is being violated. However we can make no definite assertion. We do note however that an abnormal growth in the number of sheets seems to characterize those regions of parameter space that have large errors. We consider this important for several reasons. The amount of work to compute the velocity at each of the sheets is at best $O(hN^2)$. Thus, since decreasing h results in many more sheets than decreasing $\omega_{\max}$ by the same amount, it also results in a more expensive computation. For the same reason decreasing h leads to a larger value of $\|\tilde{\omega}\|$, whereas decreasing $\omega_{\max}$ does not significantly alter $\|\tilde{\omega}\|$. Consequently, the bounds in § 4.2 that depend on $\|\omega^0\|$ deteriorate when h is decreased. Unfortunately, we cannot pursue a strategy of only decreasing $\omega_{\max}$; eventually there is no improvement, except possibly a reduction in the variance. It is therefore necessary to toe a fine line between decreasing h too rapidly and not decreasing h fast enough.

5.6. A comparison of particle-creation algorithms. In this section we present the results of a numerical experiment designed to compare the two particle-creation algorithms described in § 2.3.2. We obtained the errors in Table 7 with creation Algorithm A using a value of $\varepsilon = 10^{-6}$. Table 8 is a duplicate of Table 7, except here

TABLE 7
Discrete L^2 norm of the error. Average number of sheets and mean of the error using Algorithm A.

	h ($\Delta t = h/U_{\max}$)							
$\omega_{\max}$	0.2		0.1		0.05		0.025	
	Sheets	Error	Sheets	Error	Sheets	Error	Sheets	Error
0.2	60	0.3600	—	—	—	—	—	—
0.1	88	0.2957	289	0.2738	—	—	—	—
0.05	145	0.2525	422	0.2166	1407	0.2310	—	—
0.025	255	0.2327	676	0.1783	2021	0.1690	7060	0.1965
0.0125	477	0.2021	1150	0.1400	3109	0.1164	9835	0.1362

TABLE 8
Discrete L^2 norm of the error. Average number of sheets and mean of the error using Algorithm B.

	h ($\Delta t = h/U_{\max}$)							
$\omega_{\max}$	0.2		0.1		0.05		0.025	
	Sheets	Error	Sheets	Error	Sheets	Error	Sheets	Error
0.2	27	0.3610	—	—	—	—	—	—
0.1	53	0.2763	124	0.2766	—	—	—	—
0.05	109	0.2410	251	0.2090	638	0.2087	—	—
0.025	216	0.2378	495	0.1709	1256	0.1612	3749	0.1808
0.0125	430	0.2222	972	0.1549	2370	0.1258	6775	0.1329

we used creation Algorithm B, i.e., the same algorithm used for all other experiments in this paper. Note that here the errors are in the discrete L^2 norm. In all instances Algorithm A results in an increase in the number of sheets in the flow but with little or no improvement in accuracy. We also tried replacing $\varepsilon = 10^{-6}$ with $\varepsilon = \omega_{\max}/2$ and observed a similar result: there was no noticeable improvement in the error but more sheets than with Algorithm B. We conclude that Algorithm B is more efficient and recommend that all users of the vortex sheet method employ it.

5.7. Sheet tagging. Finally, we made a sequence of runs to examine the effectiveness of the sheet-tagging procedure. In Table 9 we present $\bar{\Theta}$ and $\tilde{\sigma}$ for the error in the L^1 norm. These figures should be compared with Table 1. It is apparent that the sheet tagging leads to neither a decrease in the error nor a reduction in the variance. We do not recommend the sheet-tagging procedure.

TABLE 9
Discrete L^1 norm of the error. Sheet tagging.

	h ($\Delta t = h/U_{\max}$)			
$\omega_{\max}$	0.2	0.1	0.05	0.025
0.2	0.5508 ± 0.2671	—	—	—
0.1	0.4574 ± 0.1933	0.3677 ± 0.1938	—	—
0.05	0.2922 ± 0.1145	0.2524 ± 0.0748	0.2844 ± 0.0777	—
0.025	0.2422 ± 0.1236	0.1729 ± 0.0399	0.2026 ± 0.0437	0.2045 ± 0.0236
0.0125	0.2406 ± 0.0893	0.1638 ± 0.0637	0.1413 ± 0.0308	0.1490 ± 0.0179

5.8. Conclusions. We have proved that one timestep of the random walk and sheet-creation process is a consistent approximation to the exact solution of the heat equation subject to the no-slip boundary condition. This is true in each of the L^1, L^2, and L^∞ norms. Furthermore, it follows from the proof that the truncation error is entirely due to our failure to satisfy the no-slip boundary condition exactly at each point on the boundary.

We have demonstrated numerically that the vortex sheet method converges when it is used to approximate Blasius flow, provided certain conditions on the computational parameters are satisfied. These include two accuracy conditions of the form $\Delta t U_{\max} \leqq h$

and $\omega_{\max} \leqq C_0 h^2/\Delta t$, where $C_0 = O(1/L)$ and a relative rate of refinement condition $\omega_{\max} = O(h^{2q})$ for some q with $2/3 \leqq q \leqq 1$. We have also shown that the error will eventually increase when the second condition is not satisfied; for example, this occurs when h goes to zero much faster than $\omega_{\max}$.

We have been unable to observe any improvement in the accuracy of the method when a second-order time-integration scheme is used, or when the other parameters are fixed and the timestep is decreased. This may be because the test problem is a stationary flow. However, we suggest workers carefully consider the advantages and disadvantages of a second-order time-integration scheme before using one with the vortex sheet method.

We have also demonstrated that the most efficient sheet-creation algorithm is one for which all sheet strengths have equal magnitude, $|\omega_j| = \omega_{\max}$. Algorithms that create sheets with varying magnitudes end up producing more sheets without improving the accuracy of the method. Finally, we have shown that the sheet-tagging procedure is not an effective variance-reduction technique and therefore should not be used.

Acknowledgments. The author would like to thank Alexandre Chorin and Ole Hald for their time, support, and encouragement. He also would like to acknowledge fruitful discussions with Phil Colella, Maciej Pindera, and Jamie Sethian.

REFERENCES

[1] C. R. ANDERSON AND C. A. GREENGARD, *On vortex methods*, SIAM J. Numer. Anal., 22 (1985), pp. 413–440.

[2] J. T. BEALE AND A. MAJDA, *Vortex methods* I: *Convergence in three dimensions*, Math. Comp., 39 (1982), pp. 1–27.

[3] ———, *Vortex methods* II: *higher order accuracy in two and three dimensions*, Math. Comp., 39 (1982), pp. 29–52.

[4] CARL DE BOOR, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[5] A. Y. CHEER, *A study of incompressible 2-D vortex flow past a circular cylinder*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 685–705.

[6] ———, *Unsteady separated wake behind an impulsively started cylinder in slightly viscous fluid*, unpublished manuscript, University of California, CA, 1986.

[7] Y. CHOI, J. A. C. HUMPHREY, AND F. S. SHERMAN, *Random vortex simulation of transient wall-driven flow in a rectangular enclosure*, J. Comp. Phys., 75 (1988), pp. 359–383.

[8] A. J. CHORIN, *Numerical study of slightly viscous flow*, J. Fluid Mech., 57 (1973), pp. 785–796.

[9] ———, *Vortex sheet approximation of boundary layers*, J. Comp. Phys., 27 (1978), pp. 428–442.

[10] A. J. CHORIN AND J. E. MARSDEN, *A Mathematical Introduction to Fluid Mechanics*, Springer-Verlag, New York, 1979.

[11] A. J. CHORIN, *Vortex models and boundary layer instability*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 1–21.

[12] K. L. CHUNG, *A Course in Probability Theory*, Harcourt, Brace, and World, New York, 1974.

[13] G. H. COTTET, *Convergence of a vortex in cell method for the two dimensional Euler equations*, Rapport Interne 108, Centre de Mathématiques Appliquées, École Polytechique, Paris, 1984.

[14] W. FELLER, *An Introduction to Probability Theory and its Applications*, 2nd ed., Vol. II, John Wiley, New York, 1971.

[15] A. F. GHONIEM, A. J. CHORIN, AND A. K. OPPENHEIM, *Numerical modeling of turbulent flow in a combustion tunnel*, Philos. Trans. Roy. Soc. London Ser. A, 304 (1982), pp. 303–325.

[16] J. GOODMAN, *Convergence of the random vortex method in two dimensions*, Commun. Pure Appl. Math., 40 (1987), pp. 189–220.

[17] C. A. GREENGARD, *Three dimensional vortex methods*, Ph.D. thesis, University of California, Berkeley, CA, 1984.

[18] O. H. HALD AND V. M. DEL PRETE, *Convergence of vortex methods for Euler's equations*, Math. Comp., 32 (1978), pp. 791–809.

[19] O. H. HALD, *The convergence of vortex methods*, II, SIAM J. Numer. Anal., 16 (1979), pp. 726–755.

[20] O. H. HALD, *Convergence of a random method with creation of vorticity*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1373–1386.

[21] ———, *Convergence of vortex methods for Euler's equations*, III, SIAM J. Numer. Anal., 24 (1987), pp. 538–582.

[22] J. M. HAMMERSLEY AND D. C. HANDSCOMB, *Monte Carlo Methods*, Methuen, London, 1964.

[23] L. D. LANDAU AND E. M. LIFSHITZ, *Fluid Mechanics*, Pergamon Press, New York, 1959.

[24] D. G. LONG, *Convergence of the random vortex method in one and two dimensions*, Ph.D. thesis, University of California, Berkeley, CA, 1986.

[25] E. G. PUCKETT, *Convergence of a random particle method to solutions of the Kolmogorov equation*, Math. Comp., 52 (1989), to appear.

[26] ———, *A study of the vortex sheet method and its rate of convergence*, Preprint 23341, Lawrence Berkeley Laboratory, University of California, Berkeley, CA, 1987.

[27] E. G. PUCKETT AND S. B. BADEN, *A fast vortex code for computing 2-d flow in a box*, Proc. First National Fluid Dynamics Congress, Cincinnati, OH, 1988.

[28] S. G. ROBERTS, *Accuracy of the random vortex method for a problem with non-smooth initial conditions*, J. Comput. Phys., 58 (1985), pp. 29–43.

[29] ———, *Convergence of a random walk method for Burgers' equation*, Ph.D. thesis, University of California, Berkeley, CA, 1985.

[30] H. SCHLICHTING, *Boundary-Layer Theory*, McGraw-Hill, New York, 1968.

[31] M. H. SCHULTZ, *Spline Analysis*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1973.

[32] J. A. SETHIAN AND A. F. GHONIEM, *Validation study of vortex methods*, J. Comput. Phys., 74 (1988), pp. 283–317.

[33] J. A. SETHIAN, *Turbulent combustion in open and closed vessels*, J. Comp. Phys., 55 (1984), pp. 425–456.

[34] ———, *Vortex Methods and Turbulent Combustion*, Lectures in Applied Mathematics 22, Springer-Verlag, New York, Berlin, 1985.

[35] D. M. SUMMERS, T. HANSON, AND C. B. WILSON, *A random vortex simulation of wind-flow over a building*, Internat. J. Numer. Methods Fluids, 5 (1985), pp. 849–871.

[36] E. TIEMROTH, *The simulation of the viscous flow around a cylinder by the random vortex method*, Ph.D. thesis, Naval Arch. Department, University of California, Berkeley, CA, 1986.

[37] F. M. WHITE, *Viscous Fluid Flow*, McGraw-Hill, New York, 1974.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 328-340, March 1989
© 1989 Society for Industrial and Applied Mathematics
006

THE l_1 SOLUTION OF LINEAR EQUATIONS SUBJECT TO LINEAR CONSTRAINTS*

ACHIYA DAX†

Abstract. The paper presents an efficient active set method for calculating the l_1 solution of linear equations subject to linear constraints. The search direction of the proposed method is obtained via the solution of a linear least-squares subproblem. It is shown that the feasible direction of steepest descent can be computed by solving the same least-squares subproblem but with simple bounds on the variables. This direction is used to prevent cycling at degenerate dead points. Numerical experiments illustrate the practical value of the new approach.

Key words. l_1 solution of linear equations, linear constraints, active set methods, degeneracy, bounded least squares, steepest descent

AMS(MOS) subject classifications. 65K99, 65D99

1. Introduction. This paper outlines an active set method for solving the following problem:

$$\begin{aligned} &\text{minimize} && F(\mathbf{x}) = \sum_{i \in B} |\mathbf{a}_i^T \mathbf{x} - b_i| \\ &\text{subject to} && \mathbf{a}_i^T \mathbf{x} = b_i \quad \text{for } i \in E, \\ &&& \mathbf{a}_i^T \mathbf{x} \geqq b_i \quad \text{for } i \in C, \end{aligned} \tag{1.1}$$

where B, E, and C are finite index sets, $\mathbf{x} \in \Re^n$ is the unknown vector, $\mathbf{a}_i$ are given vectors in $\Re^n$, and b_i are given real numbers.

A comprehensive up-to-date survey of methods for solving this problem can be found, for example, in Osborne [11]. The earlier approaches include the use of linear programming methods (e.g., Barrodale and Roberts [1]), and penalty function methods (e.g., Bartels et al. [3], or Bartels and Conn [2]). The method proposed in this paper is a projected gradient algorithm that resembles the Bartels-Conn approach. However, there are basic differences between the two approaches. One difference lies in the way constraints are handled. The Bartels-Conn method is a penalty function method, while our method starts at a feasible point and generates a sequence of feasible points until an optimal point is reached. Another difference lies in the way degeneracy is handled. Our method prevents cycling by using the steepest descent vector as a search direction at degenerate dead points. On the other hand, the Bartels-Conn method resolves degeneracy by applying a perturbation technique.

The proposed algorithm is based on the observation that a feasible descent direction (the "projected gradient") can be calculated by solving a linear least squares subproblem. It is shown that the feasible direction of steepest descent can be obtained by solving the same least squares subproblem but with certain bounds on the variables. The projected gradient serves as the usual search direction while the steepest descent direction is used at degenerate dead points. The aim of this paper is to illustrate the practical effectiveness of this approach.

The idea of handling degeneracy by solving a "bounded" least squares subproblem has been independently suggested by Watson [13], Busovača [4], and Dax [6], [7]. The works of Watson and Busovača concentrate on nonlinear l_1 optimization, while

* Received by the editors October 1986; accepted for publication (in revised form) June 14, 1988.
† Hydrological Service, P.O. Box 6381, Jerusalem 91060, Israel.

Dax's works concentrate on the multifacility location problem. The use of the new method in other applications is considered in Dax [5], which shows that the method can be viewed as an implementation of the well-known Theorems of the Alternative by Farkas, Gordan, Gale, and others. Another observation of Dax [5] is that the search direction obtained by the new method is the steepest descent vector.

Finally, we note that similar algorithms have been constructed for the analogous l_∞ problem (see Dax [9]) and linear programming (see Dax [10]). These algorithms have the same structure as the present algorithm but differ in the details of the line search and the least squares subproblem. The present algorithm can be viewed, therefore, as a special case of a general method for minimizing a polyhedral convex function.

2. Motivation and optimality conditions. We shall start by introducing some necessary terminology. Let $\mathbf{x} \in \Re^n$ be a feasible point, i.e., $\mathbf{a}_i^T\mathbf{x} = b_i$ for $i \in E$ and $\mathbf{a}_i^T\mathbf{x} \geqq b_i$ for $i \in C$. Define

$$B^* = \{i | i \in B \quad \text{and} \quad \mathbf{a}_i^T\mathbf{x} = b_i\},$$

$$C^* = \{i | i \in C \quad \text{and} \quad \mathbf{a}_i^T\mathbf{x} = b_i\},$$

$$V = \{i | i \in B \quad \text{and} \quad \mathbf{a}_i^T\mathbf{x} \neq b_i\},$$

and

$$\mathbf{g} = \sum_{i \in V} \mathbf{a}_i \operatorname{sign}(\mathbf{a}_i^T\mathbf{x} - b_i).$$

Then $\mathbf{g}$ is sometimes called the "restricted gradient" of $F(\mathbf{x})$ at $\mathbf{x}$. The set B^* includes the indices of "active" terms (i.e., linear equations in the objective function that are satisfied exactly), while C^* contains the indices of "active" inequalities. The importance of $\mathbf{g}$ and B^* lies in the following observation. Let $\mathbf{u}$ be an arbitrary vector in $\Re^n$. Then there exists a positive constant, α say, such that

$$F(\mathbf{x} + \lambda\mathbf{u}) = F(\mathbf{x}) + \lambda\mathbf{g}^T\mathbf{u} + \sum_{i \in B^*} \lambda|\mathbf{a}_i^T\mathbf{u}| \tag{2.1}$$

for all $0 \leqq \lambda \leqq \alpha$. Consequently a vector $\mathbf{u} \in \Re^n$ is said to be a descent direction at $\mathbf{x}$ if it satisfies

$$\mathbf{g}^T\mathbf{u} + \sum_{i \in B^*} |\mathbf{a}_i^T\mathbf{u}| < 0. \tag{2.2}$$

Similarly, a vector $\mathbf{u} \in \Re^n$ that satisfies

$$\mathbf{a}_i^T\mathbf{u} = 0 \quad \text{for } i \in E \quad \text{and} \quad \mathbf{a}_i^T\mathbf{u} \geqq 0 \quad \text{for } i \in C^* \tag{2.3}$$

is called a feasible direction at $\mathbf{x}$. If $\mathbf{u}$ satisfies both (2.2) and (2.3) then it is said to be a feasible descent direction at $\mathbf{x}$. The following theorem provides a simple way to obtain such a direction. For this purpose we introduce the "activity" matrix A. Let t denote the number of indices in the "active" indices set

$$N = B^* \cup E \cup C^*.$$

Then A is a $t \times n$ matrix whose rows are $\mathbf{a}_i^T$, $i \in N$. The order of the rows does not matter. Hence it is assumed here, for simplicity, that $N = \{1, 2, \cdots, t\}$.

THEOREM 1. *Let $\tilde{\mathbf{y}} \in \Re^t$ solve the problem*

$$\text{minimize} \quad \|A^T\mathbf{y} - \mathbf{g}\|_2^2, \tag{2.4}$$

and let $\mathbf{r} = A^T\tilde{\mathbf{y}} - \mathbf{g}$ denote the corresponding residual vector. If $\mathbf{r} \neq \mathbf{0}$, $\mathbf{r}$ is a feasible descent direction at $\mathbf{x}$.

Proof. The basic properties of the linear least squares problem imply that $A\mathbf{r}=\mathbf{0}$, which means that $\mathbf{r}$ is a feasible direction, and

$$\mathbf{g}^T\mathbf{r}+\sum_{i\in B^*}|\mathbf{a}_i^T\mathbf{r}|=\mathbf{g}^T\mathbf{r}=(A^T\tilde{\mathbf{y}}-\mathbf{r})^T\mathbf{r}=\tilde{\mathbf{y}}^TA\mathbf{r}-\mathbf{r}^T\mathbf{r}=-\mathbf{r}^T\mathbf{r}<0. \qquad \square$$

If $\mathbf{r}=\mathbf{0}$ then $\mathbf{x}$ is called a "dead" point. In this case the question whether there exists a feasible descent direction at $\mathbf{x}$ is answered by imposing certain bounds on the components of $\mathbf{y}$.

THEOREM 2. *Let* $\mathbf{y}^*\in\Re^t$ *solve the problem*

$$\begin{aligned} &\textit{minimize} && \|A^T\mathbf{y}-\mathbf{g}\|_2^2 \\ &\textit{subject to} && -1\leqq y_i\leqq 1 \quad \textit{for } i\in B^*, \\ &&& 0\leqq y_i \qquad\quad \textit{for } i\in C^*, \end{aligned} \tag{2.5}$$

and let

$$\mathbf{r}=A^T\mathbf{y}^*-\mathbf{g}$$

denote the corresponding residual vector. If $\mathbf{r}=\mathbf{0}$ *then* $\mathbf{x}$ solves (1.1). *Otherwise* $\mathbf{r}$ *is a feasible descent direction.*

Proof. The existence of a point $\mathbf{y}^*$ that solves (2.5) is ensured by the well-known Projection Theorem. Hence although $\mathbf{y}^*$ is not necessarily unique, $\mathbf{r}$ is unique. (A constructive proof that $\mathbf{y}^*$ exists is implied by the algorithm of § 5.) Assume first that $\mathbf{r}=\mathbf{0}$ and let $\mathbf{u}$ be a feasible direction at $\mathbf{x}$. Then, substituting $\mathbf{g}=A^T\mathbf{y}^*$ into (2.1) gives

$$\begin{aligned} F(\mathbf{x}+\lambda\mathbf{u})-F(\mathbf{x}) &= \lambda\sum_{i\in N}y_i^*\mathbf{a}_i^Tu+\lambda\sum_{i\in B^*}|\mathbf{a}_i^T\mathbf{u}| \\ &= \lambda\sum_{i\in C^*}y_i^*\mathbf{a}_i^T\mathbf{u}+\lambda\sum_{i\in B^*}(|\mathbf{a}_i^T\mathbf{u}|+y_i^*\mathbf{a}_i^T\mathbf{u})\geqq 0 \end{aligned}$$

for all $0\leqq\lambda\leqq\alpha$, which proves the first claim.

Conversely, assume that $\mathbf{r}\neq\mathbf{0}$ and consider the one-parameter functions

$$f_i(\theta)=\|A^T(\mathbf{y}^*+\theta\mathbf{e}_i)-\mathbf{g}\|_2^2=\|\theta\mathbf{a}_i+\mathbf{r}\|_2^2,$$

$i=1,\cdots,t$, where $\mathbf{e}_i$ denotes the ith column of the $t\times t$ unit matrix. If $i\in E$ then $\theta=0$ minimizes $f_i(\theta)$; if $i\in C^*$ then $\theta=0$ minimizes $f_i(\theta)$ subject to the bound $y_i^*+\theta\geqq 0$; and if $i\in B^*$ this point minimizes $f_i(\theta)$ subject to $-1\leqq y_i^*+\theta\leqq 1$. Hence the necessary conditions for minimizing $f_i(\theta)$ imply that $\mathbf{r}$ satisfies the following relations:

$$\begin{aligned} &\mathbf{a}_i^T\mathbf{r}=0 && \text{when } i\in E, \\ &\mathbf{a}_i^T\mathbf{r}=0 && \text{when } i\in C^* \text{ and } y_i^*>0, \\ &\mathbf{a}_i^T\mathbf{r}\geqq 0 && \text{when } i\in C^* \text{ and } y_i^*=0, \\ &\mathbf{a}_i^T\mathbf{r}=0 && \text{when } i\in B^* \text{ and } -1<y_i^*<1, \\ &\mathbf{a}_i^T\mathbf{r}\geqq 0 && \text{when } i\in B^* \text{ and } y_i^*=-1, \\ &\mathbf{a}_i^T\mathbf{r}\leqq 0 && \text{when } i\in B^* \text{ and } y_i^*=1. \end{aligned}$$

The above conditions indicate that $\mathbf{r}$ is a feasible direction and give

$$\sum_{i\in E}y_i^*\mathbf{a}_i^T\mathbf{r}=0, \qquad \sum_{i\in C^*}y_i^*\mathbf{a}_i^T\mathbf{r}=0$$

and

$$\sum_{i\in B^*} (y_i^* \mathbf{a}_i^T \mathbf{r} + |\mathbf{a}_i^T \mathbf{r}|) = 0.$$

These equalities lead to

$$\begin{aligned}
\mathbf{g}^T\mathbf{r} + \sum_{i\in B^*} |\mathbf{a}_i^T\mathbf{r}| &= (A^T\mathbf{y}^* - \mathbf{r})^T\mathbf{r} + \sum_{i\in B^*} |\mathbf{a}_i^T\mathbf{r}| \\
&= -\mathbf{r}^T\mathbf{r} + \sum_{i\in N} y_i^*\mathbf{a}_i^T\mathbf{r} + \sum_{i\in B^*} |\mathbf{a}_i^T\mathbf{r}| \\
&= -\mathbf{r}^T\mathbf{r} + \sum_{i\in E} y_i^*\mathbf{a}_i^T\mathbf{r} + \sum_{i\in C^*} y_i^*\mathbf{a}_i^T\mathbf{r} + \sum_{i\in B^*} (y_i^*\mathbf{a}_i^T\mathbf{r} + |\mathbf{a}_i^T\mathbf{r}|) \\
&= -\mathbf{r}^T\mathbf{r} < 0,
\end{aligned}$$

which proves the second claim. □

COROLLARY 1. *If* $\mathbf{r} \neq \mathbf{0}$, *then* $\mathbf{r}/\|\mathbf{r}\|_2$ *solves the problem*

$$\begin{aligned}
&\textit{minimize} \quad \mathbf{g}^T\mathbf{u} + \sum_{i\in B^*} |\mathbf{a}_i^T\mathbf{u}| \\
&\textit{subject to} \quad \mathbf{a}_i^T\mathbf{u} = 0 \quad \textit{for all } i \in E, \\
&\qquad\qquad \mathbf{a}_i^T\mathbf{u} \geqq 0 \quad \textit{for all } i \in C^*, \\
&\qquad\qquad \|\mathbf{u}\|_2 \leqq 1.
\end{aligned} \tag{2.6}$$

In other words, the vector $\mathbf{d} = \mathbf{r}/\|r\|_2$ *is the feasible direction of steepest descent at* $\mathbf{x}$.

Proof. Substitution of $\mathbf{g} = A^T\mathbf{y}^* - \mathbf{r}$ transforms the objective function into

$$-\mathbf{r}^T\mathbf{u} + S(\mathbf{u})$$

where

$$\begin{aligned}
S(\mathbf{u}) &= \sum_{i\in N} y_i^*\mathbf{a}_i^T\mathbf{u} + \sum_{i\in B^*} |\mathbf{a}_i^T\mathbf{u}| \\
&= \sum_{i\in E} y_i^*\mathbf{a}_i^T\mathbf{u} + \sum_{i\in C^*} y_i^*\mathbf{a}_i^T\mathbf{u} + \sum_{i\in B^*} (y_i^*\mathbf{a}_i^T\mathbf{u} + |\mathbf{a}_i^T\mathbf{u}|).
\end{aligned}$$

Now it is easy to verify that the inequalities $S(\mathbf{u}) \geqq 0$ and

$$-\mathbf{r}^T\mathbf{u} + S(\mathbf{u}) \geqq -\mathbf{r}^T\mathbf{u} \geqq -\|\mathbf{r}\|_2$$

hold whenever $\mathbf{u}$ satisfies the constraints of (2.6). Therefore, since $S(\mathbf{d}) = 0$, the vector $\mathbf{d}$ solves (2.6). □

COROLLARY 2. *Let* $\tilde{\mathbf{y}}$ *solve* (2.4). *If the resulting residual vector is zero and* $\tilde{\mathbf{y}}$ *satisfies the bounds*

$$-1 \leqq y_i \leqq 1 \quad \textit{for} \quad i \in B^* \quad \textit{and} \quad 0 \leqq y_i \quad \textit{for } i \in C^*, \tag{2.7}$$

then $\mathbf{x}$ *solves* (1.1).

If the rows of A are linearly dependent, then $\mathbf{x}$ is called "degenerate." Otherwise, when the rows of A are linearly independent, $\mathbf{x}$ is called "nondegenerate." The assumption of nondegeneracy provides an alternative way of computing a feasible descent direction at dead points.

THEOREM 3. *Let* $\mathbf{x}$ *be a nondegenerate dead point such that* $\tilde{\mathbf{y}}$, *the unique solution of* (2.4), *violates* (2.7). *Let* $j \in B^* \cup C^*$ *be an index such that* $\tilde{\mathbf{y}}_j$ *violates its bound. Let* $\hat{A}$ *denote the* $(t-1)\times n$ *matrix which is obtained from* A *by deleting its* j*th row. Let* $\hat{\mathbf{y}} \in \Re^{t-1}$ *solve the following problem*:

$$\textit{minimize} \ \|\hat{A}^T\mathbf{y} - \mathbf{g}\|_2^2, \tag{2.8}$$

and let $\hat{\mathbf{r}} = \hat{A}^T\hat{\mathbf{y}} - \mathbf{g}$ *denote the corresponding residual vector. Then* $\hat{\mathbf{r}}$ *is a feasible descent direction.*

Proof. The possibility $\hat{\mathbf{r}} = \mathbf{0}$ is excluded by the following argument. If $\hat{\mathbf{r}} = \mathbf{0}$, then the relation $\mathbf{g} = A^T\tilde{\mathbf{y}} = \hat{A}^T\hat{\mathbf{y}}$ contradicts the nondegeneracy assumption. It is also clear that $\hat{A}\hat{\mathbf{r}} = \mathbf{0}$, which gives

$$\mathbf{g}^T\hat{\mathbf{r}} = (\hat{A}^T\hat{\mathbf{y}} - \hat{\mathbf{r}})^T\hat{\mathbf{r}} = -\hat{\mathbf{r}}^T\hat{\mathbf{r}} < 0.$$

Similarly, the equality $\mathbf{g} = A^T\tilde{\mathbf{y}}$ implies

$$\mathbf{a}_j = \left(\mathbf{g} - \sum_{i\in\hat{N}} \tilde{\mathbf{y}}_i\mathbf{a}_i\right) \Big/ \tilde{y}_j$$

and

$$\mathbf{a}_j^T\hat{\mathbf{r}} = \mathbf{g}^T\hat{\mathbf{r}}/\tilde{y}_j = -\hat{\mathbf{r}}^T\hat{\mathbf{r}}/\tilde{y}_j$$

where $\hat{N} = N - \{j\}$. These relations yield the inequalities

$$\mathbf{a}_j^T\hat{\mathbf{r}} > 0 \quad \text{and} \quad \mathbf{g}^T\hat{\mathbf{r}} + \sum_{i\in B^*} |\mathbf{a}_i^T\hat{\mathbf{r}}| = -\hat{\mathbf{r}}^T\hat{\mathbf{r}} < 0 \quad \text{when } j \in C^*,$$

and

$$\mathbf{g}^T\hat{\mathbf{r}} + \sum_{i\in B^*} |\mathbf{a}_i^T\hat{\mathbf{r}}| = -\hat{\mathbf{r}}^T\hat{\mathbf{r}} - \hat{\mathbf{r}}^T\hat{\mathbf{r}}/\tilde{y}_j = -\hat{\mathbf{r}}^T\hat{\mathbf{r}}(1 + \tilde{y}_j^{-1}) < 0 \quad \text{when } j \in B^*.$$

Hence in both cases $\hat{\mathbf{r}}$ is a feasible descent direction. □

3. Moving away from a dead point. Let $\mathbf{x}$ be a feasible dead point such that $\tilde{\mathbf{y}}$, the solution of (2.4), violates (2.7). The discussion in the previous section suggests two possible ways of constructing a feasible descent direction at $\mathbf{x}$.

Strategy A. Solve the constrained least squares problem (2.5) and compute $\mathbf{r}$. If $\mathbf{r} = \mathbf{0}$, terminate. In this case $\mathbf{x}$ solves (1.1). Otherwise use $\mathbf{r}$ as the search direction.

Strategy B. If there is a clear indication of degeneracy (e.g., $t > n$), then apply Strategy A. Otherwise, when $\mathbf{x}$ is assumed to be a nondegenerate dead point, solve the unconstrained least squares problem (2.8) and compute $\hat{\mathbf{r}}$. If $\hat{\mathbf{r}} = \mathbf{0}$ the nondegeneracy assumption is false and Strategy A is applied. Otherwise use $\hat{\mathbf{r}}$ as the search direction.

4. The main algorithm. The basic iteration of the proposed method for solving (1.1) is composed of the following three steps.

Step M1: *Compute a search direction.* Let $\mathbf{x}$ denote the current feasible point. Solve (2.4) for $\tilde{\mathbf{y}}$ and compute $\mathbf{r}$. If $\mathbf{r} \neq \mathbf{0}$ set $\mathbf{u} = \mathbf{r}$ and skip to Step M3.

Step M2: *Moving away from a dead point* (*Strategy* A). If $\tilde{\mathbf{y}}$ satisfies (2.7), terminate. In this case $\mathbf{x}$ solves (1.1). Otherwise solve (2.5) for $\mathbf{y}^*$, and compute $\mathbf{r}$. If $\mathbf{r} = \mathbf{0}$, terminate. In this case $\mathbf{x}$ solves (1.1). Otherwise set $\mathbf{u} = \mathbf{r}$.

Step M3: *The line search.* Set the new point to be $\mathbf{x} + \lambda\mathbf{u}$, where λ minimizes $F(\mathbf{x} + \lambda\mathbf{u})$ subject to the restriction that $\mathbf{x} + \lambda\mathbf{u}$ is kept feasible.

Strategy B is implemented by inserting the corresponding changes in Step M2. The details of the line search and the way (2.5) is solved are given in the next sections. The finite termination of the algorithm is a consequence of the following properties:

(a) The objective function is strictly decreasing at each iteration.

(b) Each time that Step M2 is skipped, the number of "active" indices increases. Hence it is not possible to have an infinite succession of iterations in which Step M2 is skipped.

(c) Each time that Step M2 is executed, the current point is a minimizer of the objective function on the linear manifold

$$\{\mathbf{z} | \mathbf{a}_i^T \mathbf{z} = b_i \quad \text{for all } i \in N\}.$$

(d) There is a finite number of such manifolds.

A straightforward way to implement the algorithm for medium-sized dense problems is to solve the unconstrained least squares subproblems via the QR factorization of A^T. The updating of this factorization when a row is added to or deleted from A is discussed in many places (e.g., Bartels et al. [3]).

5. Solving the bounded least squares subproblem. This section describes an active set method for solving (2.5) whose basic iteration is similar to that of the main algorithm. The search direction is obtained by solving a modified form of (2.4), using the same factorization scheme as the main algorithm. A further feature that characterizes our method is that $\tilde{\mathbf{y}}$, the already known solution of (2.4), is used to provide an improved starting point. Also, the possibility of cycling is prevented in a simple way. These features result in an efficient algorithm, which is easily incorporated into the main one.

Let $\mathbf{y} \in \mathfrak{R}^t$ satisfy the constraints (2.7). Then the ith variable of $\mathbf{y}$ is said to be "*bounded*" if $i \in B^*$ and $|y_i| = 1$, or $i \in C^*$ and $y_i = 0$. Otherwise it is called "*free*." The number of free variables is denoted by $\bar{t}$, and the $\bar{t} \times n$ matrix whose rows correspond to free variables is denoted by $\bar{A}$. The basic iteration of the proposed method for solving (2.5) is composed of the following three steps.

Step B1: *Compute a search direction.* Let $\mathbf{y}$ denote the current feasible point and let $\mathbf{r} = A^T\mathbf{y} - \mathbf{g}$ denote the corresponding residual vector. If $\mathbf{r} = \mathbf{0}$, $\bar{t} = 0$, or $\bar{A}\mathbf{r} = \mathbf{0}$, continue to Step B2. Otherwise a descent direction, $\mathbf{v}$, is obtained by solving the following problem:

$$\begin{aligned} &\text{minimize} \quad S(\mathbf{v}) = \tfrac{1}{2}\|A^T\mathbf{v} - \mathbf{r}\|_2^2 \\ &\text{subject to} \quad v_i = 0 \quad \forall i \in M, \end{aligned} \tag{5.1}$$

where M denotes the subset of N, which contains the indices of "bounded" variables. Then, when $\mathbf{v}$ is computed, skip to Step B3.

Step B2: *Test for optimality.* Let $\mathbf{h} = A\mathbf{r}$ denote the gradient vector of the objective function $H(\mathbf{y}) = \tfrac{1}{2}\|A^T\mathbf{y} - \mathbf{g}\|_2^2$ at the current point. Let $\mathbf{h}^* \in \mathfrak{R}^t$ denote the corresponding vector of Lagrange multipliers. Then $\mathbf{h}^*$ is obtained from $\mathbf{h}$ by the following rule. For all $i \in N$ do as follows:

(a) Set $h_i^* = h_i$.

(b) If $i \in B^*$, $|y_i| = 1$ and $y_i h_i \leqq 0$, set $h_i^* = 0$.

(c) If $i \in C^*$, $y_i = 0$ and $h_i \geqq 0$, set $h_i^* = 0$.

The vector $\mathbf{h}^*$ enables us to test whether the Kuhn–Tucker optimality conditions hold at $\mathbf{y}$. If $\mathbf{h}^* = \mathbf{0}$, $\mathbf{y}$ solves (2.5) and the algorithm terminates. Otherwise, a feasible descent direction is obtained as follows. Compute an index j such that

$$|h_j^*| = \max\{|h_i^*|, i = 1, \cdots, t\}$$

and set

$$\boldsymbol{\nu} = -(h_j / \mathbf{a}_j^T \mathbf{a}_j)\mathbf{e}_j \tag{5.2}$$

where $\mathbf{e}_j$ denotes the jth column of the $t \times t$ unit matrix. (The motivation behind this rule is discussed in Dax [8].)

Step B3: *The line search.* Set the new point to be $\mathbf{y} + \rho\boldsymbol{\nu}$, where ρ is the largest number in the interval $[0, 1]$ that keeps this point feasible. This completes the basic

iteration. If $\rho = 1$ and the search direction was computed by (5.1), then the next iteration should start in Step B2.

The finite termination of the above method is proved by the same arguments as in the main algorithm. The solution of (5.1) is carried out by using the QR factorization of $\bar{A}^T$ to solve the unconstrained problem

$$\underset{\boldsymbol{\omega}\in\Re^{t}}{\text{minimize}} \|\bar{A}^T\boldsymbol{\omega} - \mathbf{r}\|_2^2. \tag{5.3}$$

The updating of this factorization when a row is added to or deleted from $\bar{A}$ is done exactly as in the main algorithm.

The initial point, $\mathbf{y}$, is obtained from $\tilde{\mathbf{y}}$ by "pruning" those components that violate their bound. In other words, for all $i \in N$, do as follows:

(a) Set $y_i = \tilde{y}_i$.
(b) If $i \in B^*$ and $|\tilde{y}_i| > 1$, set $y_i = \text{sign}\,(\tilde{y}_i)$.
(c) If $i \in C^*$ and $\tilde{y}_i < 0$, set $y_i = 0$.

This choice has two advantages: $\mathbf{y}$ seems to be a good starting point (see the numerical results); the initial $\bar{A}$ is obtained from A by deleting those rows for which $\tilde{y}_i$ violates its bound. Hence the QR factorization of $\bar{A}^T$ can be obtained from that of A^T by applying a few updates. Similarly, the final $\bar{A}$ matrix provides the new A matrix for the main algorithm and the factorization of this matrix is also available.

6. The line search algorithm. Define $\alpha_i = \mathbf{a}_i^T\mathbf{x} - b_i$, $\beta_i = \mathbf{a}_i^T\mathbf{u}$, and $\lambda_i = -\alpha_i/\beta_i$. Then the maximal steplength that keeps $\mathbf{x} + \lambda\mathbf{u}$ feasible is

$$\lambda^* = \min\{\lambda_i | i \in C - C^* \text{ and } \beta_i < 0\}.$$

(If this set is empty then $\lambda^* = \infty$.) Therefore, since $f(\lambda) = F(\mathbf{x} + \lambda\mathbf{u})$ is a convex piecewise linear function, the desired minimizer is either λ^* or one of the points $\lambda_j, j \in V$, or $0 \leqq \lambda_j \leqq \lambda^*$. The choice between these points is made by computing the corresponding values of $f(\lambda_j)$, using the relation

$$f(\lambda) = \sum_{i\in B} |\alpha_i + \lambda\beta_i|.$$

7. Further remarks. If the rows of A are linearly dependent then $\tilde{\mathbf{y}}$, the solution of (2.4), is not unique. Fortunately, the QR factorization of A^T provides a simple way to handle this situation: Assume that the QR factorization transforms A^T into an $n \times t$ upper trapezoidal matrix, R say, whose first ρ diagonal elements differ from zero, while the last $n - \rho$ rows of R are negligible. Then the last $t - \rho$ components of $\tilde{\mathbf{y}}$ are set to zero, which fixes $\tilde{\mathbf{y}}$ in a precise way. A similar strategy is applied in the solution of (5.3).

For large sparse problems, the solution of (2.4) and (2.5) should be done by methods that are able to take advantage of the special structure of these problems. This is demonstrated in Dax [7], where the analogue of (2.5) is solved by a relaxation technique.

The computation of a feasible initial point for the main algorithm can be done by a similar algorithm (see Dax [5]). The only differences between the feasible point algorithm and the main algorithm are that here

$$N = E \cup C^*, \qquad V = \{i | i \in C \text{ and } \mathbf{a}_i^T\mathbf{x} < b_i\},$$
$$F(\mathbf{x}) = \sum_{i\in V} (b_i - \mathbf{a}_i^T\mathbf{x}), \qquad \mathbf{g} = -\sum_{i\in V} \mathbf{a}_i.$$

8. Numerical results. This section presents numerical experiments with the new algorithm. The algorithm was programmed in FORTRAN and the computations were carried out on a CDC CYBER 170-855 computer, with machine precision of about 10^{-15}. The use of finite precision arithmetic forces us to modify the definitions of B^* and C^*. In practice these definitions took the form

$$B^* = \{i | i \in B \text{ and } |\mathbf{a}_i^T \mathbf{x} - b_i| \leqq \varepsilon\},$$

$$C^* = \{i | i \in C \text{ and } \mathbf{a}_i^T \mathbf{x} - b_i \leqq \varepsilon\}$$

where $\varepsilon = 10^{-10}$.

The first type of test problem consists of "Approximation" problems that have the following structure. Let the function $f(\theta)$ be defined in the interval $[0, 1]$. Let $P(\theta)$ be a polynomial of degree $n-1$ whose coefficients are $x_1, \cdots, x_n$, i.e.,

$$P(\theta) = \sum_{j=1}^{n} x_j \theta^{j-1}. \tag{8.1}$$

Let $\rho_i = i/m$, $i = 0, 1, 2, \cdots, m$, be $m+1$ points equally spaced in $[0, 1]$. Then the objective function is defined as follows:

$$F(\mathbf{x}) = \sum_{i=1}^{m} |P(\rho_i) - f(\rho_i)| = \sum_{i=1}^{m} \left| \sum_{j=1}^{n} (\rho_i)^{j-1} x_j - f(\rho_i) \right|. \tag{8.2}$$

There are $2m$ inequality constraints generated by forcing $P'(\rho_i)$ and $P''(\rho_i)$, $i = 1, \cdots, m$, to be nonnegative or nonpositive, according to $f(\theta)$. In addition to these constraints, there are two equality constraints that force $P(0)$ and $P'(0)$ to coincide with $f(0)$ and $f'(0)$, respectively. The exact definition of the constraints depends, of course, on the function $f(\theta)$. These details are summarized in Table 1. (The constraints of the last two examples are slightly different for obvious reasons.) The values of m

TABLE 1
Constraints and starting points of Approximation test problems.

$f(\theta)$	Constraints on $P'(\rho_i)$, $i = 1, \cdots, m$	Constraints on $P''(\rho_i)$, $i = 1, \cdots, m$	Equality constraints	Nondegenerate starting point[1]	Degenerate starting point[1]
e^{θ}	$P'(\rho_i) \geqq 0$	$P''(\rho_i) \geqq 0$	$P(0) = 1,\ P'(0) = 1$	$x_1 = x_2 = 1,\ x_3 = e-2$	$x_1 = 1,\ x_2 = 1$
$e^{-1/2\theta^2}$	$P'(\rho_i) \leqq 0$	$P''(\rho_i) \leqq 0$	$P(0) = 1,\ P'(0) = 0$	$x_1 = 1,\ x_3 = -\frac{1}{3}$	$x_1 = 1,\ x_2 = 0$
e^{θ^2}	$P'(\rho_i) \geqq 0$	$P''(\rho_i) \geqq 0$	$P(0) = 1,\ P'(0) = 0$	$x_1 = 1,\ x_3 = e-1$	$x_1 = 1,\ x_2 = 0$
$\sin\theta$	$P'(\rho_i) \geqq 0$	$P''(\rho_i) \leqq 0$	$P(0) = 0,\ P'(0) = 1$	$x_2 = 1,\ x_4 = -\frac{1}{6}$	$x_1 = 0,\ x_2 = 1$
$\cos\theta$	$P'(\rho_i) \leqq 0$	$P''(\rho_i) \leqq 0$	$P(0) = 1,\ P'(0) = 0$	$x_1 = 1,\ x_3 = -\frac{1}{2}$	$x_1 = 1,\ x_2 = 0$
$\tan\theta$	$P'(\rho_i) \geqq 0$	$P''(\rho_i) \geqq 0$	$P(0) = 0,\ P'(0) = 1$	$x_2 = 1,\ x_3 = \frac{1}{2}$	$x_1 = 0,\ x_2 = 1$
$\sqrt{\theta}$	$P'(\rho_i) \geqq 0$	$P''(\rho_i) \leqq 0$	$P(0) = 0,\ P(1) = 1$	$x_2 = \frac{5}{4},\ x_4 = -\frac{1}{4}$	$x_1 = 0,\ x_2 = 1$
$\theta^{1/10}$	$P'(\rho_i) \geqq 0$	$P''(\rho_i) \leqq 0$	$P(0) = 0,\ P(1) = 1$	$x_2 = \frac{5}{4},\ x_4 = -\frac{1}{4}$	$x_1 = 0,\ x_2 = 1$

[1] The components of $\mathbf{x}^{(0)}$ not mentioned here are zeros.

and n in our experiments are $n = 6$ and $m = 10, 20, 50$. Each of the test problems was solved twice, using two different (feasible) starting points (see Table 1). Note that one of these points is a degenerate dead point at which all the m constraints on $P''(\rho_i)$ are active. (In problems 2, 3, and 5 the constraints on $P'(\rho_i)$ are also active at this point.) The results of our experiments with this type of test problem are presented in Tables 2–7.

TABLE 2
Results of Approximation test problems with $n = 6$ and $m = 10$, using Strategy A.

	Nondegenerate starting point			Degenerate starting point		
$f(\theta)$	Major iterations	Solving (2.5)	Minor iterations	Major iterations	Solving (2.5)	Minor iterations
e^{θ}	6	2	4	10	4	10
$e^{-1/2\theta^2}$	10	3	6	10	4	10
e^{θ^2}	4	1	1	13	5	15
$\sin\theta$	10	3	10	7	2	2
$\cos\theta$	10	3	10	17	6	16
$\tan\theta$	12	3	8	11	4	9
$\sqrt{\theta}$	7	1	1	18	6	15
$\theta^{1/10}$	20	5	20	22	7	28

TABLE 3
Results of Approximation test problems with $n = 6$ and $m = 10$, using Strategy B.

	Nondegenerate starting point			Degenerate starting point		
$f(\theta)$	Major iterations	Solving (2.5)	Minor iterations	Major iterations	Solving (2.5)	Minor iterations
e^{θ}	6	0	0	8	1	1
$e^{-1/2\theta^2}$	5	0	0	5	1	4
e^{θ^2}	4	0	0	8	1	4
$\sin\theta$	8	0	0	8	1	1
$\cos\theta$	8	0	0	8	1	4
$\tan\theta$	7	0	0	8	1	1
$\sqrt{\theta}$	7	1	1	18	6	15
$\theta^{1/10}$	20	5	20	22	7	28

TABLE 4
Results of Approximation test problems with $n = 6$ and $m = 20$, using Strategy A.

	Nondegenerate starting point			Degenerate starting point		
$f(\theta)$	Major iterations	Solving (2.5)	Minor iterations	Major iterations	Solving (2.5)	Minor iterations
e^{θ}	14	4	15	14	4	12
$e^{-1/2\theta^2}$	12	3	8	12	4	11
e^{θ^2}	6	1	5	14	5	12
$\sin\theta$	14	3	12	15	5	15
$\cos\theta$	14	3	12	11	5	14
$\tan\theta$	14	4	10	20	6	14
$\sqrt{\theta}$	24	8	29	24	8	27
$\theta^{1/10}$	28	9	28	29	12	40

TABLE 5
Results of Approximation test problems with $n = 6$ and $m = 20$, using Strategy B.

	Nondegenerate starting point			Degenerate starting point		
$f(\theta)$	Major iterations	Solving (2.5)	Minor iterations	Major iterations	Solving (2.5)	Minor iterations
e^{θ}	9	0	0	10	1	1
$e^{-1/2\theta^2}$	9	0	0	9	1	3
e^{θ^2}	9	0	0	10	1	3
$\sin\theta$	10	0	0	10	1	1
$\cos\theta$	10	0	0	13	1	3
$\tan\theta$	8	0	0	12	1	1
$\sqrt{\theta}$	24	8	29	24	8	27
$\theta^{1/10}$	20	0	0	29	12	40

TABLE 6
Results of Approximation test problems with $n = 6$ and $m = 50$, using Strategy A.

	Nondegenerate starting point			Degenerate starting point		
$f(\theta)$	Major iterations	Solving (2.5)	Minor iterations	Major iterations	Solving (2.5)	Minor iterations
e^{θ}	18	5	11	22	7	20
$e^{-1/2\theta^2}$	16	5	14	16	6	17
e^{θ^2}	10	3	11	19	6	20
$\sin\theta$	27	7	25	16	5	14
$\cos\theta$	22	6	20	22	7	25
$\tan\theta$	12	3	6	29	10	28
$\sqrt{\theta}$	31	9	29	30	10	32
$\theta^{1/10}$	60	24	72	59	25	75

TABLE 7
Results of Approximation test problems with $n = 6$ and $m = 50$, using Strategy B.

	Nondegenerate starting point			Degenerate starting point		
$f(\theta)$	Major iterations	Solving (2.5)	Minor iterations	Major iterations	Solving (2.5)	Minor iterations
e^{θ}	12	0	0	13	1	1
$e^{1/2\theta^2}$	10	0	0	10	1	3
e^{θ^2}	10	0	0	13	1	3
$\sin\theta$	10	0	0	11	1	1
$\cos\theta$	13	0	0	12	1	3
$\tan\theta$	8	0	0	13	1	1
$\sqrt{\theta}$	31	9	29	30	10	32
$\theta^{1/10}$	33	2	7	40	2	10

In the columns headed "Major iterations," the number of iterations made by the main algorithm are given. The columns headed "Solving (2.5)," provide the number of major iterations in which problem (2.5) was solved. In the columns headed "Minor iterations," the overall number of iterations required by the algorithm for solving (2.5) are given. For example, the approximation of e^{θ}, using $m = 10$, Strategy A, and a nondegenerate starting point, needed six iterations of the main algorithm. The problem (2.5) was solved in two of the six iterations, which required four minor iterations (see the first row of Table 2).

The second category of test problems consists of "random" problems. Here the elements of the vectors $\mathbf{a}_i$ are uniformly distributed random numbers from the interval $[-1, 1]$. The numbers b_i, $i \in B \cup C$, are generated in the same way, whereas the right-hand sides of the equality constraints are defined by the rule $b_i = \sum_{j=1}^{n} a_{ij}$. The starting point is $\mathbf{e} = (1, 1, \cdots, 1)^T \in \mathfrak{R}^n$. In order to ensure that this point is feasible, we check the inequality constraints and multiply by -1 those that are violated at this point. The experiments were done with $n = 20$, five equality constraints, and fifteen inequality constraints. The number of terms in the objective function (i.e., the size of B) takes the values 5, 10, 20, and 40. In each of these four cases, we solved 10 different problems, and the averaged results are given in Table 8.

TABLE 8
Results of Random test problems.[1]

No. of terms in $F(\mathbf{x})$	No. of variables	No. of equalities	No. of inequalities	Major iterations	Solving (2.5)	Minor iterations
5	20	5	15	9.3	0.2	0.4
10	20	5	15	16.5	0.9	2.6
20	20	5	15	20.4	2.1	7.7
40	20	5	15	24.9	3.1	13.7

[1] Here both strategies gave the same results.

The third category of test problems consists of "integer" problems. In these problems $n = 5$ and for each $i \in B$ the elements a_{i1}, a_{i2}, a_{i3} and b_i are random integers between (including) zero and nine, $a_{i4} = a_{i1} + a_{i2} + a_{i3}$ and $a_{i5} = a_{i1} + a_{i2} - a_{i3}$. There is one equality constraint, $\sum_{j=1}^{n} x_j = 1$, and five inequalities of the form $x_j \geqq 0$, $j = 1, \cdots, 5$. The starting point is always $\mathbf{e}/5$. The number of terms in the objective function takes the values 5, 10, 15, and 20. In each case, 10 different problems of this type were solved (see Table 9).

The fourth type of test problem is called a "simple bounds" problem. These problems have the form

$$\begin{aligned} &\text{minimize} && \|A\mathbf{x} - \mathbf{b}\|_1 \\ &\text{subject to} && -1 \leqq x_j \leqq 1, \quad j = 1, \cdots, n, \end{aligned} \tag{8.3}$$

where A is a $m \times n$ matrix whose elements are uniformly distributed random numbers from the interval $[-1, 1]$. The right-hand side vector, $\mathbf{b}$, is defined as $A\mathbf{z}$, where $\mathbf{z} \in \mathfrak{R}^n$ and the components of $\mathbf{z}$ are uniformly distributed random numbers from the interval $[-2, 2]$. These problems were solved for $n = 20$ and $m = 10$, 20, 40. For each value of m, 10 different problems of this type were solved (see Table 10).

We have also solved a problem due to M. Cox (see Bartels et al. [3]) as well as a "Hilbert" test problem taken from Barrodale and Roberts [1] and based on a similar example by Powell [12] (see Tables 11 and 12, respectively).

TABLE 9
Results of Integer test problems.

No. of terms in $F(\mathbf{x})$	No. of variables	No. of equalities	No. of inequalities	Major iterations	Solving (2.5)	Minor iterations	Strategy
5	5	1	5	3.8	0.4	0.6	A
5	5	1	5	3.8	0.1	0.4	B
10	5	1	5	4.1	0.5	1.2	A
10	5	1	5	4.0	0.2	0.7	B
15	5	1	5	4.8	0.9	1.2	A
15	5	1	5	4.9	0.3	0.6	B
20	5	1	5	4.9	0.9	2.2	A
20	5	1	5	4.8	0.7	2.4	B

TABLE 10
Results of Simple Bounds test problems.

		Strategy A			Strategy B		
m	Starting point	Major iterations	Solving (2.5)	Minor iterations	Major iterations	Solving (2.5)	Minor iterations
10	**0**	20.7	0.8	0.9	21.2	0	0
10	**e**	21.8	3.7	9.3	40.6	0	0
20	**0**	23.4	1.5	4.1	24.2	0	0
20	**e**	26.7	4.4	15.6	57.3	0	0
40	**0**	26.8	2.1	7.4	32.7	0	0
40	**e**	32.4	5.3	20.9	69.7	0	0

TABLE 11
Results of Cox's test problem.

Starting point	No. of terms in $F(\mathbf{x})$	No. of variables	No. of equalities	No. of inequalities	Major iterations	Solving (2.5)	Minor iterations	Strategy
0	9	7	0	5	4	3	13	A
0	9	7	0	5	5	2	12	B
e	9	7	0	5	5	3	13	A
e	9	7	0	5	6	2	12	B

TABLE 12
Results of a Hilbert test problem.

Starting point	No. of terms in $F(\mathbf{x})$	No. of variables	No. of equalities	No. of inequalities	Major iterations	Solving (2.5)	Minor iterations	Strategy
0	20	10	0	20	27	1	1	A
0	20	10	0	20	27	0	0	B
500 **e**	20	10	0	20	41	2	2	A
500 **e**	20	10	0	20	74	0	0	B

The comparison of Strategies A and B reveals that the second strategy is advantageous in solving "approximation" test problems, while for "simple bounds" problems the situation is reversed. In the other test problems there is no significant difference between the two options.

The results of our experiments are quite encouraging. It seems that the proposed method compares favourably with the Simplex Method of Barrodale and Roberts [1], and the Penalty Function Method of Bartels and Conn [2]. It is illustrated that the number of "minor" iterations needed to solve (2.5) is usually small. The maximum number recorded is seven, while the average number is smaller than four. This indicates that the proposed algorithm for solving (2.5) provides an efficient way to resolve degeneracy.

Acknowledgment. I thank Danit Naimi (Sharon) for the superb typing of this and many of my other works.

REFERENCES

[1] I. BARRODALE AND F. D. K. ROBERTS, *An efficient algorithm for discrete l_1 linear approximation with linear constraints*, SIAM J. Numer. Anal., 15 (1978), pp. 603–611.

[2] R. H. BARTELS AND A. R. CONN, *Linearly constrained discrete l_1 problems*, ACM Trans. Math. Software, 6 (1980), pp. 594–608.

[3] R. H. BARTELS, A. R. CONN, AND J. W. SINCLAIR, *Minimization techniques for piecewise differentiable functions: the l_1 solution to an overdetermined linear system*, SIAM J. Numer. Anal., 15 (1978), pp. 224–241.

[4] S. BUSOVAČA, *Handling degeneracy in a nonlinear l_1 algorithm*, Tech. Report CS85 34, University of Waterloo, Waterloo, Ontario, Canada, 1985.

[5] A. DAX, *The computation of descent directions at degenerate points*, Tech. Report Hydrological Service of Israel, Jerusalem, Israel, 1985.

[6] ———, *A note on optimality conditions for the Euclidean multifacility location problem*, Math. Programming, 36 (1986), pp. 72–80.

[7] ———, *An efficient algorithm for solving the rectilinear multifacility location problem*, IMA J. Numer. Anal., 6 (1986), pp. 343–355.

[8] ———, *On computational aspects of bounded linear least squares problems*, Tech. Report, Hydrological Service of Israel, Jerusalem, Israel, 1986.

[9] ———, *The minimax solution of linear equations subject to linear constraints*, Tech. Report, Hydrological Service of Israel, Jerusalem, Israel, 1986.

[10] ———, *Linear programming via least squares*, Tech. Report, Hydrological Service of Israel, Jerusalem, Israel, 1986.

[11] M. R. OSBORNE, *Finite Algorithms in Optimization and Data Analysis*, John Wiley, New York, 1985.

[12] M. J. D. POWELL, *The minimax solution of linear equations subject to bounds on the variables*, in Proc. 4th Manitoba Conference on Numerical Mathematics, B. L. Hartnell and H. C. Williams, eds., Utilas Mathematica, Winnipeg, Ontario, Canada, 1975, pp. 53–107.

[13] G. A. WATSON, *Discrete l_1 approximation by rational functions*, IMA J. Numer. Anal., 4 (1984), pp. 275–288.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 341-358, March 1989
© 1989 Society for Industrial and Applied Mathematics
007

AN ACCELERATED SEQUENTIAL ALGORITHM FOR PRODUCING *D*-OPTIMAL DESIGNS*

T. G. ROBERTAZZI† AND S. C. SCHWARTZ‡

Abstract. An extremely efficient algorithm is presented for generating *D*-optimal designs. It is shown that the underlying set function of the sequential design algorithm is submodular and thus the so-called accelerated greedy algorithm may be applied to this problem. While the new algorithm's statistical basis is identical to that of the Wynn and Federov algorithm, it requires significantly fewer function evaluations. The new algorithm is particularly useful when no prior information concerning the structure of the optimal design is available.

Key words. optimal experimental design, *D*-optimal design, greedy algorithm, accelerated greedy algorithm, submodular set function, sequential design

AMS(MOS) subject classifications. 05C, 62K05, 68R05, 68R10

1. Introduction. This paper presents an efficient iterative algorithm that asymptotically produces *D*-optimal designs. These designs are based on a continuous allocation of measurements rather than on the integer allocation that has been examined by Mitchell (1974) and Galil and Kiefer (1980). The statistical foundation of this new algorithm is the same as that for the iterative procedure analyzed by Wynn (1970) and Federov (1972, p. 102). However this algorithm requires far fewer function evaluations than what shall be referred to as the standard sequential design algorithm.

It is pointed out below that the standard sequential design algorithm embodies the principle of the greedy algorithm of combinatorial optimization. It is also shown that a certain set function property, submodularity, is also present in the *D*-optimal design problem. This allows the accelerated greedy algorithm of Minoux (1977) to be used as the basis of this new "accelerated sequential design algorithm."

Recent work on experimental design algorithms has implicitly recognized, if not explicitly mentioned, that there is a close relationship between these procedures and the greedy-type algorithms of combinatorial optimization. Mori and DiStefano (1979) have developed an exchange-type greedy procedure for a measurement-scheduling problem closely related to the *D*-optimal design problem. Mitchell (1974) has developed an exchange-type algorithm specifically for the *D*-optimal design problem. Welch (1982) used a branch-and-bound technique to produce *D*-optimal designs. These authors and Galil and Kiefer (1980) and Federov (1972) have also stressed the practical importance of design algorithms that do not require an excessive amount of computation.

In § 2 the standard sequential design algorithm and related procedures will be reviewed. Section 3 presents an overview of greedy algorithms, with particular emphasis on the accelerated greedy algorithm of Minoux (1977). The relationship between the design problem and submodular set functions is described in § 4. This leads naturally to the new accelerated design algorithm.

The accelerated greedy algorithm exploits submodularity to avoid performing many of the function evaluations that the standard greedy algorithm would require.

* Received by the editors November 9, 1982; accepted for publication (in revised form) June 14, 1988. This research has been supported by the Office of Naval Research under contracts N00014-80-C-0530 and N00014-81-K-0146.

† Department of Electrical Engineering, State University of New York, Stony Brook, New York 11794.
‡ Department of Electrical Engineering, Princeton University, Princeton, New Jersey 08544.

It has been applied to network optimization problems arising in the context of operations research (Minoux (1981) and Biesel-Guitonneau (1980)). Section 5 presents examples of the use of this algorithm. For the particular set of basis functions used in this paper the standard algorithm requires approximately 40–140 times as many function evaluations as the accelerated algorithm. For instance, for one § 5 example with a 31×31 grid, the standard algorithm requires 1.6 million evaluations as opposed to the accelerated procedure, which requires only slightly more than 32,000.

Finally, in § 6, conditions for which the accelerated algorithm can efficiently produce optimal designs for dynamic models are outlined.

2. Experimental design algorithms. An experimental design is a distribution of measurements in a given sampling region within the space of independent (control) variables. An optimal experimental design maximizes some criterion expressing the amount of statistical information concerning unknown model parameters that is obtained as a result of the measurements.

Consider the linear static regression model:

$$y_k = f(x_k)\beta + \varepsilon_k, \qquad k = 1, 2, \cdots, n. \tag{2.1}$$

Here β is the $p \times 1$ vector of parameters to be estimated and $f(x_k)$ is the $1 \times p$ vector of basis function values at the kth measurement location. These locations are contained within a finite-dimensional independent parameter space. The error terms ε_k are assumed to be independent and identically distributed with zero mean and variance σ^2. If the n discrete replicated measurements are made at L distinct sampling locations, the Fisher information matrix, normalized for the number of measurements, is

$$(M_d)_{rs} = (X^T X)_{rs} = \frac{1}{n} \sum_{j=1}^{L} f_r(x_j) f_s(x_j) n_j. \tag{2.2}$$

Here $f_r(x_j)$ is the value of the rth basis function at the jth location. There are n_j out of n replicated measurements made at the jth sampling location. The subscript d indicates that the design consists of discrete measurements.

Significant progress in producing optimal designs analytically was made with the recognition by Kiefer and Wolfowitz (1959) that the realistic, discrete, measurement situation described above could be approximated by a (possibly) continuous design measure $\xi(x)$ for a given sampling region χ satisfying:

$$\xi(x) \geqq 0, \qquad x \in \chi, \tag{2.3}$$

$$\int_{\chi} \xi(dx) = 1. \tag{2.4}$$

The information matrix for the continuous ("c") design $\xi(x)$ is

$$M_c(\xi)_{rs} = \int_{\chi} f_r(x) f_s(x) \xi(dx). \tag{2.5}$$

A design $\xi^*(x)$ that maximizes the determinant of (2.5) over all possible designs in χ, which associated nonsingular $M_c(\xi)$, is said to be D-optimal. A complete exposition of D-optimal design theory for static models appears in Federov (1972), whereas St. John and Draper (1975) provide a more concise survey. Mehra (1976) and Titterington (1980) discuss D-optimal design theory for dynamic systems. Zarrop's monograph (1979) emphasizes system identification applications.

For common sets of basis functions D-optimal designs tend to consist of allocating nonzero design measure to a finite number of measurement locations. However it may not be possible to construct an optimal discrete design from a D-optimal design if the n measurements cannot be allocated according to the optimal proportions. Naturally, if n is large, the departure from optimality will be small.

The development of D-optimal designs from analytic considerations is possible only for certain manageable sets of basis functions and sampling regions (see Karlin and Studden (1966)). Computer design algorithms offer a means of overcoming this difficulty.

For D-optimal designs over continuous sampling regions a straightforward approach would be to use an iterative nonlinear optimization routine (e.g., Fletcher-Powell, etc. $\cdots$) to solve for the optimal measurement locations and allocation proportions. Unfortunately this technique requires that the number of measurement locations be known initially. There is also some question as to how many locations can practicably be handled (Federov (1972)). More importantly, it may be difficult to incorporate complicated sampling region boundaries into the optimization procedure. A more specific, and frequently mentioned approach for the continuous design problem is the standard sequential design algorithm:

STANDARD SEQUENTIAL (UNNORMALIZED) DESIGN ALGORITHM

(1) Begin with some nonsingular allocation of measurements (design) within the sampling region χ.

(2) Find the measurement location where adding a measurement to the previous design would maximize the increase in the determinant of the Fisher information matrix. Update the old design by including an extra measurement at this location. Continue step (2) until some termination test is satisfied.

Wynn (1970) showed that this procedure asymptotically produces D-optimal designs. The above version actually produces unnormalized designs but naturally any unnormalized design can be scaled into a normalized design. Wynn's procedure is a specialization of a more general class of procedures outlined by Federov (see Federov (1972)).

For designs in a small number of dimensions (of the independent variable) the evaluations for finding the location yielding the maximum improvement can be performed on an equi-spaced grid. Let $V(\hat{\beta}) = \sigma^2 I_c^{-1}$. Since

$$\frac{|M_c + f(x_k)^T f(x_k)|}{|M_c|} = \frac{\sigma^2 + f(x_k) V(\hat{\beta}) f(x_k)^T}{\sigma^2} \tag{2.6}$$

the many determinant evaluations that step (2) seems to require can be replaced by evaluations of the predicted measurement variance, $f(x_k) V(\hat{\beta}) f(x_k)^T$. This is significant because a simple recursion exists for calculating this quantity from its previous values (Federov (1972, § 2.5), Galil and Kiefer (1980)):

$$f(x_k) V(\hat{\beta})_{i+1} f(x_k)^T = f(x_k) V(\hat{\beta})_i f(x_k)^T - \frac{(f(x_k) V(\hat{\beta})_i f(x_i)^T)^2}{f(x_i) V(\hat{\beta})_i f(x_i)^T + \sigma^2}. \tag{2.7}$$

Here $f(x_i)$ is the vector of basis function values at the location where the predicted measurement variance is maximized during the ith iteration. The term at the left side of the equation is the updated predicted measurement variance at the kth location. $V(\hat{\beta})_{i+1}$ is the $i+1$st covariance matrix for the associated $i+1$st continuous design. This expression can be developed by taking the classical measurement updating equation for the covariance matrix, which is used extensively in Kalman filtering and pre- and post-multiplying it by $f(x_k)$ and $f(x_k)^T$.

The dominant computational cost in the above recursion is in evaluating the basis functions at each kth location and in forming the product in the numerator of the right-side fraction. The basis function evaluations can be avoided by precomputing and storing the np values. However this may not be possible for a very large number of gridpoints. The product formation requires about p multiplications and adds in all.

It should be noted that the differencing implicit in the above recursion can lead to numerical problems over a large number of iterations. Because of the numerical inaccuracies that will eventually appear, measurement locations other than the one corresponding to the true maximum may be selected in step (2) of the sequential design algorithm. This is not necessarily fatal and the difference in the final design may not be noticeable. However this does point out the need for performing the above recursion in high precision arithmetic and perhaps calculating the predicted measurement variance directly every so often. If the direct calculation of $f(x_k)V(\hat{\beta})_{i+1}f(x_k)^T$ were performed at each iteration, it would require approximately (p^2+p) multiplications, not counting the basis function evaluations.

The standard design algorithm that appears above is not as general as it could be. In saying that a measurement is added at the location of maximum increase in the objective function, an unnormalized situation is implied. Assume that the initial, nonsingular, design consists of a single measurement at each of L locations and another measurement is added at some location. In terms of the design measure at the $i+1$st iteration,

$$\xi_{i+1}(x)=\frac{L}{L+1}\,\xi_i(x)+\frac{1}{L+1}\,\xi_{1\,\text{location}}(x) \tag{2.8}$$

where $\xi_{1\,\text{location}}(x)$ is the allocation of design measure to the new location. This can be written more generally as

$$\xi_{i+1}(x)=(1-\alpha_i)\xi_i(x)+\alpha_i\xi_{1\,\text{location}}(x) \tag{2.9}$$

with $0\leqq\alpha_i\leqq 1$ and

$$f(x_k)V(\hat{\beta})_{i+1}f(x_k)^T=(1-\alpha_i)^{-1} \cdot\left[f(x_k)V(\hat{\beta})_i f(x_k)^T-\frac{\alpha_i(f(x_k)V(\hat{\beta})_i f(x_i)^T)^2}{\alpha_i f(x_i)V(\hat{\beta})_i f(x_i)^T+(1-\alpha_i)\sigma^2}\right]. \tag{2.10}$$

Proofs of the convergence of the standard sequential design algorithm first show that $|V(\hat{\beta})_{i+1}|\leqq|V(\hat{\beta})_i|$ for small enough α_i. Based on this we can find the α_i at each iteration that maximizes the incremental improvement. A simple closed-form expression for this α_i appears in Federov (1972, p. 102), where convergence for this choice is also proved. Federov also showed that convergence will occur for any positive sequence α_i that satisfies:

$$\lim_{i\to\infty}\alpha_i=0,\qquad\sum_{i=1}^{\infty}\alpha_i=\infty. \tag{2.11}$$

As with stochastic approximation techniques, the choice $\alpha_i=1/(c+i)$ for some constant c is most common. Wynn (1970) showed convergence with $c=1$ and Tsay (1976) empirically compared some other choices.

A practical point is that the final design may be improved by redistributing small numbers of measurements that have been allocated to points not part of a D-optimal design (Federov (1972)). Unwanted points from the initial design, for instance, will otherwise be present in the final design. A redistribution can sometimes also produce a symmetrical design, without sacrificing efficiency.

Another approach for generating D-optimal designs is Mitchell's DETMAX procedure (1974). This procedure is more suited to the discrete design problem (2.2) as it attempts to find a best design consisting of n measurements, where n is chosen a priori and is usually small (less than 30). The basic idea is to perform "excursions" from the current solution. These consist of a sequence of additions and removals of several measurements from a current n measurement design that results in an improved design. DETMAX utilizes a user-supplied list of candidate measurement locations. However it is not efficient when dealing with the large number of candidate grid points necessary to approximate a continuous sampling region.

It is not possible to directly compare sequential and exchange procedures as they are best suited for different types of problems. In the context of discrete designs, DETMAX will usually produce more nearly optimal results. However Mitchell (1974) did point out that sequentially generated designs are seldom far off, require less time to compute, and are fine for getting a "good" design quickly. Galil and Kiefer (1980) discuss some refinements of the DETMAX procedure.

3. Greedy algorithms.

3.1. Standard greedy algorithm. Many combinatorial problems to which the greedy algorithm is applicable can be formulated as an optimal subset problem (Minoux 1977)). That is, given a set E of n elements, and a function defined on the power set of E, $P(E)$, find the subset S^* such that

$$f(S^*) = \max_{S \subseteq E} f(S). \tag{3.1}$$

The standard greedy algorithm can then be written as follows.

STANDARD GREEDY ALGORITHM

(1) Begin at iteration $i = 0$ with the empty set S_0.

(2) At the current iteration i, find in the set $E - S_i$ of unused elements, that element e_j which, when appended to S_i, maximizes the incremental inprovement in f.

If the corresponding incremental improvement is negative, stop. Otherwise, set

$$S_{i+1} = S_i \cup \{e_j\}$$
$$i \leftarrow i+1$$

and return to (2).

The essential characteristic of greedy algorithms is that at each iteration an element that maximizes the immediate incremental improvement in the objective function is selected.

Greedy algorithms have been a subject of wide interest since the demonstration by Kruskal (1956) that such a procedure produces the minimal cost spanning tree for an undirected graph with edge costs. Generally though, greedy solutions are not optimal. Edmonds (1971) studied the class of problems for which the greedy algorithm always provides an optimal solution and showed that this property properly characterizes matroid optimization problems. Fisher, Nemhauser, and Wolsey (1978) developed bounds on the extent to which greedy solutions in more general situations are close to optimal solutions.

There are several variations of the greedy algorithm that are commonly used. Nominally, greedy algorithms search in a neighborhood about the current solution that is only one element deep. Certain problems are solved more easily if we search in a larger neighborhood (see Lin (1965) and Papadimitriou and Steiglitz (1982)). The procedure of Mitchell (1974) also embodies this approach.

We may also start with some initial guess as to the solution S^*, and alternately remove and add elements that provide incremental improvements of the objective function. Both Mori and DiStefano (1979) and Mitchell (1974) use this type of exchange strategy.

Finally the "stingy" algorithm starts with $S = E$ and removes elements so as to maximize the incremental improvement. The degree to which these variations represent improvements over the standard greedy algorithm is problem dependent.

The greedy approach is also present in the standard sequential design algorithm. Here E is the set of measurement locations. We begin with any nonsingular allocation of measurements. The standard sequential design algorithm selects measurement locations from E with replacement. However this is not a major difference with respect to the standard greedy algorithm. The function $f(S)$ is the determinant of the error covariance matrix associated with the measurement locations that have been selected. The number of times any particular element is chosen is recorded and we are interested in the asymptotic relative proportions of these choices.

3.2. The accelerated greedy algorithm. The accelerated greedy algorithm of Minoux (1977), (1981) assumes that a particular set function property, submodularity, is inherent in problems to which it is applied, or at least serves as a good approximation.

A set function $f(S)$, which maps the elements of $P(E)$ into the real numbers is submodular if and only if for every $S \subseteq E$, $T \subseteq E$

$$f(S)+f(T) \geqq f(S \cup T)+f(S \cap T). \tag{3.2}$$

If the inequality is reversed, supermodularity results. There is an equivalent definition that provides more insight into the nature of submodularity. The incremental improvement of the objective function obtained by appending an element e_j to a set A (which does not contain e_j) can be defined as follows:

$$\delta(A, e_j) = f(A+\{e_j\}) - f(A). \tag{3.3}$$

A set function is submodular if and only if for every $B \subseteq A \subseteq E$, and e_j not a member of A,

$$\delta(A, e_j) \leqq \delta(B, e_j). \tag{3.4}$$

Submodularity has been described as a combinatorial analogue of convexity (Minoux (1977)). We might also think of it as a combinatorial version of the law of diminishing returns. The incremental improvement is larger if e_j is appended to a subset of A, rather than to the set itself. This is suggestive of the situation in estimating statistical parameters, as in stochastic approximation-like estimators. Generally the incremental improvement in the accuracy decreases as the number of measurements already processed increases. In the following section this will be demonstrated explicitly in the context of generating D-optimal designs.

In the standard greedy algorithm at the ith iteration we evaluate $\delta(S_i, e_j)$ for all e_j not contained in the current solution S_i. The element that provides the maximum δ is chosen. If all the elements of E may eventually be selected, the entire algorithm requires at most $n^2/2$ evaluations of the incremental improvement. Minoux's accelerated greedy algorithm exploits submodularity so as to avoid performing many of these incremental improvement evaluations.

During the first iteration of the accelerated greedy algorithm, the incremental differences are computed for all the elements. The element associated with the maximum difference is selected and the remaining incremental differences are *stored* for use

during the next iteration. These subsequent iterations involve a number of subiterations. In each subiteration the maximum incremental difference in *storage* is sought. This may be an old incremental difference. It is updated. If the new, updated value is larger than all the values in storage, it must be the true maximum. This is a consequence of (3.4). None of the values in storage that could possibly be updated may increase in value.

Suppose that an updated value is not larger than all the elements in storage. A flag corresponding to the updated value is set to indicate that the updating has been performed and that value is returned to storage. The largest value in storage is found and another subiteration occurs. Eventually a subiteration produces a maximum value or the largest value found in storage has been updated previously. This element is appended to S_i to produce S_{i+1}, the flags are reset, and another iteration occurs.

A more detailed outline of this algorithm appears in Minoux (1977).

Let a solution produced by the standard greedy algorithm be called a "greedy solution." It is not unique unless at each iteration of the standard greedy algorithm there is a unique element that provides the maximum incremental improvement. Minoux (1977) has pointed out the following:

(1) If f is a submodular set function, the accelerated greedy algorithm produces a greedy solution.

(2) If f is a submodular set function and the greedy solution is unique, then the accelerated greedy algorithm produces the same solution.

The accelerated greedy algorithm achieves savings by evaluating the incremental improvement for only a small number of e_j's at each iteration rather than for all possible e_j's, as in the standard algorithm. Although the accelerated algorithm yields significant savings on a variety of combinatorial problems (Minoux (1977), Biesel-Guitonneau (1980)) it is possible to construct worst-case scenarios where the accelerated algorithm would require as many evaluations as the standard algorithm. However Minoux has shown the following:

(3) The algorithms studied by Edmonds (1971) are essentially greedy algorithms that search in a neighborhood that is 1 deep. No other algorithm in this class can provide a greedy solution to a submodular set function problem with fewer evaluations.

There is also some overhead associated with the accelerated greedy algorithm in the sense that it potentially requires finding the maximum of a set of numbers many more times than the standard greedy algorithm. This will be discussed in the following section in the context of using the accelerated greedy algorithm for the design problem.

4. The accelerated greedy algorithm and experimental design.

4.1. Submodularity and the design problem. If the accelerated greedy algorithm is to be applied to the experimental design problem, the appropriate $f(S)$ should be submodular. Specifically $f(S)$ is the determinant of the error covariance matrix associated with a set S of replicated measurements. These measurements are drawn from the set of possible measurement locations E. Because replications are allowed, the algorithm selects measurement locations from E with replacement.

This necessitates a modification of the previous definition of submodularity. Let a replication function $r_i(e_j)$ be defined over the individual elements of E. This is the number of times a measurement is made at location e_j in the design produced at the ith iteration.

Suppose there are two designs, A and B. Let $B < A$ if and only if $r_B(e_j) < r_A(e_j)$ for at least one value of j and $r_B(e_j) \leqq r_A(e_j)$ for the remaining ones. Let $B \leqq A$ if and only if $r_B(e_j) \leqq r_A(e_j)$ for all j. The expanded form of submodularity will be said to hold if, for every $B \leqq A \leqq E$, and every e_j, condition (3.4) is satisfied. Note that if a

function $f(S)$ is submodular in this situation, it is also submodular if the elements of S were sampled without replacement.

The function $f(S)$ for unnormalized designs is indeed submodular in the replicated selection framework. The incremental improvement that results from appending a single measurement to the current estimate is

$$|V(\hat{\beta})_{i+1}| - |V(\hat{\beta})_i|$$

$$= \left| \left(I - \frac{V(\hat{\beta})_i f(x_k)^T f(x_k)}{f(x_k) V(\hat{\beta})_i f(x_k)^T + \sigma_i^2} \right) (V(\hat{\beta})_i) \right| - |V(\hat{\beta})_i| \tag{4.1}$$

$$= \left(\left| I - \frac{V(\hat{\beta})_i f(x_k)^T f(x_k)}{f(x_k) V(\hat{\beta})_i f(x_k)^T + \sigma_i^2} \right| - 1 \right) |V(\hat{\beta})_i| \tag{4.2}$$

$$= \frac{-f(x_k) V(\hat{\beta})_i f(x_k)^T}{f(x_k) V(\hat{\beta})_i f(x_k)^T + \sigma_i^2} |V(\hat{\beta})_i|. \tag{4.3}$$

The incremental improvement is the product of two terms. The term at the right of (4.3), $|V(\hat{\beta})_i|$, is nonincreasing in the number of measurements that have been processed (see (2.6)). This is also true of the term at the left. To see this, note that the term is monotonic in the predicted measurement variance $f(x_k) V(\hat{\beta})_i f(x_k)^T$. This quantity is nonincreasing in the number of measurements processed (set $k = i$ in (2.7)).

The incremental improvement, then, is nonincreasing in the number of measurements that have already been processed. If $\{e_j\}$ is to be appended to a set $B \subseteq A \subseteq E$, the incremental difference would be equal to, or larger than, the incremental difference that would result in appending $\{e_j\}$ to A. This means that (3.4) is satisfied and the function is submodular.

The covariance matrix determinant is not submodular if we consider normalized designs. Naturally any unnormalized design can be scaled into a normalized design.

It is worth mentioning that if the accelerated design algorithm is constrained to sample measurement locations without replacement, the sequence of locations it selects serves as an approximate solution for a particular best measurement processing order problem (Robertazzi and Schwartz (1981a), (1981c), and (1982)).

4.2. The accelerated design algorithm. The following is the accelerated design algorithm for producing D-optimal designs. It is based on using the predicted measurement variance (see 2.6).

ACCELERATED SEQUENTIAL (UNNORMALIZED) DESIGN ALGORITHM

(1) Begin with some initial allocation of measurements. This is in terms of a set of locations S_0 and an associated replication function $r_0(e_k)$. Compute the corresponding $V(\hat{\beta})_0$.

(2) For every measurement location $e_k \in E$ compute and store

$$f(x_k) V(\hat{\beta})_0 f(x_k)^T.$$

(3) At the current iteration i:

(3a) Find $\{e_k^{\max}\}$. This is an element that satisfies

$$\left\{ e_k : \max_{e_k \in E} f(x_k) V(\hat{\beta})_i f(x_k)^T \right\}.$$

If $i = 0$ or if $\{e_k^{\max}\}$ has already been selected during this iteration, go to (4).

(3b) For $\{e_k^{\max}\}$, update the corresponding predicted measurement variance in storage:

$$f(x_k)^{\max} V(\hat{\beta})_{i+1} f(x_k)^{T\max}.$$

Here $f(x_k)^{\max}$ is the vector of basis functions associated with $\{e_k^{\max}\}$.

If this updated predicted measurement variance is less than some other value in storage, return to (3a). Otherwise go to (4).

(4) The maximum element is appended to the current solution by updating S_i and $r_i(e_k^{\max})$:

$$S_{i+1} = S_i \cup \{e_k^{\max}\},$$

$$r_{i+1}(e_k^{\max}) = r_i(e_k^{\max}) + 1.$$

(Naturally, r_{i+1} at the other locations is set equal to the value of r_i for each location.)

The $i+1$st iteration begins.

If the termination criterion is not satisfied, go to (3a). Such a criterion will indicate when there is insufficient difference between successive iterations to warrant continuing. The Kiefer bound can be used for this purpose.

The total number of function evaluations performed by the accelerated sequential design algorithm is

$$\sum_{i=1}^{i_{\text{total}}} c_{bf}(i) n. \tag{4.4}$$

Here n is the number of measurement locations (gridpoints), i_{total} is the number of iterations the algorithm performs and $c_{bf}(i)$ is the fraction of the total number of measurement locations that are evaluated at iteration i. This varies with the basis functions in question.

The number of evaluations performed by the accelerated design algorithm tends to decrease as the iterations progress. Intuitively, this is because the change in the predicted measurement variance that results from updating tends to decrease. As a consequence, there are fewer locations whose predicted measurement variance is larger than that of an updated location. This leads to fewer subiterations and fewer evaluations.

By way of comparison, the total number of evaluations performed by the standard sequential design algorithm is

$$i_{\text{total}} n. \tag{4.5}$$

It is important to realize that the cost of one standard design algorithm evaluation is somewhat less than the cost of one accelerated design algorithm evaluation. The standard algorithm can take advantage of the p multiplications per evaluation of (2.7) rather than the p^2 multiplications involved in forming the product $f(x_k) V(\hat{\beta})_i f(x_k)^T$. To achieve practical savings the accelerated design algorithm must perform a small enough number of evaluations to compensate for this. As the cost of calculating the basis function values $f(x_k)$ increases, this difference becomes less important.

4.3. Including a heap in the accelerated design algorithm. It has already been mentioned that there is also a certain cost associated with finding the maximum stored predicted measurement variance during each subiteration (step (3a) in the algorithm). Finding the maximum by direct and exhaustive comparisons would require

$$\left(\sum_{i=1}^{i_{\text{total}}} c_{bf}(i) n \right) n = \sum_{i=1}^{i_{\text{total}}} c_{bf}(i) n^2 \tag{4.6}$$

comparisons. The cost of a comparison is essentially the cost of a subtraction. With modern computing machines, this is equivalent to the cost of a multiplication. For the design problem, n is generally on the order of several thousand and the equivalent cost of a function evaluation is often below 100 multiplications. Thus the cost of finding the maximum can easily become the dominant computational burden.

The use of a heap-type data structure can significantly reduce this cost (Williams (1964)). A heap is a binary tree where the value associated with each node is greater than or equal to the values associated with its two leaves. A heap is particularly easy to store as a column vector A. The value associated with the jth node is $A(j)$. The leaves of the jth node are nodes $2j$ and $2j+1$. Their associated values are stored in $A(2j)$ and $A(2j+1)$, respectively. The root is node 1 with value $A(1)$.

For the design problem, two numbers should be associated with the jth node of the heap. One number, $A(j; 1)$ signifies to which measurement location the node corresponds. The other, $A(j; 2)$, is the most recent evaluation of the predicted measurement variance for that location. Naturally the latter quantity is used in constructing and maintaining the heap.

Initially the elements can be rearranged to form a heap using only n comparisons (see Aho, Hopcroft, and Ullman (1976)). The maximum value of the predicted measurement variances now appears in $A(1; 2)$. After each subiteration $A(1; 2)$ will be replaced with a value that is not greater and probably smaller than the old value. To reconstitute the heap, a series of exchanges is made, in the natural way, which transfers this value (actually $A(1; 1)$ and $A(1; 2)$) to an appropriate position in the tree. The new maximum element will be in the root. If there are n locations (nodes), there will be about $\log_2 n$ levels in the heap.

The number of comparisons involved in reforming the heap will be at most $\log_2 n$. The total number of comparisons necessary for finding the maximum elements is now bounded from above by

$$\sum_{i=1}^{i_{\text{total}}} c_{bf}(i) n \log_2 n. \tag{4.7}$$

While the growth rate, as a function of n, is potentially larger than that of (4.4), the cost of performing the evaluations of the incremental improvement will generally dominate the cost of making the comparisons.

5. Use of the accelerated design algorithm.

5.1. Structure and numerical accuracy. The accelerated design algorithm is most useful when no prior information concerning the structure of the optimal design is available. The algorithm can be productively used to establish this structure. With the approximate coordinates of sampling locations and the proportion of measurement effort known a standard nonlinear minimization descent algorithm is then best suited to obtaining additional accuracy. This is because discrete grid approximations of a continuous space, such as that used by sequential algorithms, are ultimately limited in accuracy because of computational considerations. The number of independent variables that a nonlinear minimization algorithm must optimize may in fact be small because of symmetry in the design.

Two examples are used below to illustrate aspects of the use-accelerated sequential design algorithm. The first is a two-dimensional example where a computational comparison is made between the accelerated sequential design algorithm and the standard one. The second example demonstrates the ability of the accelerated algorithm to determine a design's structure in the context of a three-dimensional design.

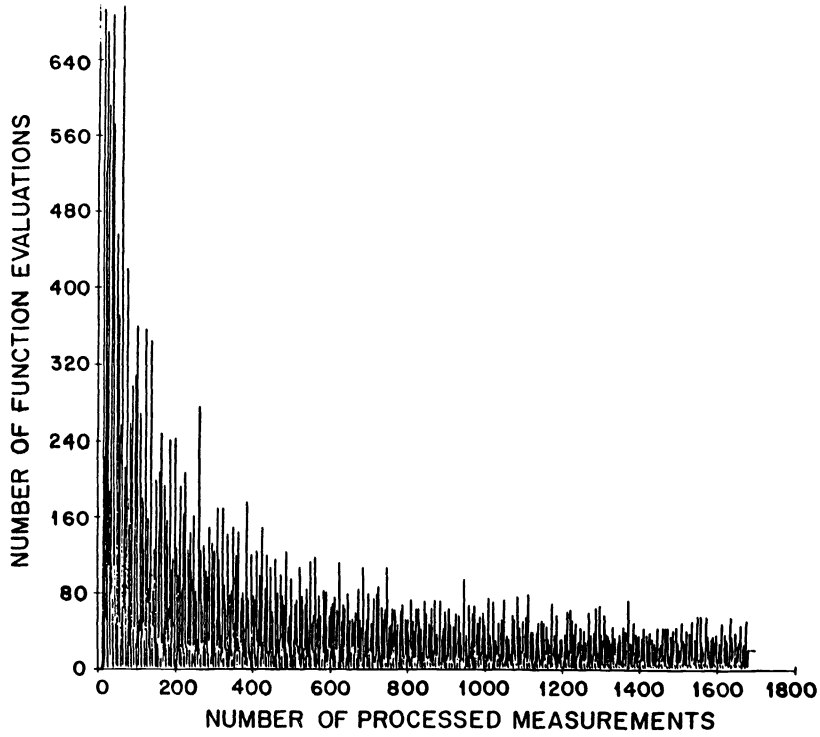


FIG. 1. *Number of function evaluations.*

5.2. Computation. Consider two-dimensional basis functions defined on the unit square ($x \in [-1, 1]$; $y \in [-1, 1]$). They are:

$$1, x, y, xy, x^2, y^2. \tag{5.1}$$

It is well known (Mahoney (1970)) that an optimal design for this set of functions consists of the equal allocation of measurements to the corners, to the midpoints of the boundaries, and to the center of the rectangular region (Fig. 2).

For the simulations summarized in Tables 1 and 2, the number of algorithm iterations was set equal to 1,681. Equi-spaced grids of size 31×31, 41×41, and 49×49

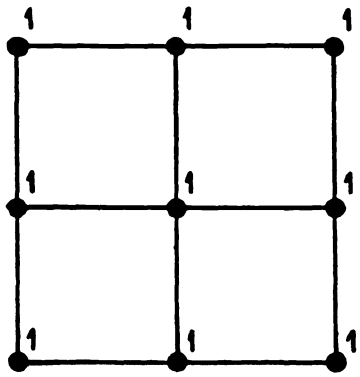


FIG. 2. *D-optimal allocation of measurements.*

TABLE 1
Number of incremental improvements as a function of the number of gridpoints.

$n(i_{\text{total}} = 1{,}681)$	961	1,681	2,401
Standard	1,615,441	2,825,761	4,036,081
Accelerated	32,281	51,261	69,965
Linear increase in n		1.75	1.43
Actual increase for acceleration		1.59	1.36

TABLE 2
Execution times as a function of the number of gridpoints.

$n(i_{\text{total}} = 1{,}681)$	961	1,681	2,401
Standard	79	131	189
Accelerated	14	21	29
Linear increase in n		1.75	1.43
Actual increase for acceleration		1.5	1.4

were used. For the remainder of this section, the number of iterations performed by any algorithm will be the listed value, less the number of locations in the initial design. The number of function evaluations for both algorithms is listed in Table 1.

For the accelerated sequential design algorithm, the growth in the number of function evaluations that occurs as M is increased can indeed be seen to be approximately linear. For the standard sequential design algorithm, the growth is exactly linear, though the number of evaluations is much greater.

The CPU time in seconds that both algorithms required is listed in Table 2. To some extent this is machine (IBM 370), language (Fortran) and program dependent. The times in Table 2 are nonetheless informative.

The increase in time required by the accelerated greedy algorithm is consistent with the increase in the number of function evaluations in the previous table. Even though one accelerated sequential design algorithm evaluation is more expensive than one standard algorithm evaluation, the accelerated algorithm required significantly less time. Figure 1 is a plot of the number of function evaluations per iteration for the 41×41 grid. Note that the number of function evaluations does indeed significantly decrease as the iterations progress.

As the total number of iterations increases, the standard design algorithm requires a proportional increase in the number of function evaluations. This is not true for the accelerated algorithm, as $c_{bf}(i)$ tends to decrease as i becomes large. This is illustrated for the previous basis functions in Table 3. The rate of growth in the number of function

TABLE 3
Number of incremental improvements as a function of the number of iterations.

$i_{\text{total}}(n = 2{,}401)$	1,681	2,401
Standard	4,036,081	5,764,801
Accelerated	69,965	80,399
Linear increase in n		1.43
Actual increase for acceleration		1.15

evaluations, as a function of the number of iterations, for the accelerated sequential design algorithm is less than linear.

The initial design in all the previous simulations consisted of a D-optimal allocation of nine measurements. It may seem odd to initialize a design algorithm with an optimal design but the rate of change in the computation required as a function of n would not be any different for any other choice of initial design. Moreover, other choices of initial designs required approximately the same number of updatings when the number of measurements in the initial design was about the same (Robertazzi (1981c)).

The final unnormalized design produced by the accelerated sequential design algorithm in the above example is illustrated in Fig. 3. Scaling can be used to produce a normalized design. Using the Kiefer bound, this achieves at least 98 percent of the efficiency of a D-optimal design.

5.3. Regression in three dimensions. Consider a set of three-dimensional basis functions defined on the cube ($x \in [-1, 1]$, $y \in [-1, 1]$, $z \in [-1, 1]$). The basis functions are as follows:

$$1, x, y, z, xy, yz, xz, x^2, y^2, z^2, \tag{5.2}$$
$$x^2y, x^2z, xy^2, y^2z, xz^2, yz^2, xyz, x^3, y^3, z^3.$$

A $21 \times 21 \times 21$ grid of measurement locations was used. Thus no a priori information concerning the structure of the design was assumed. The initial design was nonoptimal. The algorithm was run for 6,000 iterations in batches of 1,500 iterations. Between batches measurement locations with less than three measurements were eliminated. The result is illustrated in Fig. 4. The design consists of four types of measurement locations (see Fig. 5). Measurements are symmetrically located at the corners of the design space, at two locations on each edge, in a square pattern on each interior face and in an interior cube pattern:

Exterior Corners:	$(\pm 1, \pm 1, \pm 1)$;
Edges:	$(\pm 0.30, \pm 1.0, \pm 1.0)$, $(\pm 1.0, \pm 0.30, \pm 1.0)$, $(\pm 1.0, \pm 1.0, \pm 0.30)$;
Faces:	$(\pm 1.0, \pm 0.40, \pm 0.40)$, $(\pm 0.40, \pm 1.0, \pm 0.40)$, $(\pm 0.40, \pm 0.40, \pm 1.0)$;
Interior Cube:	$(\pm 0.50, \pm 0.50, \pm 0.50)$.

In the resulting design the number of measurements allocated to a location of the same type tends to be similar. We may conclude that the optimal design includes equal

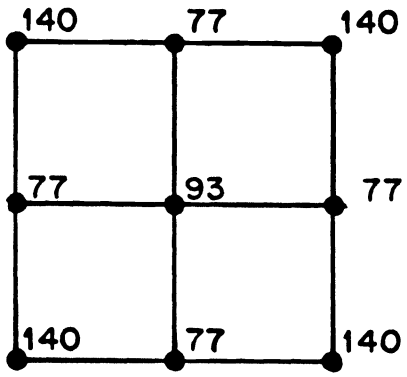


FIG. 3. *The final unnormalized design.*

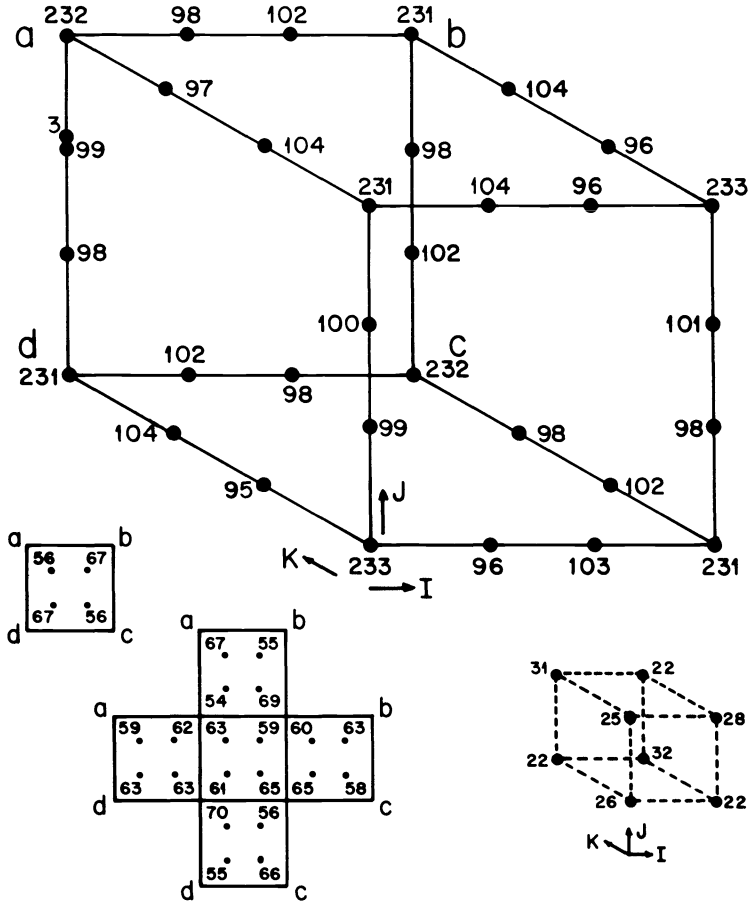


FIG. 4. *A three-dimensional design.*

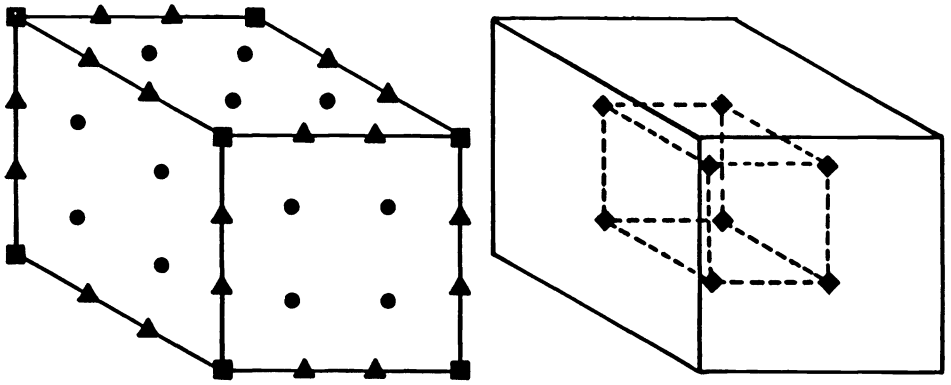


FIG. 5. *Measurement location types for three-dimensional design.*

amounts of design measure applied to the same type of location. This knowledge of the general structure of the design reduces the problem to the optimization of seven parameters. Three are for measurement location and four are for the amount of measurement allocation. This can be further reduced to six parameters since the total amount of design measure must sum to one. A standard function-minimization routine (IMSL ZXMIN) was used to optimize over these parameters. The result was:

Exterior Corner Weight:	0.039;
Edge Weight:	0.017;
Face Weight:	0.010;
Interior Cube Weight:	0.0041;
Edge Offset:	±0.30;
Face Offset:	±0.42;
Interior Cube Offset:	±0.50.

These figures are similar to those that can be inferred directly from the results of the accelerated sequential design algorithm.

In Fig. 6 an approximate Kiefer efficiency bound is plotted versus the number of iterations. The bound is approximate, as it is based on the predicted measurement variances evaluated *only* at gridpoint locations. The actual bound (evaluated over the entire design space) would be somewhat higher. The approximation is roughly monotonically increasing although there is a drop at 1,500 iterations. This is because of the elimination of locations with fewer than three measurements every 1,500 iterations. This approximate bound registers 90 percent after 2,000 iterations and 96.4 percent after 6,000 iterations.

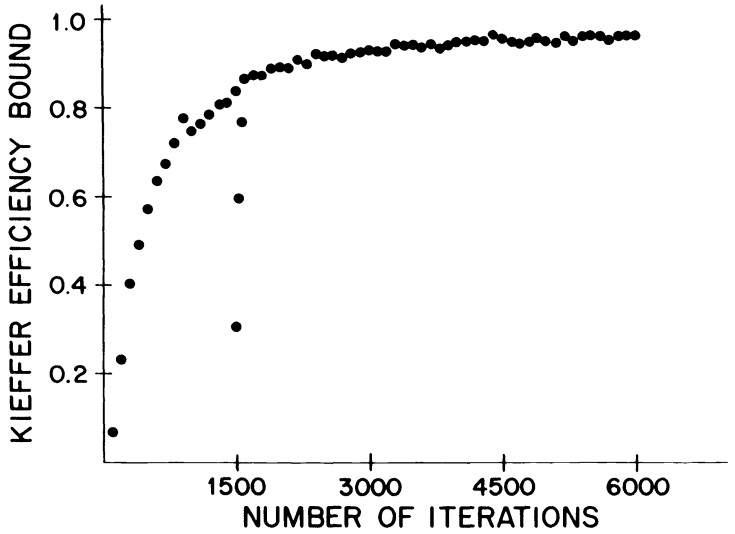


FIG. 6. *Approximate Kiefer efficiency bound.*

What is significant about this example is that the existence of design locations interior to the design space necessitates a grid search. This is efficiently performed by the accelerated sequential design algorithm. It required 4×10^5 iterations versus the 5.6×10^7 that the standard algorithm would require. We note that an alternative design appears in Brodsky et al. (1982).

6. Extension to dynamic models. The major results of D-optimal design theory for static models can be extended to dynamic models in a single independent variable and without process noise (Mehra (1976)):

$$\beta_j = T_{k,j}\beta_k, \tag{6.1}$$

$$y_j = f(x_j)\beta_j + \varepsilon_j. \tag{6.2}$$

The standard sequential design algorithm is also applicable to dynamic models. Suppose that at the ith iteration $V(\hat{\beta})_{k,i}$ is known. This is the error covariance matrix at the kth location during the ith iteration. Also let $T_{k,j}$ be the parameter (state) transition matrix from location k to j. Then the standard algorithm finds, by exhaustive evaluation, the location j at which

$$f(x_j)T_{k,j}V(\hat{\beta})_{k,i}T_{k,j}^T f(x_j)^T \tag{6.3}$$

is maximized.

Note that a new computational task is necessary for dynamic systems. This is the evaluation of

$$T_{k,j}V(\hat{\beta})_{k,i}T_{k,j}^T \tag{6.4}$$

for each j.

There are two possibilities as to the nature of this computation. Either the cost of computing $T_{k,j}$ is independent of $|j-k|$ or it is proportional to $|j-k|$. In the latter case the accelerated design algorithm may not be competitive with the standard sequential design algorithm. The standard algorithm can minimize the cost of computing transition matrices by incrementing and decrementing j one at a time. It is only if $T_{k,j}$ is independent of $|j-k|$ that the accelerated sequential design algorithm might operate efficiently.

The presence of submodularity in the dynamic (no process noise) design problem can be demonstrated in a straightforward manner. The proof is given in Appendix A.

7. Conclusion. A new design algorithm whose statistical foundations are identical to Wynn's procedure but is computationally more efficient has been presented. In showing that the design problem is submodular this paper also reinforces a broad trend (i.e., Welch (1982), Mori and DiStefano, (1979)) of attempting to deal with the design problem from the viewpoint of combinatorial optimization. Computationally, the accelerated sequential design algorithm represents a significant improvement over the standard sequential design algorithm. However a comparison with Mitchell's DETMAX is not possible, as the two algorithms are best suited for somewhat different types of problems. Therefore an appreciation of these differences may be more important than a direct comparison.

Appendix. If a measurement is made at location j, it is necessary to know the incremental improvement in the determinant of the error covariance matrix at k.

The error covariance matrix, at the ith iteration, associated with $\hat{\beta}_j$ in terms of that associated with $\hat{\beta}_k$ is

$$V(\hat{\beta})_{j,i} = T_{k,j}V(\hat{\beta})_{k,i}T_{k,j}^T. \tag{A1}$$

If a measurement is made at j, the updated value of the error covariance matrix at k is

$$V(\hat{\beta})_{k,i+1} = V(\hat{\beta})_{k,i} - \frac{V(\hat{\beta})_{k,i}T_{k,j}^T f(x_j)^T f(x_j)T_{k,j}V(\hat{\beta})_{k,i}}{f(x_j)T_{k,j}V(\hat{\beta})_{k,i}T_{k,j}^T f(x_j)^T + \sigma_j^2}. \tag{A2}$$

Proceeding as in (4.1)-(4.3), we have

$$\text{(A3)} \qquad |V(\hat{\beta})_{k,i+1}| - |V(\hat{\beta})_{k,i}| = \frac{-f(x_j) T_{k,j} V(\hat{\beta})_{k,i} T_{k,j}^T f(x_j)^T}{f(x_j) T_{k,j} V(\hat{\beta})_{k,i} T_{k,j}^T f(x_j)^T + \sigma_j^2} |V(\hat{\beta})_{k,i}|.$$

Again, the incremental difference is nonincreasing in the number of measurements already processed (see (A4) below).

To apply the accelerated greedy algorithm directly to the predicted measurement variance it must be shown that the predicted measurement variance associated with any location k should decrease as a result of making an additional measurement at any other location j. From (A2), the predicted measurement variance at k is

$$\text{(A4)} \qquad \begin{aligned} & f(x_k) V(\hat{\beta})_{k,i+1} f(x_k)^T \\ & \quad = f(x_k) V(\hat{\beta})_{k,i} f(x_k)^T - \frac{(f(x_k) V(\hat{\beta})_{k,i} T_{k,j}^T f(x_j)^T)^2}{f(x_j) T_{k,j} V(\hat{\beta})_{k,i} T_{k,j}^T f(x_j)^T + \sigma_j^2}. \end{aligned}$$

A positive-definite quantity is subtracted from the old predicted measurement variance to form the new value. Thus the predicted measurement variance does indeed decrease as new measurements are added.

REFERENCES

A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN (1976), *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA.

S. BIESEL-GUITONNEAU (1980), *Algorithmes gloutons et fonctions surmodulaires: théorie et application à un problème de securité dans les réseaux de telecommunications*, Ann. Telecomm., 35, pp. 263-271.

V. Z. BRODSKY, L. J. BRODSKY, T. J. GOLIKOVA, AND YE. P. NIKITINA, *Tablitsy Planov Eksperimenta (d ly a Faktomykh i Polinomialnykh Modelei), Tables of Experimental Designs for Factorial and Polynomial Models*, Metallurgiya, Moscow, USSR, 1982.

J. EDMONDS (1971), *Matroids and the greedy algorithm*, Math. Programming, 1, pp. 127-136.

V. V. FEDEROV (1972), *Theory of Optimal Experiments*, Academic Press, New York.

M. L. FISHER, G. L. NEMHAUSER, AND L. A. WOLSEY (1978), *An Analysis of Approximations for Maximizing Submodular Set Functions* II, Math. Prog. Study 8: Polyhedral Combinatorics, North-Holland, pp. 73-87.

Z. GALIL AND J. KIEFER (1980), *Time and space saving methods, related to Mitchell's* DETMAX, *for finding D-optimum designs*, Technometrics, 22, pp. 301-313.

S. KARLIN AND W. J. STUDDEN (1966), *Optimal experimental designs*, Ann. Math. Statist., 37, pp. 783-815.

J. KIEFER AND J. WOLFOWITZ (1959), *Optimum designs in regression problems*, Ann. Math. Statist., 30, pp. 271-294.

J. B. KRUSKAL, JR. (1956), *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proc. Amer. Math. Soc., 17, pp. 48-50.

S. LIN (1965), *Computer solutions to the traveling salesman problem*, Bell System Tech. J., 44, pp. 2245-2269.

F. J. MAHONEY, III (1970), *The design of experiments for regression analysis by computer search procedures*, Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, New York.

R. K. MEHRA (1976), *Optimization of measurement schedules and sensor design for linear dynamic systems*, IEEE Automat. Control, 21, pp. 55-64.

M. MINOUX (1977), *Accelerated greedy algorithms for maximizing submodular set functions*, in Proc. 8th IFIP Conference, Wurzburg, Part 2, Springer-Verlag, New York, pp. 234-243.

M. MINOUX AND M. GONDRAN (1981), *Minimisation de fonctions d'ensemble surmodulaires et sous-modulaires*, in Proc. 9th Triennial Conference on Operations Research (IFORS) Hamburg, F.R.G., J. P. Brans, ed., North-Holland, Amsterdam, pp. 509-521.

T. J. MITCHELL (1974), *An algorithm for the construction of D-optimal experimental designs*, Technometrics, 16, pp. 203-210.

F. MORI AND J. J. DISTEFANO III (1979), *Optimal nonuniform sampling interval and test-input design for identification of physiological systems from very limited data*, IEEE Automat. Control, 24, pp. 893-899.

C. H. PAPADIMITRIOU AND K. STEIGLITZ (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

T. G. ROBERTAZZI AND S. C. SCHWARTZ (1982), *Optimizing the measurement processing order for linear regression with multiple independent variables*, in Proc. 20th Annual Allerton Conference on Communication, Control and Computing, University of Illinois, Urbana-Champaign, pp. 535–544; J. Statist. Comput. Simulation, to appear.

——— (1981a), *Optimizing the measurement processing order for linear regression*, 15th Conference on Information Science and Systems, Baltimore, MD, pp. 507–511.

——— (1981b), *An accelerated experimental design algorithm*, in Proc. 10th IFIP Conference on System Modeling and Optimization, Springer-Verlag, New York, pp. 267–275.

T. G. ROBERTAZZI (1981c), *Measurement processing order and recursive statistical estimation*, Ph.D. thesis, Princeton University, Princeton, NJ.

R. C. ST. JOHN AND N. R. DRAPER (1975), *D-optimality for regression designs: a review*, Technometrics, 17, pp. 15–23.

D. M. TITTERINGTON (1980), *Aspects of optimal design in dynamic systems*, Technometrics, 22, pp. 287–299.

J. Y. TSAY (1976), *On the sequential construction of D-optimal designs*, J. Amer. Statist. Assoc., 71, pp. 671–674.

W. J. WELCH (1982), *Branch and bound search for experimental designs based on D-optimality and other criteria*, Technometrics, 24, pp. 41–48.

J. W. J. WILLIAMS (1964), *Algorithm* 232: *Heapsort*, Commun. ACM, 6, pp. 347–348.

H. P. WYNN (1970), *The sequential generation of D-optimal experimental designs*, Ann. Math. Statist., 41, pp. 1655–1664.

M. B. ZARROP (1979), *Optimal Experiment Design for Dynamic System Identification*, Springer-Verlag, Berlin.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 359-371, March 1989
© 1989 Society for Industrial and Applied Mathematics
008

A ONE-SIDED JACOBI ALGORITHM FOR COMPUTING THE SINGULAR VALUE DECOMPOSITION ON A VECTOR COMPUTER*

P. P. M. DE RIJK†

Abstract. An old algorithm for computing the singular value decomposition, which was first mentioned by Hestenes [*SIAM J. Appl. Math.*, 6 (1958), pp. 51-90], has gained renewed interest because of its properties of parallelism and vectorizability. Some computational modifications are given and a comparison with the well-known Golub-Reinsch algorithm is made. Comparative experiments on a CYBER 205 are reported.

Key words. singular value decomposition, one-sided Jacobi algorithm, vector computing

AMS(MOS) subject classifications. 65F15, 65F20, 65F25, 15A18

1. Introduction. Let A be a real $m \times n$ matrix and $p = \min(m, n)$. Then there exists an $m \times m$ orthogonal matrix U, an $n \times n$ orthogonal matrix V, and an $m \times n$ diagonal matrix Σ with diagonal elements σ_i, $i = 1, 2, \cdots, p$, such that

$$A = U\Sigma V^T \tag{1.1}$$

and

$$\sigma_1 \geqq \sigma_2 \geqq \cdots \geqq \sigma_p \geqq 0. \tag{1.2}$$

For a proof see Golub and Van Loan (1983, p. 17).

The decomposition (1.1) is called the singular value decomposition (SVD) of A. The scalars σ_i are the singular values of A, the columns of U are the left singular vectors of A and the columns of V are the right singular vectors of A. From (1.1) it is evident that

$$A = U_p \Sigma_p V_p^T, \tag{1.3}$$

where U_p is the $m \times p$ matrix formed by the first p columns of U, V_p is the $n \times p$ matrix formed by the first p columns of V, and $\Sigma_p = \operatorname{diag}(\sigma_1, \sigma_2, \cdots, \sigma_p)$ is the square pth order upper left part of Σ. Note that since $p = \min(m, n)$, $V_p = V$ if $m \geqq n$ and $U_p = U$ otherwise. If $r = \operatorname{rank}(A) < p$, then $\sigma_{r+1} = \cdots = \sigma_p = 0$ and consequently (1.3) can be reduced to

$$A = U_r \Sigma_r V_r^T, \tag{1.4}$$

with obvious definitions for U_r, V_r, and Σ_r. In the literature, the factorizations (1.3) and (1.4) as well are referred to as the SVD of A. Since the decomposition (1.3) is most frequently used, we shall call this one the *standard* SVD, whereas (1.1) and (1.4) are referred to as the *extended* and *reduced* SVD, respectively.

In this paper, we discuss a one-sided orthogonalization method for computing the SVD, which was first mentioned by Hestenes (1958). The method is mathematically equivalent to the iterative Jacobi algorithm for solving the eigenproblem of A^TA. However, instead of repeatedly applying orthogonal similarity transformations as is done in traditional Jacobi methods, Hestenes' method deals only with postmultiplications by orthogonal matrices. Though it was superseded by the Golub-Reinsch

* Received by the editors December 8, 1986; accepted for publication (in revised form) July 22, 1988.
† Faculty of Mathematics and Computer Science, Department of Computer Systems, University of Amsterdam, Kruislaan 409, 1098 SJ Amsterdam, the Netherlands.

algorithm (see Golub and Reinsch (1970)), there are some specific situations in which this method could be preferred:

(1) Chartres (1962) has suggested an adaption of the standard (cyclic) Jacobi method for a computer with small main memory but with magnetic backing store. Then he briefly discusses how his formulation of the Jacobi method can be generalized to calculate the SVD of an arbitrary real matrix. Nash (1975) used Chartres' method in an implementation for minicomputers.

(2) Luk (1980) preferred Hestenes' method to the Golub–Reinsch SVD algorithm for an implementation on the ILLIAC IV, because then he could make better use of the special architecture of this machine.

(3) There is now a renewed interest in the method for the following two reasons:
   (a) It is extremely suitable for parallel computation (e.g., see Brent and Luk (1985)).
   (b) As we shall see in this paper, it is extremely suitable for implementation on machines with vector processors, such as the CYBER 205 and the CRAY-1. Considering both parallelism and vectorizability, it becomes particularly attractive on machines that have a number of independently working vector processors in one system (CRAY X-MP, ETA-10).

We will show that the one-sided orthogonalization transformations in Hestenes' method can easily be restructured in such a way that they can take full advantage of possible vector capabilities of the computer. By extending this restructuring, we will eliminate the need for copying operations. Moreover, choosing a suitable way for storing the singular vectors strongly reduces the startup times for vector instructions. In addition, we enhance the iteration sequence as used by Hestenes, Nash, and Luk by a new pivoting strategy similar to the usual column pivoting in other algorithms (e.g., QR-factorization) to speed up the convergence. Finally, we discuss the use of thresholds to eliminate unnecessary transformations.

The paper is outlined as follows. In § 2 the algorithm is described in general for both serial and parallel computation. In addition, some different iteration schemes are discussed. In § 3 computational modifications are given and the aspect of vectorizability is discussed. The results of some numerical experiments are presented in § 4.

Throughout this paper, it is assumed that $m \geqq n$. This is no restriction, since the case $n > m$ can simply be derived from A^T: if $A = U\Sigma V^T$, then $A^T = V\Sigma^T U^T$.

2. A one-sided Jacobi algorithm. In this section we describe the basic idea of the algorithm and discuss some different iterative schemes.

2.1. Outline of the algorithm. Given a real $m \times n$ matrix A $(m \geqq n)$, we construct a sequence of matrices $\{X_k\}$ by the recurrence relation

$$\begin{aligned} &X_0 = A, \\ &X_k = X_{k-1} R_k, \qquad k = 1, 2, 3, \cdots. \end{aligned} \tag{2.1.1}$$

Here the $R_k = (r_{ij}^{(k)})$ are $n \times n$ Jacobi plane rotations, such that two columns in X_k become orthogonal. More specifically, to every k there corresponds a pair (p, q) of two indices p and q satisfying $1 \leqq p < q \leqq n$ and an angle θ such that R_k differs from the identity matrix only in the following four entries:

$$\begin{aligned} r_{pp}^{(k)} &= \cos\theta, \qquad r_{pq}^{(k)} = -\sin\theta, \\ r_{qp}^{(k)} &= \sin\theta, \qquad r_{qq}^{(k)} = \cos\theta. \end{aligned} \tag{2.1.2}$$

Here the angle θ is determined such that the pth and qth columns in X_k are orthogonal. In other words, R_k zeros the off-diagonal entries (p, q) and (q, p) in the symmetric

matrix $X_k^T X_k$. So, the transition from X_{k-1} to X_k implicitly applies a minor step in some Jacobi algorithm on A^TA. By choosing the indices p and q and the angle θ appropriately in each step, the sequence $\{X_k\}$ converges to a real $m \times n$ matrix X, which has the property that X^TX is a diagonal matrix with the eigenvalues of A^TA on its main diagonal. Hence, if the matrix V is the product of all R_k, then V is orthogonal and

$$AV = X, \tag{2.1.3}$$

where the columns of X are mutually orthogonal.

From (2.1.3) the SVD of A can be obtained in the following way. Let $X = [x_1, x_2, \cdots, x_n]$ and $V = [v_1, v_2, \cdots, v_n]$ be a column partitioning of X and V, respectively, and let us assume that the columns of X are (re-)ordered so that

$$\|x_i\|_2 \geqq \|x_j\|_2 \tag{2.1.4}$$

for all i and j with $1 \leqq i < j \leqq n$. Define

$$\sigma_j = \|x_j\|_2, \qquad j = 1, 2, \cdots, r,$$

$$u_j = x_j / \sigma_j, \qquad j = 1, 2, \cdots, r,$$

where r is the rank of A. Then the reduced SVD (1.4) is given by the matrices $U_r = [u_1, u_2, \cdots, u_r]$, $V_r = [v_1, v_2, \cdots, v_r]$, and $\Sigma_r = \operatorname{diag}(\sigma_1, \sigma_2, \cdots, \sigma_r)$.

2.2. The Jacobi rotations. The rotations R_k introduced in (2.1.1) can easily be determined. Let $X_k = [x_1^{(k)}, \cdots, x_n^{(k)}]$, $c = \cos\theta$ and $s = \sin\theta$; then, since postmultiplication by R_k affects only two columns, we have

$$[x_p^{(k)}, x_q^{(k)}] = [x_p^{(k-1)}, x_q^{(k-1)}] \begin{bmatrix} c & -s \\ s & c \end{bmatrix}, \tag{2.2.1}$$

so

$$x_p^{(k)} = c x_p^{(k-1)} + s x_q^{(k-1)}, \qquad x_q^{(k)} = -s x_p^{(k-1)} + c x_q^{(k-1)}, \tag{2.2.2}$$

whereas for all other columns

$$x_i^{(k)} = x_i^{(k-1)}. \tag{2.2.3}$$

For convenience, define $\omega_p^{(k-1)} = \|x_p^{(k-1)}\|_2^2$, $\omega_q^{(k-1)} = \|x_q^{(k-1)}\|_2^2$, and $\xi = (x_p^{(k-1)})^T (x_q^{(k-1)})$. The requirement that the rotation angle θ is to be chosen so that the pth and qth columns in X_k are orthogonal, leads to (cf. Rutishauser (1966))

$$\tan 2\theta = \frac{2\xi}{\omega_p^{(k-1)} - \omega_q^{(k-1)}}. \tag{2.2.4}$$

Using some simple trigonometry, we find that $t = \tan\theta$ satisfies

$$t^2 \tan 2\theta + 2t - \tan 2\theta = 0. \tag{2.2.5}$$

Choosing the root of smaller magnitude yields

$$t = \tan 2\theta / (1 + \sqrt{1 + \tan^2 2\theta}). \tag{2.2.6}$$

If we define $Q = \omega_p^{(k-1)} - \omega_q^{(k-1)}$, then substituting (2.2.4) in (2.2.6) gives

$$t = 2\xi / (Q + \operatorname{sgn}(Q)\sqrt{Q^2 + 4\xi^2}), \tag{2.2.7}$$

where the function sgn is defined as

$$\text{sgn}(x) = \textbf{if } x < 0 \textbf{ then } -1 \textbf{ else } 1 \textbf{ fi}.$$

In the degenerated case where both $\xi = 0$ and $Q = 0$, t can be set to zero. Note that this choice of t ensures that

$$|\theta| \leqq \pi/4, \tag{2.2.8}$$

and so from t we can calculate

$$c = 1/\sqrt{1+t^2}, \qquad s = c \cdot t. \tag{2.2.9}$$

In some future step of the algorithm, the columns $x_p^{(k)}$ and $x_q^{(k)}$ must be set orthogonal to some other columns in the updated matrix X_k. Then, as we see from (2.2.7), the squares of the Euclidean norms of these columns will be needed. However, they do not have to be calculated in each orthogonalization step, since they can be updated from the norms of $x_p^{(k-1)}$ and $x_q^{(k-1)}$, i.e., from $\omega_p^{(k-1)}$ and $\omega_q^{(k-1)}$. With $\omega_p^{(k)} = \|x_p^{(k)}\|_2^2$, $\omega_q^{(k)} = \|x_q^{(k)}\|_2^2$ and using the fact that they correspond to diagonal elements in $X_k^T X_k$, we know from Rutishauser (1966) that

$$\omega_p^{(k)} = \omega_p^{(k-1)} + t\xi, \qquad \omega_q^{(k)} = \omega_q^{(k-1)} - t\xi. \tag{2.2.10}$$

Note, that the choice of t given by (2.2.7) keeps the ordering of $\omega_p^{(k)}$ and $\omega_q^{(k)}$ the same as that of $\omega_p^{(k-1)}$ and $\omega_q^{(k-1)}$, i.e., $\omega_p^{(k)} \geqq \omega_q^{(k)}$ if and only if $\omega_p^{(k-1)} \geqq \omega_q^{(k-1)}$.

2.3. Strategy and convergence. To have a complete algorithm, we need a strategy for determining the rotation index pair (p, q) in each step. Since the method for calculating the SVD of a matrix A as described in §§ 2.1 and 2.2 is essentially the same as a Jacobi method solving the eigenproblem for A^TA, it seems obvious to choose a strategy from the many available Jacobi methods.

A well-known class of Jacobi methods is the so-called family of (general) cyclic Jacobi methods, where in every segment of $N = n(n-1)/2$ consecutive iterative steps, each rotation pair (p, q) satisfying $1 \leqq p < q \leqq n$ occurs exactly once (but, in principle, any order is allowed). Each set of N rotations is often referred to as a sweep. The easiest to use in this class is the one in which a sweep consists of the following ordering of index pairs:

$$(1,2), (1,3), \cdots, (1,n), (2,3), \cdots, (2,n), (3,4), \cdots, (n-1,n). \tag{2.3.1}$$

This scheme is known under the name of the special cyclic Jacobi method. Forsythe and Henrici (1960) proved that using this cyclic order, the sequence $\{X_k^T X_k\}$ defined by (2.1.1), subject to (2.2.8), always converges to a fixed diagonal matrix. Though convergence can no longer be guaranteed in case of an arbitrary cyclic order, it is generally assumed that a cyclic Jacobi method does converge as long as each sweep consists of the same ordering of rotation index pairs and the rotation angle θ is suitably restricted. Moreover, it has been shown that if a cyclic Jacobi method does converge and condition (2.2.8) is always satisfied, the method ultimately converges quadratically (see Henrici (1958), Wilkinson (1962), and van Kempen (1966)).

The SVD algorithm can be seen as finding an orthogonal basis for the column space of the original matrix. From the convergence theory of Jacobi methods, we know that the orthogonalization of a column pair $(x_p^{(k-1)}, x_q^{(k-1)})$

(1) Has little or no effect when their inner product is small compared to the average level of inner products of all pairs;
(2) Might completely destroy the orthogonality of some other pair that was accomplished (with much effort) during an earlier step in the iteration process.

Thus, it might be more beneficial to skip the orthogonalization of a nearby orthogonal pair as long as there are many other column pairs, which are far from orthogonality. This can be accomplished by omitting the orthogonalization of pairs whose inner products are less than some sweep-dependent threshold. In doing so, we obtain a threshold Jacobi method (cf. Wilkinson (1965, pp. 277–278)), from which it is not only known that convergence is guaranteed (whatever cyclic order is used), but also that the rate of convergence is accelerated.

Observe that each inner product $(x_p^{(k-1)})^T(x_q^{(k-1)})$ can be written as

$$(x_p^{(k-1)})^T(x_q^{(k-1)}) = \gamma_{pq}^{(k-1)}\|x_p^{(k-1)}\|_2\|x_q^{(k-1)}\|_2, \tag{2.3.2}$$

where $\gamma_{pq}^{(k-1)}$ is the cosine of the angle between the vectors $x_p^{(k-1)}$ and $x_q^{(k-1)}$ and so $\gamma_{pq}^{(k-1)}$ satisfies $0 \leqq |\gamma_{pq}^{(k-1)}| \leqq 1$. In fact, $|\gamma_{pq}^{(k-1)}|$ is a relative measure for the orthogonality of $x_p^{(k-1)}$ and $x_q^{(k-1)}$. Thus, we suggest skipping the orthogonalization of the pair (p, q) whenever $|x_{pq}^{(k-1)}|$ is less than some (sweep-dependent) threshold. In addition, (2.3.2) indicates that the convergence rate could be accelerated (even more) when the columns with larger norms are treated first during the orthogonalization process. This can be accomplished by either imposing a similar kind of threshold on $\|x_i^{(k-1)}\|_2$ or by rearranging the computational order (see the next section).

2.4. A pivoting strategy. In the original algorithm presented by Hestenes (1958), the special cyclic order is used for selecting the rotation index pairs. In addition, he suggests that during the orthogonalization process the columns can be rearranged such that finally their (Euclidean) norms appear in nonincreasing order (cf. condition (2.1.4)). This is achieved in the following way. Every time before the columns $x_p^{(k-1)}$ and $x_q^{(k-1)}$ are orthogonalized, first check whether $\|x_p^{(k-1)}\| < \|x_q^{(k-1)}\|$ and if so, interchange them. In the implementations of Nash (1975) and Luk (1980) the need for a possible explicit interchange has been eliminated by incorporating an implicit interchange in the rotation. That is, when two columns are orthogonalized by a Jacobi rotation, the rotation angle θ is chosen so that $\|x_p^{(k)}\| \geqq \|x_q^{(k)}\|$. By allowing θ to be in the interval $[-\frac{1}{2}\pi, \frac{1}{2}\pi]$, this extra condition can always be satisfied.

Naturally, it is nice when sorting and orthogonalization can be done simultaneously, as long as it does not effect the convergence rate. Unfortunately, the (explicit or implicit) interchange of two columns might result in a situation where certain pairs of columns are not treated for some sweeps. To illustrate this, consider the matrix

$$X_0 = [a, b, c] = \begin{bmatrix} 2 & 0 & 1 \\ 0 & 3 & 0 \\ 1 & 0 & 4 \end{bmatrix}.$$

When we apply Hestenes' method to this matrix (the variants of Nash and Luk are similar), the first sweep unfolds as follows:

(1) In the first step the pair (1, 2), i.e., the columns a and b, are treated. Since $\|a\| < \|b\|$ and $a^Tb = 0$, a and b are interchanged and no rotation is performed. Thus, the resulting matrix X_1 is $[b, a, c]$.

(2) In the second step the pair (1, 3), i.e., the columns b and c, are treated. Since $\|b\| < \|c\|$ and $b^Tc = 0$, b and c are interchanged and no rotation is performed. Thus, the resulting matrix X_2 is $[c, a, b]$.

(3) In the third step the pair (2, 3), i.e., the columns a and b (again!), are treated. This step is equivalent to the first one and so the resulting matrix X_3 is $[c, b, a]$.

Note that the column pair (a, c) is never handled during this sweep and in fact this first sweep only sorts the original matrix X_0, but in a very expensive way.

To overcome this problem, we suggest the following strategy.

Pivoting Strategy 2.4.1. Split each sweep into $n-1$ consecutive *subsweeps*, where the pth *subsweep* $(1 \leqq p < n)$ looks as follows. First determine the column in $[x_p^{(k-1)}, x_{p+1}^{(k-1)}, \cdots, x_n^{(k-1)}]$ with largest Euclidean norm and interchange this one with $x_p^{(k-1)}$; then orthogonalize the column pairs (p, q) for $q = p+1, \cdots, n$.

Using this strategy, all possible pairs are treated and since a single rotation keeps the ordering of $x_p^{(k-1)}$ and $x_q^{(k-1)}$ the same as that of $x_p^{(k)}$ and $x_q^{(k)}$ (see § 2.2), the same effect of Hestenes' method is achieved. The number of interchanges in a sweep is at most $n-1$ and consequently does not influence the required time for the method, whereas the number of comparisons is the same as in Hestenes' method.

Another advantage of this pivoting strategy is that it tends to treat the columns with larger norms first, which, as we have seen earlier (cf. the last paragraph of § 2.3), could improve the convergence rate.

2.5. Parallelism. Observe that during an orthogonalization step only the elements of two columns of the matrix X are accessed and changed. Similarly, to accumulate the rotations R_k into the matrix V, we need only access two columns of this matrix. In addition, during an orthogonalization step only a small amount of extra data is needed, which is independent of other pairs. This implies that (at most) $\lfloor n/2 \rfloor$ pairs of columns can be treated independently of each other, i.e. in parallel. Brent and Luk (1985) give a cyclic order for the rotation index pairs such that a maximum number of pairs can be treated simultaneously.

3. Modified Jacobi rotations. For convenience we change the notation as follows. We omit the superscripts $k-1$ and replace the superscripts k with the tilde symbol. Then (2.2.2) have the form

$$\tilde{x}_p = cx_p + sx_q, \qquad \tilde{x}_q = -sx_p + cx_q. \tag{3.1}$$

Using these formulae, the calculation of $\tilde{x}_p$ and $\tilde{x}_q$ in scalar mode (i.e., the way it is done on a conventional machine) requires $4m$ multiplications plus $2m$ additions. On a machine where vectors can be handled as elementary quantities, the calculations must be reordered in order to reduce the number of vector operations. For instance, since θ is restricted to (2.2.8), the following scheme can be used:

$$\begin{aligned} y_p &= x_p + tx_q, \qquad y_q = x_q - tx_p, \\ \tilde{x}_p &= cy_p, \qquad \tilde{x}_q = cy_q. \end{aligned} \tag{3.2}$$

Here y_p and y_q are temporary vectors of length m. The latter two equations of (3.2) are of the form *scalar times vector*, whereas the former two are, using CYBER 205 terminology, so-called *linked triads*, i.e., a triadic operation that involves one scalar operand and two vector operands and which can be performed by one vector instruction. On a machine with a CRAY-like architecture, the same effect can be achieved by *chaining*, which means that the output of one vector operation (*scalar times vector*) immediately becomes the input of another (*vector plus vector*). Schemes other than (3.2) are possible, but the necessary number of vector operations is at least four.

From the replacement of Givens rotations by so-called modified or fast Givens rotations (e.g., see Hammarling (1974)), we know that the number of operations can be reduced. This can be achieved by postponing the calculation of $\tilde{x}_p$ and $\tilde{x}_q$ in (3.2). Instead, the new matrix $\tilde{X}$ is held in factored form YD, where the diagonal matrix D must hold the contribution of the multiplier c in positions p and q.

With this remark in mind, we can reformulate each orthogonalization step in the following way. Suppose that the real $m \times n$ matrix $X = [x_1, x_2, \cdots, x_n]$ is given in

factored form

$$X = YD, \tag{3.3}$$

where $Y = [y_1, y_2, \cdots, y_n]$ is a real $m \times n$ matrix and $D = \operatorname{diag}(d_1, d_2, \cdots, d_n)$ is an $n \times n$ diagonal matrix with positive elements. Our task is to compute $\tilde{X} = XR(p, q, \theta)$ in the form

$$\tilde{X} = \tilde{Y}\tilde{D}, \tag{3.4}$$

where $\tilde{D}$ may be chosen at our convenience. Looking at columns p and q, we see that

$$[\tilde{x}_p, \tilde{x}_q] = [x_p, x_q]\begin{bmatrix} c & -s \\ s & c \end{bmatrix} = [y_p, y_q]\begin{bmatrix} cd_p & -sd_p \\ sd_q & cd_q \end{bmatrix} = [y_p, y_q]\begin{bmatrix} 1 & -\alpha \\ \beta & 1 \end{bmatrix}\begin{bmatrix} cd_p & 0 \\ 0 & cd_q \end{bmatrix}, \tag{3.5}$$

where α and β are given by

$$\alpha = t\frac{d_p}{d_q}, \qquad \beta = t\frac{d_q}{d_p}. \tag{3.6}$$

By choosing

$$\tilde{d}_p = cd_p, \qquad \tilde{d}_q = cd_q \tag{3.7}$$

and

$$\tilde{y}_p = y_p + \beta y_q, \tag{3.8a}$$

$$\tilde{y}_q = y_q - \alpha y_p, \tag{3.8b}$$

we have

$$[\tilde{x}_p, \tilde{x}_q] = [\tilde{y}_p, \tilde{y}_q]\begin{bmatrix} \tilde{d}_p & 0 \\ 0 & \tilde{d}_q \end{bmatrix}. \tag{3.9}$$

Until now, this factorization can be applied to any plane rotation and, in fact, our choice for $\tilde{d}_p$ and $\tilde{d}_q$ is most frequently used in the modified Givens rotations. By using the fact that in our case the Jacobi rotation $R = R(p, q, \theta)$ is specifically designed to orthogonalize the vectors x_p and x_q (and consequently the vectors y_p and y_q), the calculations can be done more directly, without the need for calculating t, c, and s. Defining $\rho = y_p^T y_q$, we have from (2.2.7) and (3.6)

$$\alpha = 2\rho d_p^2/(Q + \operatorname{sgn}(Q)\sqrt{Q^2 + 4\rho d_p^2 d_q^2}) \tag{3.10a}$$

and

$$\beta = 2\rho d_q^2/(Q + \operatorname{sgn}(Q)\sqrt{Q^2 + 4\rho d_p^2 d_q^2}). \tag{3.10b}$$

Again using (3.6), we calculate from equations (2.2.10)

$$\tilde{\omega}_p = \omega_p + \beta\rho d_p^2, \qquad \tilde{\omega}_q = \omega_q - \alpha\rho d_q^2. \tag{3.11}$$

Equations (3.10) and (3.11) imply that only the squares of d_p and d_q are needed. This is also convenient for updating these quantities: since $c^2 = 1/(1+t^2) = 1/(1+\alpha\beta)$, (3.7) gives

$$\tilde{d}_p^2 = d_p^2/(1+\alpha\beta), \qquad \tilde{d}_q^2 = d_q^2/(1+\alpha\beta). \tag{3.12}$$

Comparing the modified version with the original rotations, we see that the calculations of $\tilde{y}_p$ and $\tilde{y}_q$ according to (3.8) in scalar mode need $2m$ fewer multiplications than the calculations of $\tilde{x}_p$ and $\tilde{x}_q$ in (3.1). When performing the calculations in vector mode, we need only two linked triads, two fewer vector operations than in scheme (3.2).

This scheme can be made more efficient, especially in the case that one or more vector processors of memory-to-memory type (e.g., the CYBER 205) are used. Since

in practice y_i and $\tilde{y}_i$, $i = 1, 2, \cdots, n$, occupy the same physical memory, either y_p or y_q has to be copied before $\tilde{y}_p$ and $\tilde{y}_q$ can be calculated. Though this copying can be vectorized and will, in general, be cheaper than an arithmetic vector operation, it would be nice to avoid this copying. Fortunately, our choice for $\tilde{d}_p$ and $\tilde{d}_q$ allows for a reformulation of (3.8) in which this is possible. From (3.8a) we know that

$$y_p = \tilde{y}_p - \beta y_q.$$

Substitution in (3.8b) gives

$$\tilde{y}_q = (1 + \alpha\beta)(y_q - \hat{\alpha}\tilde{y}_p),$$

where $\hat{\alpha} = \alpha/(1+\alpha\beta)$. Thus, if we define $\hat{d}_q = (1+\alpha\beta)\tilde{d}_q$ and

$$\hat{y}_q = y_q - \hat{\alpha}\tilde{y}_p, \tag{3.8c}$$

we may replace the factorization (3.9) by

$$[\tilde{x}_p, \tilde{x}_q] = [\tilde{y}_p, \hat{y}_q] \begin{bmatrix} \tilde{d}_p & 0 \\ 0 & \hat{d}_q \end{bmatrix}. \tag{3.13}$$

For the calculation of $\hat{d}_q^2$, we may replace the second equation in (3.12) by

$$\hat{d}_q^2 = (1+\alpha\beta) d_q^2. \tag{3.14}$$

Now $\tilde{y}_p$ and $\hat{y}_q$ in factorization (3.13) can be calculated directly, without first making a copy of y_p or y_q.

Another optimization can be obtained when the right singular vectors (the matrix V) are required. Then the plane rotations must be accumulated; i.e., we calculate

$$V_k = V_{k-1} R_k,$$

with $V_0 = I$. Thus, updating matrix V_{k-1} is analogous to updating matrix X_{k-1}. Consequently, the modifications as described earlier in this section can be applied and both matrices may use the same scaling matrix. In addition, this equivalence may be used to reduce the number of vector operations necessary to perform these updates. By storing matrices X and V such that for each i $(i = 1, \cdots, n)$, columns x_i of X and v_i of V are contiguous in memory, each update of x_i and v_i can be executed by one vector operation. In this way, the startup times necessary to perform the column operations on V are eliminated, meaning that vectorization becomes attractive even for small n, say $n < 50$ (think about application of the SVD in least-squares problems, where n might be small, but $m \gg n$). Note that such a storage scheme could also be beneficial in other circumstances: when large matrices are to be handled in virtual memory (e.g., CYBER 205); when secondary storage is to be used (as swapping between main and secondary memory is made easier); or when pairs of columns are orthogonalized in parallel (the data and/or instruction streams are simplified; e.g. see Brent and Luk (1985)).

Remark. From (3.12) and (3.14) we see that every time an orthogonalization step is performed, the scaling factors d_i^2 are multiplied or divided by the *growth factor* $1+\alpha\beta$. Since the rotation angle θ is restricted to (2.2.8) and $\alpha\beta = t^2$, this growth factor is bounded by

$$1 \leqq 1 + \alpha\beta \leqq 2.$$

In order to avoid under- or overflow, all y_i and d_i must occasionally be rescaled. Note however, that the upper bound for this growth factor is very pessimistic, especially during the later sweeps, where the rotation angle tends to zero. Thus, in practice, rescaling will seldom be necessary and it will certainly not influence the performance of the algorithm.

4. Experiments. Four different versions of the Jacobi-like algorithm for computing the SVD as described in §§ 2 and 3 have been compared with each other in order to investigate the influence of using thresholds and pivoting. In addition, they were compared with routines from LINPACK and NAG, which implement the Golub-Reinsch algorithm. The experiments were carried out on the CYBER 205 computer of the academic computer centre SARA in Amsterdam (a one-pipe vector machine with machine precision $\varepsilon \approx 10^{-14}$). All methods have been implemented in FTN200, an extension of FORTRAN 77 in which the programmer has access to the complete set of machine instructions in the form of a special vector syntax.

4.1. Test matrices. The following sets of test matrices were used:

(1) The $n \times n$ Hilbert matrix segments H_n, defined by $H_n(i,j) = 1/(i+j-1)$, for $i,j = 1, 2, \cdots, n$. For "large" n, these matrices are almost singular and when n is sufficiently large, most singular values are smaller than $\varepsilon \|H_n\|$.

(2) The $n \times n$ matrices $A_n = D_n U_n$ $(n = 2^k)$, where each D_n is an $n \times n$ diagonal matrix and each U_n is orthogonal. Thus, for each n, A_n has orthogonal rows and its singular values are the absolute values of the elements of D_n. The following choices for D_n were made:
   (a) $D_n = \text{diag}(1, 2, \cdots, n)$; thus the singular values of A_n are well separated.
   (b) $D_n = \text{diag}(n/2, (n-1)/2, \cdots, 1, 0, -1, \cdots, -(n-1)/2)$; thus A_n is singular and all singular values, except the largest and smallest, are grouped in pairs.

(3) The $n \times n$ matrices $T_n = D_n R_n$, where the D_n are $n \times n$ diagonal matrices with $D_n(i,i) = s^{i-1}$, for $i = 1, 2, \cdots, n$, and the R_n are upper triangular matrices with all diagonal elements equal to one and all upper off-diagonal elements equal to $-c$, where $c^2 + s^2 = 1$; the choice $c = 0.2$ was made. These matrices are often used for testing the performance of SVD algorithms (e.g., see Golub and Reinsch (1970)).

(4) Rectangular $m \times n$ matrices $(m = 2n)$, whose elements are randomly chosen in the interval $(-1, 1)$.

Remark. The SVD of a rectangular matrix can also be calculated by using Chan's algorithm, in which the original matrix A is transformed into a square matrix. The SVD of A is then easily obtained from the SVD of this square matrix (see Chan (1982)). As long as, apart from vectorization, the calculations are performed sequentially, Chan's algorithm becomes profitable as soon as the difference between m and n becomes significant (approximately when $m \geqq 2n$).

4.2. Influence of thresholds and pivoting. Since the CYBER 205 has, apart from pipelines, no parallel capabilities, we have chosen the rotation index pairs in all considered variants of the Jacobi-like SVD algorithm according to the special cyclic method (2.3.1). In addition, all considered variants use the modified Jacobi rotations "without copying," as described in § 3 (i.e., factorization (3.13)), and they all are highly optimized (vectorized). They differ from each other in the following way:

- In variant CJSVD the iteration process is continued until all columns in matrix X can be considered mutually orthogonal. That is, when for all indices (p, q) satisfying $1 \leqq p < q \leqq n$, one of the following conditions holds:

$$\|x_p\|_2 \leqq \varepsilon \|X\|_2, \qquad \|x_q\|_2 \leqq \varepsilon \|X\|_2, \qquad |x_p^T x_q| \leqq \mu \|x_p\|_2 \|x_q\|_2, \tag{4.2.1}$$

with $\mu = \sqrt{m}\,\varepsilon$. Furthermore, during the process, the orthogonalization of

columns x_p and x_q is skipped whenever one of the conditions (4.2.1) is satisfied; we shall call this a "trivial threshold" strategy.

- In variant CJSVDT this threshold strategy is extended in the following way (cf. § 2.3): the orthogonalization of a column pair is omitted if their normalized inner product (γ_{pq} in (2.3.2)) is less than τ_S, where τ_S is a (fixed) threshold during sweep S as specified below.
- Variant PJSVD is variant CJSVD extended with the Pivoting Strategy 2.4.1.
- Variant PJSVDT is like PJSVD, but the same threshold strategy as in CJSVDT is used.

The thresholds in the variants CJSVDT and PJSVDT are chosen by the recurrence relation

$$\begin{aligned} &\tau_{-1} = 1, \qquad \tau_0 = \tfrac{1}{2}\gamma_{\max}, \\ &\tau_S = \tfrac{1}{2}\tau_{S-1}\tau_{S-2} \quad \text{for } S = 1, 2, 3, 4, \\ &\tau_S = \mu \qquad\qquad\ \ \text{for } S > 4. \end{aligned} \tag{4.2.2}$$

Here $\gamma_{\max}$ is the maximum of normalized inner products of n randomly chosen column pairs from the original matrix (thus, being an estimation of the maximum of all normalized inner products).

The four variants were compared to each other with respect to the following items:

- the convergence speed, i.e., the number of sweeps necessary to orthogonalize the matrix;
- the number of orthogonalizations (rotations) that have been performed;
- the number of calculated inner products during the iteration process.

The results are given in Table 1. The columns labeled S contain the number of sweeps for the specified method. The columns labeled R and P contain the number of performed orthogonalizations and the number of calculated inner products in the specified methods respectively, both divided by a factor 1,000 and rounded afterward. In case of Test 4, the average results for 10 different random matrices of the given size are displayed. The timings of the worst and best performing variant (CJSVD and PJSVDT, respectively) can be found in Table 2.

Comparing CJSVD with CJSVDT and PJSVD with PJSVDT, we see that the use of nontrivial thresholds greatly influences the number of necessary orthogonalizations. In addition, comparing PJSVD with CJSVD and CJSVDT, we see that, looking at the test matrices 1, 2a, 2b and 4, the use of the Pivoting Strategy 2.4.1 can improve the convergence speed; but, as can be seen from test problem 3, it does not always do so. It is the combination of using both nontrivial thresholds and pivoting in PJSVDT that gives the best performing algorithm. In all test cases it requires the fewest numbers of orthogonalizations (for "large" n, sometimes 40–50 percent less than CJSVD) and in nearly all cases it needs the fewest number of sweeps (and consequently the fewest number of inner product calculations), though sometimes one sweep more than in PJSVD is needed. In such a case, however, taking into account that a rotation involves more work than an inner product calculation, the overhead for calculating extra inner products is sufficiently compensated for by fewer orthogonalizations.

Note that Table 1 shows a remarkable difference between the number of calculated inner products for the Hilbert matrices and the matrices of comparable size in the other test cases. This is because during the early sweeps the norms of a large number of columns are reduced so much that they can be discarded during the rest of the orthogonalization process. The corresponding singular values cannot be numerically distinguished from zero.

TABLE 1
Comparison of different variants of Jacobi-like SVD.

Test	n	CJSVD S	CJSVD R	CJSVD P	CJSVDT S	CJSVDT R	CJSVDT P	PJSVD S	PJSVD R	PJSVD P	PJSVDT S	PJSVDT R	PJSVDT P
1	32	10	2	3	11	3	3	6	2	2	7	2	2
	64	10	8	8	10	9	11	7	6	6	7	6	7
	128	12	27	28	12	28	36	7	20	20	8	16	24
	256	12	77	78	11	62	98	8	95	95	8	51	78
2a	32	8	3	4	7	2	3	7	2	3	8	2	4
	64	9	13	18	8	8	16	7	11	14	8	8	16
	128	10	54	81	9	33	73	8	45	65	8	31	65
	256	10	239	326	10	187	326	8	185	261	8	143	261
2b	32	8	3	4	9	3	4	7	2	3	7	2	3
	64	10	12	20	10	9	20	8	10	16	8	8	16
	128	11	51	89	10	37	81	8	46	65	9	31	73
	256	13	219	424	11	148	359	9	198	293	9	120	293
3	100	10	31	50	10	21	50	11	35	54	10	20	50
	200	10	124	199	10	81	199	10	135	198	10	74	198
	300	10	278	448	10	172	448	11	298	492	10	157	447
	400	10	500	797	10	301	797	12	536	956	10	270	796
4	50	9	8	11	9	8	11	8	7	10	8	6	10
	100	10	36	50	10	33	50	9	30	44	9	28	44
	200	11	152	225	11	143	225	9	126	183	10	117	193

TABLE 2
CPU *times in seconds for various routines on the* CYBER 205.

Test	n	F02WCF	SSVDC	CJSVD	PJSVDT
1	32	0.120	0.073	0.056	0.045
	64	0.569	0.333	0.207	0.168
	128	3.033	1.684	0.931	0.672
	256	18.244	8.967	4.044	3.186
2a	32	0.214	0.104	0.071	0.058
	64	1.114	0.446	0.349	0.246
	128	6.364	2.042	1.911	1.267
	256	40.499	10.616	11.625	7.778
2b	32	0.198	0.082	0.068	0.055
	64	1.017	0.360	0.351	0.248
	128	*5.673*	*1.677*	*1.875*	*1.293*
	256	35.884	8.446	11.803	7.313
3	100	2.839	1.246	1.011	0.790
	200	16.107	5.317	5.485	4.074
	300	47.245	13.779	15.688	11.337
	400		27.901	34.053	23.968
4	50	0.644	0.325	0.237	0.201
	100	3.453	1.574	1.371	1.142
	200		8.395	8.920	7.249

Remark. When a large number of parallel processors is available, the reduction of the number of sweeps becomes more important than the skipping of individual

rotations. Table 1 indicates that the pivoting strategy has a large influence on the number of sweeps. Unfortunately, it is difficult to incorporate the pivoting strategy in a parallel scheme. On the other hand, recall that a part of the effect of pivoting, namely the fact that during the orthogonalization process the columns with larger norms are treated first, can also be achieved by using a more elaborate threshold strategy.

4.3. Comparison with the Golub–Reinsch algorithm. Our main interest in the Jacobi-like algorithm lies in the fact that it is easily adapted for parallel computation. On the other hand, as we have seen in the introduction of this paper, there are some situations in which the algorithm becomes attractive even when the computations are performed in a sequential way. The question which then arises is: How does the algorithm perform compared with the Golub-Reinsch algorithm.

Let us first give an estimation of the number of operations, using the number of multiplications as a measure of work. Though this does not always give the right impression of the amount of work, as can be seen from the two different modifications of the Jacobi rotations, it gives an indication of what can be expected in practice.

In each minor step of the orthogonalization process, the most time-consuming parts are the following:

- The calculation of an inner product at the cost of m multiplications;
- The orthogonalization of two columns by a Jacobi rotation at the cost of $2m$ multiplications when using the modified form;
- The accumulation of the right singular vectors by a Jacobi rotation, which can be done in a similar fashion, at the cost of $2n$ multiplications.

Thus, since in each sweep at most $\frac{1}{2}n(n-1) \approx \frac{1}{2}n^2$ pairs of columns are treated, we have a total of $S(\frac{3}{2}m+n)n^2$ multiplications for the whole algorithm, where S is the total number of sweeps necessary to orthogonalize the given matrix. Table 1 indicates that, on the average, $S \approx 10$.

From Chan (1982), we know that the Golub-Reinsch SVD algorithm usually takes about $7mn^2+(11/3)n^3$ multiplications. Looking at the case $m=n$, this gives $(32/3)n^3$ multiplications, whereas the Jacobi-like algorithm costs roughly $25n^3$ multiplications. Thus, the difference is almost a factor of $2\frac{1}{2}$. However, Table 1 indicates that, at least in our experiments, the operation counts as given here are far too pessimistic. Even in the "trivial threshold" cases, the number of performed orthogonalizations is much less than the number of calculated inner products, whereas in the pivoting variants the number of performed orthogonalizations varies from a third to a half of the number of calculated inner products.

To see how it turns out in practice, we compared our routines with the subroutines F02WCF from NAG (see NAG (1984)) and SSVDC from LINPACK (see Dongarra et al. (1979)), which are both (standard) FORTRAN implementations of the Golub-Reinsch SVD algorithm. F02WCF is not adapted to the architecture of the CYBER 205. SSVDC is "column biased" and all column operations are expressed in BLAS calls (see Lawson et al. (1979)). These BLAS are (manufacturer-)optimized for the CYBER 205. Both NAG and LINPACK deliver the SVD in standard form (see (1.3)), whereas the Jacobi-like SVD algorithm as presented in § 2 only calculates the reduced SVD. In order to have a fair comparison, we extended our routines by calculating the QR-factorization (with column pivoting) of matrix X in (2.1.3):

$$X\Pi = QR,$$

where Π is the permutation matrix reflecting the column pivoting. Since X, and consequently R, has orthogonal columns, it follows that R is diagonal. Thus, the

diagonal elements of R are, apart from the signs, the singular values. By calculating only the first n columns of Q, we have the SVD in standard form. Note that these additional calculations give an overhead in time that approximately equals the time necessary to perform two sweeps in the orthogonalization process (and which in many applications is not needed at all).

The CPU times needed for F02WCF, SSVDC, and for the variants CJSVD and PJSVDT are given in Table 2. The remarkable differences in performance between NAG and LINPACK are explained by the fact that NAG is not vectorized, whereas LINPACK is. In addition, LINPACK could still be improved by replacing the BLAS by the FTN200 vector instructions.

Though in all tested cases PJSVDT is the fastest routine, the relative differences in CPU times between SSVDC and PJSVDT decrease when the sizes of the matrices increase, except for the Hilbert matrices, where the opposite is the case (as explained in the previous section). Thus, comparing the two algorithms, the conclusion that one is better or worse than the other is not justified, particularly since our routines are highly optimized, whereas NAG and LINPACK are not. Still, because of its simplicity, and considering the fact that it can easily be adapted to and modified (optimized) for specific computer environments, the Jacobi-like algorithm can be seen as a competitive (and sometimes superior) alternative of the Golub–Reinsch method.

REFERENCES

R. P. BRENT AND F. T. LUK (1985), *The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays*, SIAM J. Sci. Statist. Comput., 6, pp. 69–84.

T. F. CHAN (1982), *An improved algorithm for computing the singular value decomposition*, ACM Trans. Math. Software, 8, pp. 72–83.

B. A. CHARTRES (1962), *Adaptation of the Jacobi method for a computer with magnetic-tape backing store*, Comput. J., 5, pp. 51–60.

J. J. DONGARRA, C. B. MOLER, J. R. BUNCH, AND G. W. STEWART (1979), LINPACK *User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia.

G. E. FORSYTHE AND P. HENRICI (1960), *The cyclic Jacobi method for computing the principal values of a complex matrix*, Trans. Amer. Math. Soc., 94, pp. 1–23.

G. H. GOLUB AND C. REINSCH (1970), *Singular value decomposition and least squares solutions*, Numer. Math., 14, pp. 403–420.

G. H. GOLUB AND C. F. VAN LOAN (1983), *Matrix Computations*, North Oxford Academic, Oxford.

S. HAMMARLING (1974) *A note on modifications to the Givens plane rotation*, J. Inst. Math. Appl., 13, pp. 215–218.

P. HENRICI (1958), *On the speed of convergence of cyclic and quasicyclic Jacobi methods for computing the eigenvalues of hermitian matrices*, SIAM J. Applied Math., 6, pp. 144–162.

M. R. HESTENES (1958), *Inversion of matrices by biorthogonalization and related results*, SIAM J. Appl. Math., 6, pp. 51–90.

H. P. M. VAN KEMPEN (1966), *On quadratic convergence of the special cyclic Jacobi method*, Numer. Math., 9, pp. 19–22.

C. LAWSON, R. HANSON, D. KINCAID, AND F. KROGH (1979), *Basic linear algebra subprograms for* FORTRAN *usage*, ACM Trans. Math. Software, 5, pp. 308–323.

F. LUK (1980), *Computing the singular value decomposition on the* ILLIAC IV, ACM Trans. Math. Software, 6, pp. 524–539.

J. C. NASH (1975), *A one-sided transformation method for the singular value decomposition and algebraic eigenproblem*, Comput. J., 18, pp. 74–76.

NAG (1984), NAG FORTRAN *Library Manual*, Mark 11, Numerical Algorithms Group, Oxford.

H. RUTISHAUSER (1966), *The Jacobi method for real symmetric matrices*, Numer. Math, 9, pp. 1–10.

J. H. WILKINSON (1962), *Note on the quadratic convergence of the cyclic Jacobi process*, Numer. Math., 4, pp. 296–300.

———, (1965), *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 372-381, March 1989
© 1989 Society for Industrial and Applied Mathematics
009

ORTHOGONAL REDUCTION ON VECTOR COMPUTERS*

R. BRUCE MATTINGLY†, CARL D. MEYER‡, AND JAMES M. ORTEGA§

Abstract. This paper concerns the implementation of the QR factorization by Givens and Householder transformations on vector computers.

Key words. Householder reduction, Givens reduction, orthogonal reduction, vector computers, QR factorization, orthogonal factorization

AMS(MOS) subject classifications. 65F25, 65F05, 68O4, 15A23, 15O4

1. Introduction. Following the analysis of Dongarra, Gustavson, and Karp [2] for Gaussian elimination, various **ijk forms** for both Givens and Householder transformations are investigated. Conclusions concerning which of these forms have desirable or undesirable properties for vector computers are presented. These *ijk* forms utilize only rows or columns as the basic entities of computation. Data flow organizations such as those given by Bojanczyk et al. [1] or the various annihilation patterns for the Givens method such as those considered by Modi and Clarke [7] are not considered. Furthermore, the pipelined Givens and the windowed Householder schemes developed for parallel machines by Dongarra et al. [5] are not discussed. We report on experiments run on a **CRAY-1** and conclude from these experiments together with our analysis which of the *ijk* forms are most promising on this machine. This work complements the results of Dongarra et al. [4].

2. The ijk forms of Householder reduction. For an $n \times n$ matrix $\mathbf{A}$, the Householder reduction to triangular form proceeds by multiplications by elementary reflectors of the form

$$\mathbf{P} = \mathbf{I} - \mathbf{w}\mathbf{w}^{\mathbf{T}} \quad \text{where } \mathbf{w}^{\mathbf{T}}\mathbf{w} = 2.$$

The first stage produces zeros in the first column of $\mathbf{PA}$ below the main diagonal and is effected by taking

$$\begin{aligned} &s = \sigma\left(\sum_{i=1}^{n} a_{i1}^2\right)^{1/2} \quad \text{where } \sigma = \operatorname{sign}(a_{11}), \\ &\mu^2 = (sa_{11} + s^2)^{-1}, \\ &\mathbf{u}^{\mathbf{T}} = [a_{11} + s, a_{21}, \cdots, a_{n1}], \\ &\mathbf{w} = \mu\mathbf{u}. \end{aligned} \tag{2.1}$$

The new $(1, 1)$ element is s and the major work in the computation is the update of the remaining $n-1$ columns of $\mathbf{A}$. Note that in the above description and the following codes, we have not scaled the first column in order to guard against overflow or underflow (see, e.g., Stewart [8]). This can easily be added.

* Received by the editors April 6, 1987; accepted for publication (in revised form) June 7, 1988.

† Department of Mathematics, Youngstown State University, Youngstown, Ohio 44555.

‡ Department of Mathematics and Center for Research in Scientific Computation, North Carolina State University, Raleigh, North Carolina 27695-8205. The work of this author was supported in part by National Science Foundation grant DMS-8521154.

§ Department of Applied Mathematics, University of Virginia, Charlottesville, Virginia 22901. The work of this author was supported in part by National Aeronautics and Space Administration grant NAG1-46.

There are two common ways of obtaining the update (e.g., see Golub and Van Loan [6]). The first is the **inner product form**

$$\mathbf{A}_{*j} := \mathbf{A}_{*j} - \mu^2(\mathbf{u}^T\mathbf{A}_{*j})\mathbf{u} \quad \text{for } j = 2, 3, \cdots, n \tag{2.2}$$

where $\mathbf{A}_{*j}$ is the jth column of $\mathbf{A}$. The update (2.2) arises from writing

$$(\mathbf{I} - \mathbf{w}\mathbf{w}^T)\mathbf{A} = \mathbf{A} - \mathbf{w}[\mathbf{w}^T\mathbf{A}_{*1}, \cdots, \mathbf{w}^T\mathbf{A}_{*n}].$$

On the other hand, if $\mathbf{A}_{i*}$ denotes the ith row of $\mathbf{A}$, then

$$(\mathbf{I} - \mathbf{w}\mathbf{w}^T)\mathbf{A} = \mathbf{A} - \mathbf{w}(\mathbf{w}^T\mathbf{A}) = \mathbf{A} - \mathbf{w}\left(\mu \sum_{i=1}^{n} u_i\mathbf{A}_{i*}\right).$$

This produces the **rank-one update form**

$$\mathbf{A}_{i*} := \mathbf{A}_{i*} - z_i\mathbf{v}^T \quad \text{where } \mathbf{z} = \mu^2\mathbf{u} \quad \text{and} \quad \mathbf{v}^T = \sum_{i=1}^{n} u_i\mathbf{A}_{i*}. \tag{2.3}$$

The framework for a Householder code that uses the rank-one update (2.3) is given in Fig. 1. The first **ij** loop computes the vector $\mathbf{v}_k$, according to (2.3), as a linear combination of the rows of **A**. The second **ij** loop then updates **A**. Access to **A** is by rows and the vector operations are the linked triads (i.e., **SAXPY**'s) $\mathbf{v}_k + z_{ik}\mathbf{A}_{i*}$ in the first **ij** loop and $\mathbf{A}_{i*} - u_{ik}\mathbf{v}_k$ in the second **ij** loop. The last statement in the code is analogous to the corresponding statement

$$a_{ij} = a_{ij} - l_{ik}a_{kj}$$

in Gaussian elimination where u_{ik} plays the role of the multiplier l_{ik} and v_{kj} plays the role of $\mathbf{a}_{kj}$. With this interpretation, the rank-one update code corresponds as well as could be expected to the **kij** form of Gaussian elimination as given in Dongarra et al. [2]. This is the motivation for the terminology **kij form of Householder reduction**.

By interchanging the indices i and j in the rank-one update code, the **kji (inner product) form** of Householder reduction, given in Fig. 2, is obtained.

```
For k = 1 to n − 1
    Compute s_k, μ_k^2, u_k, z_k according to (2.1) and (2.3)
    For i = k to n
      For j = k + 1 to n
        v_kj = v_kj + z_ik a_ij
    For i = k to n
      For j = k + 1 to n
        a_ij = a_ij − u_ik v_kj
```

FIG. 1. *The* **kij** *(rank-one update) form of Householder reduction.*

```
For k = 1 to n − 1
    Compute s_k, μ_k^2, u_k, z_k according to (2.1) and (2.3)
    For j = k + 1 to n
      For i = k to n
        v_kj = v_kj + z_ik a_ij
    For j = k + 1 to n
      For i = k to n
        a_ij = a_ij − u_ik v_kj
```

FIG. 2. *The* **kji** *(inner product) form of Householder reduction.*

In this code, the first **ji** loop computes $(\mu_k^2)\mathbf{z_k}\mathbf{A_{*j}}$. The second **ji** loop then updates $\mathbf{A_{*j}}$. Therefore, this is the inner product form of Householder reduction given in (2.2). Note that in the inner product form access to **A** is now by columns. Moreover, the vector operations are inner products in the first **ji** loop and linked triads in the second **ji** loop.

K. Fong and T. Jordan recognized in 1977 that Gaussian elimination could be implemented more efficiently on the **CRAY-1** by delaying the updating of columns of **A** until the time that the column is to receive final processing. This minimizes stores to memory and helps to circumvent the one-line-to-memory problem on the **CRAY-1**. The Fong-Jordan algorithm is the **jki form of Gaussian elimination** given by Dongarra et al. [2]. The corresponding Householder code is given in Fig. 3.

```
For j = 2 to n
    Compute s_{j-1}, μ²_{j-1}, u_{j-1}, z_{j-1} according to (2.1) and (2.3)
    For k = 1 to j-1
      For i = k to n
        v_kj = v_kj + z_ik a_ij
      For i = k to n
        a_ij = a_ij - u_ik v_kj
```

FIG. 3. *The* **jki** *(delayed update) form of Householder reduction.*

Note that this code is somewhat different in structure than the previous ones in that there is now a single **k** loop with two subsidiary **i** loops. The first **i** loop computes $\mu_k^2\mathbf{z_k^T}\mathbf{A_{*j}}$. The second **i** loop then updates $\mathbf{A_{*j}}$. Therefore, this form also carries out the inner product formulation (2.2) but updating of the jth column of **A** is delayed until the jth stage. We therefore call this the **delayed update form.**

Dongarra et al. [2] have given six forms for Gaussian elimination, each corresponding to one of the six permutations of the indices i, j, and k. For the Householder reduction, however, three of the possible six forms do not give useful algorithms. The **ikj** form should correspond to the **kij** form except that the updating of rows is delayed. However, an update of the, say, third row at the second stage requires that all rows have been updated at the first stage in order to compute the proper **v**. Similar problems arise with the **ijk** and **jik** forms. We therefore conclude that from the six possibilities, the only viable forms of Householder reduction are the **kij, kji,** and **jki** forms. We will now discuss in more detail the properties of these three forms on vector computers.

The **kij** form is ideal for the **CDC CYBER 205** and similar machines because its vector operations are all linked triads that can be done very efficiently. Access to **A** is by rows but if column storage is preferred, then we can work with $\mathbf{A^T}$ and use right-hand Householder transformations $\mathbf{A^T(I-ww^T)}$. The **kji** form uses inner products as well as linked triads, and hence it will be less satisfactory for the **CYBER 205**. Similarly, the **jki** form has little merit for this machine.

For the **CRAY-1**, the **kij** form is potentially quite satisfactory. In principle, the formation of the vector $\mathbf{v_k}$ can be accomplished by accumulation in a vector register. In practice, loop unrolling (Dongarra and Hinds [3]) is required to approximate this accumulation in FORTRAN. Moreover, the update of the rows is a sequence of individual linked triads which does not allow the possibility of accumulation. Dongarra et al. [4] have addressed this latter problem by combining two successive Householder transformations. This leads to rank-two updates, somewhat similar to the loop-unrolling artifice.

For the **CRAY-1**, the **kji** form seems less satisfactory than the **kij** form since the accumulation of linked triads in the formation of the **v**'s is replaced by inner products while, again, the updates are individual linked triads. The **jki** form allows the possibility of accumulation in the updates, as in Gaussian elimination. However, if $n > 64$, the number of elements in a vector register, then this possibility is defeated by the need to use the entire current vector $\mathbf{A}_{*j}$ to obtain the inner product. Therefore, it is not possible to retain $\mathbf{A}_{*j}$ in a vector register while the updates are proceeding. (With assembly language programming and judicious use of the vector registers, we could handle $n \leqq 128$.) In general, the **jki** form seems to have no advantage over the **kji** form. Our experiments, discussed in § 4, confirm that the **kij** form with loop unrolling is superior to the **kji** form, which in turn is almost equivalent to the **jki** form in FORTRAN.

It is well known (see Golub and Van Loan [6]) that Householder transformations can also be used to annihilate only a portion of the elements in a column. Techniques used by Dongarra et al. [2] for fast matrix multiplication on the **CRAY-1** when vector lengths exceed the number of elements in a vector register suggest a possible modification to the Householder reduction based on partial annihilation. At the first stage, choose **w** so that the first column of the transformed matrix has zeros in positions 2 through m. This is achieved by taking

$$
\begin{aligned}
& s = \sigma\left(\sum_{i=1}^{m} a_{i1}^2\right)^{1/2} \quad \text{where } \sigma = \operatorname{sign}(a_{11}), \\
& \mu^2 = (s a_{11} + s^2)^{-1}, \\
& \mathbf{w}^{\mathrm{T}} = \mu[a_{11} + s, a_{21}, \cdots, a_{m1}, 0, \cdots, 0].
\end{aligned} \tag{2.4}
$$

At the next stage, we can choose either the transformation to zero out the next group of $m-1$ elements in the first column or the first $m-2$ elements below the main diagonal in the second column. In either case, zeros produced in the previous stage are preserved. For the **CRAY-1**, we elect to move across the columns of **A**, with $m = 64$. In the first stage, elements 2 through 64 in the first column are annihilated, then elements 3 through 64 in the second column at the second stage, and so on until a triangle of zeros has been completed at stage 63. Stage 64 annihilates the next 63 elements in the first column; stage 65 annihilates the next 63 elements in the second column; and so on. The updates are performed by a modified **jki** form of Householder reduction so that updates are delayed. For example, at stage 5, all needed updates are performed on the first 64 elements of the fifth column. For these updates, this portion of the fifth column can be held in a vector register until the updates are completed. In this way, the desired delayed updating effect is obtained.

There are rather heavy additional costs in this modified Householder algorithm.

(i) The code is more complex.

(ii) $\mathbf{O}(n^2/64)$ square roots are required as opposed to $n-1$ square roots for regular Householder reduction.

(iii) The vector lengths are never more than 64 and they reduce to 1 as each triangle of zeros is produced.

We have not been able to test this approach and so it remains unclear whether it would provide a viable alternative algorithm.

3. Givens reduction. We now consider the various **ijk** forms of Givens reduction. Throughout the following discussion, s_{ki} and c_{ki} will designate the sine-cosine pair computed at the kth stage using the current (k, k) and (i, k) entries. The explicit expressions for c_{ki} and s_{ki} are not written. Their appearance in an algorithm indicates the point at which they are to be computed.

In the **kij** form given in Fig. 4, it is assumed that **A** is stored by rows. The notation $\hat{a}_{kj}$ distinguishes entries of the new kth row. This is necessary because the statement following the one containing $\hat{a}_{kj}$ uses elements from the old kth row. The **kij** form is the usual row-oriented organization of Givens reduction. The inner loop computes linear combinations of the current k and i rows of **A**. The corresponding column form of Givens reduction is given in Fig. 5.

In the **kji** form given in Fig. 5, superscripts indicate that the values of the (k, j) element are from the previous pass through the ***i*** loop. All of the sine-cosine pairs for the annihilation of the kth column are computed at the beginning of the kth stage. This requires continual updating of the (k, k) element so that the preparatory **h** loop is entirely scalar arithmetic. The first arithmetic statement in the **ji** loop is the updating of the (k, j) element, which is modified each time through the **i** loop. In terms of vector operations, the inner **i** loop may be written as follows:

$$\mathbf{p} = \mathbf{s_k} \otimes \mathbf{A_{*j}}, \qquad \mathbf{r} = \mathbf{c_k} \otimes \mathbf{A_{*j}}$$

For $i = k+1$ *to* n

$$a_{ij}^{(i+1)} = c_{ki} a_{kj}^{(i)} + p_i$$

$$q_i = -s_{ki} a_{kj}^{(i)}$$

$$\mathbf{A_{*j}} = \mathbf{r} + \mathbf{q}$$

where $\otimes$ denotes component-by-component vector-vector multiplication. Thus, the inner **i** loop of the **kji** form of the Givens reduction can be written in terms of three vector operations—two vector-vector multiplies and one vector-vector addition—together with one scalar loop. It is clear that the **kji** form of Givens reduction does not vectorize well, and the numerical results in § 4 show that it is not competitive with the **kij** form.

Both the **kij** and **kji** forms of Givens reduction are immediate update algorithms. We now consider the **delayed update ikj** and **jki** forms.

For $k = 1$ *to* $n-1$
  For $i = k+1$ *to* n
    Compute s_{ki}, c_{ki}
    For $j = k$ *to* n
      $\hat{a}_{kj} = c_{ki} a_{kj} + s_{ki} a_{ij}$
      $a_{ij} = -s_{ki} a_{kj} + c_{ki} a_{ij}$

FIG. 4. *The* **kij** *form of Givens reduction.*

For $k = 1$ *to* $n-1$
  For $h = k+1$ *to* n
    Compute s_{kh}, c_{kh}
    $a_{kk} = c_{kh} a_{kk} + s_{kh} a_{hk}$
  For $j = k+1$ *to* n
    For $i = k+1$ *to* n
      $a_{kj}^{(i+1)} = c_{ki} a_{kj}^{(i)} + s_{ki} a_{ij}$
      $a_{ij} = -s_{ki} a_{kj}^{(i)} + c_{ki} a_{ij}$

FIG. 5. *The* **kji** *form of Givens reduction.*

Just as in the **kij** form, the **ikj** form given in Fig. 6 assumes that **A** is stored by rows and the last statement uses the previous value of a_{kj} and not the updated value $\hat{a}_{kj}$. The annihilation pattern is across rows rather than down columns. For $i=3$, element (3, 1) is annihilated followed by element (3, 2), etc. In general, the annihilation pattern is indicated below.

$$\begin{pmatrix} * & * & * & * & * & \cdots \\ 1 & * & * & * & * & \cdots \\ 2 & 3 & * & * & * & \cdots \\ 4 & 5 & 6 & * & * & \cdots \\ 7 & 8 & 9 & 10 & * & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

The column-oriented **jki** form given in Fig. 7 is analogous to the **kji** form with the exception that updates are delayed. All updating of the jth column is done at the jth stage.

The remaining two forms, **ijk** and **jik**, do not seem to lead to vector algorithms. The only codes that we have been able to devise for these two forms are almost completely serial and they will not be discussed.

We now consider in more detail the vector properties of the four forms **kij**, **kji**, **ikj**, and **jki** of Givens reduction. The **kij** form is quite suitable for the **CYBER 205** and should be the best of the four forms for this machine. Note that in this form, at the kth stage, the kth row is successively used in all of the rotations. Hence, on the **CRAY-1**, we wish to keep the current updates of this row in a vector register and store only at the end of the kth stage. This is the implementation recommended by Dongarra et al. [4]. With this implementation, the **kij** form should run at slightly better than vector speed.

As already noted, the **kji** form has less suitable vector properties and should be inferior to the **kij** form on both the **CYBER 205** and **CRAY-1.** The delayed update **jki** form allows the possibility of accumulation in a vector register, but the same undesirable

For $i=2$ *to* n
  For $k=1$ *to* $i-1$
    Compute s_{ki}, c_{ki}
    For $j=k$ *to* n
      $\hat{a}_{kj} = c_{ki}a_{kj} + s_{ki}a_{ij}$
      $a_{ij} = -s_{ki}a_{kj} + c_{ki}a_{ij}$

FIG. 6. *The* **ikj** *form of Givens reduction.*

For $j=2$ *to* n
  For $h=j$ *to* n
    Compute $c_{j-1,h}$, $s_{j-1,h}$
    $a_{j-1,j-1} = c_{j-1,h}a_{j-1,j-1} + s_{j-1,h}a_{hj}$
  For $k=1$ *to* $j-1$
    For $i=k+1$ *to* n
      $a_{kj}^{(i+1)} = c_{ki}a_{kj}^{(i)} + s_{ki}a_{ij}$
      $a_{ij} = -s_{ki}a_{kj} + c_{ki}a_{ij}$

FIG. 7. *The* **jki** *form of Givens reduction.*

vector properties as in the **kji** form will probably decrease its speed to less than the **kij** form. This is indeed the case for our experiments. The delayed update **ikj** form seems potentially the best for the **CRAY-1**. At the ith stage, the ith row will be updated $i-1$ times and the successively updated row will be kept in a vector register until the updating is complete.

4. Experimental results. Several of the previous algorithms were run on the **CRAY-1** at the University of Minnesota, using the Cray FORTRAN compiler **CFT 1.11**. The algorithms were tested on square dense matrices ranging in size from 500×500 to 920×920. The matrix entries were randomly generated numbers between 0 and 99. The codes used no nonstandard FORTRAN statements other than the subroutines which were used to obtain the elapsed CPU time and to generate random numbers.

First, the Householder inner product **kji** form and the rank-one update **kij** form were compared. Initially, no loop unrolling was attempted. Since the **kij** form accesses the matrix by rows, it was expected that this form would be susceptible to some slowdown due to memory bank conflicts, primarily when the number of rows in the matrix was a multiple of 16 and, to a lesser extent, when the number of rows was an odd multiple of eight. This is because the **CRAY-1**'s memory consists of 16 interleaved banks, and in FORTRAN matrices are ordinarily stored by columns. Test results have confirmed that slowdowns do indeed occur. One way of alleviating this problem would be to store the transpose of the matrix so that the access pattern of the algorithm would be down the columns. A different approach, which we used, is to store the matrix in the usual way but increase the row dimension by one. This gave a dramatic improvement in the running times, as can be seen from Table 1.

TABLE 1
The effect of increasing the row dimension by one on the Householder **kij** *(rank-one) update form.*

N	Original run times (seconds)	Improved run times (seconds)	Percent difference
520	7.34	5.52	24.8
560	13.48	6.87	49.0
600	11.24	8.41	25.2
640	20.05	10.17	49.3
680	16.23	12.16	25.1
720	28.45	14.39	49.4
760	22.56	16.86	25.3
800	39.01	19.63	49.7
840	30.39	22.61	25.6
880	51.73	26.05	49.6
920	39.77	29.71	25.2

As expected, the **kji** (inner product) form of Householder reduction proved not to be sensitive to memory bank conflicts. Table 2 shows that the **kij** (rank-one update) form performed 15–18 percent faster than the **kji** form.

Unrolling the loops in the **kji** form of Householder reduction gave a slight improvement (about 9 percent) in running time, but the **kij** form was still faster. Loop unrolling gave a more significant improvement in the **kij** form. Experiments indicate that unrolling loops to different depths gives slightly different timings, but the loops unrolled to a depth of six gave the best results. Table 3 summarizes the timings obtained using the **kij** form.

TABLE 2
Comparison of the **kji** *and* **kij** *forms of Householder reduction.*

N	**kji** time (seconds)	**kij** time (seconds)	Percent difference
500	6.0	4.9	18
600	10.0	8.4	16
700	15.7	13.0	17
800	23.1	19.6	15
900	32.6	27.8	15

TABLE 3
The effect of unrolling loops to a depth of six in the **kij** *form of Householder reduction.*

N	Not unrolled (seconds)	Unrolled (seconds)	Percent difference
500	4.9	3.7	24
600	8.4	6.3	25
700	13.0	10.0	23
800	19.6	15.1	23
900	27.8	21.3	23

The last Householder variant we implemented was the delayed update **jki** form. Table 4 shows that this form is roughly equivalent to the **kji** inner product form. Unrolling loops in the **jki** form gave no significant improvement in speed. A major disadvantage of this form is that much more storage is required. In particular three $n \times n$ arrays are needed, whereas only one $n \times n$ array is required by the other forms. For our testing on the **CRAY-1**, this constrained us to matrices in which $n \leqq 540$. By clever storage techniques, the number of arrays might be decreased from three to two. However, this will almost certainly introduce additional overhead which will degrade the overall performance.

We next consider Givens reduction. As Table 5 shows, the **kji** and **jki** forms proved not to be competitive, whereas the times for the **kij** and **ikj** forms were almost identical and slightly faster than the **kji** inner product form of Householder reduction. However, they were slightly slower than the basic **kij** Householder form and considerably slower when loops were unrolled in this Householder form. Experiments with loop unrolling for the Givens algorithms did not produce any significant improvements. We note that since the **kij** and **ijk** forms of Givens reduction are both row oriented, we used the transpose so as to produce the column storage desired by FORTRAN.

TABLE 4
Comparison of the **kji** *and* **jki** *forms of Householder reduction.*

N	**kji** time (seconds)	**jki** time (seconds)
500	6.0	5.9
520	6.7	6.7
540	7.4	7.4

TABLE 5
Comparison of the various forms of Givens reduction.

N	**kij** time (seconds)	**kji** time (seconds)	**ikj** time (seconds)	**jki** time (seconds)
500	5.4	27.8	5.4	32.8
600	9.2		9.2	
700	14.4		14.5	
800	21.3		21.4	
900	30.1		30.1	

Of the several forms of Givens and Householder reduction that were tested, the **kij** form of Householder reduction is best, and its superiority becomes even more pronounced when loop unrolling is used.

5. Megaflop rates and theoretical running times. For our purposes, we consider a "*flop*" to be any one of the four basic arithmetic operations. With this definition, the Householder codes require $\mathbf{O}(\frac{4}{3}n^3)$ flops and the Givens codes require $\mathbf{O}(2n^3)$ flops. Note that all codes perform the same computations—they merely perform them in different orders. Based on these counts, megaflop rates for $n = 900$ are reported in Table 6 for the different Householder and Givens codes.

Note that the Givens codes have higher megaflop rates than the Householder codes, but the Givens codes run slower in actual time. This, of course, is because the operation count for Givens reduction is higher than that for Householder reduction.

To estimate theoretical running times for the **kij** forms of the Householder and Givens codes, a computational model for each was formulated to account for the following parameters:

- The number of clock cycles for the various arithmetic operations.
- The vector startup time.
- The memory access time.
- The clock cycle time.

Our model neglects overhead for DO-LOOPS and subroutine calls and we assume no memory bank conflicts. For problems of dimension n, we estimated that on the **CRAY-1** the Householder codes should require at least $2.94 \times 10^{-8} \times n^3$ seconds, whereas the Givens codes should require at least $3.07 \times 10^{-8} \times n^3$ seconds. Table 7 compares the theoretical running times (in seconds) with running times we observed in practice. In light of our assumptions, we believe that the theoretical numbers in Table 7 provide a reasonable qualitative description of the behavior of the algorithm.

TABLE 6
Megaflop rates. ($n = 900$.)

Algorithm	Megaflop rate
HH **kji**	29.83
HH **kji** (unrolled)	32.40
HH **kij**	35.18
HH **kij** (unrolled)	45.63
Givens **kij**	48.44
Givens **ikj**	48.44

TABLE 7
Theoretical versus actual timings.

	Householder **kij**			Givens **kij**		
N	Theory	Actual	Percent difference	Theory	Actual	Percent difference
500	3.8	4.9	22	4.2	5.4	22
700	10.3	13.0	21	11.3	14.4	22
900	21.8	27.8	22	23.7	30.1	21

We did not attempt to construct models for the other codes. The major difference between the various codes is not the number of operations performed, but rather the order in which operations are performed. Therefore, the major difference in running times would be due to factors (e.g., memory bank conflicts) for which our models do not account.

REFERENCES

[1] A. BOJANCZYK, R. BRENT, AND H. KUNG, *Numerically stable solution of dense systems of linear equations using mesh-connected processors*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 95–104.

[2] J. DONGARRA, F. GUSTAVSON, AND A. KARP, *Implementing linear algebra algorithms for dense matrices on a vector pipeline machine*, SIAM Rev., 26 (1984), pp. 91–112.

[3] J. DONGARRA AND A. HINDS, *Unrolling loops in Fortran*, Software Pract. Exper., 9 (1979), pp. 219–229.

[4] J. DONGARRA, L. KAUFMAN, AND S. HAMMARLING, *Squeezing the most out of eigenvalue solvers on high-performance computers*, Linear Algebra Appl., 77 (1986), pp. 113–136.

[5] J. DONGARRA, A. SAMEH, AND D. SORENSEN, *Implementation of some concurrent algorithms for matrix factorization*, Parallel Comput., 3 (1986), pp. 25–34.

[6] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, 1983.

[7] J. MODI AND M. CLARKE, *An alternative Givens ordering*, Numer. Math., 43 (1984), pp. 83–90.

[8] G. W. STEWART, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 382-396, March 1989
© 1989 Society for Industrial and Applied Mathematics
010

A NEW METHOD FOR SOLVING TRIANGULAR SYSTEMS ON DISTRIBUTED-MEMORY MESSAGE-PASSING MULTIPROCESSORS*

GUANGYE LI† AND THOMAS F. COLEMAN‡

Abstract. Efficient triangular solvers for use on message-passing multiprocessors are required, in several contexts, under the assumption that the matrix is distributed by columns (or rows) in a wrap fashion. In this paper a new efficient parallel triangular solver for these problems is described. This new algorithm is based on the previous method of Li and Coleman [1988] but is considerably more efficient when n/p is relatively modest, where p is the number of processors, and n is the problem dimension.

A useful theoretical analysis is provided as well as extensive numerical results obtained on an Intel iPSC with $p \leqq 128$.

Key words. hypercube multiprocessor, parallel triangular solver, parallel Gaussian elimination, parallel matrix factorization

AMS(MOS) subject classification. 65F05

0. Introduction. An important part of many computations is the solution of a triangular system of equations. On a sequential computer this is a simple task; however, an effective parallel procedure on a message-passing multiprocessor is far from trivial. Indeed, recently there has been a flurry of research activity directed at just this issue (e.g., Heath and Romine [1988], Eisenstat et al. [1988], Li and Coleman [1988], Moler [1986a], [1986b], and Romine and Ortega [1986]). In this paper a new algorithm is described: this procedure is an enhancement of the recent method proposed by Li and Coleman [1988] and represents a significant improvement.

The algorithm proposed by Li and Coleman [1988] is applicable on any message-passing multiprocessor (with no shared memory) on which it is possible to embed a ring. Theoretically the new algorithm proposed in this paper can get by with just ring connectivity as well; however, this new algorithm will usually be more efficient when there is more connectivity than this. The precise architectural design is unimportant, provided a ring can be embedded; we have conducted all our experiments on hypercube computers.

We assume that the system to be solved is $Ux = b$, where U is an upper triangular matrix of order n. We also assume that U is distributed to the nodes (processors) by column. This is a natural assumption in many cases. For example, the finite-difference estimation of Jacobian matrices yields a matrix column by column. (A similar algorithm exists for the row-distributed case; Li and Coleman [1988] discuss the base row algorithm. The generalization of this row algorithm is analogous to the column generalization presented here.) Finally, we assume that the columns of U are assigned to the nodes in a wrap fashion. So, for example, if the node containing column j is $P(j)$ then $P(j) = P(k)$ if and only if $j = k \pmod p$, where there are p nodes altogether. A wrap mapping is chosen because it seems a very reasonable choice for the matrix factorization stage (e.g., Geist and Heath [1986], Chamberlain [1986]), which often precedes a triangular solve.

* Received by the editors March 3, 1987; accepted for publication June 14, 1988.

† Computer Center, Jilin University, People's Republic of China. The research of this author was partially supported by the U.S. Army Research Office through the Mathematical Sciences Institute, Cornell University, Ithaca, New York 14853. This work was performed while the author was visiting Cornell University, Ithaca, New York 14853.

‡ Computer Science Department, Cornell University, Ithaca, New York 14853. The research of this author was partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-86ER25013.A000.

The original algorithm proposed by Li and Coleman [1988] is capable of effectively solving triangular systems in many cases. In particular, numerical experience suggests it is among the best algorithms when n/p is large; however, the original Li-Coleman algorithm begins to lose its relative advantage as n/p decreases. Numerical results are reported by Li and Coleman [1988], Moler [1986b], Heath and Romine [1988], and Eisenstat et al. [1988].

The purpose of this paper is to describe a modification of the original Li-Coleman algorithm. This modified version does not degrade in performance as p increases relative to n. Moreover, it actually reduces to the original Li-Coleman algorithm when n/p is relatively large and therefore the modified algorithm maintains a high level of performance under these circumstances as well.

The proposed algorithm can be used on any distributed-memory message-passing multiprocessor in which a ring can be embedded, provided **send** and **receive** primitives are available. We assume that when control of a node program passes to a **send** statement, the **send** is executed immediately, in time zero, and then control passes on to the next executable statement in this node program. Of course this does not imply the message is received immediately; we discuss this transfer time below. We also assume that when control passes to a **receive** statement in a node program, execution of this node program is suspended until the message is physically received, which happens when the appropriate transfer time elapses.

This paper is organized as follows. In § 1 the new algorithm is motivated and described; results of numerical experiments are given. An analysis of the new algorithm is provided in § 2 (with the proofs supplied in the Appendix). In § 3 we briefly discuss a modified version of the new algorithm—this modified version achieves a well-balanced work distribution. Section 4 contains a summary and concluding remarks.

1. The algorithm: motivation, description, and numerical results.

1.1. Motivation. The parallel triangular solver PCTS, proposed by Li and Coleman [1988], is based on a ring architecture and assumes the columns are assigned to the nodes in a wrap fashion. In particular, if the node that contains column i is $P(i)$ then $P(j)=P(k)$ if and only if $j(\text{mod } p)=k(\text{mod } p)$. In the embedded ring, node $P(i(\text{mod } p))$ is connected to $P(i+1(\text{mod } p))$ and $P(i-1(\text{mod } p))$, $i=1:p$.

Mechanically, algorithm PCTS is simple; the $p-1$ array SUM passes around the ring going from $P(j)$ to $P(j-1)$ for $j=n:2$. When SUM arrives at node $P(j)$, $P(j)$ determines $x(j)$, modifies SUM (p flops), and then forwards SUM to node $P(j-1)$. Finally, node $P(j)$ modifies the first $j-p$ elements of array $PSUM$ using column j of U ($j-p$ flops).

The procedure PCTS is executed by every node: the following initial conditions are assumed.

If myname $= P(n)$: $SUM(1:p)=b(n:n-p+1)$
$PSUM(1:n-p)=b(1:n-p)$

If myname $\neq P(n)$: $SUM(1:p)=0$
$PSUM(1;n-p)=0.$

Procedure PCTS ($x[1:m]$, $SUM(1:p)$, $PSUM(1:n-p)$, $U(1:n,[1:m])$
For $j=n:1$
If myname $= P(j)$
Receive $SUM(1:p-1)$ [if $j<n$]
$x(j)\leftarrow(SUM(1)+PSUM(j))/U(j,j)$

$SUM(1:p-2) \leftarrow SUM(2:p-1) - U(j-1:j-(p-2), j) \times x(j)$
$\quad + PSUM(j-1:j-(p-2))$
$SUM(p-1) \leftarrow -U(j-(p-1), j) \times x(j) + PSUM(j-(p-1))$
Send $SUM(1:p-1)$ to node $P(j-1)$ [if $j>1$]
$PSUM(1:j-p) \leftarrow PSUM(1:j-p) - U(1:j-p, j) \times x(j)$

End

Note. For brevity, we follow the convention that if an array index is out of bounds, the returned value is assumed to be zero. Each node has at most $m = \lceil n/p \rceil$ columns of the upper triangular matrix U.

We have listed all the arrays used, and their dimensions, in the procedure statement. The square brackets indicate indirect addressing. For example, $x[1:m]$ says that there are at most m components of the vector x on this node but they are not necessarily the first m components of the n-vector x. In particular, the components of x are assigned to the nodes in a wrap mapping consistent with the assignment of columns. Rather than introduce indirect indexing into the body of the procedure, we refer to the components directly. So for example, $x(j)$ refers to the jth component of the solution x, not the jth component of the array $x[1:m]$ on this node. Of course for this reference to be valid, this component must be assigned to this node.

The mechanism behind PCTS can be described as a distributed outer product. On each node k the array *PSUM* holds part of the outer product corresponding to processed columns on that node; the traveling array *SUM* funnels the distributed sums together, as needed.

Assuming p is fixed, Li and Coleman have shown that the running time of PCTS, $T(n)$, is a function that is linear up to a threshold value of n, after which it is quadratic. In the quadratic range $T(n)$ represents essentially optimal speedup (this is because node $P(n)$ is almost always busy doing useful floating point computations—the other nodes are not quite as busy as $P(n)$ due to variation in column size). However, $T(n)$ reflects less than optimal speed in the linear region and since the threshold value of n can be quite large, it is worthwhile trying to improve PCTS in this region.

In Theorem 2.1 of Li and Coleman [1988] it is shown that $T(n)$ is linear when the busiest node, node $P(n)$, has idle time in every cycle of *SUM*. Therefore, it is easy to see that under these circumstances $T(n)$ is dominated by the time it takes *SUM* to complete a cycle. This cycle time represents the time it takes *SUM* to visit each node on the ring (p flops per node) plus the time it takes *SUM* to traverse the t links. Hence,

$$\text{cycle time} \cong p(t+p) \tag{1.1}$$

where t is the maximum number of flops that can be executed on a single processor during the time in which a single "small" message is **sent** by one node and then **received** by a waiting adjacent node.

In this context we define "small" to be a message of size less than or equal to p/q double precision words (64 bits each), where p is the number of processors and q is a positive integer to be discussed later. Obviously, if *SUM* could be reduced to size $p/q-1$, for $q \geqq 1$, then the time to complete a cycle would be

$$\text{cycle time} \cong \left(\frac{p}{q}\right) p + tp \tag{1.2}$$

provided there are no other compromising ill effects. In particular, the size of q must be restricted because (1.2) is no longer valid if *SUM* arrives back at node $P(n)$ before node $P(n)$ is ready to process *SUM*. Hence, the following principle guides our choice

of q: choose q as large as possible subject to the constraint that node $P(n)$ is always (just) ready to process *SUM* when it arrives. Note that this principle suggests we choose $q=1$ for n sufficiently large, i.e., resort to the original PCTS algorithm.

The step from algorithm PCTS to the new algorithm PCTS$^+$ can best be understood, perhaps, by considering the first cycle of PCTS in which the last p components of x are solved for. In particular, notice that the determination of $x(n-1)$ does not involve $x(n)\times U(n-p+1:n-2, n)$. Similarly, the determination of $x(n-2)$ does not involve $x(n)\times U(n-p+1:n-3, n)$ nor does it involve $x(n-1)\times U(n-p:n-3, n-1)$. Hence it may be possible to ship information **across** the ring (provided the communication links are there) while maintaining essentially the same algorithmic form. So, for example, if $q=2$ the vector $x(n)\times U(n-p+1:n-p/2, n)$ could be shipped "directly" to node $P(n-p/2)$.

Generalizing this notion, in the first cycle node $P(n)$ computes the vector $x(n)\times U(n-p+1:n, n)$ in q packets, each of size p/q. The first packet (*SUM*) is sent to node $P(n-1)$, the second packet is sent to node $P(n-p/q)$, the third packet is sent to node $P(n-2p/q)$, and so on. Figure 1.1 shows the communication pattern for $p=16$ and $q=4$. For simplicity we also assume that $n=16$.

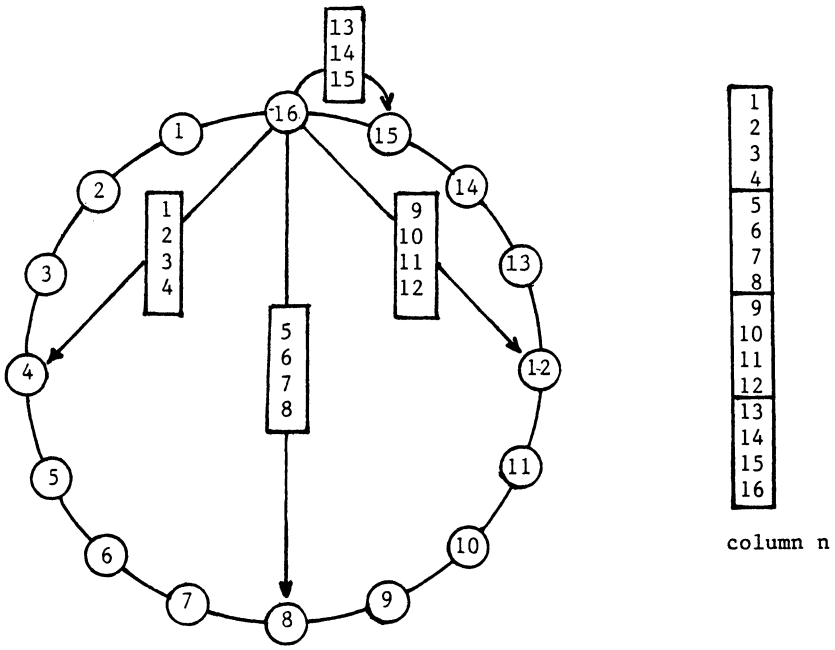


FIG. 1.1

Of course a similar communication pattern can be repeated at all nodes. Therefore, in general during a given cycle, a node $P(j)$ will receive q packets, each of size p/q, before solving for the next variable and then successively computing and sending off q packets, each of size p/q (actually, one of the packets, *SUM*, is of size $(p/q-1)$. After this node $P(j)$ will then modify the remaining $j-p$ components. Figure 1.2 illustrates the incoming commmunication pattern of node $P(3)$, for $p=16$ and $q=4$.

1.2. Description. Algorithm PCTS$^+$ is a generalization of PCTS; when $q=1$ the two algorithms are identical. In general, for $q\geqq 1$, a $p/q-1$ array *SUM* passes around

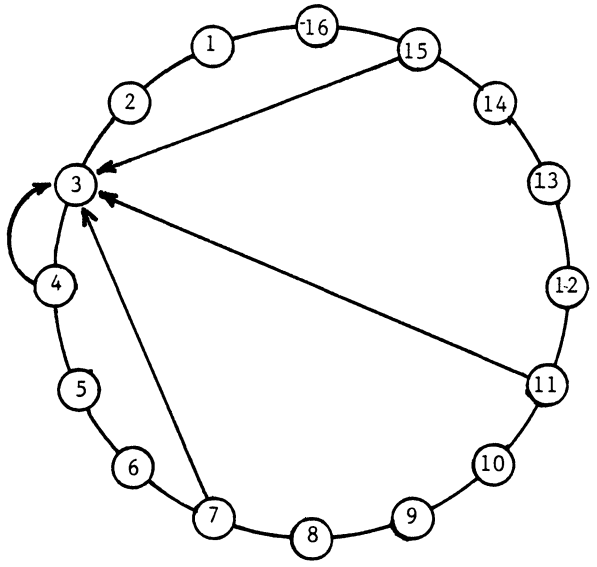


FIG. 1.2

the ring going from $P(j)$ to $P(j-1)$ for $j=n:2$. Before *SUM* arrives, node $P(j)$ receives $q-1$ packets, each of size p/q, from $q-1$ predecessor nodes (node $P(j)$ modifies vector *PSUM* using each packet it receives). When *SUM* arrives at node $P(j)$, $P(j)$ determines $x(j)$, modifies *SUM* using p/q flops, and then forwards *SUM* to node $P(j-1)$. Node $P(j)$ then modifies $p-p/q$ elements of the local array *PSUM* in packets of size p/q; the first packet is sent to node $P(j-p/q)$, the second packet is sent to node $P(j-2p/q)$, and so on. Node $P(j)$ then proceeds to modify the remaining $j-p$ elements of array *PSUM*, using the corresponding elements of column j of U. Node $P(j)$ is then ready to begin again, this time with respect to $x(j-p)$.

We assume p/q is an integer and define $\bar{p}=p/q$. The array w identifies $q-1$ of the recipients of the $\bar{p}$-packets. So for example, let k be the lowest numbered column on a node. Then $w(q-1)=P(k-\bar{p})$, $w(q-2)=P(k-2\bar{p})$ and so forth.

Procedure PCTS$^+$ $(x[1:m], SUM(1:p), PSUM(1:n-p), U(1:n,[1:m]),$
$BUF(1:\bar{p}), w(1:q-1))$

For $j=n:1$
If myname $=P(j)$
For $i=1:q-1$
If $j \leqq n-i\times\bar{p}$
Receive $BUF(1:\bar{p})$
$PSUM(j+\bar{p}+1:j) \leftarrow PSUM(j-\bar{p}+1:j)+BUF(1:\bar{p})$
Receive $SUM(1:\bar{p}-1)$ [if $j<n$]
$x(j) \leftarrow (SUM(1)+PSUM(j))/U(j,j)$
$SUM(1:\bar{p}-2) \leftarrow SUM(2:\bar{p}-1)-U(j-1:j-(\bar{p}-2),j)\times x(j)$
$+PSUM(j-1:j-(\bar{p}-2))$
$SUM(\bar{p}-1) \leftarrow -U(j-(\bar{p}-1),j)\times x(j)+PSUM(j-(\bar{p}-1))$

Send $SUM(1:\bar{p}-1)$ to node $P(j-1)$ [if $j>1$]
For $i=q-1:1$
  If $j>(q-i)\times\bar{p}$
    $PSUM(j-(q-i+1)\times\bar{p}+1:j-(q-i)\times\bar{p})$
      $\leftarrow PSUM(j-(q-i+1)\times\bar{p}+1:j-(q-i)\times\bar{p})$
          $-U(j-(q-i+1)\times\bar{p}+1:j-(q-i)\times\bar{p},j)\times x(j)$
    Send $PSUM(j-(q-i+1)\times\bar{p}+1:j-(q-i)\times\bar{p})$ to node $w(i)$
  $PSUM(1:j-p)\leftarrow PSUM(1:j-p)-U(1:j-p,j)\times x(j)$
End

Most architectures will not guarantee a direct link between sender and receiver of the $\bar{p}$-packets. In our implementation on a hypercube computer, we make no attempt to optimize the route—we let the cube-operating system take care of it (hopefully, some attempt is made to use idle nodes).

Obviously the choice of q plays an important role. As we mentioned before, a reasonable guiding principle is to choose q as large as possible subject to the constraint that when *SUM* arrives the receiving node is ready to process *SUM*. This notion can be formalized (under some simplifying assumptions on message traffic) and an optimal q can be determined analytically. We discuss this in § 2. Next, we present numerical results using different values of q.

1.3. Numerical results. Our numerical results were obtained using Intel iPSC hypercube computers. Experiments were performed using RM/Fortran, in double precision, under release 3.0 of the operating system. Experiments with $p>16$ (i.e., $p=64$, $p=128$) were performed at Intel Scientific Computers, Beaverton, Oregon with the help of Cleve Moler. The largest linear system we could solve in this environment was approximately $n=1700$. The $p=16$ experiments were performed using the Intel iPSC housed in the Advanced Computing Facility at the Cornell Center for Theory and Simulation in Science and Engineering. This cube is outfitted with extra memory boards allowing systems of approximate order $n=2000$ to be solved. Our test problems consisted of randomly generated linear systems.

In the graph in Fig. 1.3 we plot execution time (y-axis) versus various d values (x-axis), where $q=2^d$. The test problem is of size $n=1000$ and $p=128$.

Obviously the optimal choice for q in this case is $q=2^3=8$. Notice that the difference between $q=8$ and $q=1$ is greater than a factor of 4.

As we show in § 2, the optimal choice of q is dependent on the problem size. Nevertheless, Fig. 1.4 illustrates that this dependency is relatively mild in this environment. In particular, $q=8$ is almost uniformly superior for all $n\leqq 1500$ when $p=128$.

An important thing to notice is that $q=1$ and $q=8$ curves are diverging with n and therefore the significance of q increases with n (up to a point). When $n=1500$ the $q=8$ algorithm is approximately a factor 5 better than $q=1$. Notice that for the $q=1,2,4,8$ cases, $T(n)$ is clearly linear; for $q=16,32$ the plot is not quite so true; we explain this phenomenon in § 2.1.

Interestingly, Theorem 2.1 in Li and Coleman [1988], suggests that if n is below the threshold value (demarking the linear and quadratic regions) for a given p, then increasing the number of nodes p, while keeping n fixed, will cause the execution time to increase. It turns out that this is true in practice as well as theory. However, the introduction of q allows for some mitigation of this effect. Indeed, Theorem 2.3 in the next section indicates that if n is held constant and p/q is fixed, then we can expect the execution time to be constant with p. Figure 1.5 illustrates this remark.

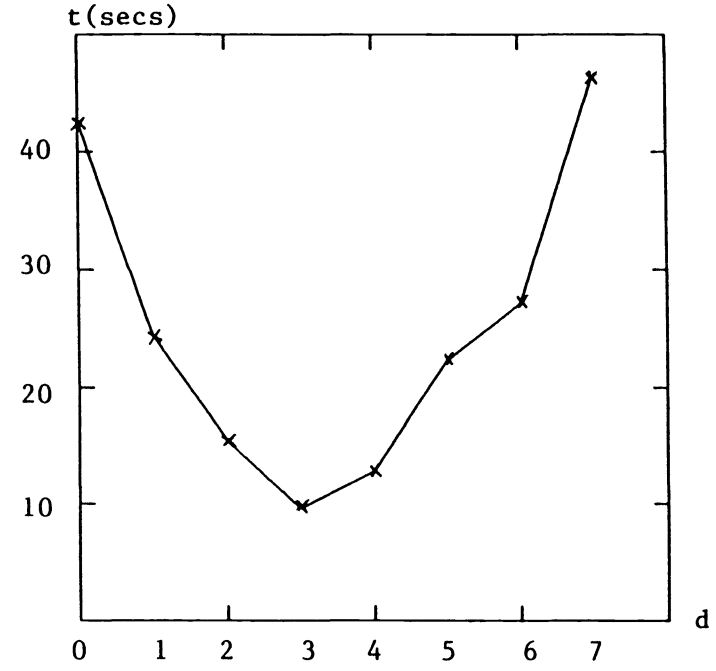


FIG. 1.3. ($n = 1000$, $p = 128$, $q = 2^d$).

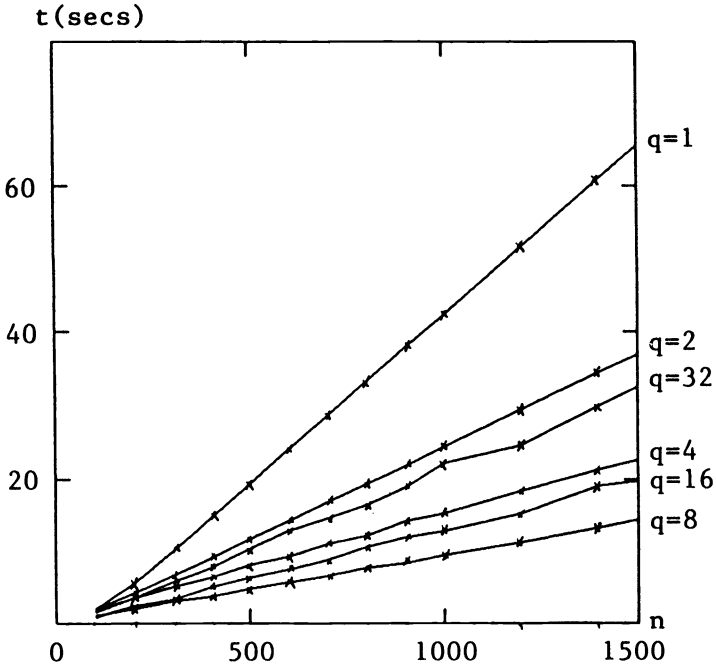


FIG. 1.4. ($p = 128$).

In Fig. 1.5 n is held constant at $n=600$ and p is varied. In the first case ($q=1$) the increase in computation time with p is evident. In the second case, q is varied always maintaining $p/q=16$. In this case there is virtually no increase in computation time.

2. Analysis. The purpose of this section is to derive an expression $T(n)$ for the running time of algorithm PCTS$^+$, measured in flops. The proofs are given in the Appendix. This can easily be done, under a simplifying assumption, and the resulting formula can be used to guide the choice of the parameter q.

One flop reflects the time required to execute the operation $y \leftarrow y + ax$, where a, x, y are scalars. We apply this definition a bit loosely, for simplicity. For example, when the following statement in PCTS$^+$ is executed,

$$PSUM(j-\bar{p}+1:j) \leftarrow PSUM(j-\bar{p}+1:j) + BUF(1:\bar{p})$$

we count one flop per assignment even though there is no multiplication. Similarly, we count just one flop each pass through the loop

$$SUM(1:\bar{p}-2) \leftarrow SUM(2:\bar{p}-1) - U(j-1:j-(\bar{p}-2),j) \times x(j) + PSUM(j-1:j-(\bar{p}-2))$$

even though there is an extra addition.

In the results to follow, we will assume that a cross-ring message of size $\bar{p} \triangleq p/q$ (a $\bar{p}$-packet in PCTS$^+$) takes time at most rt flops, where $r=\log_2(p)$. (Of course the minimum travel time is just t flops by the definition of t.) Furthermore, we assume that message forwarding accrues no overhead cost: a message is forwarded immediately by an intermediate node at zero cost. Finally, for simplicity we assume that $n=mp$ for some integer m.

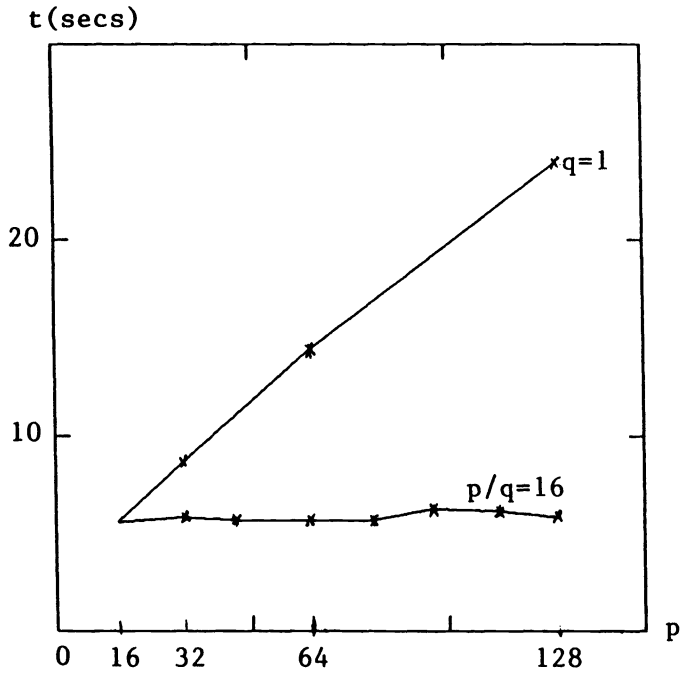


FIG. 1.5. ($n=600$).

It turns out that $T(n)$ is a linear function of n, provided q and n are both less than threshold values. We derive this expression in three stages. First, in Lemma 2.1, we show that if q is sufficiently small, then in the first cycle, *SUM* is processed ($\bar{p}$ flops) immediately upon receipt by each of the nodes $P(n-1), \cdots, P(n-(p-1))$. Second, Lemma 2.2 establishes that in the first cycle node $P(n-p)=P(n)$ is also ready to process *SUM* immediately upon receipt, provided we make the additional assumption that n is less than a threshold value of n, n^*. Finally, in Theorem 2.3 it is shown that *SUM* is always processed immediately upon receipt by *every* node in *every* cycle; the expression for T is then easily deduced.

The following reasoning leads to an intelligent upper bound on q, q^*. Upon receipt of *SUM* a node $P(k)$ updates *SUM* in $\bar{p}$ flops and then forwards *SUM* to a neighboring node on the ring. Immediately after this, $\bar{p}$ elements of *PSUM* are modified by $P(k)$ and then this $\bar{p}$-packet is forwarded to node $P(k-\bar{p})$. Now, when the cycling vector *SUM* arrives at node $P(k-\bar{p})$ it is important that $P(k-\bar{p})$ be ready to process *SUM*. An obvious necessary condition for this is that the cross-ring $\bar{p}$-packet has been received and processed by $P(k-\bar{p})$ before *SUM* arrives. This consideration immediately leads to the inequality

$$2\bar{p}+rt \leqq \bar{p}t+(\bar{p}-1)\bar{p},$$

which yields the bound

$$q \leqq \frac{2p}{(3-t)+\sqrt{(t-3)^2+4rt}}. \tag{2.0}$$

Unfortunately (2.0) does not quite do the job when the last $\bar{p}$ columns of U, columns $\bar{p}, \bar{p}-1, \cdots, 1$, are processed. The reason is that *SUM* can travel more quickly in this final stretch since less processing is required at each node. Specifically, in this last cycle node $P(j)$ processes *SUM* in just j flops instead of the usual $\bar{p}$. Hence this final stretch of $\bar{p}$ nodes leads to the inequality

$$2\bar{p}+rt \leqq \bar{p}t+\frac{\bar{p}(\bar{p}+1)}{2}-1,$$

which implies

$$q \leqq \frac{2p}{(3-2t)+\sqrt{(2t-3)^2+8(rt+1)}}. \tag{2.0'}$$

It is easy to see that (2.0')$\Rightarrow$(2.0) whenever $q \leqq p/2$, and since this latter condition is highly reasonable, we define the upper bound for q,

$$q^* = \min\left\{\frac{p}{2}, \frac{2p}{(3-2t)+\sqrt{(2t-3)^2+8(rt+1)}}\right\}.$$

LEMMA 2.1. *If $q \leqq q^*$ then, in the first cycle, SUM travels*

$$P(n) \to P(n-1) \to \cdots \to P(n-(p-1))$$

in time $p\bar{p}+(p-1)t$ flops (i.e., without delay).

Node $P(n-p)$ will be ready for *SUM* in the first cycle, provided n is less than a threshold value n^*:

$$n^* = p(t+\bar{p}-1)+\bar{p}.$$

Note that n^* depends on q; $n^*(q)$ is monotonically decreasing as q increases.

LEMMA 2.2. *If $q \leqq q^*$ and $n \leqq n^*(q)$ then, in the first cycle, node $P(n)$ is ready to process SUM upon arrival (at time $p(t+\bar{p})$).*

THEOREM 2.3. *If $q \leqq q^*$ and $n \leqq n^*(q)$ then the total running time of* PCTS$^+$ *satisfies*

$$T = (t+\bar{p})n - \frac{\bar{p}(\bar{p}-1)}{2} - t.$$

2.1. Remarks on Theorem 2.3. The main practical importance of Theorem 2.3 is that it can be used to guide the choice of q. Specifically, T decreases as q increases (for fixed p, n, t) and therefore the optimal value of q, q_* is attained by increasing q until either $n = n^*(q)$ or $q = q^*$, whichever comes first. Therefore q_* is given by

$$q_* = \max\left\{1, \min\left\{q^*, \left[\frac{p(p+1)}{n-p(t-1)}\right]\right\}\right\}.$$

Let us now consider how this theoretical estimate matches our computational experience reported in § 1. We have estimated $t \cong 40$ on the Intel iPSC (release (3.0)). Using this estimate and $p = 128$ we obtain $q^* = 19$. Hence if $n = 1000$ then $q_* = q^* = 19$. Considering the results given in Figs. 1.3 and 1.4, we see that this estimate is a bit high: i.e., the numerical results clearly indicate that $q = 8$ is superior to $q = 16$. Indeed, this overestimation is typical and can be explained, we think, by considering our assumptions. Specifically, in order to make the analysis tractable, we have assumed that the forwarding of cross-ring messages occurs at no cost: i.e., there is no interruption in the computational work performed by the forwarding nodes. However, as q increases this assumption becomes increasingly questionable since, in truth, forwarding nodes will be interrupted. Hence it is best to regard the theoretical value of q_* as an upper bound on q. Indeed, it is the violation of this assumption as q increases that accounts for the deviation from linearity exhibited by the $q = 16, 32$ curves in Fig. 1.4.

If $p = 16$ then $q^* = 4.6$ and $q_* = q^* = 4.6$, provided $n \leqq 900$. This theoretical estimate is a bit high once again, since, experimentally, we have found that the best choice for q with n in this range is $q = 2$. If $p = 64$ then $q_* = q^* = 11$ for $n \leqq 3000$. However, experimentally we obtain the results reported in Fig. 2.1. Clearly the experimental best choice for q is $q = 4$ with $q = 8$ a very close second.

It is interesting to note that the definition of the upper bound q^* does not depend on n. Hence q_* will be invariant with respect to n for some range of values of n. For example, considering the $p = 128$ example mentioned above, we will have $q_* = 19$ for all $n \leqq 5861$ (on the Intel iPSC without extra memory, this figure greatly exceeds the storage capacity). This is of considerable practical significance because it suggests that using a fixed value of q may be a reasonable thing to do despite the apparent dependence of q_* on n. Hence, in practice, it may not be necessary to compute q_* for every problem in which n changes; rather, a good value of q can be chosen given p and t. (Of course, as we have mentioned in the preceding paragraph, this choice should probably be somewhat less than the theoretical value q_*.)

Theorem 2.3 is very much related to Theorem 2.1 in Li and Coleman [1988]. Indeed, if $q = 1$ then Theorem 2.3 reduces to precisely the first half of the latter result. Theorem 2.3 says nothing about the case when $n > n^*(q)$. The reason is that Theorem 2.1 (Li and Coleman [1986]) covers the interesting ground. In particular, if $n > n^*(q = 1)$ then this theorem says that T is a quadratic function of n:

$$T = \frac{1}{2}\left\{\frac{n^2}{p} + n + p(t+p)^2 - pt - p^2 + p\right\} - t.$$

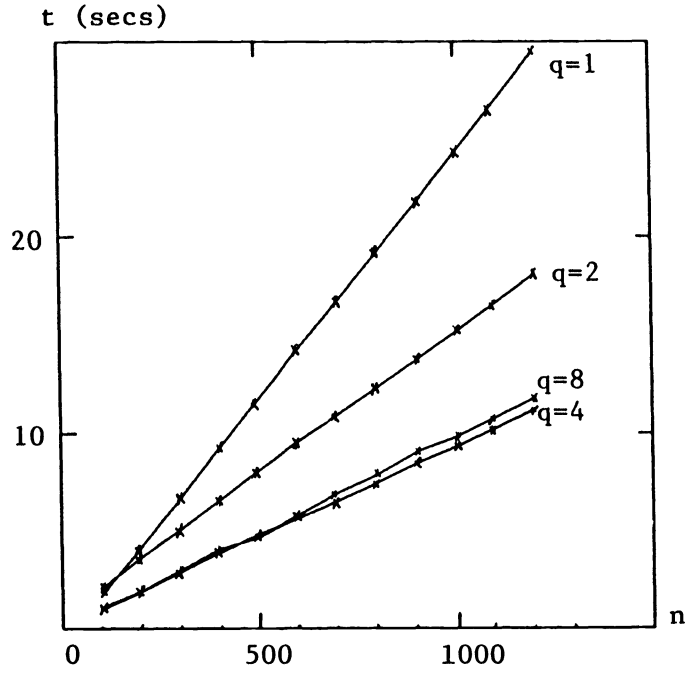


FIG. 2.1. ($p = 64$).

The case where $n \leqq n^*(q=1)$ but $n > n^*(q>1)$ is not really of interest: the practical choice is $q=1$ in this case.

It is our belief that the discrepancy between the theoretical value q_* and the observed best value of q is mainly caused by the assumption that there is no interruption in the computational work performed by the forwarding nodes. Another contributing factor is that the actual route of a cross-ring message is not known a priori; hence, a bound is used in the analysis. Note that the first difficulty evaporates on a multiprocessor with message-passing coprocessors (e.g., iPSC/2). We have not yet had the opportunity to experiment on such a system; however, we expect q_* to predict more accurately the observed performance in this environment (it remains to be seen). Determining the best q in practice on a multiprocessor without message-passing coprocessors is probably best done experimentally; the theoretical value q_* can be used as a bound. In addition, as suggested by a referee, a spline analysis of $T = T(n, p, q)$ can be used in conjunction with observed results.

3. A rectangular triangular solver. In this section we consider an improvement to algorithm PCTS$^+$; however, since our numerical experiments show that the new algorithm, RPCTS$^+$, offers only a modest gain in efficiency, we will be quite brief in our presentation. Nevertheless, there is no question that RPCTS$^+$ is never worse and is sometimes better than PCTS$^+$; the improvement may be significant in some environments.

The basic idea behind RPCTS$^+$ is based on the observation that columns of U diminish in size from right to left: therefore, the workload endured by PCTS$^+$ is not evenly balanced. Moreover, it is quite possible to postpone some of the work on the larger columns until later and thereby achieve a rectangular work distribution (instead of triangular). In particular, in every cycle of RPCTS$^+$ each node processes a "column" of U of about size $n/2$.

The rectangular triangular solver follows. The parameter h is usually chosen to be approximately equal to $n/2$. Indeed this is the choice that leads to the rectangular work distribution and is always our choice in our reprorted experiments.

Procedure RPCTS$^+$ ($x[1:m]$, $SUM(1:p)$, $PSUM(1:n-p)$,
$U(1:n,[1:m])$, $BUF(1:\bar{p})$, $w(1:q-1)$)

$l = m$
For $j = n:1$
  If myname $= P(j)$
    For $i = 1:q-1$
      If $j \leqq n - i \times \bar{p}$
        Receive $BUF(1:\bar{p})$
        $PSUM(j-\bar{p}+1:j) \leftarrow PSUM(j-\bar{p}+1:j) + BUF(1:\bar{p})$
    Receive $SUM(1:p-1)$ [if $j < n$]
    $x(j) \rightarrow (SUM(1) + PSUM(j))/U(j,j)$
    $SUM(1:2) \leftarrow SUM(2:\bar{p}-1) - U(j-1:j-(\bar{p}-2),j) \times x(j)$
      $+ PSUM(j-1:j-(\bar{p}-2))$
    $SUM(\bar{p}-1) \leftarrow -U(j-(\bar{p}-1),j) \times x(j) + PSUM(j-(\bar{p}-1))$
    Send $SUM(1:\bar{p}-1)$ to node $P(j-1)$ [if $j > 1$]
    For $i = q-1:1$
      If $j > (q-i) \times \bar{p}$
        $PSUM(j-(q-i+1) \times \bar{p}+1:j-(q-i) \times \bar{p})$
          $\leftarrow PSUM(j-(q-i+1) \times \bar{p}+1:j-(q-i) \times \bar{p})$
            $- U(j-(q-i+1) \times \bar{p}+1:j-(q-i) \times \bar{p}, j) \times x(j)$
        Send $PSUM(j-(q-i+1) \times \bar{p}+1:j-(q-i) \times \bar{p})$ to node $w(i)$
  $PSUM(j-h+1:j-p) \leftarrow PSUM(j-h+1:j-p)$
    $- U(j-h+1:j-p,j) \times x(j)$
  If $j \leqq n-h+p$
    for $i = m:l$
      $PSUM(j-2p+1:j-p) \rightarrow PSUM(j-2p+1:j-p)$
        $- U(j-2p+1:j-p, i) \times x(i)$
    $l = l-1$

End

It is important to realize that RPCTS$^+$ can potentially improve on PCTS$^+$ only when $n > n^*(q)$; otherwise, the algorithms have exactly the same running time for all feasible values of q. However, it is possible to extend the linear region beyond $n^*(q)$ with RPCTS$^+$. The reason for this is simply that the definition of the linear region is driven by the size of the largest column processed—the "column" sizes used by algorithm RPCTS$^+$ are all roughly $n/2$, which contrasts with a maximum column size of n in the case of PCTS$^+$. The following theorem formalizes this notion: notice that the bound on n is roughly twice $n^*(q)$; the expression for T is identical to that for PCTS$^+$. (We omit the proof, and it is very similar to the proof of Theorem 2.3.)

THEOREM 3.1. *If* $q \leqq q^*$ *and* $n \leqq 2p(t+\bar{p}-2)$, *then*

$$T = (t+\bar{p})n - \frac{\bar{p}(\bar{p}+1)}{2} - p - t. \qquad \square$$

3.1. Numerical results. In Fig. 3.1 we compare RPCTS$^+$ with PCTS$^+$ for $p = 16$ and $q = 1$. Note that the graphs are essentially indistinguishable up until $n \cong 1000 \cong n^*(q=1)$. (Theoretically this breakpoint occurs at $n = 900$.) At this point RPCTS$^+$ continues in a linear fashion while PCTS$^+$ begins its quadratic phase.

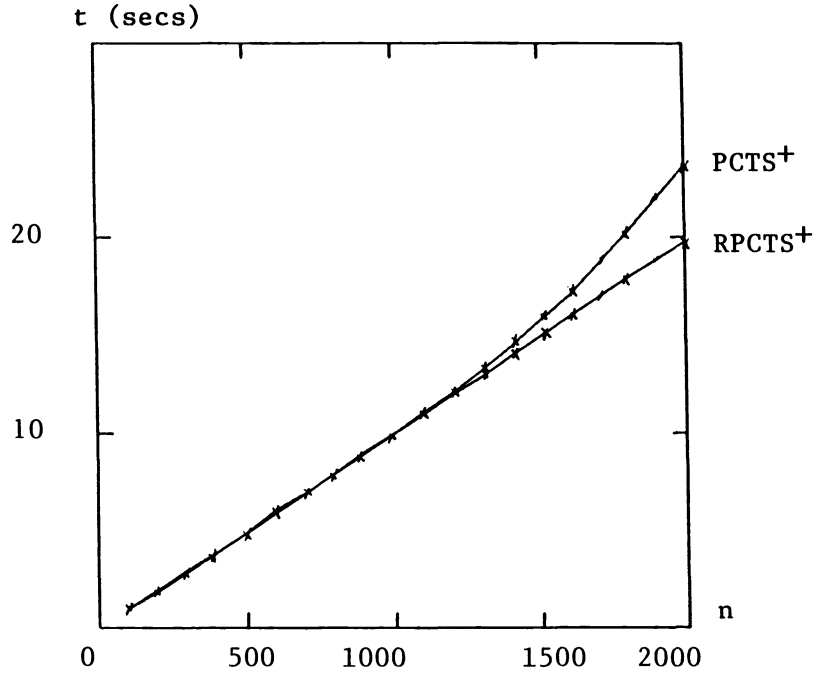


FIG. 3.1. ($p = 16$, $q = 1$).

4. Summary and conclusions. We have presented and analyzed a generalization of the Li-Coleman [1988] algorithm for solving triangular systems of equations on a multiprocessor. We have assumed that the columns of the matrix are distributed to the nodes in a wrap fashion. We note in passing that a similar algorithm can be constructed for the row-distributed case. This new solver is effective even when n/p is modest, whereas the original solver degrades in performance as n/p decreases. The new solver is applicable on a distributed-memory multiprocessor that allows for a ring embedding; however, it is most reasonable when there is additional inter-processor connectivity, beyond that of a ring. The exact nature of this connectivity is unimportant though our experiments have been restricted to a hypercube multiprocessor.

Under a slightly unrealistic assumption on cross-ring traffic (i.e., nodes forward messages at no cost) the proposed method is analytically tractable. This analysis reveals that up to threshold values of n and parameter q, the running time is a linear function of n. More importantly, the analysis yields a theoretically optimal choice for the parameter q which, in practice, serves as a very useful upper bound on q.

Finally, we note that when n/p is sufficiently large then the new algorithm reduces to the original Li-Coleman [1988] algorithm, which is quite effective in such circumstances.

Acknowledgments. We are especially grateful to Richard Chamberlain who suggested to us that the original Li-Coleman algorithm could be improved in a manner similar to the procedure described here. We thank Intel Scientific Computers of Beaverton, Oregon, and Cleve Moler in particular, for allowing us to visit and conduct many of our numerical experiments there. Finally, the balance of our numerical experimentation was performed with the assistance of the Advanced Computing Facility at the Cornell Center for Theory and Simulation in Science and Engineering, which is supported by the National Science Foundation and New York State.

Appendix. In this section we provide the proofs of the results in § 2.

Proof of Lemma 2.1. We will show that when *SUM* arrives at node $P(n-k)$, $1 \leqq k \leqq p-1$, the receiving node is ready to process *SUM*. This will be the case if node $P(n-k)$ has completed processing the required $\lfloor k/\bar{p} \rfloor$ cross-ring $\bar{p}$-packets that it will receive in the first cycle. In particular, we will show that the time between two $\bar{p}$-packet arrivals is always enough to allow the first to be processed before the second arrives. (This is true even when the second $\bar{p}$-packet is *SUM* itself.) Since processing the cross-ring $\bar{p}$-packets is the only work that node $P(n-k)$ must do before *SUM* arrives, our desired result will follow.

Assume that $\bar{p}$-packet i (say) is sent to node $P(n-k)$ at time t_i. Then, it will arrive at $P(n-k)$ by time $t_i + rt$. But $\bar{p}$-packet $i+1$ is not ready to be processed by node $P(n-k)$ until at least time $t_i + \bar{p}(\bar{p}+t) - 2\bar{p}$. (Note that $\bar{p}$-packet $i+1$ may refer to *SUM*.) Therefore, if

$$t_i + rt + \bar{p} \leqq t_i + \bar{p}(\bar{p}+t) - 2\bar{p}, \tag{2.1}$$

then it follows that node $P(n-k)$ is ready to process $\bar{p}$-packet $i+1$ upon arrival. But (2.1) is clearly implied by the assumed bound on q and therefore the result follows. □

Proof of Lemma 2.2. By Lemma 2.1, *SUM* arrives at node $P(n)$ at time $p(\bar{p}+t)$. We must now argue that node $P(n)$ has enough time to process column n of U, as well as the $(q-1)$ cross-ring $\bar{p}$-packets that arrive in staggered fashion, before *SUM* arrives. But, by algorithm PCTS$^+$, node $P(n)$ processes column n first before doing the $(q-1)$ $\bar{p}$-packet updates. Since it is clear that the last $\bar{p}$-packet, arriving from node $P(n-p)$, can be processed by $P(n)$ before *SUM* arrives (provided $P(n)$ is not otherwise busy), we can, without loss of generality, count backward. That is, assume that $P(n)$ processes this last $\bar{p}$-packet during time $[p(\bar{p}+t) - \bar{p}(\bar{p}-1), p(\bar{p}+t)]$. Similarly, we can assume that $P(n)$ processes the second to last $\bar{p}$-packet during time $[p(\bar{p}+t) - (2\bar{p}-1), p(\bar{p}+t)]$. (By definition of q^*, it is clear that $P(n)$ has processed this packet by time $p(\bar{p}+t) - (2\bar{p}-1)$.) Continuing in this fashion, assume, without loss of generality, that the interval $[p(\bar{p}+t) - ((q-1)\bar{p}-1), p(\bar{p}+t)]$ is used to process all $(q-1)$ cross-ring $\bar{p}$-packets. Hence we must only show that $P(n)$ has finished processing column n by time $p(\bar{p}+t) - (q-1)\bar{p} + 1$. But node $P(n)$ is finished processing column n of U at time n and with $n \leqq n^*(q)$ we obtain

$$n \leqq p(\bar{p}+t) - ((q-1)\bar{p}-1). \qquad \square$$

Proof of Theorem 2.3. By Lemmas 2.1 and 2.2, *SUM* is processed immediately upon receipt by nodes

$$P(n-1), P(n-2), \cdots, P(n-p) = P(n)$$

in the first cycle. But if node $P(n-p)$ is free when *SUM* arrives in the first cycle, then it is clear that every node will be free when *SUM* arrives in subsequent cycles (except possibly when the last $\bar{p}$ columns, column $\bar{p}, \cdots$, column 1, are being processed). The reason is simple: such nodes are in the same situation as node $P(n)$ is in the first cycle. In particular, each such node must process its current column of U and then the $(q-1)$ arriving $\bar{p}$-packets before *SUM* returns. But since the size of the columns of U is diminishing, and since $P(n)$ had enough time in the first cycle (by Lemma 2.2), it follows that each node will have sufficient time to do this.

However, it is necessary to consider the last $\bar{p}$ columns (i.e., columns $\bar{p}, \bar{p}-1, \cdots, 1$) separately because in this stretch the nodes $P(\bar{p}), \cdots, P(1)$ do less work in processing *SUM*. In particular, in this final stretch *SUM* requires i flops at node $P(i)$, $1 \leqq i \leqq \bar{p}$. Hence it is conceivable that *SUM* arrives at node $P(j)$, say, with $1 \leqq j \leqq \bar{p}$,

before $P(j)$ is ready. But it is easy to see that this cannot be the case: the bound on q, q^*, ensures that each node $P(j)$ has enough time to process the $\bar{p}$-packet from $P(j+p)$ before *SUM* arrives. Hence it follows that *SUM* is processed immediately on arrival at all nodes during every cycle.

We are now ready to compute T and thus prove the theorem. The total running time T is just the time for *SUM* to cycle around the ring m times. Since every node is always ready to process *SUM* upon arrival, it is straightforward to compute this cycling time. With respect to each column j, $\bar{p} < j \leqq n$, we will associate the time $\bar{p}+t$, which represents the time required to process *SUM* plus the time for *SUM* to travel to the next node on the ring. Similarly, for $2 \leqq j \leqq \bar{p}$ column j is charged $j+t$ flops; column 1 is charged just one flop. Therefore, the total time T is just the sum of the charges:

$$T = (n-\bar{p})(\bar{p}+t)+(\bar{p}-1)t+\sum_{i=1}^{\bar{p}} j,$$

which yields the results. □

REFERENCES

R. M. CHAMBERLAIN [1986], *An algorithm for* LU *factorization with partial pivoting on the hypercube*, Tech. Report CCS 86/11, Chr. Michelsen Institute, Bergen, Norway.

S. C. EISENSTAT, M. T. HEATH, C. S. HENKEL, AND CHARLES H. ROMINE [1988], *Modified cyclic algorithms for solving triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9, pp. 558–588.

G. A. GEIST AND M. T. HEATH [1986], *Matrix factorization on a hypercube multiprocessor*, in Hypercube Multiprocessors 1986, M. Heath ed., Society for Industrial and Applied Mathematics, Philadelphia, PA.

M. T. HEATH AND CHARLES H. ROMINE [1988], *Parallel solution of triangular systems on distributed-memory multiprocessors*, SIAM J. Sci. Statist. Comput., 9, pp. 558–588.

G. LI AND T. COLEMAN [1988], *A parallel triangular solver for a hypercube multiprocessor*, SIAM J. Sci. Statist. Comput., 9, pp. 485–502.

C. MOLER [1986a], Private communication.

——— [1986b], *Numerical comparisons of triangular solvers on the* Intel iPSC, presented at the Second Conference on Hypercube Multiprocessors, Knoxville, TN, September 29–October 1.

——— [1986c], *Matrix computation on distributed memory multiprocessors*, Tech. Report, Intel Scientific Computers.

C. H. ROMINE AND J. M. ORTEGA [1986], *Parallel solution of triangular systems of equations*, Tech. Report RM-86-05, Department of Applied Mathematics, University of Virginia, Knoxville, VA.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 2, pp. 397-406, March 1989
© 1989 Society for Industrial and Applied Mathematics
011

PIECEWISE MONOTONE QUADRATIC HISTOSPLINES*

ROSSANA MORANDI† AND PAOLO COSTANTINI‡

Abstract. A method is presented for constructing a quadratic spline function satisfying area-matching conditions and local monotonicity constraints, according to the frequencies on the class intervals and to the shape of a given histogram. Such a function is "as close as possible" to the quadratic spline that satisfies the area-matching conditions and the minimum curvature property and generally exhibits a visually pleasing graph.

Key words. histosplines, area-matching conditions, co-monotonicity, algorithms

AMS(MOS) subject classifications. 65D07, 65D10

1. Introduction. Let $H=\{h_i\}_{i=1}^N$ be a histogram, where h_i is the frequency for the class interval (x_{i-1}, x_i), $i=1, 2, \cdots, N$ and let $S_2^1(\Delta)$ denote the set of C^1 quadratic splines defined on a suitable partition Δ of $[x_0, x_N]$. We are interested in the construction of shape-preserving quadratic histosplines, that is, splines $s \in S_2^1(\Delta)$ satisfying the area-matching conditions $\int_{x_{i-1}}^{x_i} s(x)\,dx = h_i$, $i=1, 2, \cdots, N$, and that are subject to monotonicity constraints, according to the shape of the histogram.

The use of splines for the reconstruction of the unknown density function was first introduced in [1] and then studied in [10]. In both papers splines are computed by solving an associated linear system, and the resulting curves assume, in some cases, negative values and extraneous oscillations.

Although the problems concerning nonnegative and monotonicity-preserving histosplines arise in a natural way, the only results in this direction seem to be those proposed in [9, § 4], where sufficient conditions for the existence of increasing histosplines are given.

Here we use the general approach for locally constrained splines proposed in [3], giving a condition for the existence of piecewise monotone quadratic histosplines, and an algorithm for their construction.

It is worthwhile to note that, in this paper, splines are presented from the point of view of curve fitting, and that there are no statistical considerations, since, to quote Schoenberg " . . . statisticians must decide on the usefulness of new tools of approximation" [10, p. 278].

2. Piecewise monotone quadratic histosplines. Let $H=\{h_i\}_{i=1}^N$ be a histogram, with frequencies $h_i \in \mathbb{R}$, $i=1, 2, \cdots, N$, class intervals (x_{i-1}, x_i), $i=1, 2, \cdots, N$, and let

$$a_i = h_i/(x_i - x_{i-1}), \qquad i=1, 2, \cdots, N$$

be the height of the ith cell. Setting

$$\delta_i = (x_i - x_{i-1})/3, \qquad i=1, 2, \cdots, N$$

* Received by the editors March 19, 1986; accepted for publication (in revised form) April 6, 1988.

† Università di Firenze, Istituto Matematico "U. Dini," Viale Morgagni, 67/A, 50134 Firenze, Italy.

‡ Università di Firenze, Dipartimento di Energetica, Via di S. Marta, 3, 50139 Firenze, Italy. Present address, Università di Siena, Dipartimento di Matematica, Via del Capitano, 15, 53100 Siena, Italy.

we let $\Delta = \{t_0 < t_1 < \cdots < t_{3N}\}$ be a partition of the interval $[x_0, x_N]$, where

$$t_{3i} = x_i, \quad t_{3i+1} = x_i + \delta_{i+1}, \quad t_{3i+2} = x_i + 2\delta_{i+1}, \quad i = 0, 1, \cdots, N-1,$$
$$t_{3N} = x_N$$

and

$$S_2^1(\Delta) = \{s \in C^1[t_0, t_{3N}]: s(x) \in P_2, x \in [t_j, t_{j+1}], j = 0, 1, \cdots, 3N-1\}$$

denote the linear space of quadratic splines on Δ. We are interested in the construction of quadratic histosplines, that is $s \in S_2^1(\Delta)$ satisfying the area-matching conditions

$$\int_{x_{i-1}}^{x_i} s(x)\, dx = h_i, \qquad i = 1, 2, \cdots, N. \tag{2.1}$$

First, we observe that, setting

$$s(t_j) = \eta_j, \quad s'(t_j) = \mu_j, \quad j = 0, 1, \cdots, 3N$$

we have

$$s(x) = p_j(x) = \eta_j + \mu_j(x - t_j) + \frac{1}{2}\frac{\mu_{j+1} - \mu_j}{t_{j+1} - t_j}(x - t_j)^2,$$
$$x \in [t_j, t_{j+1}], \qquad j = 0, 1, \cdots, 3N-1, \tag{2.2}$$

and in each class interval (x_{i-1}, x_i), any $s \in S_2^1(\Delta)$ is defined by the parameters

$$\eta_j, \mu_j, \qquad j = 3i-3, \cdots, 3i,$$

which satisfy the continuity conditions

$$p_j(t_{j+1}) = \eta_{j+1}, \qquad p_j'(t_{j+1}) = \mu_{j+1}, \qquad j = 3i-3, 3i-2, 3i-1.$$

If we impose (2.1) we can express $s(x)$, $x \in [x_{i-1}, x_i]$, using the four parameters η_{3i-3}, μ_{3i-3}, η_{3i}, μ_{3i}. Setting, for notational convenience, $y_i = \eta_{3i}$, $m_i = \mu_{3i}$, $i = 0, 1, \cdots, N$, we have that any $s \in S_2^1(\Delta)$ that satisfies (2.1), is given by (2.2), where, for $i = 1, 2, \cdots, N$

$$\mu_{3i-2} = -\frac{5}{6} m_{i-1} + \frac{1}{3} m_i - \frac{2}{\delta_i} y_{i-1} - \frac{1}{\delta_i} y_i + \frac{3}{\delta_i} a_i, \tag{2.3a}$$

$$\mu_{3i-1} = \frac{1}{3} m_{i-1} - \frac{5}{6} m_i + \frac{1}{\delta_i} y_{i-1} + \frac{2}{\delta_i} y_i - \frac{3}{\delta_i} a_i, \tag{2.3b}$$

$$\eta_{3i-2} = y_{i-1} + \frac{1}{2}\delta_i(m_{i-1} + \mu_{3i-2}), \tag{2.3c}$$

$$\eta_{3i-1} = \eta_{3i-2} + \frac{1}{2}\delta_i(\mu_{3i-2} + \mu_{3i-1}). \tag{2.3d}$$

At this point we can focus our attention on the monotonicity constraints. First we introduce the sequence $H' = \{h_i'\}_{i=1}^N$, where

$$h_1' = \begin{cases} 1 & \text{if } a_1 \leqq a_2, \\ -1 & \text{if } a_1 > a_2; \end{cases}$$

$$h_i' = \begin{cases} 1 & \text{if } a_i < a_{i+1} \\ -1 & \text{if } a_i > a_{i+1} \\ h_{i-1}' & \text{if } a_i = a_{i+1} \end{cases} \quad \text{for } i = 2, \cdots, N-1;$$

and

$$h'_N = h'_{N-1},$$

and we say that the histogram is increasing (decreasing) in (x_{i-1}, x_i) if $h'_i = +1$ ($h'_i = -1$), $i = 1, 2, \cdots, N$. To preserve the symmetry, we will assume that H has no isolated extremes, that is the conditions

$$a_{i-1} < a_i > a_{i+1} \quad \text{or} \quad a_{i-1} > a_i < a_{i+1}$$

do not hold, eventually subdividing the isolated extremal cell into two cells of equal heights and half bases.

We want to construct positive piecewise monotone quadratic histosplines, that is, $s \in S_2^1(\Delta)$ which satisfies (2.1) and

$$s(t_{3i-3}) \geqq 0, \quad s(t_{3i}) \geqq 0, \quad h'_i s'(x) \geqq 0, \quad x \in (x_{i-1}, x_i), \quad i = 1, 2, \cdots, N. \tag{2.4}$$

Using (2.2), (2.3) and noting that s is increasing (decreasing) in (x_{i-1}, x_i) if and only if $s'(t_j) \geqq 0 (\leqq 0)$, $j = 3i-3, \cdots, 3i$, we can rewrite the above conditions as follows:

$$\begin{aligned} &y_{i-1} \geqq 0, \quad y_i \geqq 0, \quad h'_i m_{i-1} \geqq 0, \quad h'_i m_i \geqq 0, \\ &h'_i\left(-\frac{5}{6} m_{i-1} + \frac{1}{3} m_i - \frac{2}{\delta_i} y_{i-1} - \frac{1}{\delta_i} y_i + \frac{3}{\delta_i} a_i\right) \geqq 0, \\ &h'_i\left(\frac{1}{3} m_{i-1} - \frac{5}{6} m_i + \frac{1}{\delta_i} y_{i-1} + \frac{2}{\delta_i} y_i - \frac{3}{\delta_i} a_i\right) \geqq 0, \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad i = 1, 2, \cdots, N, \end{aligned} \tag{2.5}$$

and we seek a sequence $\{(y_i, m_i)\}_{i=0}^N$ such that $(y_{i-1}, m_{i-1}, y_i, m_i)$ satisfies (2.5).

It will be convenient to consider separately the two sequences $\{y_i\}_{i=0}^N$ and $\{m_i\}_{i=0}^N$. More specifically, let us suppose that $\{y_i\}_{i=0}^N$, $y_i \geqq 0$, $i = 0, 1, \cdots, N$, is a given sequence of parameters, and from (2.3a), (2.3b) let us construct the functions

$$M_{1,i}(u, v) = -\frac{5}{6} u + \frac{1}{3} v - \frac{2}{\delta_i} y_{i-1} - \frac{1}{\delta_i} y_i + \frac{3}{\delta_i} a_i,$$

$$M_{2,i}(u, v) = \frac{1}{3} u - \frac{5}{6} v + \frac{1}{\delta_i} y_{i-1} + \frac{2}{\delta_i} y_i - \frac{3}{\delta_i} a_i$$

and the family of sets

$$\begin{aligned} DM_i = \{(u, v) \in \mathbb{R}^2 &: h'_i u \geqq 0, h'_i v \geqq 0, h'_i M_{1,i}(u, v) \geqq 0, h'_i M_{2,i}(u, v) \geqq 0\}, \\ &i = 1, 2, \cdots, N. \end{aligned} \tag{2.6}$$

The following result, which is stated in the form of a lemma for notational convenience, follows directly from (2.5).

LEMMA 1. *Let the sequence $\{y_i\}_{i=0}^N$, $y_i \geqq 0$, $i = 0, 1, \cdots, N$ be given. There exist splines $s \in S_2^1(\Delta)$ satisfying* (2.1), (2.4) *if and only if there exist sequences $\{m_i\}_{i=0}^N$ such that $(m_{i-1}, m_i) \in DM_i$, $i = 1, 2, \cdots, N$.*

Some examples of the set DM_i are shown in Fig. 1 for $h'_i = +1$.
We note that the examples for the $h'_i = -1$ case can be obtained by symmetry transformations with respect to the origin of the uv-plane. Obviously, Lemma 1 is nontrivial when $DM_i \neq \varnothing$, $i = 1, 2, \cdots, N$. If we denote with $P_2 = (u_2, v_2)$ (see Fig. 1) the intersection of the straight lines $M_{1,i}(u, v) = 0$ and $M_{2,i}(u, v) = 0$, we have from geometrical considerations that $DM_i \neq \varnothing$ if and only if the point P_2 belongs to the first or to the third

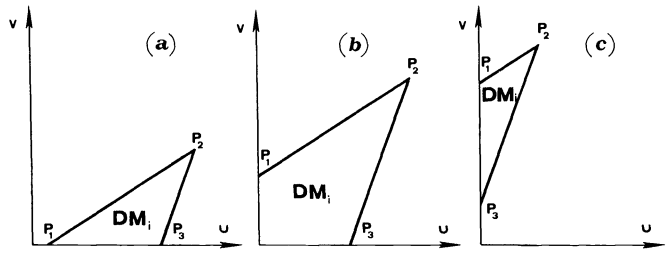


FIG. 1. *Some typical examples of the set* DM_i *when* $h_i = +1$ *for* (a) $(y_{i-1}+2y_i)/3 < a_i < (y_{i-1}+8y_i)/9$; (b) $(2y_{i-1}+y_i)/3 < a_i < (y_{i-1}+2y_i)/3$; (c) $(8y_{i-1}+y_i)/9 < a_i < (2y_{i-1}+y_i)/3$.

quadrant of the uv-plane for $h_i' = +1$ or $h_i' = -1$, respectively. Simple computation leads to

$$u_2 = (-16y_{i-1} - 2y_i + 18a_i)/7\delta_i, \qquad v_2 = (2y_{i-1} + 16y_i - 18a_i)/7\delta_i.$$

Given the functions

$$Y_{1,i}(w, z) = -8w - z + 9a_i, \qquad Y_{2,i}(w, z) = w + 8z - 9a_i$$

and the family of sets

$$DY_i = \{(w, z) \in \mathbb{R}^2 : w \geqq 0;\ z \geqq 0;\ h_i' Y_{1,i}(w, z) \geqq 0,\ h_i' Y_{2,i}(w, z) \geqq 0\}, \quad i = 1, \cdots, N \tag{2.7}$$

we obtain the following lemma.

LEMMA 2. $DM_i \neq \varnothing$, $i = 1, 2, \cdots, N$ *if and only if there exists a sequence* $\{y_i\}_{i=0}^N$, *such that* $(y_{i-1}, y_i) \in DY_i$, $i = 1, 2, \cdots, N$.

The set DY_i is shown in Fig. 2.

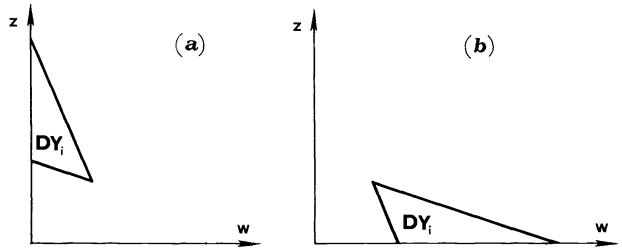


FIG. 2. *Some typical examples of the set* DY_i *for* (a) $h_i' = +1$ *and* (b) $h_i' = -1$.

It is now clear that the existence of positive, piecewise monotone quadratic histosplines is ensured if there exist two sequences of real numbers $\{y_i\}_{i=0}^N$ and $\{m_i\}_{i=0}^N$ that satisfy Lemmas 2 and 1, respectively. We note that the problems of the two lemmas are quite similar, and to solve them it will be convenient to introduce the more general formulation proposed in [3] and [4].

Let $\{D_i\}_{i=1}^N$, $D_i \subseteq \mathbb{R}^2$, $D_i \neq \varnothing$, $i = 1, 2, \cdots, N$ be an arbitrary sequence of sets, and let $\Pi_{a,b}^a : \mathbb{R}^2 \to \mathbb{R}$, $\Pi_{a,b}^b : \mathbb{R}^2 \to \mathbb{R}$ be the (algebraic) projections from the ab-plane onto the a-axis and b-axis, respectively. We give the following algorithm.

ALGORITHM A1.

(1) Let the integer N and the collection $\{D_i\}_{i=1}^N$ be given.

(2) For $i = 0, 1, \cdots, N-1$

1. $B_i = \Pi_{a,b}^a(D_{i+1})$.

(3) $B_N = \mathbb{R}$.
(4) $A_0 = B_0$; $J = N$.
(5) For $i = 1, 2, \cdots, N$
  1. $A_i = \Pi_{a,b}^{b}(\{A_{i-1} \times B_i\} \cap D_i)$.
  2. If $A_i = \varnothing$ set $J = i$ and stop.
     Otherwise continue.
(6) Stop.

We state the following result without proof (see [3, Lemma 4]).

LEMMA 3. *There exists a sequence* $\{d_i\}_{i=0}^{N}$, $d_i \in \mathbb{R}$, $i = 0, 1, \cdots, N$ *such that* $(d_{i-1}, d_i) \in D_i$, $i = 1, 2, \cdots, N$, *if and only if Algorithm* A1 *gives* $J = N$.

Now suppose that Algorithm A1 gives $J = N$ and consider Algorithm A2.

ALGORITHM A2.
(1) Let the integer N and the collections of nonempty sets $\{A_i\}_{i=0}^{N}$, $\{B_i\}_{i=0}^{N}$ be given by Algorithm A1.
(2) Choose any $d_N \in A_N$.
(3) For $i = N-1, N-2, \cdots, 0$
  1. Choose an arbitrary

$$d_i \in \{a \in \mathbb{R}: (a, d_{i+1}) \in \{A_i \times \mathbb{R}\} \cap D_{i+1}\}.$$

(4) Stop.

Then we have Lemma 4 ([3], [4]).

LEMMA 4. *Let the sequence* $\{d_i\}_{i=0}^{N}$ *be computed by Algorithm* A2. *Then* $(d_{i-1}, d_i) \in D_i$, $i = 1, 2, \cdots, N$.

Linking together Lemmas 1-4, we obtain the following existence result.

THEOREM 1. *Suppose that Algorithm* A1, *applied to the sets* (2.7) *gives* $J = N$, *and let* $\{y_i\}_{i=0}^{N}$ *be the corresponding sequence given by Algorithm* A2. *Let the family* $\{DM_i\}_{i=1}^{N}$ *be given by* (2.6). *Suppose that Algorithm* A1, *applied to the sets* (2.6), *gives* $J = N$, *and let* $\{m_i\}_{i=0}^{N}$ *be the corresponding sequence give by Algorithm* A2. *Then the spline* $s \in S_2^1(\Delta)$ *given by* (2.2), (2.3) *satisfies* (2.1), (2.4).

A geometric example for the crucial step (5)1, of Algorithm A1 is shown in Fig. 3, for $h_i' = +1$, and $D_i = DY_i$, $D_i = DM_i$.

It should be noted that, in contrast to Lemmas 1-3, the above theorem provides only a sufficient condition for the existence of positive, piecewise monotone quadratic histosplines. In fact, we consider separately the two sequences $\{y_i\}_{i=0}^{N}$, $\{m_i\}_{i=0}^{N}$ in the sense that: (a) we construct a suitable sequence $\{y_i\}_{i=0}^{N}$ (if one exists) according to

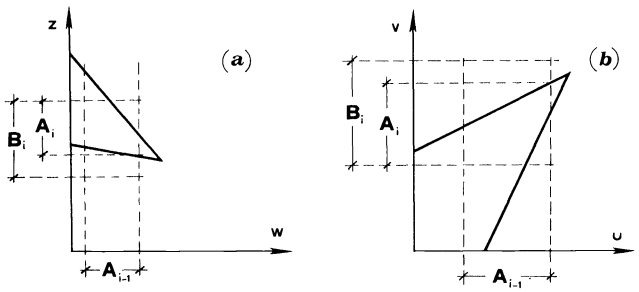


FIG. 3. *A geometric example of step* (5)1 *when Algorithm* A1 *is applied to* (a) *the set* DY_i *of Fig.* 2(a) *and* (b) *the set* DM_i *of Fig.* 1(b).

Lemma 2, and then (b) we determine a sequence $\{m_i\}_{i=0}^N$ (if it exists) according to Lemma 1, for the sets DM_i given in the previous step. In other words, at step (b) we construct a positive, piecewise monotone quadratic histospline that, moreover, satisfies the interpolatory conditions $s(x_i) = y_i$, $i = 0, 1, \cdots, N$. From a theoretical point of view the Algorithms A1–A2 (in the general formulation of [3]) could be applied to the sets of $\mathbb{R}^4$ given by (2.5) for directly obtaining the sequence $\{(y_i, m_i)\}_{i=0}^N$, and an analogue of Theorem 1 in this case would provide a necessary and sufficient condition. Unfortunately, it is very difficult to implement steps (5)1 of Algorithm A1 and (3)1 of Algorithm A2 when the D_i are convex polytopes of $\mathbb{R}^4$, and even if it were not, the computational cost would be too high, especially for application of the algorithms in computer-aided design. It is worthwhile to remark that Algorithm A2 will provide the sequence $\{d_i\}_{i=0}^N$ if Algorithm A1 gives $J = N$. However, let us suppose that Algorithm A1 gives $J = j \neq N$. This means that $A_i \neq \emptyset$, $i = 0, 1, \cdots, j-1$, $A_j = \emptyset$, and Algorithm A2 will provide a sequence $\{d_i'\}_{i=0}^j$ such that $(d_{i-1}', d_i') \in D_i$, $i = 1, \cdots, j$. Now suppose, without loss in generality, that Algorithm A1 could be completed when applied to $\{D_i\}_{i=j}^N$ and let $\{d_i''\}_{i=j}^N$, $(d_{i-1}'', d_i'') \in D_i$, $i = j+1, \cdots, N$ be given by Algorithm A2. If (the problem and consequently) the sequences are "local," that is, if a change of the element d_k does not alter $d_{k-1}, d_{k-2}, \cdots, d_0$, and $d_{k+1}, d_{k+2}, \cdots, d_N$, we can construct $\{d_i\}_{i=0}^N$ as follows:

$$d_i = \begin{cases} d_i' & \text{for } i = 0, \cdots, j-1, \\ d^* & \text{for } i = j, \\ d_i'' & \text{for } i = j+1, \cdots, N. \end{cases} \tag{2.8}$$

We note that $(d_{i-1}, d_i) \in D_i$, $i = 1, \cdots, j-1, j+1, \cdots, N$ and for a suitable choice of d^*, the sequence can have some desirable properties for $i = j$. With respect to the piecewise monotone quadratic histospline, this means that if Algorithm A1 gives $J = j \neq N$ (we will say that Algorithm A1 has a "break point" at x_j) when applied to $\{DY_i\}_{i=0}^N$ or to $\{DM_i\}_{i=0}^N$, we can construct the sequences $\{y_i'\}_{i=0}^j$, $\{m_i'\}_{i=0}^j$, $\{y_i''\}_{i=j}^N$, $\{m_i''\}_{i=j}^N$, choose suitable values for y^*, m^*, and finally obtain both $\{y_i\}_{i=0}^N$ and $\{m_i\}_{i=0}^N$ using formulas analogous to (2.8). Practical considerations suggest that it is convenient to compute y^*, m^* in such a way as to minimize the curvature norm in $[x_{j-1}, x_{j+1}]$. Consequently, the quadratic spline given by (2.2), (2.3) is surely piecewise monotone in $[x_0, x_{j-1}]$, $[x_{j+1}, x_N]$, it can be monotone in $[x_{j-1}, x_{j+1}]$ (recall that Theorem 1 gives sufficient conditions), and is generally pleasing in the whole interval $[x_0, x_N]$.

The minimization of the curvature of the spline has also been successfully used for the implementation of Algorithm A2. More specifically, we compute a histospline $\tilde{s}(x) = \tilde{s}(x; \tilde{y}_0, \tilde{m}_0, \cdots, \tilde{y}_N, \tilde{m}_N)$ that has the minimum curvature norm in $[x_0, x_N]$, that is, which satisfies

$$F(\tilde{y}_0, \tilde{m}_0, \cdots, \tilde{y}_N, \tilde{m}_N) = \min_{y_0, m_0, \cdots, y_N, m_N} F(y_0, m_0, \cdots, y_N, m_N)$$

where

$$F(y_0, m_0, \cdots, y_N, m_N) = \int_{x_0}^{x_N} |s''(x; y_0, m_0, \cdots, y_N, m_N)|^2 \, dx.$$

Then we determine y_i, m_i, $i = N, N-1, \cdots, 0$ at steps (2) and (3)1 of Algorithm A2 as the closest points of the admissible sets (A_N for step (2) and $\{a \in \mathbb{R}: (a, d_{i+1}) \in \{A_i \times R\} \cap D_{i+1}\}$ for step (3)1) to $\tilde{y}_i$ and $\tilde{m}_i$, respectively. We note that the sequence

$\tilde{y}_0, \tilde{m}_0, \cdots, \tilde{y}_N, \tilde{m}_N$ is easily computed solving the linear system

$$\frac{\partial F(y_0, m_0, \cdots, y_N, m_N)}{\partial m_i} = 0, \qquad i = 0, 1, \cdots, N,$$

$$\frac{\partial F(y_0, m_0, \cdots, y_N, m_N)}{\partial y_i} = 0, \qquad i = 0, 1, \cdots, N$$

and refer to [8] for details on the algorithm.

3. Some examples. In this section we report the graphical application of the method outlined below. All the tests are obtained running the code PMQHS1 [8] on a mini-computer Data General Eclipse S-140. The first example is given by the set of values of Table 1 and the corresponding graph is shown in Fig. 4. We imposed that the spline (and the unknown density function) is zero outside the interval $[x_0, x_N]$ simply by adding empty cells at the sides of the histogram. Other histospline fittings can be found in [1] and [10].

TABLE 1

i	0	1	2	3	4	5	6	7
x_i	15	20	25	30	35	40	45	50
h_i		7442	19261	14385	6547	2123	451	17

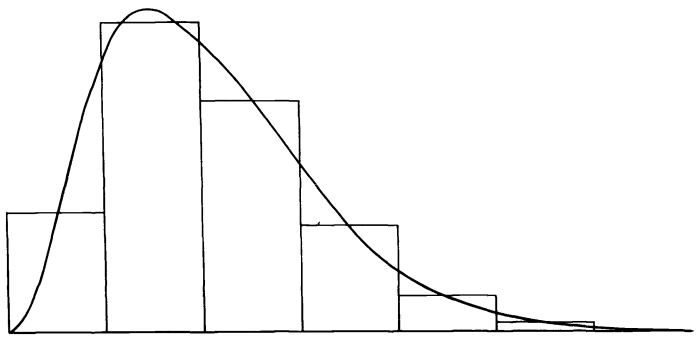


FIG. 4

Now consider the data of the first two columns of Table 2. If we want the curve to vanish outside $[x_0, x_N]$, we see that it is impossible to obtain a histospline that is decreasing in the subinterval $[23.5, 24.5]$. In fact, since $h_{13} = h_{14} = 1$, for any co-monotone histospline $s \in S_2^1(\Delta)$, we have $s(x) = 1$, $x \in [22.5, 24.5]$ and this is obviously in contrast to the request $s(x) = 0$, $x > x_{14}$. If we want to preserve the monotonicity, we can only reduce the information contained in the histogram, and consider jointly the last two cells. The graph of Fig. 5 uses the modified data of the last columns of Table 2, and should be compared with those showed in [1, Fig. 2] and [10, Fig. 2]. We note that, since Algorithm A1 has a break point at x_6 when applied to the sets (2.7) and at x_7 when applied to the sets (2.6), the spline is not monotone in $[x_6, x_7]$ and in $[x_7, x_8]$. However, the function is almost constant in $[x_7, x_8]$ (see also [8, Table 2]), and the curve has, in practice, the unimodal behaviour of the histogram. Other break

TABLE 2

i	x_i	h_i	$\bar{x}_i$	$\bar{h}_i$
0	10.5		10.5	
1	11.5	1	11.5	1
2	12.5	5	12.5	5
3	13.5	20	13.5	20
4	14.5	38	14.5	38
5	15.5	50	15.5	50
6	16.5	110	16.5	110
7	17.5	110	17.5	110
8	18.5	104	18.5	104
9	19.5	66	19.5	66
10	20.5	44	20.5	44
11	21.5	18	21.5	18
12	22.5	10	22.5	10
13	23.5	1	24.5	2
14	24.5	1		

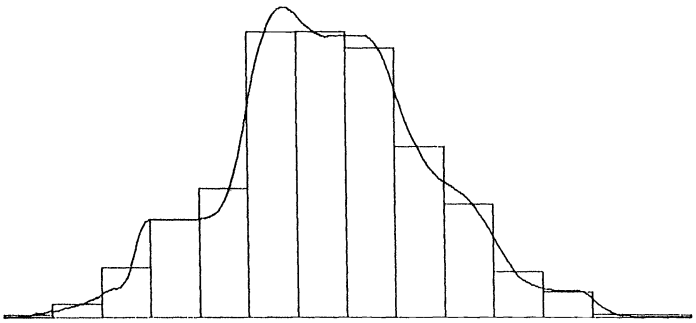


FIG. 5

points for Algorithm A1 (when applied to the sets (2.6)) are x_3, x_9, and x_{12}, but the spline is still monotone in $[x_2, x_4]$, $[x_8, x_{10}]$, and $[x_{11}, x_{13}]$.

The data of the first two columns of Table 3 are taken from [5, p. 11], and are transformed in the last two to preserve the monotonicity in [72.5, 74.5]. The corresponding spline is shown in Fig. 6.

The last example from [6, p. 43] furnishes a histogram of different shape and nonuniform classes. We imposed that the spline vanishes only at the right-hand side and obtained the graph of Fig. 7.

We conclude this section by observing that a histospline of degree n and continuity class C^k can be easily obtained differentiating a C^{k+1} spline of degree $n+1$ interpolant to the points $\{(x_i, F_i)\}_{i=0}^{N}$, where $F_0 = 0$, $F_i = F_{i-1} + h_i$, $i = 1, 2, \cdots, N$. The positivity and piecewise monotonicity will result using increasing and co-convex interpolating splines, but an analysis of the known algorithms shows that they cannot be successfully used. In fact, the monotone and convex quadratic splines of [7] are only C^1, the "taut splines" of [2] are only convex, and the splines introduced in [4] have the second derivatives at the knots identically zero.

TABLE 3

i	x_i	h_i	$\bar{x}_i$	$\bar{h}_i$
0	62.5		62.5	
1	63.5	0	63.5	0
2	64.5	1	64.5	1
3	65.5	0	65.5	0
4	66.5	2	66.5	2
5	67.5	5	67.5	5
6	68.5	9	68.5	9
7	69.5	22	69.5	22
8	70.5	16	70.5	16
9	71.5	12	71.5	12
10	72.5	8	72.5	8
11	73.5	3	73.5	8
12	74.5	1	75.5	2
13	75.5	1	76.5	0
14	76.5	0		

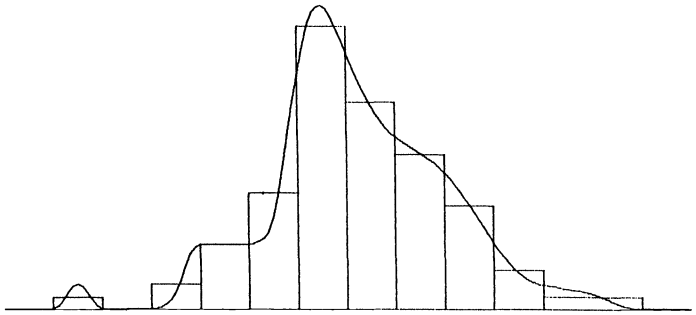


FIG. 6

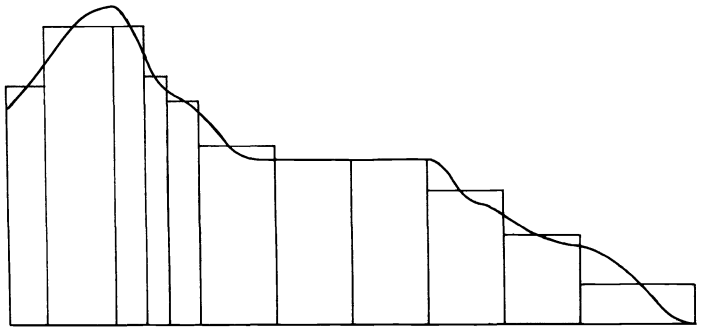


FIG. 7

TABLE 4

i	x_i	h_i
0	0	
1	5	8
2	14	18
3	18	8
4	21	5
5	25	6
6	35	12
7	45	11
8	55	11
9	65	9
10	75	6
11	90	4

REFERENCES

[1] L. I. BONEVA, D. G. KENDALL, AND I. STEFANOV, *Spline transformations: three new diagnostic aids for the statistical data-analyst*, J. Royal Statist. Soc. Ser. B, 33 (1971), pp. 1–70.

[2] C. DE BOOR, *A Practical Guide to Splines*, Springer-Verlag, Berlin, New York, 1978.

[3] P. COSTANTINI, *On monotone and convex spline interpolation*, Math. Comput., 46 (1986), pp. 203–214.

[4] ———, *An algorithm for computing shape-preserving interpolating splines of arbitrary degree*, J. Comput. Appl. Math., 22 (1988), pp. 89–136.

[5] W. J. DIXON AND F. J. MASSEY, JR., *Introduction to Statistical Analysis*, McGraw-Hill, New York, 1957.

[6] D. FREEDMAN, R. PISANI, AND R. PURVES, *Statistics*, W. W. Norton, New York, 1978.

[7] D. F. MCALLISTER AND J. A. ROULIER, *An algorithm for computing a shape preserving osculatory quadratic spline*, ACM Trans. Math. Software, 7 (1981), pp. 331–347.

[8] R. MORANDI AND P. COSTANTINI, PMQHS1: *a code for computing piecewise monotone quadratic histosplines*, Monografie di Software Matematico 46, IAC–CNR 1986.

[9] E. NEUMAN, *Shape preserving interpolation by polynomial splines*, Report N112, Institute of Computer Sciences, Wroclaw University, 1982.

[10] I. J. SCHOENBERG, *Splines and histograms*, in Spline Functions and Approximation Theory, A. Meir and A. Sharma, eds., 21, Birkhäuser, Basel, 1973, pp. 277–327.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 407-418, May 1989
© 1989 Society for Industrial and Applied Mathematics

ANALYSIS OF A RECURSIVE LEAST-SQUARES SIGNAL-PROCESSING ALGORITHM*

FRANKLIN T. LUK† AND SANZHENG QIAO‡

Abstract. This paper concerns a popular recursive least-squares algorithm for beamforming. The way in which the method's stability depends on the condition of a special matrix is analyzed in detail, and a new procedure for estimating the error in the computed solution is presented.

Key words. real time signal processing, recursive least squares, QR decomposition, plane rotations

AMS(MOS) subject classifications. 65F20, 65F25, 65G05

1. Recursive least squares. We begin with a standard linear least-squares problem. Given an $n \times p$ ($n \geqq p$) full-rank complex data matrix $X(n)$ and an n-element complex data vector $y(n)$, find a p-element complex weight vector $w(n)$ such that

$$\|X(n)w(n)+y(n)\|_2 = \min. \tag{1.1}$$

Define the residual vector

$$r(n) \equiv X(n)w(n)+y(n),$$

and use the following notation:

$$r(n) = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{pmatrix}, \quad X(n) = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix}, \quad y(n) = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

$$x_i^T = (x_{i1}, x_{i2}, \cdots, x_{ip}).$$

Unless otherwise stated in this paper, the norm $\|\cdot\|$ of a matrix denotes the Frobenius norm:

$$\|M\| = \|M\|_F = \sqrt{\sum_{i,j} |m_{ij}|^2},$$

and that of a vector the Euclidean two-norm:

$$\|v\| = \|v\|_2 = \sqrt{\sum_i |v_i|^2}.$$

Consider problem (1.1) in the setting of adaptive beamforming, an important problem in signal processing [4], [5]. An adaptive combiner operates on complex, discrete signals; at time i, for $i = 1, 2, \cdots, n$ (time $i+1$ is more recent than time i), the combiner takes as inputs a primary signal y_i and a set of p auxiliary signals x_{i1}, $x_{i2}, \cdots, x_{ip}$. Its purpose is to adjust the weight vector $w(n)$ so as to minimize the

* Received by the editors July 17, 1986; accepted for publication (in revised form) July 6, 1988. This work was supported in part by the Office of Naval Research under contract N00014-85-K-0074.

† School of Electrical Engineering, Cornell University, Ithaca, New York 14853.

‡ Department of Computer Science, McMaster University, Hamilton, Ontario, Canada L8S 4K1. The work of this author was also supported by a Mathematical Sciences Institute Graduate Fellowship funded by the Army Research Office while he was at Cornell University, Ithaca, New York.

energy of the combined output signal, viz., $\|r(n)\|_2$. Our goal, though, is to determine the power output at the present time:

$$r_n = x_n^T w(n) + y_n. \tag{1.2}$$

Often, an $n \times n$ diagonal "forgetful" matrix

$$B(n) = \text{diag}\,(\beta^{n-1}, \beta^{n-2}, \cdots, \beta^0), \qquad 0 < \beta \leqq 1,$$

is used to weigh the recent information more heavily. That is, (1.1) becomes

$$\|B(n)[X(n)w(n) + y(n)]\|_2 = \min.$$

The analysis that we present in this paper does not depend on the weights, and so we will assume $B(n) = I_n$ to simplify our presentation. Two special properties of the problem are worth emphasizing. First, we desire *only* the last residual element r_n of (1.2) and are not interested in the solution vector $w(n)$. Second, we wish to calculate this quantity *recursively* as n increases with time.

In this paper, we analyze an elegant algorithm due to McWhirter [5] that computes r_n using only plane rotations. We dispel the widespread, but mistaken, notion that the method always computes an accurate solution.

The paper is organized as follows. In § 2 we present examples where the $n \times (p+1)$-matrix $(X(n)|y(n))$ is ill conditioned and McWhirter's algorithm gives poor results. It is therefore important to estimate the computing error in r_n; we develop a novel algorithm for this purpose in § 3. A perturbation analysis for r_n is presented in § 4, where our new algorithm is compared against two well-known bounds and shown to give better numerical results. Our results in §§ 2–4 are new.

2. McWhirter's algorithm. We describe one step of the recursive algorithm of McWhirter. Suppose that $n > p$, and that we have available a QR decomposition of the $(n-1) \times p$-matrix $X(n-1)$, given by

$$X(n-1) = P(n-1)R(n-1).$$

The matrix $P(n-1)$ is $(n-1) \times p$ with orthonormal columns and the matrix $R(n-1)$ is $p \times p$ upper triangular with a nonnegative diagonal. Throughout this paper, we assume that all upper triangular matrices have nonnegative diagonal elements to ensure the uniqueness of the corresponding QR decompositions. We find r_n according to the following procedure. First, rewrite the minimization problem (1.1) as follows:

$$\left\| \begin{pmatrix} X(n-1) \\ x_n^T \end{pmatrix} w(n) + \begin{pmatrix} y(n-1) \\ y_n \end{pmatrix} \right\|_2 = \min.$$

The problem reduces to

$$\left\| \begin{pmatrix} R(n-1) \\ x_n^T \end{pmatrix} w(n) + \begin{pmatrix} v(n-1) \\ y_n \end{pmatrix} \right\|_2 = \min, \tag{2.1}$$

where

$$v(n-1) = P(n-1)^H y(n-1).$$

We define the $(p+1) \times (p+1)$-matrix

$$A \equiv \begin{pmatrix} R(n-1) & v(n-1) \\ x_n^T & y_n \end{pmatrix} \tag{2.2}$$

and let

$$k = p + 1.$$

Second, triangularize the $k \times k$-matrix A by a unitary matrix Q:

$$Q^H A = \begin{pmatrix} R(n) & v(n) \\ 0^T & \gamma \end{pmatrix} \equiv U, \tag{2.3}$$

i.e., $R(n)$ is $p \times p$ upper triangular. The matrix Q is made up of $k-1$ Givens matrices and perhaps one reflection by a diagonal matrix to force $\gamma \geqq 0$. From Q we can construct an $n \times p$ orthonormal matrix $P(n)$ such that

$$X(n) = P(n)R(n)$$

and

$$v(n) = P(n)^H y(n).$$

The residual element is given by

$$r_n = (e_k^T Q\, e_k)\gamma = (e_k^T Q\, e_k)(e_k^T Q^H A\, e_k),$$

where e_k denotes the kth unit coordinate vector.

Here are two examples where we observed sizeable computing errors in the residual; we used MATLAB on a VAX 11/750 with the following effective machine precision:

$$\varepsilon = 1.4 \times 10^{-17}.$$

We should also state that for many examples where A is well conditioned the algorithm works very well. Indeed, an example in § 4 shows that the residual may still be computed to full machine precision even when $\kappa(A) > 10^6$.

Example 1. Here is a contrived example for which we know the exact Q and U factors, and the residual element r_n, viz.,

$$A = \begin{pmatrix} 2 & 20000 & -4001\sqrt{10} & -1600\sqrt{10} & 5000 \\ 0 & 2/\sqrt{5} & 10000\sqrt{2} & -5200\sqrt{2} & 200\sqrt{5} \\ 0 & 0 & 3 & 29998 & 200\sqrt{10} \\ 0 & 0 & 0 & 3/\sqrt{10} & 1499 \\ 1 & 10001 & 3000\sqrt{10} & 1600\sqrt{10} & 5000 \end{pmatrix},$$

$$Q = \frac{1}{\sqrt{130}} \begin{pmatrix} 2\sqrt{26} & -\sqrt{13} & -\sqrt{130}/10 & -3\sqrt{10}/5 & -9\sqrt{10}/10 \\ 0 & \sqrt{65} & -\sqrt{26}/2 & -3\sqrt{2} & -9\sqrt{2}/2 \\ 0 & 0 & 3\sqrt{13} & -2 & -3 \\ 0 & 0 & 0 & 3\sqrt{10} & -2\sqrt{10} \\ \sqrt{26} & 2\sqrt{13} & \sqrt{130}/5 & 6\sqrt{10}/5 & 9\sqrt{10}/5 \end{pmatrix},$$

$$U = \frac{1}{\sqrt{10}} \begin{pmatrix} 5\sqrt{2} & 50001\sqrt{2} & -10004\sqrt{5} & -3200\sqrt{5} & 15000\sqrt{2} \\ 0 & 4 & 20001\sqrt{10} & -400\sqrt{10} & 6000 \\ 0 & 0 & 10 & 99994 & 1000\sqrt{10} \\ 0 & 0 & 0 & \sqrt{13} & 6497\sqrt{10}/\sqrt{13} \\ 0 & 0 & 0 & 0 & 2\sqrt{10}/\sqrt{13} \end{pmatrix},$$

and $r_n = 18/65 \approx 0.277$. The computed factor $\hat{Q}$ and the computed residual $\hat{r}_n$ give $\|\hat{Q} - Q\| = 6.7 \times 10^{-6}$ and

$$|\hat{r}_n - r_n| = 4.2 \times 10^{-3}.$$

In signal processing, the data matrix is usually balanced in the sense that the column norms are all approximately equal, and we may argue that the example here is unbalanced. However, we can construct a new matrix

$$\tilde{A}=\begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 & \alpha_6 \\ 0 & 2 & 20000 & -4001/\sqrt{10} & -1600\sqrt{10} & 5000 \\ 0 & 0 & 2/\sqrt{5} & 10000\sqrt{2} & -5200\sqrt{2} & 200\sqrt{5} \\ 0 & 0 & 0 & 3 & 29998 & 200\sqrt{10} \\ 0 & 0 & 0 & 0 & 3/\sqrt{10} & 1499 \\ 0 & 1 & 10001 & 3000\sqrt{10} & 1600\sqrt{10} & 5000 \end{pmatrix}$$

and choose the α_i's to give all columns equal norms. The same computing errors would be obtained.

Example 2 (cf. [6]). We make up this example using the famous matrix (shown here for $n=5$)

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

By choosing

$$A+E=\begin{pmatrix} 1 & -1 & -1 & -1 & -1 \\ 0 & 1 & -1 & -1 & -1 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ \xi & \xi & \xi & \xi & 1+\xi \end{pmatrix},$$

we get, by neglecting terms of order ξ^2, the computed factors

$$\hat{Q}\approx\begin{pmatrix} 1 & 0 & 0 & 0 & -\xi \\ 0 & 1 & 0 & 0 & -2\xi \\ 0 & 0 & 1 & 0 & -4\xi \\ 0 & 0 & 0 & 1 & -8\xi \\ \xi & 2\xi & 4\xi & 8\xi & 1 \end{pmatrix},$$

$$\hat{U}\approx\begin{pmatrix} 1 & -1 & -1 & -1 & \xi-1 \\ 0 & 1 & -1 & -1 & 2\xi-1 \\ 0 & 0 & 1 & -1 & 4\xi-1 \\ 0 & 0 & 0 & 1 & 8\xi-1 \\ 0 & 0 & 0 & 0 & 16\xi+1 \end{pmatrix}$$

and the computed residual $\hat{r}_5\approx 16\xi+1$. Again, we have $\|\hat{Q}-Q\|>8\sqrt{2}\xi$ and

$$|\hat{r}_5-r_5|\approx 16\xi.$$

In general (for n not too large), we expect that $\|\hat{Q}-Q\|>2^{n-2}\sqrt{2}\xi$ and

$$|\hat{r}_n-r_n|\approx 2^{n-1}\xi.$$

3. Error estimation. A procedure for estimating the computing error in r_n therefore has great value. Consider the triangularization procedure of (2.3). Let $A=(a_{ij})$ and $U=(u_{ij})$. Denote by $\{z_i^{(l)}\}$ the nonzero elements in the last row of A during the annihilation process. These elements are defined by the following relation:

$$\begin{pmatrix} u_{ll} & u_{l,l+1} & \cdots & u_{lk} \\ 0 & z_{l+1}^{(l)} & \cdots & z_k^{(l)} \end{pmatrix} = \begin{pmatrix} \overline{c_l} & \overline{s_l} \\ -s_l & c_l \end{pmatrix} \begin{pmatrix} a_{ll} & a_{l,l+1} & \cdots & a_{lk} \\ z_l^{(l-1)} & z_{l+1}^{(l-1)} & \cdots & z_k^{(l-1)} \end{pmatrix}, \tag{3.1}$$

for $l=1, 2, \cdots, k-1$, where

$$(z_1^{(0)}, \cdots, z_k^{(0)}) = (a_{k1}, a_{k2}, \cdots, a_{kk}).$$

The 2×2 Givens matrix is unitary; its parameters satisfy

$$|c_l|^2+|s_l|^2=1.$$

The procedure (3.1) may be written in this algorithmic form:

```
begin
for l = 1, 2, ···, k − 1 do
  begin
  u_ll = √(|a_ll|² + |z_l^(l−1)|²);   c_l = a_ll / u_ll;   s_l = z_l^(l−1) / u_ll;
  for j = l+1, l+2, ···, k, do
    begin
    u_lj = c̄_l a_lj + s̄_l z_j^(l−1);   z_j^(l) = −s_l a_lj + c_l z_j^(l−1);
    end
  end;
r_n = (c_1 ··· c_k) z_k^(k−1)
end.
```

We use $\hat{}$ to denote the corresponding computed elements, and fl to denote floating point computation. That is, we calculate the following six quantities:

$$\hat{u}_{ll} = \text{fl}\,(\sqrt{|\hat{a}_{ll}|^2+|\hat{z}_l^{(l-1)}|^2}), \quad \hat{c}_l = \text{fl}\left(\frac{\hat{a}_{ll}}{\hat{u}_{ll}}\right), \quad \hat{s}_l = \text{fl}\left(\frac{\hat{z}_l^{(l-1)}}{\hat{u}_{ll}}\right),$$

$$\hat{u}_{lj} = \text{fl}\,(\text{fl}\,(\overline{c_l})\hat{a}_{lj}+\text{fl}\,(\overline{s_l})\hat{z}_j^{(i-1)}), \qquad \hat{z}_j^{(l)} = \text{fl}\,(-\hat{s}_l\hat{a}_{lj}+\hat{c}_l\hat{z}_j^{(l-1)}),$$

$$r_n = \text{fl}\,((\hat{c}_1 \cdots \hat{c}_k)\hat{z}_k^{(k-1)}).$$

The relations between the exact and the computed quantities are defined as follows:

$$\hat{a}_{ij} = a_{ij}(1+\xi_{ij}\alpha_{ij}(\varepsilon)), \qquad \hat{u}_{ll} = u_{ll}(1+\sigma_{ll}\phi_{ll}(\varepsilon)),$$
$$\hat{c}_l = c_l(1+\sigma_{ll}\phi_{ll}^{(c)}(\varepsilon)), \qquad \hat{s}_l = s_l(1+\sigma_{ll}\phi_{ll}^{(s)}(\varepsilon)),$$
$$\hat{u}_{lj} = u_{lj}(1+\sigma_{lj}\phi_{lj}^{(u)}(\varepsilon)), \qquad \hat{z}_j^{(l)} = z_j^{(l)}(1+\eta_j^{(l)}\psi_j^{(l)}(\varepsilon)),$$
$$\hat{r}_n = r_n(1+\eta\theta(\varepsilon)),$$

where

$$|\alpha_{ij}(\varepsilon)| = O(\varepsilon), \quad |\phi_{ll}(\varepsilon)| = O(\varepsilon), \quad |\phi_{ll}^{(c)}(\varepsilon)| = O(\varepsilon), \quad |\phi_{ll}^{(s)}(\varepsilon)| = O(\varepsilon),$$
$$|\phi_{lj}^{(u)}(\varepsilon)| = O(\varepsilon), \quad |\psi_j(\varepsilon)| = O(\varepsilon), \quad |\theta(\varepsilon)| = O(\varepsilon).$$

We choose all the coefficients to be real and nonnegative and the error estimate to be

$$|\hat{r}_n - r_n| \approx |r_n|\eta\varepsilon. \tag{3.2}$$

To compute these coefficients, we need the following result.

LEMMA. *Suppose that* $\hat{\alpha}_1 = \alpha_1(1+\xi_1\phi_1(\varepsilon))$ *and* $\hat{\alpha}_2 = \alpha_2(1+\xi_2\phi_2(\varepsilon))$, *where* $\xi_1 \geqq 0$, $\xi_2 \geqq 0$, $|\phi_1(\varepsilon)| = O(\varepsilon)$, *and* $|\phi_2(\varepsilon)| = O(\varepsilon)$. *Then we have the following*:

(1) $\sqrt{|\hat{\alpha}_1|^2+|\hat{\alpha}_2|^2} = \sqrt{|\alpha_1|^2+|\alpha_2|^2}[1+\max\{\xi_1, \xi_2\}\phi_3(\varepsilon)]$,

(2) $\hat{\alpha}_1+\hat{\alpha}_2 = (\alpha_1+\alpha_2)[1+((|\alpha_1\xi_1|+|\alpha_2\xi_2|)/|\alpha_1+\alpha_2|)\phi_4(\varepsilon)]$,

where $\phi_3(\varepsilon) = O(\varepsilon)$ *and* $|\phi_4(\varepsilon)| = O(\varepsilon)$. □

We assume that initially

$$\hat{a}_{ij} = a_{ij}(1+\varepsilon), \qquad \hat{z}_j^{(0)} = z_j^{(0)}(1+\varepsilon),$$

i.e., $\xi_{ij} = 1$, $\eta_j^{(0)} = 1$, $\alpha_{ij}(\varepsilon) = \varepsilon$, and $\psi_j^{(0)}(\varepsilon) = \varepsilon$. We present the following algorithm to calculate the parameter η of (3.2).

ALGORITHM ESTIMATE.
begin
for $l = 1, 2, \cdots, k-1$ do
begin
$\sigma_{ll} = \max\{\xi_{ll}, \eta_l^{(l-1)}\}$;
for $j = l+1, l+2, \cdots, k$ do
begin

$$\sigma_{lj} = \frac{|\overline{c_l}a_{lj}\max\{\sigma_{ll}, \xi_{lj}\}| + |\overline{s_l}z_j^{(l-1)}\max\{\sigma_{ll}, \eta_j^{(l-1)}\}|}{|\overline{c_l}a_{lj} + \overline{s_l}z_j^{(l-1)}|};$$

$$\eta_j^{(l)} = \frac{|s_l a_{lj}\max\{\sigma_{ll}, \xi_{lj}\}| + |c_l z_j^{(l-1)}\max\{\sigma_{ll}, \eta_j^{(l-1)}\}|}{|s_l a_{lj} - c_l z_j^{(l-1)}|}$$

end
end;
$\eta = \max\{\sigma_{11}, \sigma_{22}, \cdots, \sigma_{k-1,k-1}, \eta_k^{(k-1)}\}$
end.

Mapping our new algorithm onto the systolic array of McWhirter [5] is a straightforward task. The implementation requires *little* additional time because the work is performed mostly by the multiply-and-accumulate processors, those that normally idle extensively during a systolic QR decomposition.

4. Perturbation analysis. In this section, we derive perturbation bounds on the error $|r_n - \hat{r}_n|$. First, we use a result of Stewart [6] to establish that a priori bounds on the coefficients ξ_{ij} given in § 3 depend on the condition number and can be very large if the matrix A is ill conditioned. Second, we use a result of Golub and Van Loan [3, p. 144] to derive a sharper bound. Third, we present three numerical examples to compare these bounds against the value provided by Algorithm Estimate.

From (2.2), the $k \times k$-matrix A has the QR decomposition

$$A = QU, \tag{4.1}$$

with Q unitary and U upper triangular. It is well known (cf. [1]) that the QR decomposition is stable in the sense that the computed factors $\hat{Q}$ and $\hat{U}$ satisfy

$$A + E = \hat{Q}\hat{U},$$

where $\|E\|/\|A\| = O(k^3, \varepsilon)$. Now, we study the perturbation of the Q and U factors. First, we need some definitions. Denote the pseudoinverse of a matrix M by M^+ and the condition number of M by

$$\kappa(M) = \|M\|\,\|M^+\|.$$

We define **T**, a linear operator that maps the space of $k \times k$ upper triangular matrices into the space of Hermitian matrices, by

$$\mathbf{T}F = U^H F + F^H U. \tag{4.2}$$

We also define

$$\tau = k\|U^{-1}\|[1+\kappa(U)/\sqrt{2}]$$

and

$$\sigma = k(2+\sqrt{2})\kappa(U).$$

THEOREM 1 [6]. *Let A, Q, and U be as in* (4.1). *Let E satisfy*

$$\|A^+\|\,\|E\| < \tfrac{1}{2}$$

and let

$$F_1 = \mathbf{T}^{-1}[U^H(Q^H E) + (E^H Q)U].$$

If

$$\tau\|F_1\| < \tfrac{1}{4},$$

then there is a unique solution of the nonlinear equation

$$\mathbf{T}F = U^H(Q^H E) + (E^H Q)U + E^H E - F^H F$$

that satisfies

$$\|F\| < 2\|F_1\| \leqq 2\sigma\|E\|.$$

Moreover,

$$A + E = (Q+W)(U+F), \tag{4.3}$$

where the matrix $Q+W$ has orthonormal columns, and

$$\|W\| \leqq \frac{3\kappa(A)(\|E\|/\|A\|)}{1-2\kappa(A)(\|E\|/\|A\|)} \tag{4.4}$$

as well as

$$\frac{\|F\|}{\|A\|} \leqq \frac{\|E\|}{\|A\|} + \|W\|\left(1+\frac{\|E\|}{\|A\|}\right) \tag{4.5}$$

assuming the diagonal elements of U and F are real. □

Stewart [6] states that (4.4) and (4.5) still hold if we replace $\kappa(A)$ by $\kappa_2(A) = \|A\|_2\|A^+\|_2$ and $\|E\|/\|A\|$ by $\|E\|/\|A\|_2$, where $\|\cdot\|_2$ denotes the spectral norm.

Let us perturb the data matrix and vector in the minimization problem (2.1), i.e., we replace the matrix A of (2.2) by $A+E$. Suppose the conditions for our theorem are satisfied and the perturbed residual element is given by

$$\hat{r}_n = [e_k^T(Q+W)\,e_k][e_k^T(Q+W)^H(A+E)\,e_k].$$

Hence

$$\hat{r}_n - r_n = (e_k^T Q\,e_k)(e_k^T Q^H E\,e_k) + (e_k^T Q\,e_k)[e_k^T W^H(A+E)\,e_k]$$
$$+ (e_k^T W e_k)[e_k^T(Q+W)^H(A+E)\,e_k],$$

from which we may derive

$$|\hat{r}_n - r_n| \leqq \|E e_k\| + 2\|W\| \|(A+E) e_k\|.$$

If $\|E\|$ is small, then

$$|\hat{r}_n - r_n| \leqq 2\|W\| \|A e_k\| \leqq \frac{6\kappa(A)\|A e_k\|(\|E\|/\|A\|)}{1-2\kappa(A)(\|E\|/\|A\|)}. \tag{4.6}$$

However, the quantity $\|E\|$ may not stay small for long. If $\|F\|$ is large, the matrix $R(n)$ and vector $v(n)$ of (2.3) will be adversely affected and the perturbation in A will be much larger for the next step of the algorithm.

A sharper bound can be obtained from Golub and Van Loan [3, p. 144] which is

$$r_n(\varepsilon) - r_n(0) = \varepsilon \dot{r}(0) + O(\varepsilon^2),$$

where $r_n(\varepsilon)$ denotes the nth component of the residual of

$$\min\{\|(A+\varepsilon E)x(\varepsilon) - (b+\varepsilon f)\|\}$$

and

$$|\dot{r}(0)| \leqq (\|f\| + \|E\| \|x\|)\|I - AA^+\| + \|r^H E\| \|A^+ e_n\|.$$

We take as an estimate

$$|\hat{r}_n - r_n| \leqq \varepsilon[(\|f\| + \|E\| \|x\|)\|I - AA^+\| + \|r^H E\| \|A^+ e_n\|]. \tag{4.7}$$

The key factor in (4.7) is the size of $\|A^+ e_n\|$. However, a backsolve is necessary to compute this value, whereas Algorithm Estimate saves the backsolve as well as provides a more accurate result.

We discuss three examples. The first two matrices are not as ill conditioned as those shown before, so that the conditions of the theorem can be satisfied.

Example 3. The matrix A is unbalanced and $\kappa(A) = 4.7 \times 10^6$:

$$A = \begin{pmatrix} 0.001 & 1000\sqrt{5} & 2\sqrt{10} & -10\sqrt{2}/7 \\ 0 & 1000 & -\sqrt{2} & 4\sqrt{10}/7 \\ 0 & 0 & 1 & 6\sqrt{5}/7 \\ 0.002 & 1000\sqrt{5} & 2\sqrt{10} & -10\sqrt{2}/7 \end{pmatrix}.$$

Its exact QR factors are:

$$Q = \begin{pmatrix} 1/\sqrt{5} & \sqrt{2}/\sqrt{5} & 3/5 & -1/5 \\ 0 & 1/\sqrt{2} & -3/(2\sqrt{5}) & 1/(2\sqrt{5}) \\ 0 & 0 & 1/\sqrt{10} & 3/\sqrt{10} \\ 2/\sqrt{5} & -1/\sqrt{10} & -3/10 & 1/10 \end{pmatrix},$$

$$U = \begin{pmatrix} 0.001\sqrt{5} & 3000 & 6\sqrt{2} & -6\sqrt{10}/7 \\ 0 & 1000\sqrt{2} & 1 & 2\sqrt{5}/7 \\ 0 & 0 & \sqrt{10} & -6\sqrt{2}/7 \\ 0 & 0 & 0 & 12\sqrt{2}/7 \end{pmatrix},$$

with the residual element $r_n = 6\sqrt{2}/35$. The computed errors are $\|\hat{Q} - Q\| = 3.5 \times 10^{-18}$ and

$$|\hat{r}_n - r_n| = 3.5 \times 10^{-18}.$$

With $\|E\|/\|A\| = \varepsilon$, the upper bounds given by (4.6) and (4.7) are

$$1.5 \times 10^{-9} \quad \text{and} \quad 9.8 \times 10^{-12},$$

respectively. Algorithm Estimate gives the much more accurate value

$$1.0 \times 10^{-17}.$$

Example 4. Here is another ill-conditioned matrix:

$$A = \begin{pmatrix} 2 & 200 & -41\sqrt{10} & -16\sqrt{10} & 50 \\ 0 & 2/\sqrt{5} & 100\sqrt{2} & -52\sqrt{2} & 2\sqrt{5} \\ 0 & 0 & 3 & 298 & 2\sqrt{10} \\ 0 & 0 & 0 & 3/\sqrt{10} & 14 \\ 1 & 101 & 30\sqrt{10} & 16\sqrt{10} & 50 \end{pmatrix},$$

with $\kappa(A) = 1.9 \times 10^{10}$. The exact factors Q and U are:

$$Q = \frac{1}{\sqrt{130}} \begin{pmatrix} 2\sqrt{26} & -\sqrt{13} & -\sqrt{130}/10 & -3\sqrt{10}/5 & -9\sqrt{10}/10 \\ 0 & \sqrt{65} & -\sqrt{26}/2 & -3\sqrt{2} & -9\sqrt{2}/2 \\ 0 & 0 & 3\sqrt{13} & -2 & -3 \\ 0 & 0 & 0 & 3\sqrt{10} & -2\sqrt{10} \\ \sqrt{26} & 2\sqrt{13} & \sqrt{130}/5 & 6\sqrt{10}/5 & 9\sqrt{10}/5 \end{pmatrix},$$

$$U = \frac{1}{\sqrt{10}} \begin{pmatrix} 5\sqrt{2} & 501\sqrt{2} & -104\sqrt{5} & -32\sqrt{5} & 150\sqrt{2} \\ 0 & 4 & 201\sqrt{10} & -4\sqrt{10} & 60 \\ 0 & 0 & 10 & 994 & 10\sqrt{10} \\ 0 & 0 & 0 & \sqrt{13} & 62\sqrt{10}/\sqrt{13} \\ 0 & 0 & 0 & 0 & 2\sqrt{10}/\sqrt{13} \end{pmatrix}.$$

In addition, $r_n = 18/65$. The errors in the computed results are $\|\hat{Q} - Q\| = 1.1 \times 10^{-11}$ and

$$|\hat{r}_n - r_n| = 7.1 \times 10^{-11}.$$

With $\|E\|/\|A\| = \varepsilon$, the error bounds given by (4.6) and (4.7) are

$$1.2 \times 10^{-4} \quad \text{and} \quad 2.98 \times 10^{-8},$$

respectively, whereas Algorithm Estimate gives

$$1.4 \times 10^{-9}.$$

Example 5. This example executes two steps of McWhirter's algorithm. Let the data matrix and right-hand vector be

$$\begin{pmatrix} 100 & 300 & 200 \\ 200 & 605 & 100 \\ 100\sqrt{5} & 300\sqrt{5} & 196\sqrt{5} \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 500 \\ 300 \\ 20\sqrt{5} \end{pmatrix},$$

respectively. Using Givens rotations in exact arithmetic, we obtain

$$R(n-1) = \begin{pmatrix} 100\sqrt{10} & 301\sqrt{10} & 138\sqrt{10} \\ 0 & \sqrt{15} & -176\sqrt{5}/\sqrt{3} \\ 0 & 0 & 2\sqrt{10}/\sqrt{3} \end{pmatrix} \quad \text{and} \quad v(n-1) = \begin{pmatrix} 120\sqrt{10} \\ 60\sqrt{5}/\sqrt{3} \\ 240\sqrt{10}/\sqrt{3} \end{pmatrix}.$$

Using Givens rotations in six decimal digit arithmetic with roundings ($\varepsilon = 10^{-6}$), we get

$$R(n-1) = \begin{pmatrix} 316.228 & 951.846 & 436.394 \\ 0 & 3.87271 & -227.216 \\ 0 & 0 & 3.64600 \end{pmatrix} \quad \text{and} \quad v(n-1) = \begin{pmatrix} 379.474 \\ 77.4490 \\ 438.181 \end{pmatrix}.$$

Suppose that new data have arrived. Using notation of the paper, we have

$$A = \begin{pmatrix} 100\sqrt{10} & 301\sqrt{10} & 138\sqrt{10} & 120\sqrt{10} \\ 0 & \sqrt{15} & -176\sqrt{5}/\sqrt{3} & 60\sqrt{5}/\sqrt{3} \\ 0 & 0 & 2\sqrt{10}/\sqrt{3} & 240\sqrt{10}/\sqrt{3} \\ 50\sqrt{10} & 151\sqrt{10} & 39\sqrt{10} & 30\sqrt{10} \end{pmatrix}$$

and

$$A+E = \begin{pmatrix} 316.228 & 951.846 & 436.394 & 379.474 \\ 0 & 3.87271 & -227.216 & 77.4490 \\ 0 & 0 & 3.64600 & 438.181 \\ 158.114 & 477.504 & 123.329 & 94.8683 \end{pmatrix}.$$

The initial coefficients, computed in six decimal digit arithmetic, for Algorithm Estimate are

$$\begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} & \sigma_{14} \\ 0 & \sigma_{22} & \sigma_{23} & \sigma_{24} \\ 0 & 0 & \sigma_{33} & \sigma_{34} \\ \eta_1^{(0)} & \eta_2^{(0)} & \eta_3^{(0)} & \eta_4^{(0)} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 301 & 301 & 1.71\times 10^3 \\ 0 & 0 & 1.78\times 10^4 & 301 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

Carrying out one step of McWhirter's algorithm, we get

$$\hat{U} = \begin{pmatrix} 353.565 & 1064.87 & 445.463 & 381.826 \\ 0 & 4.12278 & -242.535 & 43.65 \\ 0 & 0 & 4.05480 & 440.504 \\ 0 & 0 & 0 & 96.1870 \end{pmatrix}.$$

The coefficients computed by Algorithm Estimate are

$$\begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} & \sigma_{14} \\ 0 & \sigma_{22} & \sigma_{23} & \sigma_{24} \\ 0 & 0 & \sigma_{33} & \sigma_{34} \\ 0 & 0 & 0 & \eta \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 603 & 603 & 1.4\times 10^3 \\ 0 & 0 & 5.36\times 10^4 & 5.36\times 10^4 \\ 0 & 0 & 0 & 5.36\times 10^4 \end{pmatrix}.$$

The exact residual is $r_n = 480\sqrt{10}/21 = 72.2806$, and the computed residual is $\hat{r}_n = 72.6640$. So

$$|\hat{r}_n - r_n| = 0.3834.$$

We have $\kappa(A) = 2.8\times 10^5$, $\|E\|_2/\|A\|_2 = 8.7\times 10^{-6}$, $\|A\, e_k\|_2 = 592.5$. The parameter $\kappa(A)$ is too large to satisfy the conditions of Theorem 1, and (4.6) cannot be used. The bound (4.7) equals

$$58.5$$

and Algorithm Estimate gives

$$(5.36\times 10^4)\times 10^{-6}\times 72.6640 = 3.89.$$

Now we include one new row of data:

$$A=\begin{pmatrix} 250\sqrt{2} & 752\sqrt{2} & 315\sqrt{2} & 270\sqrt{2} \\ 0 & \sqrt{17} & -1000/\sqrt{17} & 180/\sqrt{17} \\ 0 & 0 & 2\sqrt{70}/\sqrt{17} & 1520\sqrt{10}/\sqrt{119} \\ 250\sqrt{2}/4 & 94\sqrt{2} & 63\sqrt{2} & 90\sqrt{2} \end{pmatrix}.$$

Our new perturbed matrix is

$$A+E=\begin{pmatrix} 353.565 & 1064.87 & 445.463 & 381.826 \\ 0 & 4.12278 & -242.535 & 43.6500 \\ 0 & 0 & 4.05480 & 440.504 \\ 88.3881 & 132.936 & 89.0952 & 127.279 \end{pmatrix}.$$

The initial coefficients for Algorithm Estimate are

$$\begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} & \sigma_{14} \\ 0 & \sigma_{22} & \sigma_{23} & \sigma_{24} \\ 0 & 0 & \sigma_{33} & \sigma_{34} \\ \eta_1^{(0)} & \eta_2^{(0)} & \eta_3^{(0)} & \eta_4^{(0)} \end{pmatrix}=\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 603 & 603 & 1.4\times 10^3 \\ 0 & 0 & 5.36\times 10^4 & 5.36\times 10^4 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

Carrying out one step of McWhirter's algorithm, we get

$$\hat{U}=\begin{pmatrix} 364.445 & 1065.32 & 453.762 & 401.296 \\ 0 & 129.360 & 13.8613 & -29.4681 \\ 0 & 0 & 243.133 & -37.2590 \\ 0 & 0 & 0 & 441.186 \end{pmatrix}.$$

The coefficients computed by Algorithm Estimate are

$$\begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} & \sigma_{14} \\ 0 & \sigma_{22} & \sigma_{23} & \sigma_{24} \\ 0 & 0 & \sigma_{33} & \sigma_{34} \\ 0 & 0 & 0 & \eta \end{pmatrix}=\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 603 & 1.28\times 10^3 & 698 \\ 0 & 0 & 5.36\times 10^4 & 5.36\times 10^4 \\ 0 & 0 & 0 & 5.36\times 10^4 \end{pmatrix}.$$

The exact residual is $r_n \approx 0.228354$, and the computed residual is $\hat{r}_n = 0.227494$. So,

$$|\hat{r}_n - r_n| = 8.6\times 10^{-2}.$$

Now, we have $\kappa(A)=31.9$, $\|E\|_2/\|A\|_2 = 8.7\times 10^{-6}$, $\|A\,e_k\|_2 = 598.4$. The bound (4.6) leads to

$$\frac{6\kappa(A)\|A\,e_k\|(\|E\|/\|A\|)}{1-2\kappa(A)(\|E\|/\|A\|)}=135.5,$$

while the bound (4.7) equals

$$1.17\times 10^{-1}.$$

Algorithm Estimate gives the much closer value of

$$(5.36\times 10^4)\times 10^{-6}\times 0.227494 = 1.2\times 10^{-2}.$$

Acknowledgments. The authors are grateful to Å. Björck for his patience and to the referee for numerous valuable suggestions.

REFERENCES

[1] W. M. GENTLEMAN, *Error analysis of* QR *decompositions by Givens transformations*, Linear Algebra Appl., 10 (1975), pp. 189–197.

[2] W. M. GENTLEMAN AND H. T. KUNG, *Matrix triangularization by systolic arrays*, in Proc. SPIE, Vol. 298, Real Time Signal Processing IV, International Society for Optical Engineering, Bellingham, WA, 1981, pp. 19-26.

[3] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1983.

[4] J. V. MCCANNY AND J. G. MCWHIRTER, *Some systolic array developments in the United Kingdom*, IEEE Computer, 20, July (1987), pp. 51-63.

[5] J. G. MCWHIRTER, *Recursive least-squares minimization using a systolic array*, in Proc. SPIE, Vol. 431, Real Time Signal Processing VI, International Society for Optical Engineering, Bellingham, WA, 1983, pp. 105-112.

[6] G. W. STEWART, *Perturbation bounds for the* QR *factorization of a matrix*, SIAM J. Numer. Anal., 14 (1977), pp. 509-518.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 419-431, May 1989
© 1989 Society for Industrial and Applied Mathematics
002

SYMMETRIC PRIME FACTOR FAST FOURIER TRANSFORM ALGORITHMS*

JAMES S. OTTO†

Abstract. The prime factor algorithm (PFA) is a fast algorithm for the evaluation of the discrete Fourier transform (DFT), applicable when the sequence length N is a product of relative primes. PFAs are presented that take advantage of the symmetry in a real-even or real-odd sequence. These algorithms require only one-fourth the real arithmetic and storage requirements of the complex PFA. As with existing state-of-the-art PFAs, these algorithms can be performed in-place and in-order.

Key words. Fast Fourier Transform (FFT), symmetric FFTs, prime factor FFT, real FFTs

AMS(MOS) subject classifications. 42-04, 65TO5

1. Introduction. A recent advance in the design of Fast Fourier Transforms (FFTs) has resulted from the systematic study of symmetry properties of these algorithms. In [11] and [3] algorithms that exploit a given symmetry in the data sequence are developed, thereby yielding symmetric compact FFTs that are more efficient, in terms of both computational and storage requirements, than the complex FFT. Furthermore, unlike previous approaches to symmetry [6] no pre- or post-processing of the data is employed by these new algorithms. The work by these authors [3], [11] was done in the context of the conventional FFT, where the classical sequence-splitting approach of the Cooley–Tukey algorithm [7] is used. The idea of doing symmetric transformations in this context was described earlier in an overlooked paper by Gentleman [9]. For other approaches to symmetry in the conventional FFT see [8] and [4] and the references therein. In [8] algorithms are introduced that minimize the number of arithmetic operations as well as the number of butterfly stages. In [4] the symmetric transforms are obtained as the results of a complex transform of the same length. The instability of the approach in [6] is also discussed in this latter reference.

Recently, Temperton [13] has utilized real symmetry to produce a symmetric prime factor algorithm (PFA). In general, the PFA is applicable in the case where the sequence length N factors into M mutually prime factors. It works by rewriting the one-dimensional transform as an M-dimensional transform and performing "short" transforms along the M dimensions. In an earlier paper, Temperton [12] has described a version of the PFA that, by virtue of efficient indexing, out-performs conventional FFTs of similar lengths. Another attractive feature of state-of-the-art PFAs is the property that the transform is performed in-place (without additional storage) and in-order (without reordering of data before or after the transform) [5], [12]. Thus, prime factor algorithms are important alternatives to conventional FFTs and it would be useful to have symmetric compact PFAs that take into account the symmetries described in [11]. With respect to real-even or real-odd sequences there is a fourfold symmetry factor over general complex sequences. So it seems that a reasonable goal in deriving a special algorithm for these cases is to achieve a reduction of work and storage by the same factor. To this end, this paper describes PFAs for real-even and real-odd symmetry, i.e., cosine and sine transforms, that reduce by a factor of four the work and storage required by the complex PFA.

* Received by the editors February 10, 1988; accepted for publication (in revised form) August 12, 1988. This work was supported by National Science Foundation grant DMS-86-11325.

† Department of Mathematics, University of Colorado, Denver, Colorado 80202.

The approach taken here is similar to that in [13] where the crucial idea is the factoring of the discrete Fourier transform (DFT) into a real transform followed by a folding or butterfly transform, resulting in a two-stage algorithm. The first stage preserves real symmetry; short (real) transforms are applied along the various dimensions of an M-dimensional array, producing an intermediate sequence of real values. The second stage involves applying the folding to produce the final conjugate-even transform sequence. This approach uses only half the arithmetic of a complex PFA. Furthermore, it can be implemented so as to halve the storage requirements of the latter. Therefore, Temperton's real PFA is *symmetric* and *compact* in perfect analogy with the real algorithms of [3] and [11]. To further the analogy, a PFA for real-even or real-odd data should perform the necessary computations in half the time using half the storage of the real PFA. We show that this can be done by using a different factorization of the DFT to preserve symmetry. This factorization is not new; it relates the DFT to the DHT, the discrete Hartley transform [1], [2].

2. Definitions and notation. Given a complex sequence $\mathbf{x}$, the DFT, $\mathbf{y}$, of $\mathbf{x}$ is given by

$$\mathbf{y}_n = \sum_{k=0}^{N-1} \mathbf{x}_k \omega_N^{kn}, \qquad n = 0, 1, \cdots, N-1, \tag{2.1}$$

where

$$\omega_N = e^{2\pi i/N}. \tag{2.2}$$

We write (2.1) in matrix-vector form as

$$\mathbf{y} = W_N \mathbf{x} \tag{2.3}$$

where W_N is the matrix of coefficients in (2.1). We also refer to the matrix W_N^r (r relatively prime to N) obtained from W_N by raising its elements to the rth power. We note that if in (2.3) we replace W_N with W_N^r the result is a permutation (determined by the particular value of r) of the elements in $\mathbf{y}$ [12]. We call the transformation induced by W_N^r a *rotated* DFT with *rotation r.*

We are interested in computing the DFT of $\mathbf{x}$ when the sequence has some special properties. Suppose that $\mathbf{x}$ is a real sequence. If in addition,

$$\mathbf{x}_k = \mathbf{x}_{N-k}, \qquad k = 0, 1, \cdots, N, \tag{2.4}$$

we say that x has *real-even symmetry.* If

$$\mathbf{x}_k = -\mathbf{x}_{N-k}, \qquad k = 0, 1, \cdots, N, \tag{2.5}$$

we say that $\mathbf{x}$ has *real-odd symmetry.* If $\mathbf{x}$ is imaginary and (2.5) holds we say that $\mathbf{x}$ has *imaginary-odd symmetry.* Note that we always evaluate indices modulo the sequence length so that there is no doubt as to the meaning of x_k for $k \geqq N$.

Let N in (2.1) factor into M relatively prime numbers, N_i, $i = 1, 2, \cdots, M$. A *prime factor algorithm* (PFA) for the DFT consists of rewriting (2.1) as an M-dimensional transform:

$$Y(\eta) = \sum_{k_M=0}^{N_M-1} \sum_{k_{M-1}=0}^{N_{M-1}-1} \cdots \sum_{k_1=0}^{N_1-1} X(\kappa) \omega_{N_1}^{s_1} \cdots \omega_{N_{M-1}}^{s_{M-1}} \omega_{N_M}^{s_M}, \tag{2.6}$$

where

$$s_i = k_i \cdot n_i \cdot r_i, \qquad 1 \leqq r_i \leqq N_i - 1$$

and η and κ are M-tuples of indices

$$\eta = (n_1, n_2, \cdots, n_M), \qquad \kappa = (k_1, k_2, \cdots, k_M).$$

In this equation the set of rotation constants $\{r_i\}_{i=1}^{M}$ depends on the particular bijections Ω_1 and Ω_2 used to define the correspondences between n and k of (2.1) and η and κ of (2.6), respectively. Here, Ω_1 and Ω_2 are mappings

$$\Omega : \{0, 1, \cdots, N-1\} \to S = S_1 \times S_2 \times \cdots \times S_M$$

where

$$S_i = \{0, 1, \cdots, N_i - 1\}, \qquad i = 1, 2, \cdots, M,$$

and "$\times$" denotes the Cartesian cross product. The ability to rewrite (2.1) as (2.6) depends on making the appropriate choices of Ω_1 and Ω_2. Good's original choice of maps [10] resulted in $r_i \equiv 1$. Later authors suggested using the same map for both Ω_1 and Ω_2 to achieve in-order, in-place algorithms [5], [12]. In particular, Temperton showed how the choice of the Chinese Remainder Theorem (CRT) map for both maps leads to an efficient method for updating the required indices. In these later algorithms it is usually the case that r_i is not unity, causing a permutation of the components of the DFTs in this dimension. However, these algorithms avoid explicit reordering of components by cleverly incorporating the permutation into the computation of the short (length N_i) DFTs.

3. PFAs for sequences with symmetry. Let $\mathbf{x}$ in (2.1) be a real sequence. We consider this property a kind of symmetry and seek to exploit it, along with any additional symmetry, to get a more efficient PFA. One approach is to attempt to preserve the given symmetry throughout the calculations and take advantage of it at each step. Following Temperton [13], we rewrite (2.6), using the Kronecker product of the transforms $W_{N_i}^{r_i}$ as

$$\mathbf{y} = Q_N^{-1} W_{N_M}^{r_M} \otimes W_{N_{M-1}}^{r_{M-1}} \otimes \cdots \otimes W_{N_1}^{r_1} P_N \mathbf{x}, \tag{3.1}$$

where the permutation matrices P_N and Q_N are determined by the mappings Ω_1 and Ω_2. Furthermore, we assume a factorization of each $W_{N_i}^{r_i}$ as

$$W_{N_i}^{r_i} = F_{N_i} R_{N_i}^{r_i}, \tag{3.2}$$

where $R_{N_i}^{r_i}$ is a real matrix that incorporates the rotation r_i and the folding matrices F_{N_i} have a particularly simple form. As a consequence of (3.2) we may rewrite (3.1) as

$$\mathbf{y} = Q_N^{-1}(F_{N_M} \otimes F_{N_{M-1}} \otimes \cdots \otimes F_{N_1})(R_{N_M}^{r_M} \otimes R_{N_{M-1}}^{r_{M-1}} \otimes \cdots \otimes R_{N_1}^{r_1}) P_N \mathbf{x}. \tag{3.3}$$

Now let

$$F_N = Q_N^{-1}(F_{N_M} \otimes F_{N_{M-1}} \otimes \cdots \otimes F_{N_1}) Q_N, \tag{3.4}$$

$$R_N = Q_N^{-1}(R_{N_M}^{r_M} \otimes R_{N_{M-1}}^{r_{M-1}} \otimes \cdots \otimes R_{N_1}^{r_1}) P_N, \tag{3.5}$$

and define the intermediate sequence $\mathbf{z}$ by

$$\mathbf{z} = R_N \mathbf{x}, \tag{3.6}$$

so that

$$\mathbf{y} = F_N \mathbf{z}. \tag{3.7}$$

Equations (3.6) and (3.7) define a two-stage version of the PFA, where the first stage, since R_N is real, preserves real symmetry. We call these two stages, respectively, the *transform stage* and the *folding stage* of the symmetric PFA. The value of this two-stage

approach is that if the folding matrices F_{N_i} are simple enough then the lion's share of the work of the PFA will be performed during the transform stage. The factorization (3.2) is chosen so that each step of the transform stage uses and preserves the given symmetry. The symmetric result **z** is then supplied to the folding stage where symmetry is again exploited to perform the remaining calculations with a minimum of effort.

4. The transform stage for real-even sequences. In this section we use the approach of § 3 and derive the transform stage for the case of real-even symmetry. Later we will note the modifications necessary for the real-odd case. Let **x** be a real-even sequence. It is well known that **y** of (2.1) must also be a real-even sequence. Because of the efficiency in indexing achieved by the CRT map [12] we elect to use it for Ω_1 and Ω_2 of § 2 (this corresponds to an in-place, in-order algorithm). To see the symmetry that arises when we map the sequence **x** via the CRT from a one-dimensional array to an M-dimensional array we will view the M-dimensional mapping as a sequence of two-dimensional mappings. Define $P_i = \prod_{j \neq i} N_j$. The transform stage has M steps. The ith step consists of mapping the current values of the one-dimensional sequence **x** to an N_i by P_i array via the CRT, and applying the rotated transform $R^{r_i}_{N_i}$ to each of the columns in this array. In practice, no actual data transfer from a one-dimensional array to a two-dimensional array occurs. We just need to be able to compute the indices (with respect to one dimension) of the elements in each column of the two-dimensional array, and the CRT map allows us to do this in a straightforward manner.

Now consider the first step of the above procedure. Since we are trying to take advantage of the symmetry in the input data, it is natural to ask what kind of symmetry arises on mapping the data to a two-dimensional array, $X(k_1, k_2)$, say. Let k correspond to (k_1, k_2) via the CRT map. Then it is not difficult to show that

$$\mathbf{x}_k = \mathbf{x}_{N-k} \quad \text{iff} \quad X(k_1, k_2) = X(N_1 - k_1, P_1 - k_2). \tag{4.1}$$

The equation on the right states that X satisfies a two-dimensional evenness property. As an example, consider the case $N = 60$ with $N_1 = 5$, $N_2 = 4$, $N_3 = 3$. Using the CRT map the set of indices $\{0, 1, \cdots, 59\}$ maps to a 5 by 12 array of indices shown in Table 1. Using the symmetry property (4.1) we see that in terms of actual data values this array has the form shown in Table 2. We conclude that each column of this array is

TABLE 1

$N_1 = 5$, $P_1 = 12$

0	25	50	15	40	5	30	55	20	45	10	35
36	1	26	51	16	41	6	31	56	21	46	11
12	37	2	27	52	17	42	7	32	57	22	47
48	13	38	3	28	53	18	43	8	33	58	23
24	49	14	39	4	29	54	19	44	9	34	59

TABLE 2

$N_1 = 5$, $P_1 = 12$

0	25	10	15	20	5	30	5	20	15	10	25
24	1	26	9	16	19	6	29	4	21	14	11
12	23	2	27	8	17	18	7	28	3	22	13
12	13	22	3	28	7	18	17	8	27	2	23
24	11	14	21	4	29	6	19	16	9	26	1

real and, in general, has even symmetry with respect to precisely one other column in the array (the first column is an exception; it pairs with itself and the same is true for column seven at the half-way point). We say that two sequences **x** and **y** with this kind of symmetry,

$$\mathbf{x}_k = \mathbf{y}_{N-k}, \tag{4.2}$$

are *relatively even.*

Since the columns of the array $X(k_1, k_2)$ are real, half the computational work of transforming complex columns can be saved when the length five DFT is applied. Also, since the columns occur in relatively even pairs, we would hope to further halve the computations. To obtain this reduction we choose R'_N in (3.2) to preserve this symmetry, i.e.,

$$\mathbf{x}_k = \mathbf{y}_{N-k} \quad \text{implies } (R\mathbf{x})_k = (R\mathbf{y})_{N-k} \tag{4.3}$$

(where we have dropped the superscripts and subscripts from the transform). Such a transform would produce, from the array in Table 2, an array of precisely the same type, i.e., one for which

$$X(k_1, k_2) = X(N_1 - k_1, P_1 - k_2) \tag{4.4}$$

and, therefore, a resulting one-dimensional sequence for which $\mathbf{x}_k = \mathbf{x}_{N-k}$. Consequently, applying such a transform in each dimension would produce at each step of the M-dimensional transform a real-even sequence. The first stage of the PFA would thus preserve the original symmetry, yielding a real-even intermediate sequence **z**. Before showing how to obtain R satisfying (4.3) from the DFT we note how the transform stage can be implemented to yield a compact algorithm.

5. Reduction of storage requirements. Suppose that $\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{\alpha(N)-1}$ are stored sequentially in a real array of length $\alpha(N)$, where

$$\alpha(N) = \frac{N}{2} + 1, \quad N \text{ even},$$

$$= \frac{N+1}{2}, \quad N \text{ odd}.$$

In our description of the algorithm so far, redundant calculations are avoided at each step by noting that the columns of the array occur in even pairs and that the DFT of only one column in each pair need be computed. In a standard PFA the locations corresponding to elements in a given column are overwritten by their transform counterparts. However, in the present context some of the indices of these elements will correspond to nonexistent storage locations.

Consider, for example, the second column of the array in Table 1 that contains the indices $\{25, 1, 37, 13, 49\}$. In this case, $\alpha(N) - 1 = 30$ so that the third and fifth indices, 37 and 49, are beyond the range of the storage array. However, since the *input* to this step is even, we may access location $N - k$ when $k > \alpha(N) - 1$ and obtain the correct input values to the short transform. Likewise, when we overwrite this column with its transform we may overwrite location $N - k$ instead of k; this value will be in its correct location since the *result* of the step is an even one-dimensional array. Taking this approach with the second column in Table 1 we see that no action need be taken regarding the last column—all its pertinent storage locations have already been accessed

and overwritten by the correct values. Now, if we transform halfway across this array (stopping after the transform of the seventh column) similar considerations show that all the columns in the second half of the array will, effectively, already have been processed. Therefore, we may proceed to the computations in the next dimension where preservation of symmetry allows us to follow the same track, and so on. In general, across the ith dimension of the M-dimensional array instead of transforming P_i columns, we may transform columns 1 through $\alpha(P_i)$ and continue to the next dimension, or stop in case $i = M$. In the latter case, storage locations zero through $\alpha(N)-1$ will contain the values that determine $\mathbf{z}$ of (3.6). Equation (4.4) implies that the CRT map provides a proper partitioning of the columns for this scheme in that members of an even pair never occur in the same half of the current two-dimensional array.

Note that when updating indices for a new column the procedure is essentially the same as that for a standard PFA with the exception that when a set of column indices has been updated an additional calculation must be performed to determine the actual compact storage location; each index k must be augmented by $k' = \min(k, N-k)$ (we say "augment" rather than "replace" since we need to retain the original index for the next update). This requires extra integer storage of length equal to the longest short factor, N_i.

So far, we have shown that given the existence of an appropriate factorization of the DFT, the first stage of our algorithm can be performed with roughly one quarter the computational work and storage of a complex PFA. The foldings still have to be applied to $\mathbf{z}$; however these foldings can be performed with little effort while preserving the compactness of the algorithm (see § 7).

6. Factorization of the DFT. In this section we show that the DFT can be factored as a transform that preserves relatively even pairs followed by a transform with a simple structure. To fulfill the requirements on the transform R'_N of (3.2) we propose the discrete Hartley transform (DHT) defined for real data by

$$\mathbf{u}_n = \sum_{k=0}^{N-1} \mathbf{v}_k \left(\cos \frac{2\pi kn}{N} + \sin \frac{2\pi kn}{N} \right), \qquad n = 0, 1, \cdots, N-1, \tag{6.1}$$

which we write in matrix form as $H_N \mathbf{v} = \mathbf{u}$. The DHT was first proposed as an alternative to the DFT by Bracewell [1]. Define the transforms C_N and S_N as the cosine and sine parts respectively of the DHT, i.e., $H_N = C_N + S_N$. Let $\mathbf{x}$ be a real sequence and $\mathbf{y}$ be a real sequence relatively even to $\mathbf{x}$. It is easily verified that the following properties hold (dropping the subscript N from the transforms):

(i) $(C\mathbf{x})_k = (C\mathbf{x})_{N-k}$;
(ii) $(S\mathbf{x})_k = -(S\mathbf{x})_{N-k}$;
(iii) $C\mathbf{x} = C\mathbf{y}$;
(iv) $S\mathbf{x} = -S\mathbf{y}$.

Therefore,

$$(H\mathbf{x})_k = (C\mathbf{x})_k + (S\mathbf{x})_k = (C\mathbf{x})_{N-k} - (S\mathbf{x})_{N-k}$$

$$= (C\mathbf{y})_{N-k} + (S\mathbf{y})_{N-k} = (H\mathbf{y})_{N-k},$$

so that H preserves relatively even pairs, as required. In his derivation of a fast Hartley transform, Bracewell [2] made use of the following factorization, which we illustrate with some particular examples, to relate the DHT to the DFT.

Consider the length five DFT represented by W_5. We can factor this matrix as $W_5 = F_5 H_5$, where

$$F_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & (1+i)/2 & 0 & 0 & (1-i)/2 \\ 0 & 0 & (1+i)/2 & (1-i)/2 & 0 \\ 0 & 0 & (1-i)/2 & (1+i)/2 & 0 \\ 0 & (1-i)/2 & 0 & 0 & (1+i)/2 \end{bmatrix},$$

$$H_5 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \operatorname{cas} 2\pi/5 & \operatorname{cas} 4\pi/5 & \operatorname{cas} 6\pi/5 & \operatorname{cas} 8\pi/5 \\ 1 & \operatorname{cas} 4\pi/5 & \operatorname{cas} 8\pi/5 & \operatorname{cas} 12\pi/5 & \operatorname{cas} 16\pi/5 \\ 1 & \operatorname{cas} 6\pi/5 & \operatorname{cas} 12\pi/5 & \operatorname{cas} 18\pi/5 & \operatorname{cas} 24\pi/5 \\ 1 & \operatorname{cas} 8\pi/5 & \operatorname{cas} 16\pi/5 & \operatorname{cas} 24\pi/5 & \operatorname{cas} 32\pi/5 \end{bmatrix},$$

and

$$\operatorname{cas} \theta = \cos \theta + \sin \theta.$$

Here, H_5 represents a length five DHT and F_5 is a folding matrix distinct from the folding matrices of [13]. We obtain a rotated DHT by replacing cas θ with cas $r\theta$ as the general element of H_N. As an even example, we have

$$W_4 = F_4 H_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & (1+i)/2 & 0 & (1-i)/2 \\ 0 & 0 & 1 & 0 \\ 0 & (1-i)/2 & 0 & (1+i)/2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \operatorname{cas} \pi/2 & \operatorname{cas} \pi & \operatorname{cas} 3\pi/2 \\ 1 & \operatorname{cas} \pi & \operatorname{cas} 2\pi & \operatorname{cas} 3\pi \\ 1 & \operatorname{cas} 3\pi/2 & \operatorname{cas} 3\pi & \operatorname{cas} 9\pi/2 \end{bmatrix}.$$

Of course, similar factorizations exist for other transform lengths.

We note at this point that any efficient algorithm for computing a rotated complex DFT of length N_i can be used to produce an efficient rotated DHT of the same length by first deleting the calculations involving the imaginary parts of the input and then summing the real and imaginary parts of the output. In particular, see [12] and [14] for short DFTs that minimize the number of required additions. Also, to further reduce redundant calculations during this stage, short DHTs for even symmetry could be developed to handle sequences such as columns one and seven of Table 2, § 4.

The transform stage represented by (3.6) consists of using the CRT map to perform rotated Hartley transforms along the various dimensions of an M-dimensional array. Redundant calculations and storage locations are avoided by transforming only halfway across each dimension. This stage of the algorithm preserves the symmetry inherent to our problem. The folding stage (3.7), to be described in the next section, involves applying the folding operators F_i to the real-even sequence $\mathbf{z}$.

7. The folding stage. In principle, this stage could be carried out by applying each folding matrix F_i to $\mathbf{z}$ in turn. This strategy would, however, produce intermediate *complex* sequences, thereby nullifying the compact storage scheme prescribed for the transform stage. A more efficient strategy is found by examining the cumulative effect of the foldings on the various components of $\mathbf{z}$. Doing so, we find that a rather simple specification can be made for each of the components of the final result $\mathbf{y}$ in terms of related components of $\mathbf{z}$.

In [13] it has been pointed out that elements in $\mathbf{z}$ fall into $M+1$ disjoint classes $T_0, T_1, \cdots, T_M$, namely, elements affected by none of the folding transforms, elements

affected by a single folding transform, etc., up to elements affected by all M folding transforms, respectively. It has also been noted that within T_i elements separate further into disjoint subgroups that are closed under folding; the final value of any element in a given subgroup is determined only by the values of the other members in that subgroup. Also, each subgroup G in T_i has 2^i elements and $\mathbf{z}_k \in G$ if and only if $\mathbf{z}_{N-k} \in G$. In Temperton's implementation of the real PFA the indices in each subgroup are computed one time by a setup routine and this information is used during the folding stage of each call to the PFA. Since the F_i's in the real case and the real-even case have the same form (the same elements of $\mathbf{z}$ are combined to form the elements in $\mathbf{y}$—only the combine constants differ) this information is the same for both cases. We show how this information is used in the present context by examining each class T_i, $1 \leqq i \leqq 4$. Consideration of these cases will allow this algorithm to be applied to the standard set of relatively prime factors that we usually encounter in the PFA literature, namely {2, 3, 4, 5, 7, 8, 9, 16}. The resulting folding formulas, along with those for the real-odd case, are presented in the Appendix. In this section we summarize those results.

For the class T_0, by definition, no foldings are required. For the class T_1, it turns out that the single folding has no effect. We can show that if $\mathbf{z}_k \in T_1$ then $\mathbf{y}_k = \mathbf{z}_k$.

For the class T_2, each subgroup contains four elements with indices $\{k, k^*, N-k, N-k^*\}$, and we have $\mathbf{y}_k = \mathbf{z}_{k^*}$, and $\mathbf{y}_{k^*} = \mathbf{z}_k$. By considering the compact storage scheme detailed above we see that the elements in $\mathbf{y}$ corresponding to each subgroup can be determined by merely exchanging values in two pertinent storage locations.

So far, no arithmetic operations have been required. This changes with T_i, $i > 2$; however for classes T_3 and T_4 we show in the Appendix that each of the required components of $\mathbf{y}$ can be determined using only one real multiplication and two real additions. When we use compact storage, then, the total operation count for the folding stage for sequence lengths with up to four factors is bounded by $N/2$ real multiplies and N real adds.

8. The real-odd case. We now describe the necessary modifications to the above algorithm when the data to be transformed has real-odd symmetry as defined by (2.5). The result, $\mathbf{y}$, of the DFT is then an imaginary-odd sequence. In this case we see that (4.1) becomes

$$\mathbf{x}_k = -\mathbf{x}_{N-k} \quad \text{iff} \quad X(k_1, k_2) = -X(N_1 - k_1, P_1 - k_2). \tag{8.1}$$

As a result, the columns in Table 1 occur in relatively odd pairs, $\mathbf{x}_k = -\mathbf{y}_{N-k}$. Since the DHT preserves even pairs and is linear, it also preserves this kind of pairwise symmetry. Therefore, the transform stage is essentially the same as before; however, at each step real-odd symmetry is preserved. This fact causes somewhat of a complication in the compact storage scheme; we may no longer simply augment the index k with k' as in § 5; we must now check whether $k > \alpha(N) - 1$. If this is the case we must change the sign of $\mathbf{x}_{k'}$ as input to the short transform, and also as a transform value. This will probably cost two integer comparisons per index per short transform in addition to the sign changes (approximately $N/2$ per dimension), resulting in a significant slow-down in the procedure. Yet, this stage should still be performed considerably faster than in the real case, besides halving the storage. The result is the real-odd sequence $\mathbf{z}$ of (3.6).

When the folding stage is applied to $\mathbf{z}$ it will produce a purely imaginary sequence with odd symmetry. We provide the appropriate folding formulas for this case in the Appendix (A2). Much is the same as before; however, no exchanges are required. For classes T_3 and T_4, we again need only one multiply and two adds per value. As with

the transform stage, it is necessary to check the size of each index and perform possible sign changes accordingly. See the Appendix. Aside from this overhead the folding stage for the two cases have almost the same cost, the real-even case being slightly more expensive due to the required exchanges for class T_2.

9. Concluding remarks. Other symmetries. We have demonstrated the existence of symmetric prime factor algorithms for real-even and real-odd sequences analogous to the symmetric FFTs presented in [11]. These symmetric PFAs roughly quarter the real arithmetic and storage required by the complex PFA, though the odd algorithm is somewhat less efficient than the even one due to additional overhead needed to preserve the compact storage scheme. These algorithms were derived by separating the work of the PFA into two stages, the first of which performs the greater part of the computation, but is able to do so in an efficient way by taking advantage of and preserving the given symmetry. Since symmetry is preserved in the first stage it may also be used by the second stage. We have demonstrated that for sequences with up to four relatively prime factors the latter can be performed with little computational effort.

The reformulation of the PFA presented in § 3 works well for the symmetries considered so far, namely, real, real-even and real-odd symmetries. Yet it is not clear that this approach will work well for other symmetries. Consider, for example, *quarter-wave (even/odd) symmetry*,

$$\mathbf{x}_k = (+/-)\mathbf{x}_{N-k-1}.$$

If the CRT or Ruritanian [5], [10], [12] map is used for Ω_1 and Ω_2 then the columns of the two-dimensional arrays will have pairwise quarterwave symmetry and a factorization of the DFT that preserves this symmetry is not obvious. On the other hand, it may be possible to define a new mapping for Ω_1 and Ω_2 that will produce pairwise even/odd columns so that the factorization used herein is applicable. The question then is whether such a mapping will allow us to rewrite the DFT as (2.6) or in a closely related form. More work needs to be done here. Hopefully, if we can extend the idea of symmetric PFAs to other symmetries then a more general reformulation of the PFA, with wide applicability to symmetry, will emerge.

Appendix. Folding formulae. In this Appendix we present the folding formulas needed to determine the final values of the elements in a typical subgroup of each class T_i, $1 \leqq i \leqq 4$. We assume that the appropriate one-dimensional indices that determine the subgroup have been supplied by a call to a routine designed for the computation of such indices. We do not show how to compute these indices. For a more thorough discussion of this topic see [13].

A1. The real-even case.

(i) Elements in T_1. Given the index pair $\{k, N-k\}$ we have

$$\begin{aligned}\mathbf{y}_k &= \frac{1+i}{2}\mathbf{z}_k + \frac{1-i}{2}\mathbf{z}_{N-k} \\ &= \tfrac{1}{2}\mathbf{z}_k + \tfrac{1}{2}\mathbf{z}_{N-k} \\ &= \mathbf{z}_k,\end{aligned}$$

since $\mathbf{z}$ is even. The folding operations have no effect on elements in T_1.

(ii) Elements in T_2. We are given $\{k, k^*, N-k, N-k^*\}$. Let k correspond to $(k_i, k_j \cdot)$ via the CRT mapping (here "$\cdot$" corresponds to indices in the dimensions

where $\mathbf{z}_k$ is not affected by foldings). Then it follows that

$$\begin{aligned}\mathbf{y}_k &= \frac{1+i}{2}\left[\frac{1+i}{2}Z(k_i, k_j, \cdot)+\frac{1-i}{2}Z(N_i-k_i, k_j, \cdot)\right] \\ &\quad +\frac{1-i}{2}\left[\frac{1+i}{2}Z(k_i, N_j-k_j, \cdot)+\frac{1-i}{2}Z(N_i-k_i, N_j-k_j, \cdot)\right] \\ &= \frac{1}{2}[Z(N_i-k_i, k_j, \cdot)+Z(k_i, N_j-k_j, \cdot)]+\frac{i}{2}[Z(k_i, k_j, \cdot)-Z(N_i-k_i, N_j-k_j, \cdot)] \\ &= \frac{1}{2}[2Z(N_i-k_i, k_j, \cdot)] \\ &= \mathbf{z}_{k^*}.\end{aligned}$$

Similarly, $\mathbf{y}_{k^*}=\mathbf{z}_k$, so that the foldings for T_2 merely exchange the values $\mathbf{z}_k$ and z_{k^*}.

(iii) Elements in T_3. Suppose $\mathbf{z}_A \in T_3$ and let

$$\mathbf{z}_A = Z(k_1, k_2, k_3, \cdot), \qquad \mathbf{z}_B = Z(N_1-k_1, k_2, k_3, \cdot),$$
$$\mathbf{z}_C = Z(k_1, N_2-k_2, k_3, \cdot), \qquad \mathbf{z}_D = Z(k_1, k_2, N_3-k_3, \cdot)$$

(to use this labeling of the k_i's and N_i's we have assumed a possible reordering of the indices) so that the subgroup in T_3 corresponding to $\mathbf{z}_A$ is determined by these four values. Then we can show that

$$\begin{aligned}\mathbf{y}_A &= \left(\frac{1+i}{2}\right)^2\left[\frac{1+i}{2}\mathbf{z}_A+\frac{1-i}{2}\mathbf{z}_B\right]+\left(\frac{1+i}{2}\right)\left(\frac{1-i}{2}\right)\left[\frac{1+i}{2}(\mathbf{z}_C+\mathbf{z}_D)+\frac{1-i}{2}(\mathbf{z}_C+\mathbf{z}_D)\right] \\ &\quad +\left(\frac{1-i}{2}\right)^2\left[\frac{1+i}{2}\mathbf{z}_B+\frac{1-i}{2}\mathbf{z}_A\right] \\ &= \frac{1}{2}[\mathbf{z}_B+\mathbf{z}_C+\mathbf{z}_D-\mathbf{z}_A],\end{aligned}$$

and a similar formula holds for each of $\mathbf{y}_B$, $\mathbf{y}_C$, and $\mathbf{Y}_D$. Now let

$$Q=\mathbf{z}_A+\mathbf{z}_D, \qquad S=\mathbf{z}_A-\mathbf{z}_D,$$
$$T=\mathbf{z}_B+\mathbf{z}_C, \qquad R=\mathbf{z}_B-\mathbf{z}_C;$$

then we can write

$$y_A=\tfrac{1}{2}[T-S], \qquad y_B=\tfrac{1}{2}[Q-R],$$
$$y_C=\tfrac{1}{2}[Q+R], \qquad y_D=\tfrac{1}{2}[T+S].$$

Using these formulas we see that the four pertinent components of $\mathbf{y}$ associated with $\mathbf{y}_A$ can be computed using two additions and one multiplication per value.

(iv) Elements in T_4. Suppose $\mathbf{z}_A \in T_4$, and let

$$\mathbf{z}_A = Z(k_1, k_2, k_3, k_4, \cdot), \qquad \mathbf{z}_B = Z(N_1-k_1, k_2, k_3, k_4, \cdot),$$
$$\mathbf{z}_C = Z(k_1, N_2-k_2, k_3, k_4, \cdot), \qquad \mathbf{z}_D = Z(N_1-k_1, N_2-k_2, k_3, k_4, \cdot),$$
$$\mathbf{z}_E = Z(k_1, k_2, N_3-k_3, k_4, \cdot), \qquad \mathbf{z}_F = Z(N_1-k_1, k_2, N_3-k_3, k_4, \cdot),$$
$$\mathbf{z}_G = Z(k_1, k_2, k_3, N_4-k_4, \cdot), \qquad \mathbf{z}_H = Z(N_1-k_1, k_2, k_3, N_4-k_4, \cdot).$$

Then it can be shown that

$$\begin{aligned}
y_A &= \left(\frac{1+i}{2}\right)^3\left[\frac{1+i}{2}\mathbf{z}_A+\frac{1-i}{2}\mathbf{z}_B\right] \\
&\quad+\left(\frac{1+i}{2}\right)^2\left(\frac{1-i}{2}\right)\left[\frac{1+i}{2}(\mathbf{z}_C+\mathbf{z}_E+\mathbf{z}_G)+\frac{1-i}{2}(\mathbf{z}_D+\mathbf{z}_F+\mathbf{z}_H)\right] \\
&\quad+\left(\frac{1-i}{2}\right)^2\left(\frac{1+i}{2}\right)\left[\frac{1+i}{2}(\mathbf{z}_D+\mathbf{z}_F+\mathbf{z}_H)+\frac{1-i}{2}(\mathbf{z}_C+\mathbf{z}_E+\mathbf{z}_G)\right] \\
&\quad+\left(\frac{1-i}{2}\right)^3\left[\frac{1+i}{2}\mathbf{z}_B+\frac{1-i}{2}\mathbf{z}_A\right] \\
&= \frac{1}{2}[\mathbf{z}_D+\mathbf{z}_F+\mathbf{z}_H-\mathbf{z}_A].
\end{aligned}$$

Similarly, we have

$$\begin{aligned}
y_B &= \tfrac{1}{2}[\mathbf{z}_C+\mathbf{z}_E+\mathbf{z}_G-\mathbf{z}_B], & \mathbf{y}_C &= \tfrac{1}{2}[\mathbf{z}_B+\mathbf{z}_E+\mathbf{z}_G-\mathbf{z}_C] \\
y_D &= \tfrac{1}{2}[\mathbf{z}_A+\mathbf{z}_F+\mathbf{z}_H-\mathbf{z}_D], & y_E &= \tfrac{1}{2}[\mathbf{z}_G+\mathbf{z}_B+\mathbf{z}_C-\mathbf{z}_E], \\
y_F &= \tfrac{1}{2}[\mathbf{z}_A+\mathbf{z}_D+\mathbf{z}_H-\mathbf{z}_F], & y_G &= \tfrac{1}{2}[\mathbf{z}_B+\mathbf{z}_C+\mathbf{z}_E-\mathbf{z}_G], \\
& & y_H &= \tfrac{1}{2}[\mathbf{z}_A+\mathbf{z}_D+\mathbf{z}_F-\mathbf{z}_H].
\end{aligned}$$

Now let

$$\begin{aligned}
P &= \mathbf{z}_A-\mathbf{z}_D, & Q &= \mathbf{z}_A+\mathbf{z}_D, \\
R &= \mathbf{z}_B-\mathbf{z}_C, & S &= \mathbf{z}_B+\mathbf{z}_C, \\
T &= \mathbf{z}_E+\mathbf{z}_G, & U &= \mathbf{z}_E-\mathbf{z}_G, \\
V &= \mathbf{z}_F+\mathbf{z}_H, & W &= \mathbf{z}_F-\mathbf{z}_H,
\end{aligned}$$

so that

$$\begin{aligned}
\mathbf{z}_A &= \tfrac{1}{2}[V-P], & \mathbf{z}_B &= \tfrac{1}{2}[T-R], & \mathbf{z}_C &= \tfrac{1}{2}[T+R], \\
\mathbf{z}_D &= \tfrac{1}{2}[V+P], & \mathbf{z}_E &= \tfrac{1}{2}[S-U], & \mathbf{z}_F &= \tfrac{1}{2}[Q-W], \\
& & \mathbf{z}_G &= \tfrac{1}{2}[S+U], & \mathbf{z}_H &= \tfrac{1}{2}[Q+W].
\end{aligned}$$

As with T_3 we see that the values in $\mathbf{y}$ corresponding to T_4 can be computed using just two additions and one multiplication for each value.

Note, in the above we assume that for computational purposes each one-dimensional index k is replaced by the corresponding index $k' = \min(k, N-k)$, in accordance with the storage scheme of the transform stage.

A2. The real-odd case.

(i) Elements in T_1. As above,

$$\mathbf{y}_k = \frac{1+i}{2}\mathbf{z}_k+\frac{1-i}{2}\mathbf{z}_{N-k}.$$

But now $\mathbf{z}$ has odd symmetry, so that

$$\begin{aligned}
\mathbf{y}_k &= \frac{i}{2}\mathbf{z}_k-\frac{i}{2}\mathbf{z}_{N-k} \\
&= i\mathbf{z}_k,
\end{aligned}$$

and since we just need the imaginary part of **y**, this operation has no effect on the corresponding components of **z**.

(ii) Elements in T_2. As above, we have

$$\begin{aligned}\mathbf{y}_k &= \frac{1}{2}[Z(N_i - k_i, k_j, \cdot) + Z(k_i, N_j - k_j, \cdot)] + \frac{i}{2}[Z(k_i, k_j, \cdot) - Z(N_i - k_i, N_j - k_j, \cdot)] \\ &= \frac{i}{2}[2Z(k_i, k_j, \cdot)] \qquad \text{(since } \mathbf{z} \text{ is odd)} \\ &= i\mathbf{z}_k,\end{aligned}$$

so that for the odd case folding has no effect on these components.

In the next two cases we assume the following procedure. Given an index B, say, check if $B > \alpha(N) - 1$. If so, change the sign of $\mathbf{z}_{N-B}$ before it is used in a calculation. Change the sign of $\mathbf{y}_B$ given by the formula below and store the result in location $N - B$.

(iii) Elements in T_3. As in the even case, suppose we are supplied with the indices (A through D of A1) of the components that determine a subgroup of T_3. Then with Q, R, S, and T defined as in A1(iii) we have

$$\mathbf{y}_A = \frac{i}{2}[T + S], \qquad \mathbf{y}_B = \frac{i}{2}[Q + R],$$

$$\mathbf{y}_C = \frac{i}{2}[Q - R], \qquad \mathbf{y}_D = \frac{i}{2}[T - S].$$

(iv) Elements in T_4. Suppose indices A through H correspond to components of a subgroup as in A1(iv). Let quantities Q through W also be defined as in that section. Then

$$\mathbf{y}_A = \frac{i}{2}[S + T], \quad \mathbf{y}_B = \frac{i}{2}[Q + V], \quad \mathbf{y}_C = \frac{i}{2}[Q - V],$$

$$\mathbf{y}_D = \frac{i}{2}[S - T], \quad \mathbf{y}_E = \frac{i}{2}[P + W], \quad \mathbf{y}_F = \frac{i}{2}[R + U],$$

$$\mathbf{y}_G = \frac{i}{2}[P - W], \qquad \mathbf{y}_H = \frac{i}{2}[R - U].$$

Acknowledgments. The author thanks Bill Briggs and Roland Sweet for motivating this work, for their reading of preliminary drafts of this paper, and their helpful comments and criticisms.

REFERENCES

[1] R. N. BRACEWELL, *Discrete Hartley transform*, J. Opt. Soc. Amer., 73 (1983), pp. 1832-1835.

[2] ———, *The fast Harley transform*, Proc. IEEE, 72 (1984), pp. 1010-1018.

[3] W. L. BRIGGS, *Further symmetries of in-place* FFTs, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 644-654.

[4] C. Y. CHU, *The fast Fourier transform on hypercube parallel computers*, Ph.D. thesis, Center for Applied Mathematics, Cornell University, Ithaca, NY, 1988. Also available as Tech. Report 87-882, Department of Computer Science, Cornell University, Ithaca, NY.

[5] 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.

[6] J. W. COOLEY, P. A. LEWIS, AND P. D. WELCH, *The fast Fourier transform algorithm: Programming considerations in the calculation of sine, cosine and Laplace transforms*, J. Sound Vib., 12 (1970), pp. 315-337.

[7] J. W. COOLEY AND J. W. TUKEY, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297-301.

[8] O. K. ERSOY, *A unified approach to the fast computation of all discrete Fourier transforms*, Proc. 1987 IEEE Internat. Conference on Acoustics Speech and Signal Processing, Dallas, TX, pp. 1843-1846.

[9] W. M. GENTLEMAN, *Implementing Clenshaw-Curtis quadrature.* II. *Computing the cosine transformation*, Comm. ACM, 15 (1972), pp. 343-346.

[10] I. J. GOOD, *The interaction algorithm and practical Fourier series*, J. Roy. Soc. London Ser. B, 20 (1958), pp. 361-372; Addendum, 22 (1960), pp. 372-375.

[11] P. SWARZTRAUBER, *Symmetric* FFTs, Math. Comp., 47 (1986), pp. 323-346.

[12] C. TEMPERTON, *Implementation of a self-sorting in-place prime factor* FFT *algorithm*, J. Comput. Phys., 58 (1985), pp. 283-299.

[13] ———, *A self-sorting in-place prime factor real/half-complex* FFT *algorithm*, J. Comput. Phys., to appear.

[14] ———, *A new set of minimum-add small-N rotated* DFT *modules*, J. Comput. Phys., to appear.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 432-446, May 1989
© 1989 Society for Industrial and Applied Mathematics
003

DISTRIBUTION OF THE AVERAGE POWER OF A NORMAL TIME SERIES*

CARL W. HELSTROM†

Abstract. The distribution of the sum of the squares of N correlated and normally distributed elements of a time series can be computed by numerical quadrature of a Laplace inversion integral involving the moment generating function (m.g.f.) of the sum. A method is presented for computing that m.g.f. for a stationary autoregressive moving-average (ARMA) process whose spectral density is a known rational function with $2n$ poles. It requires evaluating determinants of $2n \times 2n$ and $(2n+1)\times(2n+1)$ matrices, which may be much smaller than the $N \times N$ covariance matrix of the time series. A second method is described that is based on the Kalman equations and applies to time series, possibly nonstationary, generated by a discrete-time linear system driven by normal random noise. A third method, utilizing the Levinson algorithm, applies when the time series is merely stationary.

Key words. time series, inverse Laplace transform, numerical integration, probability distribution, moment generating function, rational spectral density, ARMA process, Kalman filtering, Levinson algorithm, saddlepoint integration

AMS(MOS) subject classification. 60E10

1. Introduction. The normal random variables $v_1, v_2, \cdots, v_N$ are N successive elements of a time series. Their expected values are denoted by

$$s_k = E(v_k), \qquad 1 \leqq k \leqq N, \tag{1.1}$$

and form what we shall call the signal vector **s**. Their $N \times N$ covariance matrix **A** is composed of the covariances

$$a_{ij} = E[(v_i - s_i)(v_j - s_j)]. \tag{1.2}$$

The average power of this time series is defined as

$$P_{\text{av}} = N^{-1} \sum_{k=1}^{N} v_k^2, \tag{1.3}$$

and its cumulative probability distribution is to be computed. To simplify our formulas we treat instead the sum of squares

$$V' = \frac{1}{2} \sum_{k=1}^{N} v_k^2 = \frac{NP_{\text{av}}}{2}, \tag{1.4}$$

which is proportional to P_{av}.

The calculation of the distribution $\Pr(V' < V)$ of V' is a special case of the extensively treated problem of calculating the distribution of an arbitrary quadratic form in correlated normal variates. A long list of references is found in Chapter 29 of Johnson and Kotz [10]. Recent work by Gideon and Gurland [4], Rice [16], and Brown [2] should also be cited. In most of these treatments the distribution is expressed in powers of V or in a series of Laguerre or chi-squared distributions. An exception is the paper by Rice [16], in which the Laplace transform of the distribution is inverted by numerical integration in the complex plane.

* Received by the editors June 8, 1987; accepted for publication June 20, 1988. This work was performed while the author was at the Laboratoire des Signaux et Systèmes, Centre National de Recherche Scientifique-ESE, Plateau du Moulon, F91190 Gif-sur-Yvette, France.

† Department of Electrical and Computer Engineering, R-007, University of California, San Diego; La Jolla, California 92093.

All of these methods presuppose that the quadratic form has been reduced to a weighted sum of squares of uncorrelated random variables. In the special case of V' in (1.4) this requires finding the N eigenvalues and the N eigenvectors of the covariance matrix $\mathbf{A}$, a lengthy computation when the number N of terms in (1.4) is large. This paper considers three situations in which it can be avoided: (i) the time series v_1, $v_2, \cdots, v_N$ is stationary and possesses a known rational spectral density; (ii) the random components $v'_k = v_k - s_k$ of the series are generated by a discrete-time linear system driven by normal random noise; and (iii) the v'_k constitute a stationary time series, whose covariance matrix $\mathbf{A}$ is known to have the Toeplitz form

$$a_{ij} = a_{i-j}. \tag{1.5}$$

Although the autoregressive moving-average (ARMA) process of (i) is a special case of both (ii) and (iii), it permits so efficient a method that it will be treated separately.

The cumulative distribution of the sum of squares V' is to be computed, as in [16], by numerical quadrature of the inverse Laplace transform of its moment generating function (m.g.f.)

$$\begin{aligned} h(z) &= E(e^{-zV'}), \\ q_-(V) &= \Pr(V' < V) = \int_{C_+} z^{-1} h(z)\, e^{Vz}\, dz/2\pi i, \end{aligned} \tag{1.6}$$

in which the contour C_+, initially a straight vertical line to the right of the origin, has been deformed in such a way as to make the numerical integration especially efficient. The complementary cumulative distribution is similarly

$$q_+(V) = \Pr(V' > V) = \int_{C_-} (-z)^{-1} h(z)\, e^{Vz}\, dz/2\pi i, \tag{1.7}$$

in which the contour C_- passes to the left of the origin, but to the right of all the singularities of the m.g.f. $h(z)$. We compute $q_-(V)$ for $V < E(V')$, $q_+(V)$ for $V > E(V')$.

In Rice's method [16] the contours of integration are straight lines inclined at a suitable angle to the Re z-axis. We have economized the number of steps of numerical quadrature by adopting a parabolic contour passing through a saddlepoint of the integrand of (1.6) or (1.7) and having the same curvature there as the path of steepest descent of the integrand [5]. Because the path of steepest descent has nearly a parabolic form, the actual contour of integration lies close to it, and along that contour the value of the integrand decreases rapidly as the point z moves away from the saddlepoint.

The m.g.f. of the variate V' in (1.4) is [10, Chap. 29]

$$h(z) = [\det(\mathbf{I} + z\mathbf{A})]^{-1/2} \exp[-\tfrac{1}{2} z \mathbf{s}^+ (\mathbf{I} + z\mathbf{A})^{-1} \mathbf{s}], \tag{1.8}$$

where $\mathbf{I}$ is the $N \times N$ identity matrix, $\mathbf{A}$ is the covariance matrix of the v_k's, $\mathbf{s}$ is the column vector of mean values s_k, and $\mathbf{s}^+$ is its transposed row vector.

The technique of saddlepoint integration is described in [5] and [6], and we shall not review it here in detail. We write the integrand of (1.6) and (1.7) as $\exp[\Psi(z)]$, where the "phase" $\Psi(z)$ is

$$\Psi(z) = Vz - \tfrac{1}{2} \ln \det(\mathbf{I} + z\mathbf{A}) - \tfrac{1}{2} z \mathbf{s}^+ (\mathbf{I} + z\mathbf{A})^{-1} \mathbf{s} - \ln(\pm z); \tag{1.9}$$

$(\pm z) = z$ in (1.6), $(\pm z) = -z$ in (1.7). The saddlepoint of $\Psi(z)$ through which the contour of integration passes lies on the Re z-axis and is the root of

$$\frac{d\Psi}{dx} = \Psi'(x) = 0, \qquad x = \operatorname{Re} z. \tag{1.10}$$

For $V<E(V')$ we use the saddlepoint to the right of the origin. For $V>E(V')$ the saddlepoint lies in the segment $x_1=-\lambda_1^{-1}<x<0$, where λ_1 is the largest eigenvalue of the covariance matrix $\mathbf{A}$. It is advisable to compute x_1 at the beginning by solving the equation

$$\det(\mathbf{I}+x_1\mathbf{A})=0$$

by any convenient root-finding algorithm, in which we can use whichever of the formulas given for this determinant in the following sections is most appropriate. The saddlepoint is efficiently located by solving (1.10) by Newton's method, computing the derivatives involved numerically.

This paper is devoted to methods of computing the m.g.f. $h(z)$ at points z on the real axis, as during the search for the saddlepoint, and on the contour of integration in the three cases (i), (ii), and (iii). The results apply whether the contour is a straight line, as in Rice's method [16] and in [6], or a parabola, as in [5] and [7]. In case (i), analyzed in § 2, the determinant and the quadratic form in (1.8) and (1.9) can be expressed as determinants of $2n\times 2n$ and $(2n+1)\times(2n+1)$ matrices, respectively, where $2n$ is the number of poles of the rational spectral density of the time series v_1', $v_2', \cdots, v_N'$; N may be much larger than $2n$. Standard computer routines for solving algebraic equations and evaluating determinants can be applied. Case (ii) is treated in § 3 by adapting the state-variable method of Kalman [11]. Section 4 shows how the elements of the m.g.f. $h(z)$ can be calculated in case (iii) by the Levinson algorithm [13].

In analyses of radar and communications systems, as in the work of Kanter on correlated Rayleigh-fading signals [12], we need the distribution of a similar sum of squares,

$$V''=\frac{1}{2}\sum_{k=1}^{N}|w_k|^2, \tag{1.11}$$

in which the w_k are circular complex Gaussian random variables with complex means $E(w_k)=s_k$ and complex covariances:

$$\tfrac{1}{2}E[(w_k^*-s_k^*)(w_l-s_l)]=a_{kl}, \qquad E[(w_k-s_k)(w_l-s_l)]=0. \tag{1.12}$$

The m.g.f. of V'' is much like that in (1.8),

$$E(e^{-zV''})=\det(\mathbf{I}+z\mathbf{A})^{-1}\exp[-\tfrac{1}{2}z\mathbf{s}^+(\mathbf{I}+z\mathbf{A})^{-1}\mathbf{s}], \tag{1.13}$$

in which $\mathbf{A}=\|a_{kl}\|=\mathbf{A}^+$ and $^+$ denotes the transpose conjugate. The methods to be described here can be directly extended to computing the distribution of V'' as well.

2. Autoregressive moving-average process. The time series $v_k'=v_k-s_k$, $1\leqq k\leqq N$, is now assumed to be stationary and to have autocovariances a_{i-j} such that its spectral density

$$A(w)=\sum_{p=-\infty}^{\infty}a_p w^{-p}, \qquad a_p=a_{-p}, \tag{2.1}$$

is a rational function of w with $2n<N$ poles $\mu_1, \cdots, \mu_{2n}$; that is, it is an ARMA process. We assume at the beginning that these are simple poles. Of these poles, $\mu_1, \cdots, \mu_n$ lie outside the unit circle and are called *exterior* poles; $\mu_{n+1}, \cdots, \mu_{2n}$ lie inside and are *interior* poles; $\mu_{k+n}=\mu_k^{-1}$, $1\leqq k\leqq n$. The spectral density may have the form

$$A(w)=R+\Phi(w) \tag{2.2}$$

with $\Phi(w)\to 0$ as $w\to 0$ and $w\to\infty$; R accounts for a white-noise component of the random process and contributes only to a_0.

Our formulas involve the $2n$ roots $\beta_1, \beta_2, \cdots, \beta_{2n}$ of the algebraic equation of degree $2n$ arising from the equation

$$1 + zA(w) = 0, \qquad w = \beta_1, \cdots, \beta_{2n}. \tag{2.3}$$

These roots are arranged so that $\beta_1, \cdots, \beta_n$ lie outside the unit circle and the rest inside; $\beta_{k+n} = \beta_k^{-1}$, $1 \leqq k \leqq n$.

We suppose that the sequence s_p of mean values has been expressed in the form

$$s_p = \sum_{r=1}^{N} \sigma_r u_r^{p-1}, \qquad 1 \leqq p \leqq N, \tag{2.4}$$

where

$$u_r = \exp[2\pi i(r-1)/N]. \tag{2.5}$$

The σ_r are the discrete Fourier coefficients

$$\sigma_r = N^{-1} \sum_{p=1}^{N} s_p u_p^{-(r-1)}, \qquad 1 \leqq r \leqq N. \tag{2.6}$$

If, as often, the mean values are constant, $s_p \equiv s$, then $\sigma_1 = s$ and $\sigma_r \equiv 0$, $r > 1$.

In the Appendix we show that the m.g.f. $h(z)$ in (1.8) can be expressed as

$$h(z) = [D(z)]^{-1/2} \exp[-\tfrac{1}{2} z Q(z)], \tag{2.7}$$

where

$$\begin{aligned} D(z) &= \det(\mathbf{I} + z\mathbf{A}) \\ &= (1+zR)^N \left(\prod_{k=n+1}^{2n} \mu_k^N \right) \frac{\det \mathbf{G}_N(z)}{\det \mathbf{G}_0(z)}. \end{aligned} \tag{2.8}$$

Here $\mathbf{G}_N(z)$ is a $2n \times 2n$ matrix whose elements are

$$\begin{aligned} G_{jk}^{(N)} &= \frac{\beta_k^N}{\beta_k - \mu_j}, \qquad 1 \leqq j \leqq n, \quad 1 \leqq k \leqq 2n, \\ &= \frac{1}{\beta_k - \mu_j}, \qquad n+1 \leqq j \leqq 2n, \quad 1 \leqq k \leqq 2n. \end{aligned} \tag{2.9}$$

The elements of $\mathbf{G}_0(z)$ are obtained by setting $N = 0$ in (2.9).

The quadratic form in (1.8) and (2.7) is

$$Q(z) = \mathbf{s}^+(\mathbf{I} + z\mathbf{A})^{-1}\mathbf{s} = J_1(z) + J_2(z), \tag{2.10}$$

where

$$J_1(z) = N \sum_{j=1}^{N} \frac{|\sigma_j|^2}{1 + zA(u_j)} \tag{2.11}$$

and

$$\begin{aligned} J_2(z) &= [\det \mathbf{G}_N(z)]^{-1} \det \bar{\mathbf{G}}_N(z), \\ \bar{\mathbf{G}}_N(z) &= \begin{bmatrix} 0 & \mathbf{E}^+ \\ \mathbf{F} & \mathbf{G}_N \end{bmatrix}, \end{aligned} \tag{2.12}$$

in which $\mathbf{E}^+$ is a $2n$-element row vector with elements

$$E_k = \sum_{r=1}^{N} \sigma_r^* \frac{1 - \beta_k^N}{1 - \beta_k u_r^{-1}}, \tag{2.13}$$

and $\mathbf{F}$ is a $2n$-element column vector with elements

$$F_j = \sum_{r=1}^{N} \frac{\sigma_r}{[1+zA(u_r)](u_r-\mu_j)}. \tag{2.14}$$

The u_r's are given in (2.5). These summations contain a single term ($r=1$) when the means s_p are equal. The matrix $\bar{\mathbf{G}}_N(z)$ in (2.12) is $(2n+1)\times(2n+1)$.

The sizes of the matrices involved in this method depend only on the order n of the time series $v'_1, \cdots, v'_N$, and n may be much smaller than the number N of terms in the sum in (1.4) and the numbers of rows and columns of the covariance matrix $\mathbf{A}$ in (1.8). The m.g.f. $h(z)$ can be calculated, through (2.8)-(2.14), by standard computer routines for solving algebraic equations with complex coefficients and for evaluating determinants of complex matrices. With $y = w + w^{-1}$, (2.3) becomes an algebraic equation of degree n in y, and each root y yields an interior and an exterior β as solutions of the quadratic equation

$$\beta^2 - y\beta + 1 = 0.$$

When N is large, the matrices $\mathbf{G}_N(z)$ and $\bar{\mathbf{G}}_N(z)$ in (2.9) and (2.12) should be modified to avoid overflow. With the exterior β's indexed from 1 through n, we factor β_k^N from the kth of the first n columns of $\det \mathbf{G}_N(z)$ and from column $k+1$ of $\det \bar{\mathbf{G}}_N(z)$, $1 \leqq k \leqq n$. Then for use in $\Psi(z)$ in (1.9), (2.8) is replaced by

$$\ln \det (\mathbf{I} + z\mathbf{A}) = N \ln (1+zR) + N \sum_{k=1}^{n} \ln\left(\frac{\beta_k}{\mu_k}\right) + \ln\left[\frac{\det \mathbf{H}_N(z)}{\det \mathbf{G}_0(z)}\right], \tag{2.15}$$

where the first n columns of $\mathbf{H}_N(z)$ have elements

$$\begin{aligned} H_{jk}^{(N)} &= \frac{1}{\beta_k - \mu_j}, \quad 1 \leqq j \leqq n, \quad 1 \leqq k \leqq n, \\ &= \frac{\beta_k^{-N}}{\beta_k - \mu_j}, \quad n+1 \leqq j \leqq 2n, \quad 1 \leqq k \leqq n; \qquad \mu_{j+n} = \mu_j^{-1}, \quad 1 \leqq j \leqq n. \end{aligned} \tag{2.16}$$

The last n columns are the same as in $\mathbf{G}_N(z)$ defined by (2.9). The first n elements of the row vector $\mathbf{E}^+$ now become

$$E'_k = \sum_{r=1}^{N} \sigma_r^* \frac{\beta_k^{-N} - 1}{1 - \beta_k u_r^{-1}}, \qquad 1 \leqq k \leqq n; \tag{2.17}$$

the last n elements remain the same as in (2.13). With $\mathbf{E}'^+$ denoting the modified vector, (2.12) becomes

$$\begin{aligned} J_2(z) &= [\det \mathbf{H}_N(z)]^{-1} \det \bar{\mathbf{H}}_N(z), \\ \bar{\mathbf{H}}_N(z) &= \begin{bmatrix} 0 & \mathbf{E}'^+ \\ \mathbf{F} & \mathbf{H}_N \end{bmatrix}. \end{aligned} \tag{2.18}$$

If the spectral density $\Phi(w)$ possesses a double pole, that pole can be thought of as composed of two poles, say μ_1 and μ_2, separated by an infinitesimal distance ε. The two associated rows of each of the three matrices $\mathbf{G}_N$, $\mathbf{G}_0$, and $\bar{\mathbf{G}}_N$ in (2.8) and (2.12) will be nearly equal as long as ε is finite. Replace the second row by the difference of the first two rows. The value of the determinant is thereby unchanged. The difference is approximately ε times the derivative of the first row with respect to μ_1. The ε's will cancel from (2.8) and (2.12), and the second row of each matrix will have been replaced by the derivative of the first with respect to μ_1; its elements will be

$$\beta_k^N (\beta_k - \mu_1)^{-2}, \qquad 1 \leqq k \leqq 2n.$$

As $\mu_{n+1} = \mu_1^{-1}$ will then also be a double pole of $\Phi(w)$, row $n+2$ of each of the matrices must be similarly modified. Its elements will be

$$(\beta_k - \mu_{n+1})^{-2}, \qquad 1 \leqq k \leqq 2n.$$

The second and $(n+1)$th element of the column vector $\mathbf{F}$ must also be modified in this way. By an extension of this confluential procedure, expressions for the determinant $D(z)$ and for $J_2(z)$ can be derived as needed for time series whose spectral densities have multiple poles, and there is no point to our writing them here.

Continuous-time versions of (2.8), (2.11), and (2.12) are found in [7] and [8]. They have similar forms, but the variable w in (2.2) et seq. is replaced by the angular frequency ω, $-\infty < \omega < \infty$, and factors such as β^N are replaced by $\exp(\beta T)$, where T is the observation interval.

3. State-variable formulation. When as in § 2 the spectral density $A(w)$ of the process $v_1', \cdots, v_N'$ is a rational function of w, the process can be considered as generated by a discrete-time linear system driven by normal (Gaussian) noise. The terms in the m.g.f. $h(z)$ in (1.8) can then be computed by an adaptation of the prediction and estimation equations of Kalman [11]. If the system is not time invariant or the driving noise is nonstationary, the series $v_1', \cdots, v_N'$ will be nonstationary, and the method of § 2 will be inapplicable. We therefore present this adapted Kalman method in a form suitable for a nonstationary time series. How to determine an appropriate state-variable model for generating a given time series has been treated in the literature [14] and will not concern us here.

The n state variables of this discrete-time system at "time" k constitute a column vector $\mathbf{x}_k'$, which evolves according to the vector state equation

$$\mathbf{x}_{k+1}' = \mathbf{H}_k \mathbf{x}_k' + \mathbf{w}_{k+1}', \qquad k = 0, 1, 2, \cdots, N-1, \tag{3.1}$$

in which $\mathbf{H}_k$ is the $n \times n$ dynamical matrix and $\mathbf{w}_k'$ is a sequence of independently random n-vectors with normally distributed components whose mean values are zero and whose $n \times n$ covariance matrix is

$$\mathbf{Q}_k' = E(\mathbf{w}_k' \mathbf{w}_k'^{+}); \tag{3.2}$$

$^+$ indicates the transpose. The $n \times n$ initial covariance matrix

$$\mathbf{\Phi}_0' = E(\mathbf{x}_0' \mathbf{x}_0'^{+}) \tag{3.3}$$

is assumed known. From the state vector $\mathbf{x}_k'$ the noise components $v_k - s_k = v_k'$ of our time series are determined by

$$v_k' = \mathbf{M}_k \mathbf{x}_k' + u_k', \qquad k = 1, 2, \cdots, N, \tag{3.4}$$

in which $\mathbf{M}_k$ is an n-element row vector and the u_k' are independent normally distributed variates with mean zero and known variances R_k'. The process $v_1', v_2', \cdots, v_N'$ has covariance matrix $\mathbf{A}$, which is a function of $\mathbf{M}_k$, $\mathbf{Q}_k'$, $\mathbf{H}_k$, R_k', and $\mathbf{\Phi}_0'$, $1 \leqq k \leqq N$.

In the m.g.f. $h(z)$ of (1.8), as we shall demonstrate,

$$\det(\mathbf{I} + z\mathbf{A}) = \prod_{k=1}^{N} (\mathbf{M}_k \mathbf{P}_k \mathbf{M}_k^{+} + R_k), \tag{3.5}$$

and

$$\mathbf{s}^{+}(\mathbf{I} + z\mathbf{A})^{-1}\mathbf{s} = \sum_{k=1}^{N} \frac{(s_k - \mathbf{M}_k \mathbf{H}_{k-1} \boldsymbol{\sigma}_{k-1})^2}{\mathbf{M}_k \mathbf{P}_k \mathbf{M}_k^{+} + R_k}. \tag{3.6}$$

Here

$$R_k = 1 + zR'_k \tag{3.7}$$

and as in the Kalman equations [11]

$$\mathbf{P}_{k+1} = \mathbf{H}_k \mathbf{\Phi}_k \mathbf{H}_k^+ + z\mathbf{Q}'_{k+1}, \qquad 0 \leqq k \leqq N-1, \tag{3.8}$$

and

$$\mathbf{\Phi}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{M}_k)\mathbf{P}_k, \qquad 1 \leqq k \leqq N, \tag{3.9}$$

with

$$\mathbf{K}_k = \mathbf{P}_k \mathbf{M}_k^+ (\mathbf{M}_k \mathbf{P}_k \mathbf{M}_k^+ + R_k)^{-1} \tag{3.10}$$

an n-element column vector, and furthermore

$$\boldsymbol{\sigma}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{M}_k)\mathbf{H}_{k-1}\boldsymbol{\sigma}_{k-1} + \mathbf{K}_k \mathbf{s}_k, \qquad \boldsymbol{\sigma}_0 = \mathbf{0}, \tag{3.11}$$

$\mathbf{0}$ a column vector of n zeros.

Starting with the known initial conditions $\mathbf{\Phi}_0 = z\mathbf{\Phi}'_0$ and $\boldsymbol{\sigma}_0 = \mathbf{0}$, we generate—for each value of z needed in evaluating (1.6) and (1.7)—the determinant in (3.5) and the quadratic form in (3.6), substituting these then into (1.8) to calculate the m.g.f. $h(z)$. To avoid overflow when $N \gg n$, the logarithm of $\det(\mathbf{I} + z\mathbf{A})$ can be accumulated for substitution into $\Psi(z)$ of (1.9).

Taking z real and positive for the nonce, we define new normal random variables by

$$\mathbf{x}_{k+1} = \mathbf{H}_k \mathbf{x}_k + \mathbf{w}_{k+1}, \tag{3.12}$$

$$y_k = \mathbf{M}_k \mathbf{x}_k + u_k, \tag{3.13}$$

in which the covariance matrix of the vector $\mathbf{w}_k$ is

$$\mathbf{Q}_k = E(\mathbf{w}_k \mathbf{w}_k^+) = z\mathbf{Q}'_k \tag{3.14}$$

and the variances of the independent random variables u_k are R_k as defined in (3.7). According to our construction, the covariance matrix of the time series $y_1, y_2, \cdots, y_N$ is then $\mathbf{I} + z\mathbf{A}$, and the joint probability density function (p.d.f.) of these normal random variables is

$$p(y_1, y_2, \cdots, y_N) = (2\pi)^{-N/2}[\det(\mathbf{I} + z\mathbf{A})]^{-1/2} \exp[-\tfrac{1}{2}\mathbf{y}^+(\mathbf{I} + z\mathbf{A})^{-1}\mathbf{y}], \tag{3.15}$$

where $\mathbf{y}$ is the column vector of the y_k's and $\mathbf{y}^+$ its transposed row vector. Comparing (3.15) with (1.8) we see that the same determinants figure in both and that the quadratic form in (3.15) is the same as that in (1.8), except that $\mathbf{y}$ is replaced by $\mathbf{s}$.

Denoting the partial sequence $(y_1, y_2, \cdots, y_k)$ by Y_k, and following Jazwinski [9, Chaps. 5, 7], we express the p.d.f. in (3.15) as

$$p(y_1, y_2, \cdots, y_N) = \prod_{k=1}^{N} p(y_k \mid Y_{k-1}), \tag{3.16}$$

where $p(y_k \mid Y_{k-1})$ is the conditional normal p.d.f. of y_k, given the previous values Y_{k-1};

$$p(y_1 \mid Y_0) = p(y_1). \tag{3.17}$$

As shown in deriving the Kalman equations [9], the y_k have conditional expected values

$$E(y_k \mid Y_{k-1}) = \hat{y}_k = \mathbf{M}_k \mathbf{H}_{k-1} \hat{\mathbf{x}}_{k-1}, \tag{3.18}$$

where

$$\hat{\mathbf{x}}_k = E(\mathbf{x}_k \mid Y_{k-1}) = \mathbf{H}_{k-1}\hat{\mathbf{x}}_{k-1} + \mathbf{K}_k(y_k - \hat{y}_k), \qquad \hat{\mathbf{x}}_0 = \mathbf{0}, \tag{3.19}$$

with $\mathbf{K}_k$ defined by (3.10). The conditional variance of y_k, given Y_{k-1}, is

$$\operatorname{Var}(y_k \mid Y_{k-1}) = \mathbf{M}_k\mathbf{P}_k\mathbf{M}_k^+ + R_k, \tag{3.20}$$

in which the $\mathbf{P}_k$ are generated as in (3.8)-(3.10).

The conditional p.d.f.s in (3.16) therefore have the form

$$p(y_k \mid Y_{k-1}) = (2\pi)^{-1/2}(\mathbf{M}_k\mathbf{P}_k\mathbf{M}_k^+ + R_k)^{-1/2} \exp\left[-\tfrac{1}{2}(y_k - \hat{y}_k)^2/(\mathbf{M}_k\mathbf{P}_k\mathbf{M}_k^+ + R_k)\right], \tag{3.21}$$

$1 \leqq k \leqq N$, with $\hat{y}_1 = 0$. Multiplying these as in (3.16), we obtain the joint p.d.f.

$$p(y_1, y_2, \cdots, y_N) = (2\pi)^{-N/2} \prod_{k=1}^{N} (\mathbf{M}_k\mathbf{P}_k\mathbf{M}_k^+ + R_k)^{-1/2} \times \exp\left[-\frac{1}{2}\sum_{k=1}^{N} \frac{(y_k - \hat{y}_k)^2}{\mathbf{M}_k\mathbf{P}_k\mathbf{M}_k^+ + R_k}\right], \tag{3.22}$$

which must be identical with that in (3.15). Equating the product in this equation with the determinant in (3.15), we obtain (3.5); and equating the quadratic forms and thereafter replacing y_k by s_k, $\hat{\mathbf{x}}_k$ by $\boldsymbol{\sigma}_k$, and $\hat{y}_k$ by the expressions in (3.18) and (3.19), we obtain (3.6) with (3.11). Both sides of (3.5) are polynomials in z, and both sides of (3.6) rational functions of z. These expressions are finite and purely algebraic, and having been shown to hold for real, positive values of z, they must hold everywhere in the z-plane.

Although the matrices involved in this procedure are at most $n \times n$ and no larger than those in § 2, the time to compute $h(z)$ is now proportional to N, and for stationary processes the previous method is preferable when $N \gg 2n$.

4. Stationary time series. When the time series $v_1', \cdots, v_N'$ is stationary, its covariance matrix $\mathbf{A}$ has the Toeplitz form, with elements a_{i-j}. The matrix $\mathbf{I} + z\mathbf{A}$ figuring in the m.g.f. $h(z)$, (1.8), has then also the Toeplitz form, and both the determinant and the quadratic form in (1.8) can be computed for each value of z by means of the Levinson algorithm [13].

The quadratic form in (1.8) can be written as

$$Q(z) = \mathbf{s}^+(\mathbf{I} + z\mathbf{A})^{-1}\mathbf{s} = \mathbf{s}^+\mathbf{t} = \sum_{j=1}^{N} s_j t_j \tag{4.1}$$

where

$$\mathbf{t} + z\mathbf{A}\mathbf{t} = \mathbf{s}; \tag{4.2}$$

the t_i are the solutions of the linear equations

$$t_i + z\sum_{j=1}^{N} a_{i-j}t_j = s_i, \qquad 1 \leqq i \leqq N, \tag{4.3}$$

to which the Levinson algorithm is directly applicable. Following Blahut [1], but with a different normalization, we summarize it as a prelude to deriving a method for computing $\det(\mathbf{I} + z\mathbf{A})$ as well.

We denote the diagonal elements of the matrix $\mathbf{I} + z\mathbf{A}$ by

$$b_0 = 1 + za_0, \tag{4.4}$$

and define the row vectors $\boldsymbol{\beta}_r = (\beta_{r1}, \cdots, \beta_{rr})$ as the sequences with elements

$$\beta_{rj} = za_{r+1-j}, \qquad 1 \leqq j \leqq r. \tag{4.5}$$

At the rth stage we have a column vector $\boldsymbol{\tau}_r$ with r elements; $\tilde{\boldsymbol{\tau}}_r$ denotes the same vector turned upside down. These vectors are generated sequentially by

$$\boldsymbol{\tau}_{r+1} = (1-k_r^2)^{-1}\left[\begin{pmatrix}\boldsymbol{\tau}_r\\0\end{pmatrix} - k_r\begin{pmatrix}0\\\tilde{\boldsymbol{\tau}}_r\end{pmatrix}\right] \tag{4.6}$$

where

$$k_r = \boldsymbol{\beta}_r\boldsymbol{\tau}_r. \tag{4.7}$$

The initial vector $\boldsymbol{\tau}_1$ has a single element b_0^{-1}, and $k_1 = za_1/b_0$.

A second sequence of r-element column vectors $\mathbf{f}_r$ is generated by the recurrence

$$\mathbf{f}_r = \begin{pmatrix}\mathbf{f}_{r-1}\\0\end{pmatrix} + (s_r - \sigma_r)\tilde{\boldsymbol{\tau}}_r, \tag{4.8}$$

where

$$\sigma_r = \boldsymbol{\beta}_{r-1}\mathbf{f}_{r-1}, \qquad \sigma_1 = 0, \tag{4.9}$$

and initially

$$\mathbf{f}_1 = (s_1/b_0), \qquad \sigma_2 = za_1s_1/b_0. \tag{4.10}$$

At the end, the vector $\mathbf{t}$ in (4.1) is

$$\mathbf{t} = \mathbf{f}_N,$$

and (4.1) then yields the value of the quadratic form $Q(z)$ in (1.8).

To derive an algorithm for computing the determinant $\det(\mathbf{I}+z\mathbf{A})$ in (1.8), we consider a signal vector $\mathbf{s}$ that has a one in its pth place and zeros elsewhere. We denote by $\mathbf{B}_p$ the Toeplitz matrix that is the upper left-hand $p\times p$ block of the matrix $\mathbf{I}+z\mathbf{A}$. Then we find by this procedure that the vectors $\mathbf{f}_r$, $1\leqq r\leqq p$, contain only zeros, and by (4.8) and (4.3), with $N=p$, the element at the bottom of the column vector $\mathbf{f}_p$ is equal to

$$\frac{\det \mathbf{B}_{p-1}}{\det \mathbf{B}_p} = \tau_{p1}, \qquad 2\leqq p\leqq N, \tag{4.11}$$

where τ_{p1} is the top element of the column vector $\boldsymbol{\tau}_p$. Because $\det \mathbf{B}_1 = b_0$, we find by multiplying these $N-1$ equations

$$\det(\mathbf{I}+z\mathbf{A}) = \det \mathbf{B}_N = b_0\prod_{p=2}^{N}\tau_{p1}^{-1}, \tag{4.12}$$

and by (4.6) the factors τ_{p1}^{-1} are generated by the recurrence

$$\tau_{p+1,1}^{-1} = (1-k_p^2)\tau_{p1}^{-1}, \qquad \tau_{11}^{-1} = b_0, \tag{4.13}$$

with the k_p's given by (4.7). By means of (4.12) and (4.13) the determinant figuring in (1.8) can be computed. When N is large, computer overflow can be avoided by accumulating

$$\ln\det(\mathbf{I}+z\mathbf{A}) = \ln b_0 + \sum_{p=2}^{N}\ln(\tau_{p1}^{-1})$$

instead and substituting this into (1.9) to compute the phase $\Psi(z)$.

Appendix. Derivation of (2.8)–(2.14). We begin by assuming that the mean values of the time series are simply

$$s_p = u^{p-1}, \qquad 1\leqq p\leqq N, \tag{A1}$$

where u is one of the u_r in (2.5). At the end we superpose the corresponding solutions by means of (2.4). We write the quadratic form in (2.10) as

$$Q(z)=\sum_{k=1}^{N} s_k t_k = \mathbf{s}^{+}\mathbf{t} \tag{A2}$$

with

$$\mathbf{t}=(\mathbf{I}+z\mathbf{A})^{-1}\mathbf{s} \quad \text{or} \quad \mathbf{s}=\mathbf{t}+z\mathbf{A}\mathbf{t};$$

that is,

$$t_p+z\sum_{r=1}^{N} a_{p-r}t_r=s_p, \qquad 1\leqq p\leqq N. \tag{A3}$$

We define the sequence $t_p^{(\infty)}$ as the solution of

$$t_p^{(\infty)}+z\sum_{r=-\infty}^{\infty} a_{p-r}t_r^{(\infty)}=s_p=u^{p-1}. \tag{A4}$$

By substitution we find

$$t_p^{(\infty)}=[1+zA(u)]^{-1}u^{p-1}. \tag{A5}$$

Subtracting (A4) from (A3), we obtain the equation

$$t_p-t_p^{(\infty)}+z\sum_{r=-\infty}^{\infty} a_{p-r}(t_r-t_r^{(\infty)})=\left(\sum_{r=-\infty}^{0}+\sum_{r=N+1}^{\infty}\right)a_{p-r}t_r. \tag{A6}$$

We take $t_p-t_p^{(\infty)}$ to have the form

$$t_p-t_p^{(\infty)}=\sum_{r=1}^{2n}\gamma_r\beta_r^{p-1}, \tag{A7}$$

where the β_r are the $2n$ roots of (2.3), and the coefficients γ_r are to be determined. The left-hand side of (A6) then vanishes, and we can write (A6) as

$$\left(\sum_{r=-\infty}^{0}+\sum_{r=N+1}^{\infty}\right)a_{p-r}(t_r-t_r^{(\infty)})+\left(\sum_{r=-\infty}^{0}+\sum_{r=N+1}^{\infty}\right)a_{p-r}t_r^{(\infty)}=0, \qquad 1\leqq p\leqq N. \tag{A8}$$

When the spectral density (2.1) possesses only simple poles $\mu_1, \mu_2, \cdots, \mu_{2n}$, the covariances can be written as follows:

$$\begin{aligned} a_r &= \sum_{k=1}^{n}\alpha_k\mu_{k+n}^{r}, \qquad r>0, \\ &= \sum_{k=1}^{n}\alpha_k\mu_k^{r}, \qquad r<0, \end{aligned} \tag{A9}$$

with certain coefficients α_k; the first n poles $\mu_1, \cdots, \mu_n$ lie outside, the last n inside the unit circle. Substituting (A5), (A7), and (A9) into (A8) and carrying out the summations over r, we find

$$\sum_{k=1}^{n}\alpha_k\mu_{k+n}^{p}\left(\sum_{l=1}^{2n}\frac{\gamma_l}{\beta_l-\mu_{k+n}}+\frac{C}{u-\mu_{k+n}}\right)$$
$$+\sum_{k=1}^{n}\alpha_k\mu_k^{p-N}\left(\sum_{l=1}^{2n}\frac{\beta_l^{N}\gamma_l}{\beta_l-\mu_k}+\frac{Cu^{N}}{u-\mu_k}\right)=0, \qquad 1\leqq p\leqq N,$$

with $C=[1+zA(u)]^{-1}$.

These equations will hold for $1 \leqq p \leqq N$, $N > 2n$, only if the terms in parentheses vanish, and with $u^N = 1$ by (2.5) we obtain $2n$ linear simultaneous equations for the unknowns γ_l in (A7),

$$\begin{aligned} &\sum_{l=1}^{2n} \frac{\beta_l^N}{\beta_l - \mu_k} \gamma_l = -\frac{C}{u - \mu_k}, \qquad 1 \leqq k \leqq n, \\ &\sum_{l=1}^{2n} \frac{1}{\beta_l - \mu_k} \gamma_l = -\frac{C}{u - \mu_k}, \qquad n+1 \leqq k \leqq 2n. \end{aligned} \tag{A10}$$

The coefficients of the $2n$ unknowns γ_l form the matrix $\mathbf{G}_N(z)$ defined in (2.9). Putting $u = u_r$ in (A10), multiplying by the Fourier coefficients σ_r in (2.4), and summing over $1 \leqq r \leqq N$, we find that the column vector $\boldsymbol{\gamma}$ of the coefficients γ_r in (A7) is given by

$$\boldsymbol{\gamma} = -\mathbf{G}_N^{-1}\mathbf{F} \tag{A11}$$

with the $2n$ elements of the column vector $\mathbf{F}$ as in (2.14).

We now write the quadratic form in (2.10) and (A2) as

$$Q(z) = \sum_{p=1}^{N} s_p t_p^{(\infty)} + \sum_{p=1}^{N} s_p (t_p - t_p^{(\infty)}) = J_1 + J_2. \tag{A12}$$

By similarly weighting and summing (A5),

$$t_p^{(\infty)} = \sum_{r=1}^{N} \sigma_r [1 + zA(u_r)]^{-1} u_r^{p-1}, \tag{A13}$$

and by the orthogonality of the vectors

$$(1, u_r, \cdots, u_r^{N-1}), \qquad 1 \leqq r \leqq N,$$

the first term

$$J_1 = \sum_{p=1}^{N} s_p t_p^{(\infty)} \tag{A14}$$

reduces to (2.11).

Substituting (A7) and the conjugate of (2.4) into the second term of (A12) and carrying out the summation over p, we find

$$\begin{aligned} J_2 = \sum_{p=1}^{N} s_p (t_p - t_p^{(\infty)}) &= \sum_{l=1}^{2n} \gamma_l \sum_{r=1}^{N} \sigma_r^* \frac{1 - (\beta_l u_r^{-1})^N}{1 - \beta_l u_r^{-1}} \\ &= \sum_{l=1}^{2n} \gamma_l E_l = \mathbf{E}^+ \boldsymbol{\gamma} \end{aligned}$$

where, with $u_r^{-N} = 1$, the E_l are given by (2.13). Thus with (A11),

$$J_2 = -\mathbf{E}^+ \mathbf{G}_N^{-1} \mathbf{F}, \tag{A15}$$

which can be written as in (2.12) by the well-known rule for the determinant of a matrix divided into blocks [3, p. 46].

To demonstrate (2.8), we consider first the determinant

$$D_0(z) = \det (\mathbf{I} + z\boldsymbol{\phi}), \tag{A16}$$

where $\boldsymbol{\phi}$ is the Toeplitz matrix of covariances ϕ_{i-j},

$$\phi_p = a_p, \quad p \neq 0, \quad \phi_0 = a_0 - R.$$

The transform of the sequence ϕ_p is $\Phi(w)$ in (2.2);

$$\Phi(w) = N(w)/P(w)$$

is a rational function of w whose denominator $P(w)$ is a polynomial of degree $2n$ and whose numerator $N(w)$ is a polynomial of degree $2m < 2n$. In effect, we are assuming $R = 0$ for now.

We shall show that

$$D_0(z) = \left(\prod_{k=1}^{n} \mu_{k+n}^{N} \right) \frac{\det \mathbf{G}_N(z)}{\det \mathbf{G}_0(z)}, \tag{A17}$$

where $\mathbf{G}_N(z)$ is as in (2.9), except that the β's are determined by (2.3) with $A(w)$ is replaced by $\Phi(w)$. Equation (2.3) is now equivalent to the algebraic equation of degree $2n$:

$$P(w) + zN(w) = 0, \qquad w = \beta_1, \cdots, \beta_{2n}, \tag{A18}$$

where

$$P(w) = \prod_{k=1}^{2n} (w - \mu_k). \tag{A19}$$

With these changes, and with a_{p-r} replaced by ϕ_{p-r}, our analysis in (A1)-(A10) continues to hold. The white-noise term R will be reintroduced at the end.

In terms of the N eigenvalues λ_k of the covariance matrix $\boldsymbol{\phi}$,

$$D_0(z) = \prod_{k=1}^{N} (1 + \lambda_k z) \tag{A20}$$

is a polynomial in z of degree N with N zeros $z_k = -\lambda_k^{-1}$, $1 \leqq k \leqq N$. At these N values of z, neither the simultaneous equations (A10) nor those in (A3) from which they were derived have a solution. The matrix $\mathbf{G}_N(z)$ of the coefficients in (A10), like the matrix $\mathbf{I} + z\boldsymbol{\phi}$, is singular at $z = z_k$, $1 \leqq k \leqq N$; and (A17) and (A20) possess the same zeros. We must yet show that the right-hand side of (A17), which we denote by RHS, is also a polynomial of degree N in z and that it agrees with (A20) at at least one other value of z.

First we must prove that RHS has no singularities in the finite part of the z-plane. The elements of the matrix $\mathbf{G}_0(z)$ are, by (2.9) with $N = 0$,

$$G_{jk}^{(0)} = (\beta_k - \mu_j)^{-1}, \qquad 1 \leqq (j, k) \leqq 2n. \tag{A21}$$

If we expand the determinants in the numerator and denominator of (A17) and combine all their terms, we obtain two vast fractions, the denominators of which are identical. When the two determinants are divided as in (A17), those denominators cancel. Any singularities in RHS must arise from the vanishing of $\det \mathbf{G}_0(z)$.

By subtracting any column of $\det \mathbf{G}_N(z)$ from any other, we can show that $\det \mathbf{G}_N(z)$ is proportional to

$$P_\beta = \prod_{i=1}^{n} \prod_{j=i+1}^{n} (\beta_i - \beta_j). \tag{A22}$$

Furthermore [15, pp. 348-349],

$$\det \mathbf{G}_0(z) = \frac{\prod_{i=1}^{n} \prod_{j=i+1}^{n} [(\beta_i - \beta_j)(\mu_j - \mu_i)]}{\prod_{i=1}^{n} \prod_{j=1}^{n} (\beta_i - \mu_j)}, \tag{A23}$$

which also contains P_β. The product P_β cancels when $\det \mathbf{G}_N(z)$ is divided by $\det \mathbf{G}_0(z)$ in RHS. Because of our assumption that the poles of $\Phi(w)$ are simple, what is left of $\det \mathbf{G}_0(z)$ cannot vanish for any value of z. The quotient in RHS possesses no singularities except a pole at infinity.

We now show that RHS is proportional to z^N when $|z|$ is large. Denote by $2m$ the degree of the polynomial $N(w)$ in (A18); $m < n$. The spectral density $\Phi(w)$ possesses m zeros $\varepsilon_1, \cdots, \varepsilon_m$ outside the unit circle and m zeros $\varepsilon_{n+k} = \varepsilon_k^{-1}$ inside, $1 \leqq k \leqq m$. As z goes to infinity, m of the exterior β_k's approach those m exterior zeros; the remaining n-m exterior β_k's go to infinity. The interior β_k's pass either to the m interior zeros or to the zero of order n-m at the origin.

Now divide the determinant $\det \mathbf{G}_N(z)$ into four $n \times n$ blocks:

$$\det \mathbf{G}_N(z) = \begin{vmatrix} \mathbf{G}_{11}^{(N)} & \mathbf{G}_{12}^{(N)} \\ \mathbf{G}_{21} & \mathbf{G}_{22} \end{vmatrix}. \tag{A24}$$

Only the blocks with superscript (N) depend on N; the others are the same as in $\det \mathbf{G}_0(z)$. When $|z| \gg 1$, the upper left-hand block $\mathbf{G}_{11}^{(N)}$ dominates the determinant, as can be seen from (2.9). Its elements are

$$\beta_k^N (\beta_k - \mu_j)^{-1}, \qquad 1 \leqq k \leqq n,$$

and n-m of these β_k's have large absolute values. Because [3, p. 46]

$$\det \mathbf{G}_N(z) = \det (\mathbf{G}_{11}^{(N)} - \mathbf{G}_{12}^{(N)} \mathbf{G}_{22}^{-1} \mathbf{G}_{21}) \det \mathbf{G}_{22} \tag{A25}$$

and

$$\det \mathbf{G}_{11}^{(N)} = \prod_{k=1}^{N} \beta_k^N \det \mathbf{G}_{11}^{(0)}, \tag{A26}$$

we find in this limit $|z| \to \infty$

$$\frac{\det \mathbf{G}_N(z)}{\det \mathbf{G}_0(z)} \simeq \prod_{k=1}^{N} \beta_k^N \left[\frac{\det \mathbf{G}_{11}^{(0)} \det \mathbf{G}_{22}}{\det \mathbf{G}_0(z)} \right]. \tag{A27}$$

The factor in square brackets remains bounded as $|z| \to \infty$, as can be seen by inspection of the elements of the determinants involved. The RHS of (A17) is therefore proportional to the nth power of the product of the exterior β_k's when $|z| \to \infty$. We now show that this product is proportional to z in this limit.

The spectral density $\Phi(w)$ depends on w only through $y = w + w^{-1}$, and as it is a rational function, it can be written

$$\Phi(w) = N_1(y)/P_1(y),$$

where $P_1(y)$ is a polynomial of degree n and $N_1(y)$ a polynomial of degree $m < n$. Equation (A18) is equivalent to

$$P_1(y) + zN_1(y) = 0 \tag{A28}$$

and possesses n roots $y_1, y_2, \cdots, y_n$;

$$y_k = \beta_k + \beta_k^{-1}, \qquad 1 \leqq k \leqq n. \tag{A29}$$

The product of these roots equals the quotient of the coefficient of y^0 and that of y^N in (A28), and when $|z| \gg 1$ this quotient is proportional to z:

$$\prod_{k=1}^{N} y_k \simeq K_0 z, \qquad |z| \gg 1,$$

for some constant K_0. Let $\beta_1, \cdots, \beta_{n-m}$ denote those roots of (A18) that go to infinity when $|z| \to \infty$. By (A29), $y_k \to \beta_k$, $1 \leqq k \leqq n-m$, in this limit; and as the remaining y_k's and the remaining exterior β_k's approach finite values in this limit,

$$\prod_{k=1}^{N} \beta_k \simeq K_1 z, \qquad |z| \gg 1$$

for some other constant K_1. Because by (A27) RHS is proportional to z^N as $|z| \to \infty$ and because it possesses no singularities in the finite portion of the z-plane, it must, like (A20), be a polynomial in z of degree N.

When the point z goes to the origin, the $2n$ roots of (A18) approach the $2n$ poles $\mu_1, \cdots, \mu_{2n}$ of the spectral density $\Phi(w)$. In both $\det \mathbf{G}_N(z)$ and $\det \mathbf{G}_0(z)$ the diagonal elements then dominate, and as $z \to 0$ the RHS of (A17) approaches

$$\prod_{k=1}^{N} \mu_{k+n}^{N} \beta_k^{N},$$

which goes to 1 because $\beta_k = \beta_{k+n}^{-1} \to \mu_{k+n}^{-1}$. The right-hand sides of both (A17) and (A20) are equal to 1 at $z = 0$, and as both are polynomials of degree N with the same zeros, they must be equal to all values of z.

Returning to (2.8), we use

$$\mathbf{A} = R\mathbf{I} + \boldsymbol{\phi}$$

from (2.2) to write the determinant as

$$\begin{aligned} D(z) &= \det [(1+zR)\mathbf{I} + z\boldsymbol{\phi}] \\ &= (1+zR)^N \det [\mathbf{I} + (1+zR)^{-1} z\boldsymbol{\phi}]. \end{aligned}$$

The determinant on the right is just $D_0(z(1+zR)^{-1})$ with $D_0(\cdot)$ as in (A17) and (A20). The β's in the RHS of (A17) are now the roots of (A18) with z replaced by $z(1+zR)^{-1}$, that is, of

$$1 + z(1+zR)^{-1}\Phi(w) = 0,$$

which by (2.2) is the same as (2.3). In this manner (2.8) follows from (A17).

Acknowledgment. I thank Professor B. Picinbono for his hospitality during my stay at the Laboratoire des Signaux et Systèmes.

REFERENCES

[1] R. E. BLAHUT, *Fast Algorithms for Digital Signal Processing*, Addison-Wesley, New York, 1985, Chap. 11, pp. 352-358.

[2] R. H. BROWN, *The distribution function of positive definite quadratic forms in normal random variables*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 689-695.

[3] F. R. GANTMACHER, *The Theory of Matrices*, Chelsea, New York, 1960.

[4] R. A. GIDEON AND J. GURLAND, *Series expansions for quadratic forms in normal variables*, J. Amer. Statist. Assoc., 71 (1976), pp. 227-232.

[5] C. W. HELSTROM AND J. A. RITCEY, *Evaluating radar detection probabilities by steepest descent integration*, IEEE Trans. Aerospace Electron. Systems, 20 (1984), pp. 624-634.

[6] ———, *Evaluation of the noncentral F-distribution by numerical contour integration*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 505-514.

[7] C. W. HELSTROM, *Distribution of the filtered output of a quadratic rectifier computed by numerical contour integration*, IEEE Trans. Inform. Theory, 32 (1986), pp. 450-463.

[8] C. W. HELSTROM, *Computation of photoelectric counting distributions by numerical contour integration. Part* II. *Light with rational spectral density*, J. Opt. Soc. Amer. A, 4 (1987), pp. 1245-1255.
[9] A. H. JAZWINSKI, *Stochastic Processes and Filtering Theory*, Academic Press, New York, 1970.
[10] N. L. JOHNSON AND S. KOTZ, *Distributions in Statistics-Continuous Univariate Distributions*, Vol. 2, Houghton-Mifflin, New York, 1970.
[11] R. E. KALMAN, *A new approach to linear filtering and prediction problems*, Trans. ASME Ser. D, J. Basic Engrg., 82 (1960), pp. 35-45.
[12] I. KANTER, *Exact detection probability for partially correlated Rayleigh targets*, IEEE Trans. Aerospace Electron. Systems, 22 (1986), pp. 184-196.
[13] N. LEVINSON, *The Wiener* RMS *(root mean square) error criterion in filter design and prediction*, J. Math. and Phys., 25 (1946), pp. 261-278.
[14] L. LJUNG AND T. SÖDERSTRÖM, *Theory and Practice of Recursive Estimation*, MIT Press, Cambridge, MA, 1983, § 3.8, pp. 122-130.
[15] T. MUIR, *A Treatise on the Theory of Determinants*, Chelsea, New York, 1960.
[16] S. O. RICE, *Distribution of quadratic forms in normal random variables—evaluation by numerical contour integration*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 438-448.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 447-474, May 1989
© 1989 Society for Industrial and Applied Mathematics
004

RESOLUTION OF RECONSTRUCTION METHODS IN COMPUTERIZED TOMOGRAPHY*

HERMANN KRUSE†

Abstract. The essential step in many reconstruction methods is the evaluation of certain integrals by the trapezoidal rule, using the data function Rf and a weight function q. Hence the accuracy of the reconstruction depends on the accuracy of the trapezoidal rule and thus on the number and location of the points where the data are sampled. In parallel geometry Lindgren and Rattey have suggested using a grid that contains only 50 percent of the points of the standard grid. Using a special error estimate for the trapezoidal rule, it can be shown that to reconstruct details of a certain size the filtered backprojection method needs only the data on the small grid. Similar results can be obtained for fan-beam geometry; in this case Marr's algorithm successfully works on a small grid.

Key words. computerized tomography, numerical integration, ill-posed problems, integral equations, Radon transform

AMS(MOS) classifications. 65R20, 65D30, 68G10, 92A07

1. Introduction. The mathematical problem in computerized tomography (CT) is the reconstruction of a function f from the values of its Radon transform Rf at certain points. By "resolution" we mean the following: how many data (i.e., evaluations of Rf) are necessary and where must these data be sampled in order to reconstruct details of a certain size?

The number of data implies the computational effort and in X-ray tomography it determines the dose of radiation. Thus the number of data is kept as small as possible.

The size of the smallest detail that can be recognized in an image is determined by the bandwidth of the representing function. Hence bandlimited functions are very important in our analysis. We shall deal with two kinds of sampling schemes: parallel geometry and fan-beam geometry. The set of points where the data are sampled will be called a grid.

For the standard grid in parallel geometry it is well known that $2b^2/\pi$ data are necessary to reconstruct a b-bandlimited function. (See Bracewell and Riddle [2]; Crowther, de Rosier and Klug [5].) Cormack [3] has suggested what we shall call the interlaced grid that contains only half as many data points. Precisely this grid has been found by Lindgren and Rattey [9], [14], through their study of the support of the Fourier transform of Rf. It turns out that the Radon transform Rf of an essentially b-bandlimited function f is determined by the values of Rf on either of these grids (up to an exponentially decreasing term), and that from these values f can be reconstructed reliably by the filtered backprojection method.

Furthermore Natterer [11] has proved for fan-beam geometry that Rf is essentially determined if the classical fan-beam grid contains at least $4rb^2 \arcsin(1/r)/\pi$ points (with r being the radius of the circle on which the X-ray source surrounds the region of reconstruction that is the unit disk). If redundancy occurs this number can be reduced by 50 percent. The same result can be obtained for a grid that is important in positron emission tomography, the so-called PET-grid. It roughly contains $b^2/2$ points, which is considerably fewer than for the usual fan-beam grid. The PET-grid too had been mentioned already by Cormack [4].

* Received by the editors July 6, 1987; accepted for publication (in revised form) May 15, 1988.

† Institut für Numerische und instrumentelle Mathematik, Einsteinstrasse 62, 4400 Münster, Federal Republic of Germany.

Hence we know that the Radon transform Rf of an essentially b-bandlimited function f is determined very accurately by the values on one of the small grids.

In this paper we are interested in algorithms that actually achieve the possible resolution. We shall see that for the interlaced geometry, the well-known filtered backprojection algorithm is such an algorithm, while for the PET-grid, Marr's algorithm [10] does the job.

To obtain these results we need some notation and tools, which we introduce in § 2. In § 3 we give exact definitions of the grids and the sampling conditions; in § 4 we briefly present the reconstruction algorithms. We see that the accuracy of such a reconstruction essentially depends on the accuracy of the trapezoidal rule applied to certain integrals. So in § 5 we deduce an error estimate for the trapezoidal rule by means of a Poisson summation formula on a subgroup of $\mathbb{R}^n \times [0, 2\pi)^m$. Afterwards we are able to estimate the error we commit applying filtered backprojection to the data on the interlaced grid in § 6; we estimate the error of Marr's algorithm on the data of the PET-grid in § 7. Numerical results in § 8 confirm these results.

2. Notation and tools. Let $\mathcal{S}(\mathbb{R}^n)$ be the Schwartz space, S^{n-1} the unit sphere in $\mathbb{R}^n$, $Z = \mathbb{R} \times S^{n-1}$ the unit cylinder in $\mathbb{R}^{n+1}$, and $\mathcal{S}(Z)$ the C^∞-functions on Z that are rapidly decreasing in the first argument. Then the Radon transform is defined as follows.

DEFINITION 2.1. The operator $R: \mathcal{S}(\mathbb{R}^n) \to \mathcal{S}(Z)$

$$Rf(s, \omega) = \int_{\omega^\perp} f(s \cdot \omega + x)\, dx \tag{2.1}$$

is called the Radon transform.

Let $\mathcal{S}^{n,m}$ be the space of C^∞-functions on $\mathbb{R}^{n+m}$ that are in $\mathcal{S}(\mathbb{R}^n)$ with respect to the first n arguments and 2π-periodic with respect to the last m arguments. For $n = 2$ Rf can be viewed as a function in $\mathcal{S}^{1,1}$.

Similarly $L^{n,m}$ is the space of functions on $\mathbb{R}^{n+m}$ that are in $L_1(\mathbb{R}^n)$ with respect to the first n arguments, C^∞ and 2π-periodic with respect to the last m arguments. With $T := [0, 2\pi)$ the Fourier transform on $L^{n,m}$ reads

$$\hat{f}\begin{pmatrix} \xi \\ k \end{pmatrix} = (2\pi)^{-(n+m)/2} \int_{\mathbb{R}^n} \int_{T^m} f\begin{pmatrix} x \\ y \end{pmatrix} e^{-ix\cdot\xi}\, e^{-iy\cdot k}\, dy\, dx. \tag{2.2}$$

The inverse Fourier transform of a function f is denoted by $\tilde{f}$. Functions in $L^{n,m}$ can be convolved with functions in $\mathcal{S}^{n,m}$, and in this case the classical convolution theorems hold. Radon transform and Fourier transform are connected by the projection theorem.

THEOREM 2.1. *If $f \in \mathcal{S}(\mathbb{R}^n)$, then*

$$(Rf)\hat{}\,(\sigma, \omega) = (2\pi)^{(n-1)/2} \hat{f}(\sigma \cdot \omega). \tag{2.3}$$

Here $(Rf)\hat{}$ is the one-dimensional Fourier transform with respect to the first variable, and $\hat{f}$ is the usual n-dimensional Fourier transform.

We define the backprojection operator R^* on $\mathcal{S}(Z)$ by

$$R^* g(x) = \int_{S^{n-1}} g(x \cdot \omega, \omega)\, d\omega.$$

Then from Theorem 2.1 we obtain Radon's inversion formula

$$f = \tfrac{1}{2}(2\pi)^{1-n} R^* I^{n-1} Rf, \tag{2.4}$$

with $(I^{n-1}h)\hat{}\,(\sigma) = |\sigma|^{n-1} h(\sigma)$.

Bandlimited functions are of special interest in digital image processing: $f \in \mathcal{S}(\mathbb{R}^n)$ is called b-bandlimited, if $\hat{f}(\xi) = 0$ for $|\xi| > b$. An image represented by a b-bandlimited

function f does not contain details of size smaller than $2\pi/b$. Since functions of compact support cannot be strictly bandlimited, we work with essentially bandlimited functions.

For $d \in \mathbb{Z}$, $f \in \mathcal{S}(\mathbb{R}^n)$, $b \geqq 0$ define

$$\varepsilon_d(f, b) = \int_{|\xi|>b} |\xi|^d |\hat{f}(\xi)|\, d\xi. \tag{2.5}$$

f is essentially b-bandlimited, if $\varepsilon_d(f, b)$ is very small for $d \geqq 0$.

By $\eta(\theta, b)$, $0 < \theta < 1$, we denote any quantity that admits an estimate of the form

$$0 \leqq \eta(\theta, b) \leqq C(\theta)\, e^{-\lambda(\theta)\cdot b} \tag{2.6}$$

if $b \geqq B(\theta)$. Here $C(\theta), \lambda(\theta), B(\theta) > 0$ are positive constants. For fixed $\theta \in (0, 1)$, $\eta(\theta, b)$ decays exponentially as b increases beyond $B(\theta)$. We shall denote different quantities by the same symbol $\eta(\theta, b)$ if they satisfy (2.6).

Examples for such quantities are

$$\begin{aligned} &\eta_1(\theta, m) = \sup_{|r|\leqq 1} \int_{-\theta m}^{\theta m} |J_m(r\sigma)|\, d\sigma, \\ &\eta_2(\theta, b) = \sum_{m \geqq b} \eta_1(\theta, m), \qquad \eta_{3,d}(\theta, b) = \sum_{m \geqq b} m^d \eta_1(\theta, m), \\ &b \cdot \eta(\theta, b) \quad \text{and} \quad \eta\left(\theta, \left(\frac{1}{\theta} - 1\right) b\right), \end{aligned} \tag{2.7}$$

with

$$J_k(x) = \frac{(-i)^k}{2\pi} \int_0^{2\pi} e^{ix\cos\varphi - ik\varphi}\, d\varphi \tag{2.8}$$

the Bessel functions of the first kind. (See Natterer [11, Chap. III.2].)

3. Sampling schemes in CT. We shall restrict ourselves to the case $n = 2$; then the Radon transform Rf is the evaluation of line integrals. The region of reconstruction will be the unit disk, i.e., supp $(f) \subset D = \{x \in \mathbb{R}^2 \,|\, |x| \leqq 1\}$. Some results of Natterer [11] are essential for understanding the following sections.

3.1. Parallel geometry. In the parallel case $Rf(s, \varphi)$ is sampled at discrete points s_i, φ_j which we want to describe by means of a (2, 2)-matrix W:

$$\begin{pmatrix} s_l \\ \varphi_l \end{pmatrix} = W \cdot l, \qquad l = \begin{pmatrix} l_1 \\ l_2 \end{pmatrix} \in \mathbb{Z}^2.$$

The standard grid in parallel geometry is defined by

$$W = \begin{pmatrix} 1/q & 0 \\ 0 & \pi/p \end{pmatrix}, \qquad p, q \in \mathbb{N}. \tag{3.1}$$

Since $Rf(s, \varphi) = 0$ for $|s| > 1$, and because of the 2π-periodicity of Rf with respect to the second argument it is sufficient to sample at the points (s_l, φ_l) for $l_1 = -q, \cdots, q$; $l_2 = 0, \cdots, p-1$. (See Fig. 3.1.)

The next theorem provides us with a proper choice of p and q:

THEOREM 3.1. *Let $f \in C_0^\infty(D)$ and $g(s, \varphi) = Rf(s, \varphi) \in \mathcal{S}^{1,1}$. For $0 < \theta < 1$, $b \geqq 1$ let the set K be defined by*

$$K = \left\{(\sigma, k) \in \mathbb{R} \times \mathbb{Z} \,\middle|\, |\sigma| < b, |k| < \max\left(|\sigma|/\theta, \left(\frac{1}{\theta} - 1\right) b\right)\right\}. \tag{3.2}$$

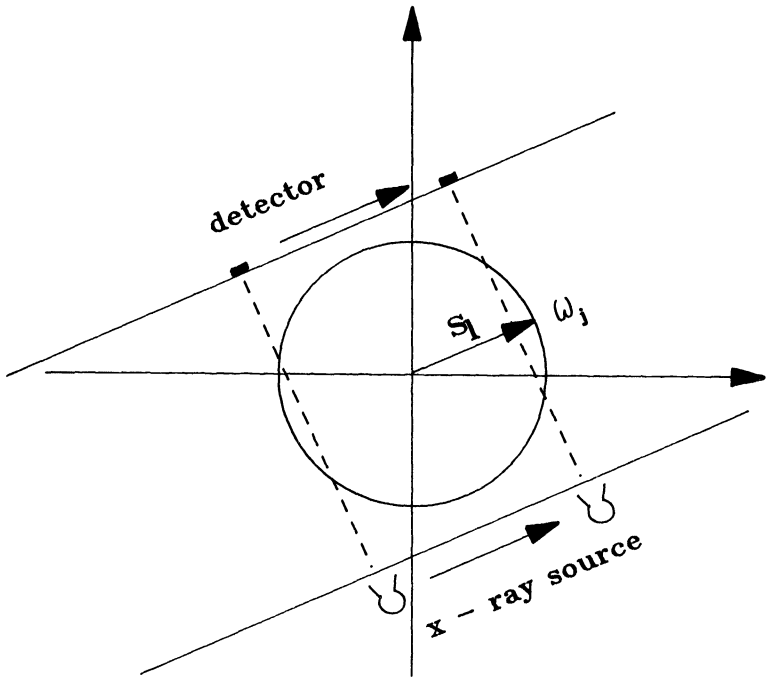


FIG. 3.1

(*See Fig.* 3.2.)

(i) *Then the Fourier transform* $\hat{g}(\sigma, k)$ *satisfies the estimate*

$$\sum_{k\in\mathbb{Z}} \int_{(\frac{\sigma}{k})\notin K} |\hat{g}(\sigma, k)|\, d\sigma \leqq \eta(\theta, b)\|f\|_{L_1(D)} + \frac{8}{\theta}\varepsilon_0(f, b). \tag{3.3}$$

(ii) *Let* W *be a nonsingular* $(2,2)$*-matrix such that the sets* $K + 2\pi(W^{-1})^T$, $l\in\mathbb{Z}^2$, *are mutually disjoint. If* $g(Wl) = 0$ *for* $l\in\mathbb{Z}^2$, *then*

$$\|Rf\|_{L_\infty(Z)} \leqq \eta(\theta, b)\|f\|_{L_1(D)} + \frac{8}{\pi\theta}\varepsilon_0(f, b). \tag{3.4}$$

Proof. See Natterer [11, Thm. III.3.1] for the proof of Theorem 3.1. □

This theorem tells us that for essentially b-bandlimited functions f (i.e., $\varepsilon_0(f, b)$ is very small) the Radon transform Rf is determined up to a negligible quantity by the values of Rf on the grid $\{Wl \mid l\in\mathbb{Z}^2\}$ if the sets $K + 2\pi(W^{-1})^T l$, $l\in\mathbb{Z}^2$, are mutually disjoint.

For the standard grid we have

$$2\pi(W^{-1})^T l = \begin{pmatrix} 2\pi q & 0 \\ 0 & 2p \end{pmatrix}$$

and looking at Fig. 3.2 we immediately see that the equations

$$p \geqq b/\theta, \qquad q \geqq b/\pi \tag{3.5}$$

must be satisfied in order to keep the translated sets disjoint.

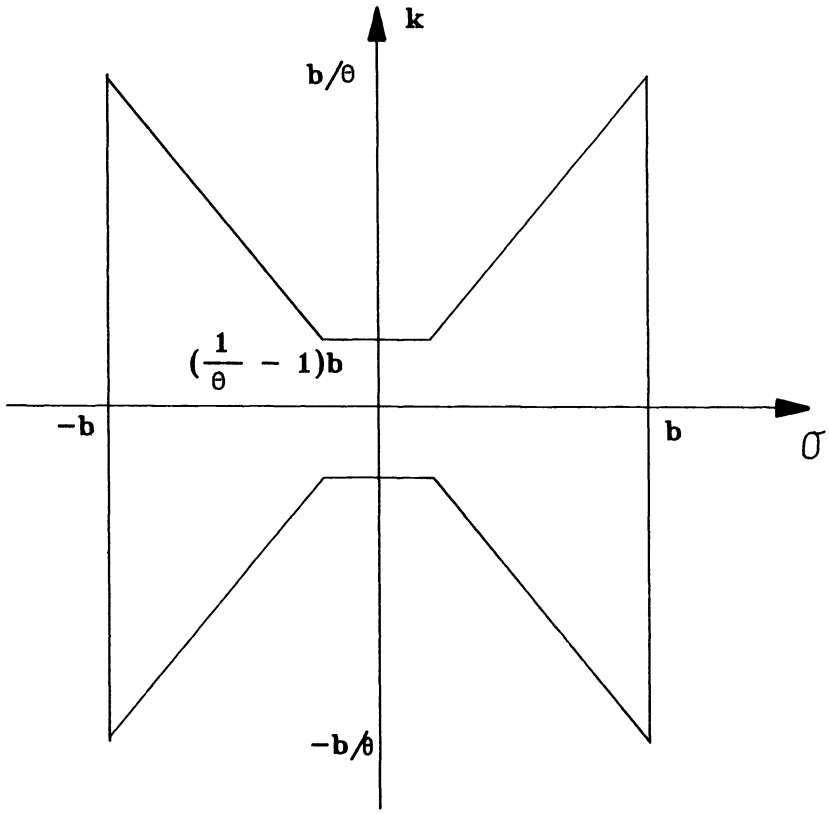


FIG. 3.2

But it is also obvious that the matrix

$$2\pi(W^{-1})^T = \begin{pmatrix} 2b & b \\ 0 & b/\theta' \end{pmatrix}, \qquad 0 < \theta' < \frac{\theta}{2-\theta}$$

keeps the sets mutually disjoint, too. In this case we have

$$W = \begin{pmatrix} \pi/b & 0 \\ -\pi\theta'/b & 2\pi\theta'/b \end{pmatrix} = \begin{pmatrix} 1/q & 0 \\ -\pi/p & 2\pi/p \end{pmatrix} \tag{3.6}$$

with $p = b/\theta'$ and $q = b/\pi$.

This (2, 2)-matrix W defines the interlaced grid. To keep the number of sampling points small it is desirable to have $\theta, \theta' \sim 1$.

Figure 3.3 illustrates both grids for $p = 2$, $q = 4$. Obviously the interlaced grid contains only half as many points as the standard grid.

3.2. Fan-beam geometry. Very important for practical applications is fan-beam geometry, where other coordinates are used: the X-ray source runs on a circle with radius $r > 1$. Then the data

$$g(\alpha, \beta) = \begin{cases} Rf(r\cos\alpha, \beta - \alpha), & 0 \leqq \alpha \leqq \pi, \\ 0, & \pi < \alpha \leqq 2\pi \end{cases} \tag{3.7}$$

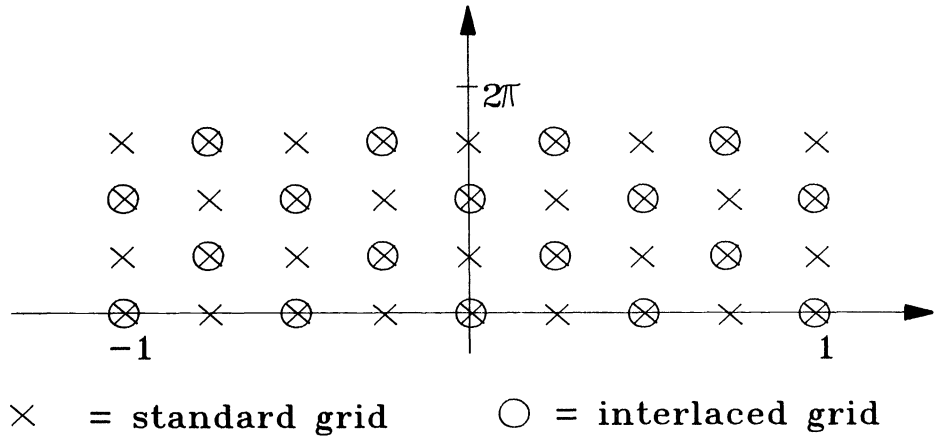


FIG. 3.3

are sampled at the points $\binom{\alpha_l}{\beta_l} = Wl$, $l \in \mathbb{Z}^2$, with

$$W = \begin{pmatrix} \pi/2q & 0 \\ 0 & 2\pi/p \end{pmatrix}. \tag{3.8}$$

(See Fig. 3.4.)

THEOREM 3.2. *Let* $f \in C_0^\infty(D)$, $g \in \mathcal{S}^{0,2}$ *be defined by* 3.7. *Let* $0 < \theta < 1$, $b \geqq 1$, $b' = \min(b, b \ln r)$ *and*

$$K = \left\{ (m, k) \in \mathbb{Z}^2 \,\middle|\, |k| \leqq \frac{b}{\theta}, |k+m| < \frac{rb}{\theta}, |k| \leqq \max\left(|k+m|, \left(\frac{1}{\theta} - 1 \right) rb \right) \right\}. \tag{3.9}$$

(i) *Then the Fourier transform* $\hat{g}(m, k)$ *satisfies the estimate*

$$\sum_{(m,k) \notin K} |\hat{g}(m, k)| \leqq \eta(\theta, b') \| f \|_{L_1(D)} + \frac{12r}{\theta^2} \varepsilon_1(f, b). \tag{3.10}$$

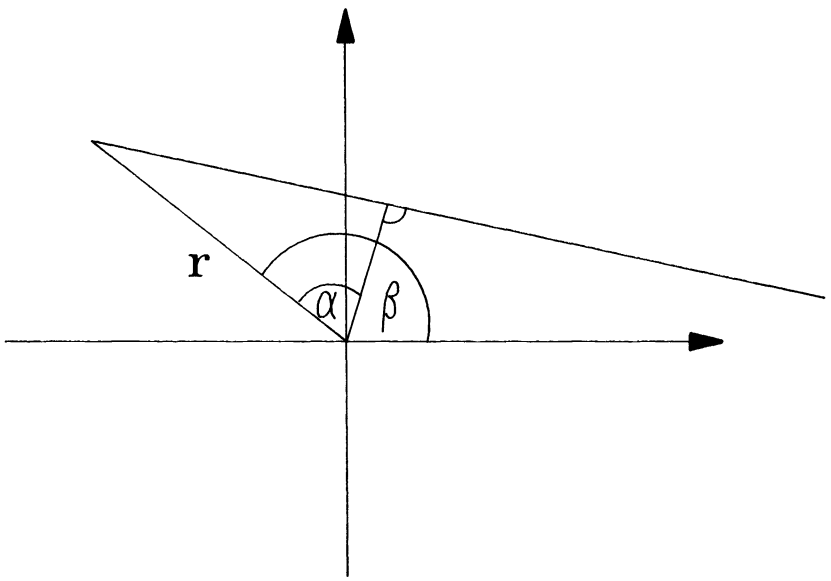


FIG. 3.4

(ii) *Let W be a nonsingular $(2,2)$-matrix, such that the sets $K+2\pi(W^{-1})^T l$, $l\in\mathbb{Z}^2$, are mutually disjoint. Then, if $g(Wl)=0$ for $l\in\mathbb{Z}^2$ we have*

$$\|Rf\|_{L_\infty(Z)} \leqq \eta(\theta, b')\|f\|_{L_1(D)} + \frac{12r}{\pi\theta^2}\varepsilon_1(f, b). \tag{3.11}$$

Proof. The proof is analogous to Theorem III.3.2 in Natterer [11]. □

For the fan-beam grid we have

$$2\pi(W^{-1})^T = \begin{pmatrix} 4q & 0 \\ 0 & p \end{pmatrix}.$$

Figure 3.5 immediately shows us that the sets $K+2\pi(W^{-1})^T l$, $l\in\mathbb{Z}^2$, are mutually disjoint if the sampling conditions

$$q \geqq \frac{rb}{2\theta} \quad \text{and} \quad p \geqq \frac{2b}{\theta} \tag{3.12}$$

are satisfied.

If we add the four tips cut off by the lines $|k| = b/\theta$, i.e., if we choose

$$K = \left\{ (m,k)\in\mathbb{Z}^2 \;\middle|\; |k+m| \leqq \frac{rb}{\theta},\ |k| \leqq \max\left(|k+m|, \left(\frac{1}{\theta}-1\right)rb\right)\right\}, \tag{3.13}$$

then for $r \simeq 1$ we obtain a denser nonoverlapping covering of the plane using the matrix

$$2\pi(W^{-1})^T = \begin{pmatrix} 2rb/\theta' & 0 \\ 0 & rb/\theta' \end{pmatrix}, \qquad 0 < \theta' < \frac{\theta}{2-\theta}. \tag{3.14}$$

This leads to

$$W = \begin{pmatrix} \pi\theta'/rb & 0 \\ 0 & 2\pi\theta'/rb \end{pmatrix} = \begin{pmatrix} \pi/p & 0 \\ 0 & 2\pi/p \end{pmatrix} \quad \text{with } p = \frac{rb}{\theta'}. \tag{3.15}$$

(See Fig. 3.5.) This grid plays an important role in positron emission tomography and therefore it is called "PET-grid."

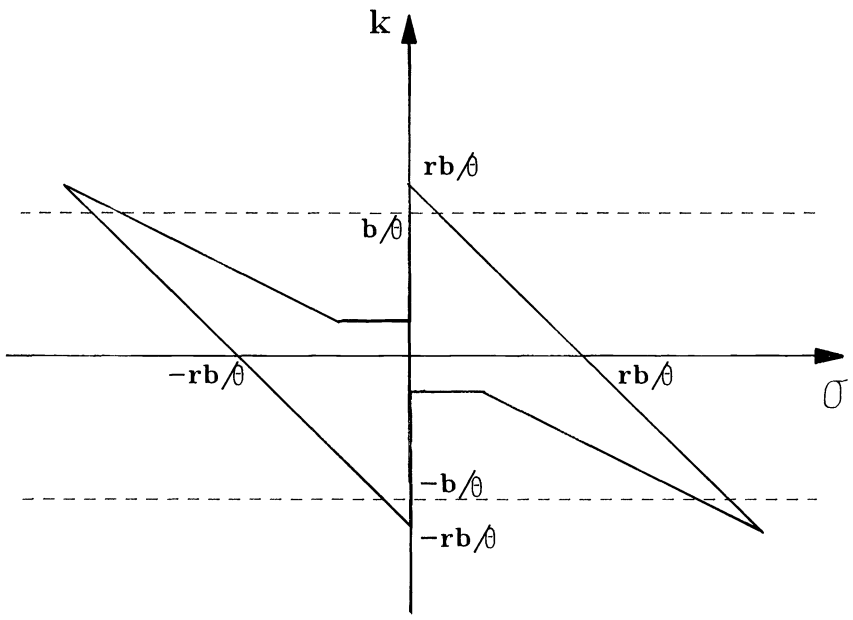


FIG. 3.5

The fan-beam grid (for b large and $\theta, \theta' \sim 1$) contains about $(\arcsin(1/r)/\pi)$ $(4rb^2/\theta^2)$ points, for $p \in \mathbb{N}$; the PET-grid, because of a certain redundancy in the data, consists of only $(\arcsin(1/r)/\pi)$ (r^2b^2/θ'^2) points.

The same redundancy may occur for the fan-beam grid, so for $r \simeq 1$ the PET-grid is 50–75 percent smaller than the fan-beam grid. (See Natterer [11] or Kruse [8] for details.)

4. Reconstruction methods.

4.1. Filtered backprojection. Filtered backprojection is the standard method in CT (see Natterer [11], Herman [7]). It is based on (2.4), which for $n=2$ reads

$$f(x) = \frac{1}{2}\left(\frac{1}{2\pi}\right)^{3/2} \int_0^{2\pi} \int_{-\infty}^{\infty} \widehat{(Rf)}(\sigma, \varphi)|\sigma|\, e^{i\sigma\omega\cdot x}\, d\sigma\, d\varphi.$$

Inserting a cutoff function $\hat{\Phi}(\sigma/b)$ into the inner integral, with

$$0 \leqq \hat{\Phi}(\sigma) \leqq 1, \quad \hat{\Phi}(-\sigma) = \hat{\Phi}(\sigma), \quad \hat{\Phi}(\sigma) = 0 \quad \text{for } |\sigma| > 1, \tag{4.1}$$

this filtering of Rf is simply a filtering of f (see Smith [16]). With

$$\begin{aligned} \hat{w}_b(\sigma) &= \frac{1}{2}(2\pi)^{-3/2}|\sigma|\hat{\Phi}\left(\frac{\sigma}{b}\right), \\ \hat{W}_b(\xi) &= \frac{1}{2\pi}\hat{\Phi}\left(\frac{|\xi|}{b}\right), \end{aligned} \tag{4.2}$$

we obtain

$$W_b * f(x) = \int_0^{2\pi} \int_{-\infty}^{\infty} w_b(x\cdot\omega - s) Rf(s, \varphi)\, ds\, d\varphi. \tag{4.3}$$

That is, instead of the function f, we compute a filtered version of it. This is done by evaluating the multiple integral by the trapezoidal rule. To reduce the computational effort, an interpolation step is introduced; we neglect this step in our analysis.

Examples for the filter functions $\hat{\Phi}$, w_b respectively can be found, e.g., in Ramachandran and Lakshminarayanan [13] or Shepp and Logan [15].

4.2. Marr's algorithm. Marr's algorithm [10] is deduced from a singular value decomposition of the Radon transform: Let $Q_{n,k}$ be polynomials of degree k, such that

$$\int_0^1 Q_{n,k}(t)t^{n+l}\, dt = 0 \quad \text{for } n, k \in \mathbb{N}_0, \quad l = 0, \cdots, k-1,$$

normed by $Q_{n,k}(1) = 1$, then the functions

$$\begin{aligned} F_{\pm n,k}(x_1, x_2) &= (x_1 \pm ix_2)^n Q_{n,k}(x_1^2 + x_2^2) \\ &= e^{\pm in\Psi}|x|^n Q_{n,k}(|x|^2), \end{aligned} \tag{4.4}$$

with

$$(x_1, x_2) = |x|\begin{pmatrix}\cos\psi \\ \sin\psi\end{pmatrix} \in \mathbb{R}^2$$

form a complete orthogonal system in $L_2(D)$.

Defining

$$g_{\pm n,k}(s, \varphi) = \frac{2}{|n| + 2k + 1} w(s) U_{|n|+2k}(s)\, e^{\pm in\varphi}$$

with U_m the Chebyshev polynomial of the second kind of degree m and $w(s) = (1-s^2)^{1/2}$, these functions form a complete orthogonal system in $L_2([-1,1]\times T, 1/w)$ and we have

$$RF_{n,k}(s,\varphi) = g_{n,k}(s,\varphi),$$
$$\|g_{n,k}\|_{L_2(Z,1/w)} = \frac{2\pi}{|n|+2k+1}. \tag{4.5}$$

For $r>1$, $D_r = \{x\in\mathbb{R}^2 \,|\, |x| \leqq r\}$ the functions $F^r_{n,k}(x_1,x_2) := F_{n,k}(x_1/r, x_2/r)$, $g^r_{n,k}(s,\varphi) := g_{n,k}(s/r,\varphi)$ form complete orthogonal systems in $L_2(D_r)$, $L_2(Z_r, 1/w_r)$, respectively, with $Z_r = [-r,r]\times[0,2\pi)$ and $w_r(s) := w(s/r)$, and we have

$$RF^r_{n,k}(s,\varphi) = rg^r_{n,k}(s,\varphi),$$
$$\|g_{n,k}\|_{L_2(Z_r,1/w_r)} = \sqrt{r}\,\frac{2\pi}{|n|+2k+1}.$$

Each $f\in L_2(D_r)$ can be written as

$$f = \sum_{n=-\infty}^{\infty}\sum_{k=0}^{\infty} \gamma_{n,k}F^r_{n,k}. \tag{4.6}$$

Applying the Radon transform, we obtain

$$Rf = \sum_{n=-\infty}^{\infty}\sum_{k=0}^{\infty} \gamma_{n,k} r g^r_{n,k}.$$

Hence

$$\begin{aligned}\gamma_{n,k} &= \frac{1}{r^2}\left(\frac{|n|+2k+1}{2\pi}\right)^2 \langle Rf, g^r_{n,k}\rangle_{L_2(Z_r,1/w_r)} \\ &= \frac{1}{r}\left(\frac{|n|+2k+1}{2\pi}\right)^2 \int_0^{2\pi}\int_0^{\pi} Rf(r\cos\psi,\varphi)\sin((|n|+2k+1)\psi)\cdot e^{-in\varphi}\,d\psi\,d\varphi.\end{aligned} \tag{4.7}$$

Using

$$g(\alpha,\beta) = \begin{cases} Rf(r\cos\alpha, \beta-\alpha), & 0\leqq\alpha\leqq\pi, \\ 0, & \text{else,}\end{cases}$$

we obtain

$$\gamma_{n,k} = \frac{1}{r}\left(\frac{|n|+2k+1}{2\pi}\right)^2 \int_0^{2\pi}\int_0^{\pi} g(\alpha,\beta)\sin((|n|+2k+1)\alpha)\, e^{-in\beta}\, e^{in\alpha}\,d\alpha\,d\beta. \tag{4.8}$$

If we sample the data on the PET-grid and apply the trapezoidal rule to (4.8), we obtain an approximate solution:

$$\begin{aligned}\Gamma_{n,k} = \frac{1}{r}\left(\frac{|n|+2k+1}{2\pi}\right)^2 \cdot 2\left(\frac{\pi}{p}\right)^2 \sum_{l=1}^{p-1}\left(\sum_{j=0}^{p-1} g(\alpha_l,\beta_j)\, e^{-2\pi inj/p}\right) \\ \cdot e^{inl\pi/p}\sin((|n|+2k+1)l\pi/p)\end{aligned} \tag{4.9}$$

with $(\alpha_l,\beta_j) = (l\pi/p, 2\pi j/p)$.

In the series expansion (4.6) we take into account only the polynomials of degree $\leqq M$, i.e., we compute

$$f_M = \sum_{n=-M}^{M}\sum_{k=0}^{[(M-|n|)/2]} \gamma_{n,k}F^r_{n,k}. \tag{4.10}$$

Then, with $\gamma_{n,k} = \lambda_{n,k} + i\zeta_{n,k}$, $\gamma_{n,k} = \bar{\gamma}_{-n,k}$, for $x = |x|\binom{\cos\psi}{\sin\psi} \in D_r$, we must evaluate the sum

$$f_M(x) = \sum_{k=0}^{[M/2]} \lambda_{0,k} Q_{0,k}\left(\frac{|x|^2}{r^2}\right) + 2\sum_{n=1}^{M}\sum_{k=0}^{[M-n/2]} (\lambda_{n,k}\cos(n\psi) - \zeta_{n,k}\sin(n\psi))\left(\frac{|x|}{r}\right)^n Q_{n,k}\left(\frac{|x|^2}{r^2}\right). \tag{4.11}$$

This can be done by the following algorithm.

Step 1. Compute $\Gamma_{m,k} = \Lambda_{n,k} + iZ_{n,k}$, $n = 0, \cdots, M$, $k = 0, \cdots, [(M-n)/2]$ by means of the trapezoidal rule.

Step 2. For $r_l = l/N$, $l = 0, \cdots, N$ compute

$$\tau_n(r_l) = \sum_{k=0}^{[M-n/2]} \Lambda_{n,k}\left(\frac{r_l}{r}\right)^n Q_{n,k}\left(\left(\frac{r_l}{r}\right)^2\right), \qquad n = 0, \cdots, M,$$

$$\sigma_n(r_l) = \sum_{k=0}^{[M-n/2]} Z_{n,k}\left(\frac{r_l}{r}\right)^n Q_{n,k}\left(\left(\frac{r_l}{r}\right)^2\right), \qquad n = 1, \cdots, M.$$

Step 3. For $x = |x|\binom{\cos\psi}{\sin\psi} \in D$ find j such that $r_j \leqq |x| < r_{j+1}$, and compute an approximation F_M to f_M by interpolation:

$$F_M(x) = N\Big\{(r_{j+1} - |x|)\tau_0(r_j) + (|x| - r_j)\tau_0(r_{j+1}) + 2\sum_{n=1}^{M} \cos(n\psi)((r_{j+1} - |x|)\tau_n(r_j) + (|x| - r_j)\tau_n(r_{j+1})) - \sin(n\psi)((r_{j+1} - |x|)\sigma_n(r_j) + (|x| - r_j)\sigma_n(r_{j+1}))\Big\}.$$

Doing so for all points x in a $Q \times Q$-grid, we find that this is an algorithm of complexity $O(p^3)$, if M, N, Q are of the same magnitude p.

Marr's algorithm is optimal in a certain sense: If

$$M < p - 1, \tag{4.12}$$

then

$$\|f - f_M\|_{L_2(D_r)} \leqq \|f - h\|_{L_2(D_r)} \tag{4.13}$$

for all polynomials h of degree $\leqq M$. (See Marr [10].)

5. Error estimate for the trapezoidal rule. The most important steps in the presented algorithms are evaluations of multiple integrals using the trapezoidal rule where the functions to be integrated are known on grids of the form $\{Wl \mid l \in \mathbb{Z}^n\}$.

First we must make some assumptions on the matrix W:

DEFINITION 5.1. A nonsingular $(n+m, n+m)$-matrix W is called a feasible (n, m)-grid-matrix if there is an integer $(n+m, m)$-matrix L such that $WL = \binom{0}{2\pi I_m}$.

A feasible (n, m)-grid-matrix generates a grid in $\mathbb{R}^{n+m}$ that is 2π-periodic in the last m arguments, i.e.,

$$\binom{x}{y} \in \{Wl \mid l \in \mathbb{Z}^{n+m}\}, \qquad x \in \mathbb{R}^n, \quad y \in \mathbb{R}^m$$

implies

$$\binom{x}{y + 2\pi j} \in \{Wl \mid l \in \mathbb{Z}^{n+m}\} \quad \text{for all } j \in \mathbb{Z}^m.$$

If W is a nonsingular (n, n)-matrix and $g \in C(\mathbb{R}^n)$ is W-periodic, i.e., $g(x+Wj) = g(x)$ for all $j \in \mathbb{Z}^n$, then the Fourier-series for g reads

$$g(x) = \sum_{k \in \mathbb{Z}^n} \hat{g}_k \exp(2\pi i (W^{-1})^T k \cdot x)$$

with

$$\begin{aligned} \hat{g}_k &= \int_{[0,1]^n} g(Wx)\, e^{-2\pi i k x}\, dx \\ &= |\det W|^{-1} \int_{W([0,1]^n)} g(x) \exp(-2\pi i (W^{-1})^T k \cdot x)\, dx. \end{aligned} \tag{5.1}$$

Let $T = [0, 2\pi)$ and $A = \{j \in \mathbb{Z}^{n+m} \mid Wj \in \mathbb{R}^n \times T^m\}$; then $\{Wj \mid j \in A\}$ is a subgroup of $\mathbb{R}^n \times T^m$ if W is a feasible (n, m)-grid-matrix. To obtain an estimate for the trapezoidal rule we use the Poisson summation formula for this subgroup.

LEMMA 5.1. *Let W be a feasible (n, m)-grid-matrix and for $j \in \mathbb{Z}^{n+m}$ define $U_j \subset \mathbb{R}^{n+m}$ by $U_j = \{Wj + Wx \mid x \in [0, 1]^{n+m}\}$. Let $f \in L^{n,m}$ be 2π-periodic in the last m arguments. Then for $k \in \mathbb{Z}^{n+m}$,*

$$\int_{\cup_{j \in A} U_j} f(z) \exp(-2\pi i (W^{-1})^T k \cdot z)\, dz = \int_{\mathbb{R}^n \times T^m} f(z) \exp(-2\pi i (W^{-1})^T k \cdot z)\, dz. \tag{5.2}$$

Proof.

$$W^{-1} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}$$

is the inverse of a feasible (n, m)-grid-matrix W if and only if $2\pi w_{12}$ and $2\pi w_{22}$ are integers. Hence the function $f(z) \exp(-2\pi i (W^{-1})^T k \cdot z)$ is 2π-periodic with respect to the last m arguments. The 2π-periodicity in the last m arguments of the grid $\{Wj \mid j \in \mathbb{Z}^{n+m}\}$ then implies (5.2). □

LEMMA 5.2 (Poisson summation formula). *Let W and A be defined as in Lemma 5.1, $f \in L^{n,m}$, such that the series $\sum_{j \in A} f(x + Wj) =: g(x)$ converges absolutely for all x. Let the W-periodic function g be continuously differentiable. Then*

$$|\det W| \sum_{j \in A} f(Wj) = (2\pi)^{n+m/2} \sum_{l \in \mathbb{Z}^{n+m}} \hat{f}(2\pi (W^{-1})^T l). \tag{5.3}$$

Proof. g is W-periodic. Hence

$$g(x) = \sum_{k \in \mathbb{Z}^{n+m}} \hat{g}_k \exp(2\pi i (W^{-1})^T k \cdot x),$$

with

$$\begin{aligned} \hat{g}_k &= |\det W|^{-1} \sum_{j \in A} \int_{W([0,1]^{n+m})} f(x + Wj) \exp(-2\pi i (W^{-1})^T k \cdot x)\, dx \\ &= |\det W|^{-1} \int_{\cup_{j \in A} U_j} f(x) \exp(-2\pi i (W^{-1})^T k \cdot x)\, dx, \end{aligned}$$

the sets U_j being defined as in Lemma 5.1. Using (5.2) we obtain

$$\hat{g}_k = |\det W|^{-1} (2\pi)^{n+m/2} \hat{f}(2\pi (W^{-1})^T k).$$

(Note that the last m arguments of $2\pi (W^{-1})^T k$ are integers!) Choosing $x = 0$ yields

$$|\det W| \sum_{j \in A} f(Wj) = |\det W| g(0) = (2\pi)^{n+m/2} \sum_{k \in \mathbb{Z}^{n+m}} \hat{f}(2\pi (W^{-1})^T k). \qquad \square$$

As an immediate consequence of Lemma 5.2 we obtain a first result related to the error of the trapezoidal rule.

COROLLARY 5.1. *Let f, W, and A be defined as in Lemma 5.2. Then*

$$|\det W| \sum_{j\in A} f(Wj) - \int_{\mathbb{R}^n\times T^m} f(x)\,dx = (2\pi)^{n+m/2} \sum_{\substack{k\neq 0\\ k\in\mathbb{Z}^{n+m}}} \hat{f}(2\pi(W^{-1})^T k). \tag{5.4}$$

THEOREM 5.1. *Let W be a feasible (n, m)-grid-matrix, $A=\{j\in\mathbb{Z}^{n+m} \mid Wj\in\mathbb{R}^n \times T^m\}$, and let $f_1\cdot\bar{f}_2\in L^{n,m}$ be such that the series $\sum_{j\in A} f_1(x+Wj)\bar{f}_2(x+Wj)=:g(x)$ converges absolutely and g is continuously differentiable. Let the set $K\subset\mathbb{R}^n\times\mathbb{Z}^m$ be such that the translated sets $K+2\pi(W^{-1})^T l$, $l\in\mathbb{Z}^{n+m}$ are mutually disjoint and let $\hat{f}_1$ be bounded almost everywhere on K by a constant $c(f_1, K)$. Then*

$$|\det W| \sum_{j\in A} f_1(Wj)\bar{f}_2(Wj) = \int_{\mathbb{R}^n\times T^m} f_1(x)\bar{f}_2(x)\,dx + e,$$

with

$$\begin{aligned} |e| \leqq & \left| \sum_{\substack{l\neq 0\\ l\in\mathbb{Z}^{n+m}}} \sum_{k\in\mathbb{Z}^m} \int_{\binom{\xi}{k}\notin K} \hat{f}_1\binom{\xi}{k}\bar{\hat{f}}_2\left(\binom{\xi}{k} - 2\pi(W^{-1})^T l\right) d\xi \right| \\ & + c(f_1, K) \sum_{k\in\mathbb{Z}^m} \int_{\binom{\xi}{k}\notin K} \left|\hat{f}_2\binom{\xi}{k}\right| d\xi. \end{aligned} \tag{5.5}$$

Proof. Using Lemma 5.2 and the convolution theorem, we obtain

$$|\det W| \sum_{j\in A} (f_1\bar{f}_2)(Wj) = \sum_{l\in\mathbb{Z}^{n+m}} \sum_{k\in\mathbb{Z}^m} \int_{\mathbb{R}^n} \hat{f}_1\binom{\xi}{k}\bar{\hat{f}}_2\left(\binom{\xi}{k} - 2\pi(W^{-1})^T l\right) d\xi.$$

Hence

$$\begin{aligned} & |\det W| \sum_{j\in A} (f_1\bar{f}_2)(Wj) - \int_{\mathbb{R}^n\times T^m} f_1(x)\bar{f}_2(x)\,dx \\ & = \sum_{\substack{l\neq 0\\ l\in\mathbb{Z}^{n+m}}} \sum_{k\in\mathbb{Z}^m} \int_{\mathbb{R}^n} \hat{f}_1\binom{\xi}{k}\bar{\hat{f}}_2\left(\binom{\xi}{k} - 2\pi(W^{-1})^T l\right) d\xi \\ & =: e. \end{aligned}$$

Defining $U_l = K - 2\pi(W^{-1})^T l$ we have

$$\begin{aligned} |e| \leqq & \left| \sum_{\substack{l\neq 0\\ l\in\mathbb{Z}^{n+m}}} \sum_{k\in\mathbb{Z}^m} \int_{\binom{\xi}{k}\notin K} \hat{f}_1\binom{\xi}{k}\bar{\hat{f}}_2\left(\binom{\xi}{k} - 2\pi(W^{-1})^T l\right) d\xi \right| \\ & + c(f_1, K) \sum_{\substack{l\neq 0\\ l\in\mathbb{Z}^{n+m}}} \sum_{k\in\mathbb{Z}^m} \int_{\binom{\xi}{k}\in U_l} \left|\hat{f}_2\binom{\xi}{k}\right| d\xi. \end{aligned}$$

The sets U_l, $l\in\mathbb{Z}^{n+m}$ are mutually disjoint; thus

$$\sum_{\substack{l\neq 0\\ l\in\mathbb{Z}^{n+m}}}{}^{\cdot} \sum_{k\in\mathbb{Z}^m} \int_{\binom{\xi}{k}\in U_l} \left|\hat{f}_2\binom{\xi}{k}\right| d\xi \leqq \sum_{k\in\mathbb{Z}^m} \int_{\binom{\xi}{k}\notin K} \left|\hat{f}_2\binom{\xi}{k}\right| d\xi. \qquad \square$$

There are some immediate conclusions.

COROLLARY 5.2. *Let W, A, f_1, f_2, K be as in Theorem 5.1 and $\bigcup_{l\in\mathbb{Z}^{n+m}} K - 2\pi(W^{-1})^T l = \mathbb{R}^n \times \mathbb{Z}^m$. Then*

$$
\begin{aligned}
&|\det W| \sum_{j\in A} f_1(Wj)\bar{f}_2(Wj) - \int_{\mathbb{R}^n\times T^m} f_1(x)\bar{f}_2(x)\,dx \\
&= \sum_{\substack{l\neq 0\\ l\in\mathbb{Z}^{n+m}}} \sum_{k\in\mathbb{Z}^m} \int_{\binom{\xi}{k}\notin K} \hat{f}_1\binom{\xi}{k}\bar{f}_2\left(\binom{\xi}{k} - 2\pi(W^{-1})^T l\right) d\xi \qquad (5.6)\\
&\quad + \sum_{k\in\mathbb{Z}^m} \int_{\binom{\xi}{k}\notin K} \hat{f}_1\left(\binom{\xi}{k} \bmod K\right)\bar{f}_2\binom{\xi}{k}\,d\xi
\end{aligned}
$$

with $\binom{\xi}{k} \bmod K$ defined as follows:

$\tau = \binom{\xi}{k} \bmod K \Leftrightarrow \tau \in K$ *and there is* $j\in\mathbb{Z}^{n+m}$ *such that* $\tau + 2\pi(W^{-1})^T j = \binom{\xi}{k}$.

COROLLARY 5.3. *Let $f_1, f_2 \in \mathcal{S}(\mathbb{R}^n)$, let W be a nonsingular (n, n)-matrix, $K\subset\mathbb{R}^n$ such that the translated sets $K + 2\pi(W^{-1})^T l$, $l\in\mathbb{Z}^n$ are mutually disjoint, and let $\bigcup_{l\in\mathbb{Z}^n} K + 2\pi(W^{-1})^T l = \mathbb{R}^n$. Then*

$$
\begin{aligned}
&|\det W| \sum_{j\in\mathbb{Z}^n} f_1(Wj)\bar{f}_2(Wj) - \int_{\mathbb{R}^n} f_1(x)\bar{f}_2(x)\,dx \\
&= \sum_{\substack{l\neq 0\\ l\in\mathbb{Z}^n}} \int_{\mathbb{R}^n\setminus K} \hat{f}_1(\xi)\bar{f}_2(\xi - 2\pi(W^{-1})^T l)\,d\xi \qquad (5.7)\\
&\quad + \int_{\mathbb{R}^n\setminus K} \hat{f}_1(\xi \bmod K)\bar{f}_2(\xi)\,d\xi.
\end{aligned}
$$

It is obvious from the following corollary that the supports of the Fourier transform of the functions f_1, f_2 play an important role.

COROLLARY 5.4. *If in Theorem 5.1 there is the additional assumption that* $\operatorname{supp}(\hat{f}_1)\cup\operatorname{supp}(\hat{f}_2)\subset K$, *then*

$$
|\det W| \sum_{j\in A} f_1(Wj)\bar{f}_2(Wj) = \int_{\mathbb{R}^n\times T^m} f_1(x)\bar{f}_2(x)\,dx. \qquad (5.8)
$$

6. Resolution of filtered backprojection. The essential step in the filtered backprojection method is the evaluation of the integral

$$
W_b * f(x) = \int_0^{2\pi}\int_{-\infty}^{\infty} w_b\left(x\cdot\binom{\cos\varphi}{\sin\varphi} - s\right) Rf(s,\varphi)\,ds\,d\varphi.
$$

Let the function $q: D\times\mathbb{R}\times[0, 2\pi]\to\mathbb{R}$ be defined by

$$
q(x, s, \varphi) = w_b\left(x\cdot\binom{\cos\varphi}{\sin\varphi} - s\right). \qquad (6.1)
$$

Remark. Occasionally we do not distinguish between row vectors and column vectors, i.e., we write, e.g., $(s,\varphi) + \binom{t}{\psi} = (s+t, \varphi+\psi)$.

LEMMA 6.1. *Let $x = |x|\binom{\cos\psi}{\sin\psi} \in D$; then for the Fourier transform of $q(x,\cdot,\cdot)$ with respect to s and φ (i.e., in the sense of $L^{1,1}$) we have*

$$
\hat{q}(x,\sigma,k) = \frac{1}{2}\frac{1}{2\pi} i^{-k}|\sigma|\hat{\Phi}\left(\frac{\sigma}{b}\right) J_k(\sigma|x|)\, e^{-ik\psi} \qquad (6.2)
$$

with Φ *defined by* (4.1) *and* J_k *the Bessel functions of the first kind. Furthermore, the estimates*

$$
\begin{aligned}
|\hat{q}(x,\sigma,k)| &\leqq \frac{1}{2}\frac{1}{2\pi}|\sigma|\hat{\Phi}\begin{pmatrix}\sigma\\ b\end{pmatrix}\sup_{0\leqq r\leqq 1}|J_k(\sigma r)| \\
&\leqq \frac{1}{2}\frac{1}{2\pi}|\sigma|\hat{\Phi}\begin{pmatrix}\sigma\\ b\end{pmatrix} \qquad (6.3)\\
&\leqq \frac{1}{2}\frac{1}{2\pi}b
\end{aligned}
$$

hold for all $x\in D$.

Proof. Using the definition of q, we obtain

$$
\begin{aligned}
\hat{q}(x,\sigma,k) &= \frac{1}{2\pi}\int_0^{2\pi}\int_{-\infty}^{\infty} w_b\left(x\cdot\begin{pmatrix}\cos\varphi\\ \sin\varphi\end{pmatrix}-s\right)e^{-is\sigma}\,ds\,e^{-i\varphi k}\,d\varphi\\
&= \frac{1}{2\pi}\int_0^{2\pi}\int_{-\infty}^{\infty} w_b(t)\,e^{it\sigma}\,dt\,\exp\left(-i|x|\sigma\cos(\varphi-\psi)\right)e^{-i\varphi k}\,d\varphi\\
&= \sqrt{2\pi}\,\hat{w}_b(-\sigma)i^k J_k(-|x|\sigma)\,e^{-ik\psi}\quad\text{(see (2.8))}\\
&= \frac{1}{2}\frac{1}{2\pi}\,i^{-k}|\sigma|\hat{\Phi}\begin{pmatrix}\sigma\\ b\end{pmatrix}J_k(\sigma|x|)\,e^{-ik\psi}\quad\text{(see (4.2)).}
\end{aligned}
$$

The estimates $|\hat{\Phi}(\sigma)|\leqq 1$ (see (4.1)) and $|J_k(\sigma)|\leqq 1$ (see Abramowitz and Stegun [1, formula 9.1.60]) immediately imply the estimates (6.3). □

By means of this lemma we are able to prove an estimate for filtered backprojection.

THEOREM 6.1. *Let* $f\in C_0^\infty(D)$, $g(s,\varphi)=Rf(s,\varphi)$, *and* K, W, A *be defined as in Theorem* 5.1. *Then*

$$
|\det W|\sum_{j\in A} q(x,Wj)g(Wj) = W_b * f(x) + e_1 + e_2
$$

with

$$
\begin{aligned}
|e_1| &\leqq \frac{1}{2\pi}\,b\sum_{m\in\mathbb{Z}}\int_{\binom{\sigma}{m}\notin K}|\hat{g}(\sigma,m)|\,d\sigma,\\
|e_2| &\leqq \frac{1}{8\pi^2}\,b|\det W|\sum_{j\in A}|g(Wj)|\cdot\sum_{m\in\mathbb{Z}}\sup_{0\leqq r\leqq 1}\int_{\substack{\binom{\sigma}{m}\notin K\\ |\sigma|\leqq b}}|J_m(\sigma r)|\,d\sigma.
\end{aligned}
\qquad (6.4)
$$

Proof. Let χ_K be the characteristic function of the set K, i.e.,

$$
\chi_K(\sigma,m)=\begin{cases}1 & \text{if } (\sigma,m)\in K,\\ 0 & \text{else;}\end{cases}
$$

then

$$
\begin{aligned}
\sum_{j\in A} q(x,Wj)g(Wj) &= \frac{1}{2\pi}\sum_{j\in A}\sum_{m\in\mathbb{Z}}\int_{\binom{\sigma}{m}\notin K}\hat{q}(x,\sigma,m)\,e^{iWj\cdot\binom{\sigma}{m}}\,d\sigma\,g(Wj)\\
&\quad+\sum_{j\in A}(\hat{q}\cdot\chi_K)^{\sim}(x,Wj)g(Wj),
\end{aligned}
$$

and

$$W_b * f(x) = \int_0^{2\pi} \int_{\mathbb{R}} q(x, s, \varphi) g(s, \varphi)\, ds\, d\varphi$$

$$= \sum_{m \in \mathbb{Z}} \int_{\mathbb{R}} \hat{q}(x, \sigma, m) \bar{\hat{g}}(\sigma, m)\, d\sigma \quad \text{(Parseval's relation)}$$

$$= \sum_{m \in \mathbb{Z}} \int_{\binom{\sigma}{m} \notin K} \hat{q}(x, \sigma, m) \bar{\hat{g}}(\sigma, m)\, d\sigma$$

$$+ \int_0^{2\pi} \int_{\mathbb{R}} (\hat{q} \cdot \chi_K)^{\vee}(x, s, \varphi) g(s, \varphi)\, ds\, d\varphi.$$

Hence,

$$|\det W| \sum_{j \in A} q(x, Wj) g(Wj) - W_b * f(x) = e_{11} - e_{12} + e_2,$$

with

$$e_{11} = |\det W| \sum_{j \in A} (\hat{q} \cdot \chi_K)^{\vee}(x, Wj) g(Wj)$$

$$- \int_0^{2\pi} \int_{\mathbb{R}} (\hat{q} \cdot \chi_K)^{\vee}(x, s, \varphi) g(s, \varphi)\, ds\, d\varphi$$

$$e_{12} = \sum_{m \in \mathbb{Z}} \int_{\binom{\sigma}{m} \notin K} \hat{q}(x, \sigma, m) \bar{\hat{g}}(\sigma, m)\, d\sigma,$$

$$e_2 = \frac{|\det W|}{2\pi} \sum_{j \in A} \sum_{m \in \mathbb{Z}} \int_{\binom{\sigma}{m} \notin K} \hat{q}(x, \sigma, m)\, e^{iWj \cdot \binom{\sigma}{m}}\, d\sigma\, g(Wj).$$

Since $g(s, \varphi) = 0$ for $|s| > 1$ and $(\hat{q} \cdot \chi_K)^{\vee}$ is continuously differentiable the series

$$\sum_{j \in A} (\hat{q} \cdot \chi_K)^{\vee}(x, (s, \varphi) + Wj) g((s, \varphi) + Wj)$$

satisfies the assumptions of Theorem 5.1 and we obtain

$$|e_{11}| \leqq \sup_{\binom{\sigma}{m} \in K} |\hat{q}(x, \sigma, m)| \sum_{m \in \mathbb{Z}} \int_{\binom{\sigma}{m} \notin K} |\hat{g}(\sigma, m)|\, d\sigma.$$

Applying Lemma 6.1 we find

$$|e_{11}| \leqq \frac{1}{2} \frac{1}{2\pi} b \sum_{m \in \mathbb{Z}} \int_{\binom{\sigma}{m} \notin K} |\hat{g}(\sigma, m)|\, d\sigma.$$

Lemma 6.1 also implies

$$|e_{12}| \leqq \frac{1}{2} \frac{1}{2\pi} b \sum_{m \in \mathbb{Z}} \int_{\binom{\sigma}{m} \notin K} |\hat{g}(\sigma, m)|\, d\sigma$$

and we obtain the desired estimate for $e_1 = e_{11} - e_{12}$. Again from (6.3) and (4.1) we deduce the estimate

$$|e_2| \leqq \frac{1}{8\pi^2} b |\det W| \sum_{j \in A} |g(Wj)| \sum_{m \in \mathbb{Z}} \sup_{0 \leqq r \leqq 1} \int_{\substack{\binom{\sigma}{m} \notin K \\ |\sigma| \leqq b}} |J_m(\sigma r)|\, d\sigma. \qquad \square$$

In the parallel case we now can estimate the integration error for every grid defined by a feasible $(1, 1)$-grid-matrix W. First we do this for the standard grid.

THEOREM 6.2. (Estimate for the standard grid.) *Let $f \in C_0^\infty(D)$, $g(s, \varphi) = Rf(s, \varphi)$, $0 < \theta < 1$, $b \geqq 1$ and $2p = 2b/\theta \in \mathbb{N}$, $\pi q = b$. Let the standard grid be defined by the matrix*

$$W = \begin{pmatrix} \pi/b & 0 \\ 0 & \pi\theta/b \end{pmatrix} = \begin{pmatrix} 1/q & 0 \\ 0 & \pi/p \end{pmatrix}$$

and let $A = \{j \in \mathbb{Z}^2 \mid Wj \in \mathbb{R} \times T\}$. Then

$$|\det W| \sum_{j \in A} q(x, Wj) g(Wj) = W_b * f(x) + e_1 + e_2,$$

with

$$|e_1| \leqq \eta(\theta, b) \|f\|_{L_1(D)} + \frac{4}{\pi\theta} b\varepsilon_0(f, b),$$

$$|e_2| \leqq \eta\left(\theta, \frac{b}{\theta}\right) \|f\|_{L_\infty(D)}. \tag{6.5}$$

Proof. W is a feasible $(1, 1)$-grid-matrix and for the rectangle $K = \{(\sigma, m) \in \mathbb{R} \times \mathbb{Z} \mid -b \leqq \sigma < b, -b/\theta \leqq m < b/\theta\}$, we have $\bigcup_{j \in \mathbb{Z}^2} K + 2\pi(W^{-1})^T j = \mathbb{R} \times \mathbb{Z}$, and the translated sets $K + 2\pi(W^{-1})^T j, j \in \mathbb{Z}^2$, are mutually disjoint. Applying Theorem 6.1, we obtain

$$|\det W| \sum_{j \in A} q(x, Wj) g(Wj) = W_b * f(x) + e_1 + e_2,$$

with

$$|e_1| \leqq \frac{b}{2\pi} \sum_{m \in \mathbb{Z}} \int_{\binom{\sigma}{m} \notin K} |\hat{g}(\sigma, m)| \, d\sigma,$$

$$|e_2| \leqq \frac{1}{8\pi^2} b |\det W| \sum_{j \in A} |g(Wj)| \sum_{m \in \mathbb{Z}} \sup_{0 \leqq r \leqq 1} \int_{\substack{\binom{\sigma}{m} \notin K \\ |\sigma| \leqq b}} |J_m(\sigma r)| \, d\sigma.$$

In Theorem 3.1 we have defined the set

$$\tilde{K} = \left\{(\sigma, m) \in \mathbb{R} \times \mathbb{Z} \,\middle|\, |\sigma| < b, |m| < \max\left(|\sigma|/\theta, \left(\frac{1}{\theta} - 1\right) b\right)\right\}.$$

Obviously, $\tilde{K} \subset K$ and (3.3) implies

$$\sum_{m \in \mathbb{Z}} \int_{\binom{\sigma}{m} \notin K} |\hat{g}(\sigma, m)| \, d\sigma \leqq \eta(\theta, b) \|f\|_{L_1(D)} + \frac{8}{\theta} \varepsilon_0(f, b),$$

and the estimate for e_1 immediately follows.

It is easily seen that there are at most $2p(2q+1)$ points with $g \neq 0$ in the grid. Therefore with $\det W = \pi/pq$ we have

$$|\det W| \sum_{j \in A} |g(Wj)| \leqq \frac{\pi}{pq} 2p(2q+1) \|g\|_{L_\infty(Z)}$$

$$\leqq \pi\left(4 + \frac{2}{q}\right) \cdot 2\|f\|_{L_\infty(D)}.$$

Furthermore,

$$\sum_{m\in\mathbb{Z}} \sup_{0\leqq r\leqq 1} \int_{\substack{(\sigma)\notin K \\ |\sigma|\leqq b}} |J_m(\sigma r)|\, d\sigma \leqq 2 \sum_{m\geqq b/\theta} \sup_{0\leqq r\leqq 1} \int_{-m\theta}^{m\theta} |J_m(\sigma r)|\, d\sigma$$

$$= 2 \sum_{m\geqq b/\theta} \eta_1(\theta, m) \quad \text{(see (2.7))}$$

$$= 2\eta_2\left(\theta, \frac{b}{\theta}\right)$$

$$= \eta\left(\theta, \frac{b}{\theta}\right).$$

Hence,

$$|e_2| \leqq \frac{b}{4\pi}\left(4+\frac{2}{q}\right) \|f\|_{L_\infty(D)} \eta\left(\theta, \frac{b}{\theta}\right)$$

$$= \eta\left(\theta, \frac{b}{\theta}\right) \|f\|_{L_\infty(D)}. \qquad \square$$

For the interlaced grid we obtain about the same result.

THEOREM 6.3. (Estimate for the interlaced grid.) *Let* $f\in C_0^\infty(D)$, $g(s,\varphi) = Rf(s,\varphi)$, $0<\theta<1$, $0<\theta'<\theta/(2-\theta)$, $b\geqq 1$, $p = b/\theta' \in \mathbb{N}$, $\pi q = b$ *and*

$$W = \frac{\pi}{b}\begin{pmatrix} 1 & 0 \\ -\theta' & 2\theta' \end{pmatrix} = \begin{pmatrix} 1/q & 0 \\ -\pi/p & 2\pi/p \end{pmatrix}.$$

Furthermore, let A be defined as in Theorem 6.2. *Then*

$$|\det W| \sum_{j\in A} q(x, Wj) g(Wj) = W_b * f(x) + e_1 + e_2,$$

with

$$|e_1| \leqq \eta(\theta, b)\|f\|_{L_1(D)} + \frac{4}{\pi\theta} b\varepsilon_0(f, b),$$

$$|e_2| \leqq \eta\left(\theta, \left(\frac{1}{\theta}-1\right)b\right)\|f\|_{L_\infty(D)}. \tag{6.6}$$

Proof. Let K be the set defined in Theorem 3.1: $K = \{(\sigma, m)\in\mathbb{R}\times\mathbb{Z} \,|\, |\sigma|<b, |m| \leqq \max(|\sigma|/\theta, (1/\theta-1)b)\}$. Then W and K satisfy the assumptions of Theorem 6.1 and Theorem 5.1 and we again obtain

$$|\det W| \sum_{j\in A} q(x, Wj) g(Wj) = W_b * f(x) + e_1 + e_2.$$

In the same way as in the proof of Theorem 6.2 we can deduce the estimate for e_1.

The number of gridpoints where g does not vanish is $p(2q+1)$, $\det W = 2\pi/pq$. Hence,

$$|\det W| \sum_{j\in A} |g(Wj)| \leqq 4\pi\left(2+\frac{1}{q}\right)\|f\|_{L_\infty(D)}.$$

In the same way as in the proof of Theorem 6.2 we obtain

$$\sum_{m\in\mathbb{Z}} \sup_{0\leqq r\leqq 1} \int_{\substack{(\sigma_m)\notin K\\ |\sigma|\leqq b}} |J_m(\sigma r)|\, d\sigma \leqq 2 \sum_{m\geqq(1/\theta-1)b} \sup_{0\leqq r\leqq 1} \int_{-m\theta}^{m\theta} |J_m(\sigma r)|\, d\sigma$$

$$= \eta\left(\theta, \left(\frac{1}{\theta}-1\right)b\right),$$

and the second estimate follows. □

Remarks. Theorem 6.2 coincides with Theorem V.1.1 in Natterer [11]. Theorem 6.3 tells us that the filtered version $W_b * f$ of an essentially b-bandlimited function f (i.e., $\varepsilon_0(f, b)$ is small) for large b can be reliably computed from the data on the interlaced grid.

7. Resolution of Marr's algorithm. Marr's algorithm provides a polynomial approximation $f_M = \sum\sum_{|n|+2k\leqq M} \gamma_{n,k} F^r_{n,k}$ of $f \in L_2(D)$. There are two possible sources of error. First the coefficients

$$\gamma_{n,k} = \frac{1}{r}\left(\frac{|n|+2k+1}{2\pi}\right)^2 \int_0^{2\pi}\int_0^{\pi} g(\alpha,\beta)\sin((|n|+2k+1)\alpha)\, e^{-in\beta}\, e^{in\alpha}\, d\alpha\, d\beta$$

are computed by means of the trapezoidal rule. Here we have an integration error. Second we have a truncation error.

Let us have a look at the integration error first. The function

$$g(\alpha,\beta) = \begin{cases} Rf(r\cos\alpha, \beta-\alpha), & 0\leqq\alpha\leqq\pi \\ 0, & \pi<\alpha\leqq 2\pi \end{cases} \quad \text{is } 2\pi\text{-periodic}$$

with respect to both arguments. If we define

$$q_{n,k}(\alpha,\beta) = e^{-in\beta}\, e^{in\alpha} \sin((|n|+2k+1)\alpha), \tag{7.1}$$

then $q_{n,k}$ is 2π-periodic, too, and we have

$$\gamma_{n,k} = \frac{1}{r}\left(\frac{|n|+2k+1}{2\pi}\right)^2 \int_0^{2\pi}\int_0^{2\pi} q_{n,k}(\alpha,\beta)\bar{g}(\alpha,\beta)\, d\alpha\, d\beta.$$

To estimate the integration error by means of Theorem 5.1 we need some information about the Fourier transform of $q_{n,k}$.

LEMMA 7.1. (i) *Let*

$$\delta_m = \begin{cases} 1, & m=0, \\ 0, & \textit{else}; \end{cases}$$

then

$$\hat{q}_{n,k}(l,m) = \frac{\pi}{i}\,\delta_{n+m}(\delta_{n-l+|n|+2k+1} - \delta_{n-l-|n|-2k-1}). \tag{7.2}$$

(ii) *Let* $|n|+2k\leqq M$, $k\geqq 0$ *and define* $J\subset\mathbb{Z}^2$ *by*

$$J = \{(l,m)\in\mathbb{Z}^2 \,|\, |m+l|\leqq M+1,\ |m|<|m+l|\}; \tag{7.3}$$

then $\hat{q}_{n,k}(l,m) = 0$, *for all* $(l,m)\notin J$.

(iii)

$$|\hat{q}_{n,k}(l,m)|\leqq\pi. \tag{7.4}$$

Proof. Item (i) can be calculated easily; then (iii) is obvious.

(ii) Divide J into $J = J_1 \cup J_2 \cup J_3 \cup J_4$, with

$$J_1 = \{(l, m) \mid m > 0,\ m + l \geqq -(M+1),\ 2m + l \leqq 0\},$$

$$J_2 = \{(l, m) \mid m > 0,\ m + l \leqq M+1,\ l \geqq 0\},$$

$$J_3 = \{(l, m) \mid m \leqq 0,\ m + l \leqq M+1,\ 2m + l > 0\},$$

$$J_4 = \{(l, m) \mid m \leqq 0,\ m + l \geqq -(M+1),\ l \leqq 0\}.$$

(See Fig. 7.1.) From (i) we know that

$$\hat{q}_{n,k}(l, m) = \frac{\pi}{i}\,\delta_{n+m} \begin{cases} \delta_{2n-l+2k+1} - \delta_{2k+l+1}, & n \geqq 0, \\ \delta_{2k-l+1} - \delta_{2n-l-2k-1}, & n < 0. \end{cases}$$

Assume $n \geqq 0$; then $q_{n,k}(l, m) \neq 0$ implies the following:

(1) $m = -n \leqq 0$ and $2n - l + 2k + 1 = 0$, or

(2) $m = -n \leqq 0$ and $2k + l + 1 = 0$.

In the first case we have

$$-2m - l + 2k + 1 = 0$$

$$\Rightarrow 2m + l = 2k + 1 > 0,$$

$$m + l = n + 2k + 1 \leqq M + 1;$$

hence $(l, m) \in J_3$. In the second case we have

$$0 < 2k + 1 = -l,$$

$$-m - l = n + 2k + 1 \leqq M + 1;$$

hence $(l, m) \in J_4$. In the same way it is possible to treat the case $n < 0$. □

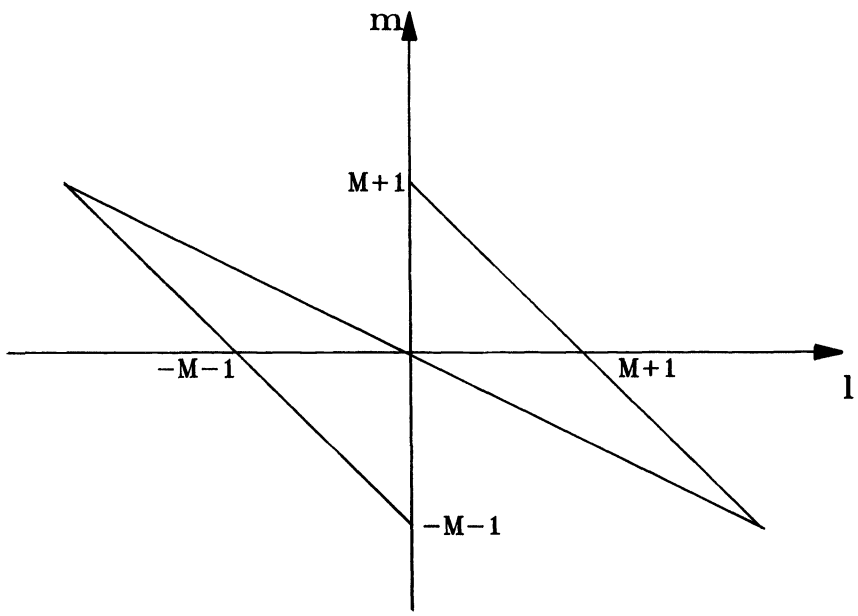


FIG. 7.1

The shape of the set J reminds us of the set

$$K=\left\{(l,m)\in\mathbb{Z}^2\,\middle|\,|l+m|\leqq\frac{rb}{\theta},\ |m|\leqq\max\left(|l+m|,\left(\frac{1}{\theta}-1\right)rb\right)\right\},$$

$0<\theta<1$, $b\geqq 1$, defined in (3.9) in connection with the PET-grid.

It is very important for our estimate that we assure $J\subset K$.

LEMMA 7.2. *If the* PET-*grid is defined by the matrix*

$$W=\begin{pmatrix}\pi/p & 0\\ 0 & 2\pi/p\end{pmatrix},\quad p=\frac{rb}{\theta'},\quad 0<\theta<1,\quad 0<\theta'<\frac{\theta}{2-\theta},$$

then

$$J\subset K\Leftrightarrow M+1\leqq\frac{rb}{\theta}. \tag{7.5}$$

If $J\subset K$, *then* $M<p-1$.

Remark. The inequality $M<p-1$ is just the optimality condition (4.12) for Marr's algorithm.

THEOREM 7.1. *Let* $f\in C_0^\infty(D)$, $g(\alpha,\beta)=Rf(r\cos\alpha,\beta-\alpha)$, $0<\theta<1$, $0<\theta'<\theta/(2-\theta)$, $b\geqq 1$, $r>1$, $p=rb/\theta'\in\mathbb{N}$ *and* $M+1\leqq rb/\theta$. *If the* PET-*grid is defined by the matrix*

$$W=\begin{pmatrix}\pi/p & 0\\ 0 & 2\pi/p\end{pmatrix}\quad\text{and}\quad A=\{j\in\mathbb{Z}^2\,|\,Wj\in[0,2\pi)\times[0,2\pi)\},$$

then with $b'=\min(b, b\ln r)$ *we have*

$$\begin{aligned}&|\det W|\sum_{j\in A}q_{n,k}(Wj)g(Wj)=\int_0^{2\pi}\int_0^{2\pi}q_{n,k}(\alpha,\beta)g(\alpha,\beta)\,d\alpha\,d\beta+e_1\\ &|e_1|\leqq\eta(\theta,b')\|f\|_{L_1(D)}+\frac{12\pi r}{\theta^2}\varepsilon_1(f,b).\end{aligned} \tag{7.6}$$

Proof. Again we apply Theorem 5.1 and obtain

$$|\det W|\sum_{j\in A}q_{n,k}(Wj)g(Wj)=\int_0^{2\pi}\int_0^{2\pi}q_{n,k}(\alpha,\beta)g(\alpha,\beta)\,d\alpha\,d\beta+e_1,$$

with

$$|e_1|\leqq\sup_{m\in K}|\hat{q}_{n,k}(m)|\sum_{m\in\mathbb{Z}^2\setminus K}|\hat{g}(m)|,$$

because $J\subset K$ (Lemma 7.2) and $\hat{q}_{n,k}(m)=0$ for $m\notin J$ (Lemma 7.1).

Then applying (7.4) and Theorem 3.2 yields the estimate

$$\begin{aligned}|e_1|&\leqq\pi\sum_{m\in\mathbb{Z}^2\setminus K}|\hat{g}(m)|\\ &\leqq\pi\left(\eta(\theta,b')\|f\|_{L_1(D)}+\frac{12r}{\theta^2}\varepsilon_1(f,b)\right)\\ &=\eta(\theta,b')\|f\|_{L_1(D)}+\frac{12\pi r}{\theta^2}\varepsilon_1(f,b).\end{aligned}$$

□

COROLLARY 7.1. *If the assumptions of Theorem* 7.1 *are valid, then*

$$\left| f_M(x) - \sum_{|n|+2k\leqq M} \left(|\det W| \sum_{j\in A} q_{n,k}(Wj) g(Wj) \right) F^r_{n,k}(x) \right| \tag{7.7}$$
$$\leqq \left(r\eta(\theta, b') \|f\|_{L_1(D)} + \frac{3}{\pi} \left(\frac{rb}{\theta^2} \right)^2 \varepsilon_1(f, b) \right) \sum\sum_{|n|+2k\leqq M} |F^r_{n,k}(x)|.$$

Thus we see that if f is an essentially b-bandlimited function, i.e., $\varepsilon_1(f, b)$ is very small, then f_M can be computed very accurately from the data on the PET-grid.

Now we are interested in the approximation of f by f_M.

LEMMA 7.3. *Let $F^r_{n,k}$ be given by* (4.4). *Then for all $|\xi| \leqq \theta(|n|+2k+1)/r$, $0<\theta<1$, the estimate*

$$|\hat{F}^r_{n,k}(\xi)| \leqq r^2(|n|+2k+1)^{-3/2}\eta(\theta, |n|+2k+1) \tag{7.8}$$

is valid.

Proof. Using Theorem 2.1 and (4.5) we obtain, for $\xi = |\xi|\binom{\cos\psi}{\sin\psi}$,

$$\hat{F}^r_{n,k}(\xi) = (2\pi)^{-1/2} r \hat{g}^r_{n,k}(|\xi|, \psi)$$
$$= \frac{r^2}{2\pi} \frac{2}{|n|+2k+1} \int_{-1}^{1} w(s) U_{|n|+2k}(s)\, e^{-isr|\xi|}\, ds\, e^{in\psi}$$
$$= r i^{|n|} (-1)^{n+k} e^{in\psi} J_{|n|+2k+1}(r|\xi|)/|\xi|$$

(see Gradshteyn and Ryzhik [6, formula 7.321]).

Using the estimate for Bessel functions in Watson [17, p. 255], and following Natterer [11, p. 198], we can deduce the estimate

$$0 \leqq \frac{J_\nu(\nu\tau)}{\tau} \leqq e^{4/3}(2\pi\nu)^{-1/2}(1-\tau^2)^{-1/4} \exp\left(-(1-\tau^2)^{3/2} \cdot \frac{\nu}{3} \right) \quad \text{for } 0<\tau<1.$$

With $r|\xi|/(|n|+2k+1) \leqq \theta < 1$ the estimates

$$0 \leqq J_{|n|+2k+1}(r|\xi|)/|\xi|$$
$$\leqq \frac{r}{|n|+2k+1} e^{4/3}(2\pi(|n|+2k+1))^{-1/2}(1-\theta^2)^{-1/4} \exp\left(-\frac{|n|+2k+1}{3}(1-\theta^2)^{3/2} \right)$$
$$= r(|n|+2k+1)^{-3/2}\eta(\theta, |n|+2k+1)$$

and (7.8) hold. □

LEMMA 7.4. *The coefficients $\gamma_{n,k}$ satisfy the estimate*

$$|\gamma_{n,k}| \leqq \frac{1}{r^2} \frac{1}{2\pi} (|n|+2k+1)^3 \|f\|_{L_1(D)}. \tag{7.9}$$

Proof. With Abramowitz and Stegun [1, Formula 22.14.6], we obtain

$$|\gamma_{n,k}| = \frac{1}{r^2} \left(\frac{|n|+2k+1}{2\pi} \right)^2 \left| \int_0^{2\pi} \int_{-r}^{r} Rf(s, \varphi) U_{|n|+2k}\left(\frac{s}{r} \right) e^{-in\varphi}\, ds\, d\varphi \right|$$
$$\leqq \frac{1}{r^2} \left(\frac{1}{2\pi} \right)^2 (|n|+2k+1)^3 \int_0^{2\pi} \int_{-r}^{r} |Rf(s, \varphi)|\, ds\, d\varphi$$
$$= \frac{1}{r^2} \left(\frac{1}{2\pi} \right)^2 (|n|+2k+1)^3 \int_0^{2\pi} \int_{-1}^{1} \int_{-\sqrt{1-s^2}}^{\sqrt{1-s^2}} \left| f\left(s\binom{\cos\varphi}{\sin\varphi} + t\binom{-\sin\varphi}{\cos\varphi} \right) \right| dt\, ds\, d\varphi$$
$$= \frac{1}{r^2} \frac{1}{2\pi} (|n|+2k+1)^3 \|f\|_{L_1(D)}.$$

□

THEOREM 7.2. *If* $0<\theta<1$, $M+1\geqslant rb/\theta$, *then*

$$(f-f_M)(x)=h_1(x)+h_2(x),$$

with

$$\begin{aligned} &|h_1(x)|\leqq \eta(\theta, M)\|f\|_{L_1(D)},\\ &\hat{h}_2(\xi)=0 \quad \text{for } |\xi|<b. \end{aligned} \tag{7.10}$$

Proof. If $|n|+2k>M$, then for $|\xi|\leqq b$ we have

$$|\xi|\leqq b\leqq \theta\frac{M+1}{r}<\theta\frac{|n|+2k+1}{r}.$$

Using Lemmas 7.3 and 7.4 we obtain

$$\begin{aligned} |(f-f_M)\hat{}(\xi)| &\leqq \sum\sum_{|n|+2k>M} |\gamma_{n,k}||\hat{F}^r_{n,k}(\xi)|\\ &\leqq \frac{1}{2\pi}\|f\|_{L_1(D)} \sum\sum_{|n|+2k>M} (|n|+2k+1)^{3/2}\eta(\theta, |n|+2k+1). \end{aligned}$$

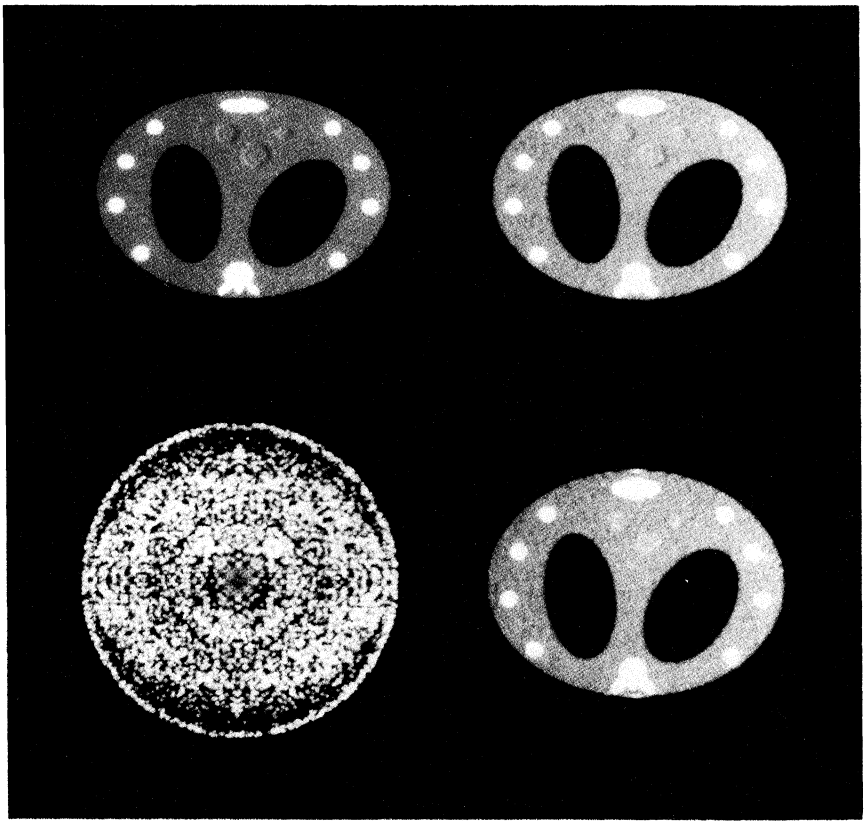


FIG. 8.1. (a) *Upper left*; (b) *upper right*; (c) *lower left*; (d) *lower right.*

For a given $l \in \mathbb{N}$ there are at most $l+1$ points $(n, k) \in \mathbb{Z} \times \mathcal{N}_0$, such that $|n|+2k = l$. Therefore

$$|(f-f_M)\hat{}(\xi)| \leqq \frac{1}{2\pi}\|f\|_{L_1(D)} \sum_{l=M+1}^{\infty} (l+1)^{5/2}\eta(\theta, l+1)$$
$$= \eta(\theta, M+2)\|f\|_{L_1(D)} \quad \text{(see (2.7))}$$

for all $|\xi| \leqq b$.

Let $D_b \subset \mathbb{R}^2$ be the circle with radius b and χ_{D_b} the characteristic function of D_b. Then with

$$\hat{h}_1 = (f-f_M)\hat{}\chi_{D_b} \quad \text{and} \quad \hat{h}_2 = (f-f_M)\hat{}(1-\chi_{D_b})$$

we have

$$(f-f_M)(x) = h_1(x)+h_2(x).$$

Of course h_2 satisfies

$$\hat{h}_2(\xi) = 0 \quad \text{for all } |\xi| < b$$

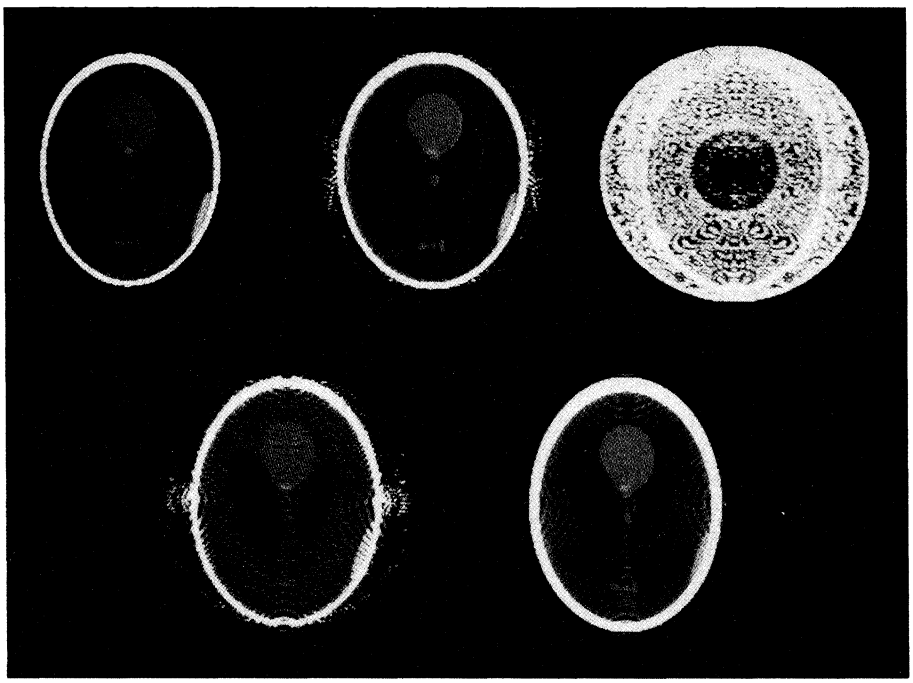


FIG. 8.2. (a) *Upper left*; (b) *upper middle*; (c) *upper right*; (d) *lower left*; (e) *lower right.*

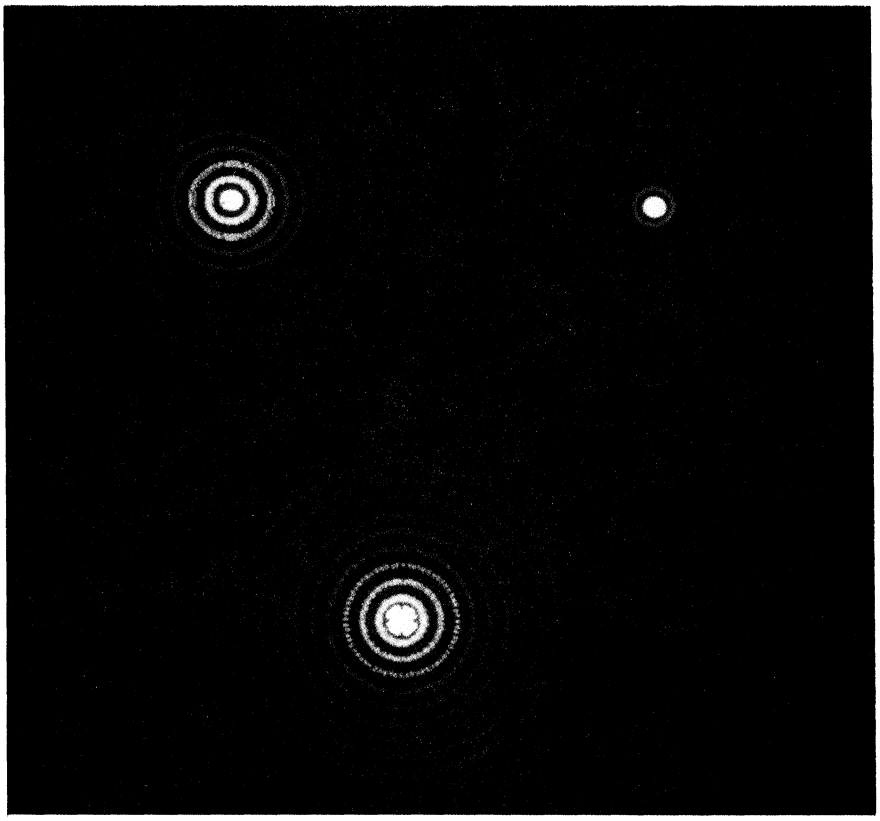


FIG. 8.3. (a) *Upper left*; (b) *upper right*; (d) *lower middle.*

and for h_1 we obtain

$$\begin{aligned}|h_1(x)| &\leqq \frac{1}{2\pi}\int_{D_b} |(f-f_M)\hat{}(\xi)|\, d\xi \\ &\leqq \frac{1}{2\pi}\pi b^2 \eta(\theta, M+2)\|f\|_{L_1(D)} \\ &\leqq \frac{1}{2}\frac{\theta^2}{r^2}(M+1)^2\eta(\theta, M+2)\|f\|_{L_1(D)} \\ &= \eta(\theta, M)\|f\|_{L_1(D)}. \end{aligned}$$

□

Remark. If $0<\theta_1,\ \theta_2<1$, $0<\theta'<\theta_2/(2-\theta_2)$ and if we choose $rb/\theta_1 \leqq M+1 \leqq rb/\theta_2$ and $p = rb/\theta'$, then for essentially b-bandlimited functions Marr's algorithm provides a good approximation for f_M (see Corollary 7.1). Theorem 7.2 then tells us that the Fourier transform of the difference $(f-f_M)$ nearly vanishes in a circle with radius b, i.e., $(f-f_M)$ is a highly oscillating function that can be eliminated by standard methods of digital image processing.

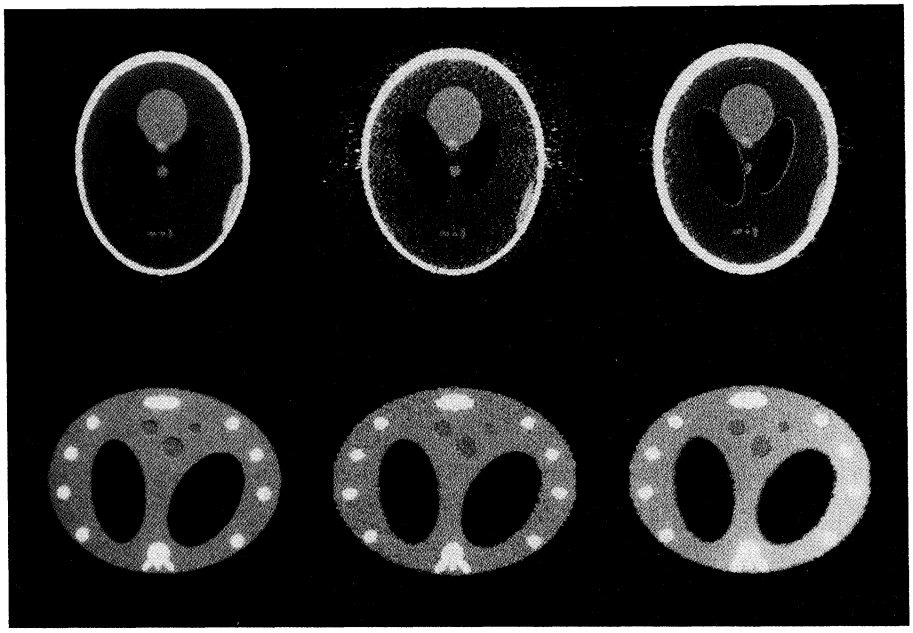


FIG. 8.4. (a) *Upper left*; (b) *upper middle*; (c) *upper right*; (d) *lower left*; (e) *lower middle*; (f) *lower right.*

8. Numerical results. We have applied filtered backprojection and Marr's algorithm to three different mathematical phantoms, two of which consist of ellipses, and the third of which is a bandlimited function. Object I (see Fig. 8.2(a)) is the so-called "Shepp-Logan-Head," representing the cross-section of a human head. Object II (see Fig. 8.1(a)) is a thorax phantom and was first used in Natterer [11]. Object III is the function

$$f(x) = \begin{cases} 1{,}800 J_1(b|x|), & |x| > 0, \\ 900, & x = 0. \end{cases}$$

Because

$$\hat{f}(\xi) = \begin{cases} 1{,}800/b^2, & |\xi| \leqq b, \\ 0, & \text{else}, \end{cases}$$

f is b-bandlimited. We have used $b = 64$. Of course the support of f is not compact, but we can ignore the data outside the unit disk.

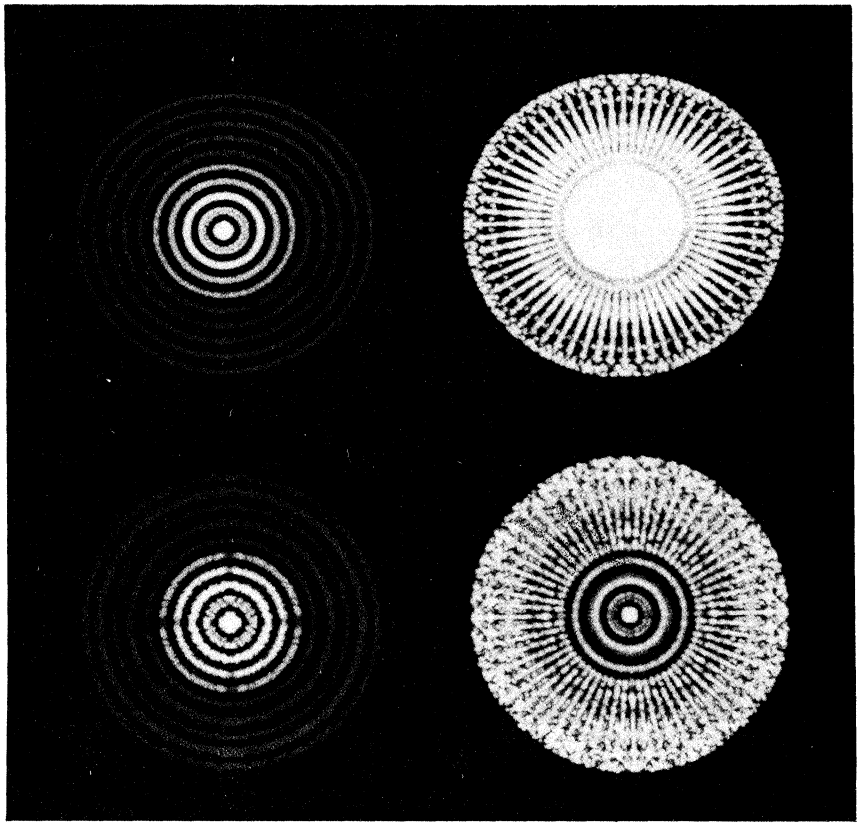


FIG. 8.5. (a) *Upper left*; (b) *upper right*; (c) *lower left*; (d) *lower right.*

8.1. Parallel geometry. Applying filtered backprojection to the data on the standard grid we obtain Fig. 8.1(b). Here we used $b = 400$, $p = 400$, and $q = 128$. All our reconstructions were done on a 160×160 grid. Obviously the reconstruction is very good. If we use the data on the interlaced grid we obtain unacceptable results. The reason for this is the interpolation step mentioned in § 4. The classical filtered backprojection algorithm computes the convolution $q(s, \varphi) = (Rf * w_b)(s, \varphi)$ just at the points s_l where the data are sampled, i.e.,

$$s_l = l/q, \qquad l = -q, \cdots, q.$$

If we use the interlaced grid there are only half the number of points in s-direction. This reduces the accuracy in the interpolation. If we compute the convolution at twice the number of points as in the classical algorithm the pictures do not become much better. Figure 8.1(c) is a reconstruction with $b = 400$, $p = 400$, and $q = 128$. Note that for the interlaced grid $q = 128$ means 128 beams for each direction, for the standatd grid $q = 128$ means $2q + 1 = 257$ beams. The reason for the bad reconstruction is probably the fact that $(Rf)\hat{}$ is not small enough outside the set K. Increasing K by

increasing p and q by 10 percent only (i.e., $p = 440$, $q = 140$) gives the reconstruction in Fig. 8.1(d). In Figs. 8.2(a), 8.2(b), 8.2(c), and 8.2(d) we have done the same calculations for object I. Smoothing reconstruction Fig. 8.2(d) by standard methods of digital image processing (see, i.e., Pratt [12, Chap. 12.3]) yields Fig. 8.2(e). The most interesting experiment is the application of the reconstruction methods to the data of object III. In this case f is bandlimited and in Fig. 8.3 we see that the original Fig. 8.3(a) is reconstructed very well indeed by filtered backprojection using the data on the standard grid (Fig. 8.3(b)) or the data on the interlaced grid (Fig. 8.3(c)). For the reconstruction in Fig. 8.3 we used $b = 64$, $p = 64$, and $q = 20$.

8.2. Fan-beam geometry. We also applied Marr's algorithm to the objects I, II, and III. We used the data on the PET-grid. The results can be seen in Fig. 8.4. In Figs. 8.4(a) and 8.4(d) we see the originals, Figs. 8.4(b) and 8.4(e) show reconstructions by Marr's algorithm. We used $p = 768$, $M = 766$, and $r = 1.0$, i.e., the X-ray source is located on the boundary of the region of reconstruction. Applying smoothing methods, we obtain Figs. 8.4(c) and 8.4(f). Finally we did reconstructions of object III. In Fig. 8.5(a) we see the original and in Fig. 8.5(c) the reconstruction by Marr's algorithm, which is quite good. In Fig. 8.5(b) there is a reconstruction provided by the standard method for fan-beam geometry, a modification of filtered backprojection. (See Natterer [11, Chap. V.1] for details.) The best reconstruction we could get from the filtered backprojection algorithm using various filters is shown in Fig. 8.5(d). Obviously filtered backprojection does not attain the optimal resolution, whereas Marr's algorithm does.

REFERENCES

[1] M. ABRAMOWITZ AND I. A. STEGUN, *Handbook of Mathematical Functions*, Dover, New York, 1970.

[2] R. N. BRACEWELL AND A. C. RIDDLE, *Inversion of fan beam scans in radio astronomy*, The Astrophys. J., 150 (1967), pp. 427–434.

[3] A. M. CORMACK, *Sampling the Radon transform with beams of finite width*, Phys. Med. Biol., 23 (1978), pp. 1141–1148.

[4] ———, *An exact subdivision of the radon transform and scanning with a positron ring camera*, Phys. Med. Biol., 25 (1980), pp. 543–544.

[5] R. A. CROWTHER, D. J. DE ROSIER, AND A. KLUG, *The reconstruction of a three-dimensional structure from projections and its application to electron microscopy*, Proc. Roy. Soc. London Ser. A, 317 (1970), pp. 319–340.

[6] I. S. GRADSHTEYN AND I. M. RYZHIK, *Table of Integrals, Series and Products*, Academic Press, New York, 1980.

[7] G. T. HERMAN, *Image Reconstruction from Projections: The Fundamentals of Computerized Tomography*, Academic Press, New York, 1980.

[8] H. KRUSE, *Die Auflösung von Rekonstruktionsalgorithmen in der Computer-Tomographie*, Ph.D. thesis, Münster, 1986.

[9] A. G. LINDGREN AND P. A. RATTEY, *The inverse discrete Radon transform with applications to tomographic imaging using projection data*, Adv. in Electronics and Electron Phys., 56 (1981), pp. 359–410.

[10] R. B. MARR, *On the reconstruction of a function on a circular domain from a sampling of its line integrals*, J. Math. Anal. Appl., 45 (1974), pp. 357–374.

[11] F. NATTERER, *The Mathematics of Computerized Tomography*, Wiley/Teubner, New York, Stuttgart, 1986.

[12] W. K. PRATT, *Digital Image Processing*, John Wiley, New York, 1978.

[13] G. N. RAMACHANDRAN AND A. V. LAKSHMINARAYANAN, *Three-dimensional reconstruction from radiographs and electron micrographs: Application of convolution instead of Fourier transforms*, Proc. Nat. Acad. Sci. U.S.A., 68 (1971), pp. 2236–2240.

[14] P. A. RATTEY AND A. G. LINDGREN, *Sampling the* 2-D *Radon transform with parallel- and fan-beam projections*, Tech. Report 5-33285-01, Department of Electrical Engineering, University of Rhode Island, Kingston, RI 2881, 1980.

[15] L. A. SHEPP AND B. F. LOGAN, *The Fourier reconstruction of a head section*, IEEE Trans. Nucl. Sci., 21 (1974), pp. 21-43.

[16] K. T. SMITH, *Reconstruction formulas in computed tomography*, Proc. Amer. Math. Soc., 27 (1983), pp. 7-23.

[17] G. N. WATSON, *A Treatise on Bessel Functions*, University Press, Cambridge, 1952.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 475-487, May 1989
© 1989 Society for Industrial and Applied Mathematics
005

A FAST ALGORITHM FOR THE NUMERICAL EVALUATION OF CONFORMAL MAPPINGS*

S. T. O'DONNELL† AND V. ROKHLIN†

Abstract. An algorithm is presented for the construction of conformal mappings from arbitrary simply connected regions in the complex plane onto the unit disk. The algorithm is based on a combination of the Kerzman-Stein integral equation (see [*Math. Anal.*, 236 (1978), pp. 85-93]) and the Fast Multipole Method for the evaluation of Cauchy-type integrals (see [V. Rokhlin, *J. Comput. Phys.*, 60 (1985), pp. 187-207], [L. Greengard and V. Rokhlin, *J. Comput. Phys.*, 73 (1987), pp. 325-348], [J. Carrier, L. Greengard, and V. Rokhlin, *SIAM J. Sci. Statist. Comput.*, 9 (1988), pp. 669-686], [L. F. Greengard, Ph.D. thesis, Department of Computer Science, Yale University, New Haven, CT, 1987]). Previously published methods for the construction of conformal mappings via the Kerzman-Stein equation have an asymptotic CPU time estimate of the order $O(n^2)$, where n is the number of nodes in the discretization of the boundary of the region being mapped. The method presented here has an estimate of the order $O(n)$, making it an approach of choice in many situations. The performance of the algorithm is illustrated by several numerical examples.

Key words. conformal mapping, Szegö kernel, fast multipole method, second kind integral equations

AMS(MOS) subject classifications. 30C30, 65E05, 65R20, 70F10

1. Introduction.

Conformal mappings have been a popular analytical tool in the theory of partial differential equations for more than a century (see, for example, [8], [11], [15]). As a mathematical problem, the construction of a conformal mapping from a more or less arbitrary simply connected region Ω onto a disk is solved satisfactorily by a whole range of methods (see [21], [11]). The connection between conformal mappings and Szegö and Bergman kernels has also been well known for a long time. For practical purposes, constructing a conformal mapping of a region Ω onto a disk is equivalent to evaluating a column of the Szegö kernel corresponding to Ω (see, for example, [8], [11], [19]).

Unfortunately, constructing conformal mappings (or evaluating Szegö kernels) numerically is by no means a trivial problem, and during the last several years the interest in it has increased significantly (see [21], [11]). Several approaches have been proposed. One group assumes that the boundary Γ of Ω is smooth and constructs a first kind integral equation on Γ, whose solution is used to evaluate a conformal mapping $\Omega \to D$ via a simple procedure (see [16]). The resulting integral equation has a logarithmically singular kernel, and this fact is used in [16] to construct an efficient numerical algorithm for its solution. Another group of algorithms is based on the numerical nonlinear optimization, and constructs conformal mappings of the unit disk or the upper half-plane onto polygons or regions bounded by smooth curves. The fastest of such methods have an asymptotic CPU time estimate proportional to $n \log(n)$, where n is the number of nodes in the discretization of the boundary of the region (see [23], [7], [12]).

Both groups of algorithms present certain numerical problems, and we refer the reader to the original papers [20], [16] for a detailed discussion of these difficulties, as well as the ways of dealing with them.

The approach we take is based on the result originally published in [13], and is closely related to the subsequent work published in [22], [14]. Given a fairly general

* Received by the editors December 21, 1987; accepted for publication (in revised form) July 14, 1988. The work of these authors was supported in part by the Office of Naval Research under grant N00014-86-K-0310.

† Department of Computer Science, Yale University, New Haven, Connecticut 06520.

region Ω in C^1, a linear second kind integral equation is derived in [13], whose solution is one column of the Szegö kernel associated with Ω. The numerical solution of linear second kind integral equations leads to linear systems with asymptotically bounded condition numbers (see § 3 below), and does not involve nonlinear iterative schemes that might or might not converge. In [22], [14], the analytical apparatus of [13] is used to construct two versions of the algorithm for the numerical evaluation of conformal mappings of arbitrarily shaped simply connected regions onto a disk. The version described in [14] requires order $O(n^3)$ operations to construct the mapping, where n is the number of nodes in the discretization of the boundary of the region, and the version described in [22] requires order $O(n^2)$ operations.

In this paper, we describe a modification of the algorithm of [22], [14] with an asymptotic operation count proportional to $O(n)$. The approach is based on discretizing the integral equation of [13] via the Nyström method and solving the resulting system of linear algebraic equations by means of the conjugate residual algorithm, which is quite similar (but not identical) to the scheme used in [22]. Each iteration of the process requires that the matrix of the system be applied to a vector, which is normally an order n^2 procedure, since the matrix in question is dense. However, it turns out that the Fast Multipole Method (see [17], [10], [3], [9]) can be used to speed up the process, resulting in an order $O(n)$ algorithm for the construction of a conformal mapping of an arbitrarily shaped region in C^1 onto a disk. A computer program implementing the procedure has been written, and we present numerical examples demonstrating that the computation times do indeed scale linearly with n, and that even large-scale problems result in acceptable computation times.

Remark 1.1. While the algorithm of this paper has a CPU time estimate proportional to n, the constant of proportionality is quite large. Therefore, in many situations of interest, the schemes of [23], [7], [12] are likely to be considerably more efficient, despite the formal advantage of the $O(n)$ estimate over $O(n \log(n))$. On the other hand, the performance of the present method is not affected by crowding (see § 7 below), making it a method of choice for regions with large aspect ratios.

2. Mathematical preliminaries. In this section we introduce the mathematical apparatus to be used in the remainder of the paper. D will denote the closed unit disk centered at the origin. Suppose that Ω is a simply connected bounded region in C^1 whose boundary $\partial\Omega$ is smooth, and a is a point in the interior of Ω. Suppose also that $\gamma: [0, L] \to R^2$ is a parameterization of $\partial\Omega$ by its length. f_a will denote the conformal mapping $f_a: \Omega \to C^1$ such that $f_a(a) = 0$, $f_a'(a)$ is real and positive, and $f_a(\Omega) = D$. The existence and uniqueness of f_a are known as the Riemann mapping theorem and can be found, for example, in [4].

In accordance with standard practice, we will denote by $L^2(\partial\Omega)$ the linear space of all square-integrable functions on $\partial\Omega$, and by $\mathscr{H}^2(\partial\Omega)$ the subspace of $L^2(\partial\Omega)$ consisting of all functions on $\partial\Omega$ that are restrictions of analytic functions on Ω.

The Cauchy projector associated with the region Ω is a mapping $\mathbf{H}: L^2(\partial\Omega) \to \mathscr{H}^2(\partial\Omega)$ defined by the formula

$$\mathbf{H}(u(w)) = \int_{\partial\Omega} H(w, z) u(z)\, d\sigma_z, \tag{2.1}$$

where

$$H(w, z) = \frac{1}{2\pi i} \frac{\gamma'(z)}{(z - w)} \tag{2.2}$$

is referred to as the Cauchy kernel, $\gamma'(z)$ is the counterclockwise unit tangent to $\partial\Omega$ at z, $d\sigma$ is the arc length on $\partial\Omega$, and $\gamma'(z)d\sigma_z = dz$. The Cauchy projector is an oblique one, and the orthogonal projector $\mathbf{S}: L^2(\partial\Omega) \to \mathscr{H}^2(\partial\Omega)$ is referred to as the Szegö projector. The Szegö projector is known to be a Cauchy type integral operator, i.e., it can be represented by the formula

$$\mathbf{S}(u(w)) = \int_{\partial\Omega} S(w, z)u(z)\, d\sigma_z, \tag{2.3}$$

and $S: \Omega \times \Omega \to C^1$ is usually referred to as the Szegö kernel.

The Szegö projector is related to the Cauchy projector by the formula

$$\mathbf{S} + (\mathbf{H}^* - \mathbf{H})\mathbf{S} = \mathbf{H}^*, \tag{2.4}$$

and therefore each column of the Szegö kernel satisfies the second kind integral equation

$$S(z, a) + \int_{t\in\partial\Omega} (\bar{H}(t, z) - H(z, t))S(t, a)\, d\sigma_t = \bar{H}(a, z), \tag{2.5}$$

with $z \in \partial\Omega$. The kernel of (2.5) is known as the Kerzman–Stein kernel, and is defined by the formula

$$\begin{aligned} A(z, w) &= \bar{H}(w, z) - H(z, w) \quad \text{for } z \neq w \\ &= 0 \quad \text{for } z = w. \end{aligned} \tag{2.6}$$

A is smooth, skew-Hermitian, and is identically zero if Ω is a disk.

Evaluation of Szegö kernels for regions in the complex plane is closely related to conformal mappings of such regions into the unit disk, and the following are the classical formulae implementing this connection. For any $a \in \Omega \backslash \partial\Omega$,

$$S(a, a) = \int_{\partial\Omega} \overline{s(z, a)} S(z, a)\, d\sigma_z. \tag{2.7}$$

For any $z \in \Omega$,

$$f_a'(z) = 2\pi \frac{S^2(z, a)}{S(a, a)}. \tag{2.8}$$

For any $z \in \partial\Omega$,

$$f_a(z) = \frac{\gamma'(z)}{i} \frac{f_a'(z)}{|f_a'(z)|}. \tag{2.9}$$

Remark 2.1. Obviously, given the function $\Psi(z) = S(a, z)$, the conformal mapping $f_a: \Omega \to D$ can be easily obtained either by quadratures using (2.8) and (2.7) or directly by combining (2.8), (2.7), and (2.9). Therefore, the problem of finding f_a has been reduced to the problem of solving (2.5).

The proofs of all results in this section can be found in [13] and [14].

3. Numerical preliminaries. In this section, the numerical techniques used to solve the integral equation (2.5) and compute the mapping f_a on the boundary of Ω are described. The Nyström algorithm is used to discretize the integral equation, and the conjugate residual algorithm is used to solve the resulting system of complex linear algebraic equations. Each iteration of the conjugate residual algorithm includes a matrix-vector multiplication, which in existing algorithms has a cost proportional to n^2. This is the only computation that forces the operation count in the conjugate residual algorithm to be proportional to n^2 rather than n. The Fast Multipole Method (see [17], [10], [3]) provides a technique for multiplying Hilbert matrices by arbitrary vectors in time proportional to n, and we use this method to rapidly apply the matrix

to the vector in the conjugate residual algorithm (see § 3.2). Once the equation (2.5) is solved numerically, we obtain the mapping f_a by combining (2.7), (2.8), and (2.9).

3.1. The trapezoidal quadrature rule for periodic functions. We will define an n-point quadrature rule η on the interval $[0, L]$ as a finite sequence of pairs $\{x_i, w_i\}$, $i = 1, \cdots, n$, where $x_i \in [0, L]$ for all $i \in [1, n]$. For a function $\phi : [0, L] \to C^1$, we will view the sum

$$\eta(\phi) = \sum_{i=1}^{n} w_i \cdot \phi(x_i) \tag{3.1}$$

as an approximation to the integral

$$\int_0^L \phi(x)\, dx. \tag{3.2}$$

The family of quadrature formulae $\eta_n = \{x_{ni}, w_{ni}\}$, $i = 1, \cdots, n$, has a rate of convergence m ($m \geqq 1$) for the function $\phi : [0, L] \to R^1$ if there exist $A > 0$ and $N > 0$ such that

$$\left| \eta_n(\phi) - \int_0^L \phi(x)\, dx \right| > \frac{A}{n^m} \tag{3.3}$$

for all $n > N$. The n-point trapezoidal quadrature rule is defined by

$$x_i = (i-1)\frac{L}{n} \quad \text{for } i = 1, \cdots, n, \tag{3.4}$$

$$w_1 = w_n = \frac{L}{2n} \quad \text{and} \quad w_i = \frac{L}{N} \quad \text{for } i \in [2, n-1].$$

Using the Euler-MacLaurin formula, it is easy to show that if the function $\phi : [0, L] \to R^1$ is periodic and has k continuous periodic derivatives ($k \geqq 1$), then the trapezoidal rule for this function has order of convergence $k+1$.

3.2. The Nyström algorithm. The Nyström algorithm associated with an n-point quadrature formula $\eta = \{z_i, w_i\}$ replaces the integral equation

$$\phi(z) + \int_0^L K(z, t) \cdot \phi(t)\, dt = g(z) \tag{3.5}$$

with the system of linear algebraic equations

$$\phi(z_i) + \sum_{j=1}^{n} w_j \cdot K(z_i, z_j) \cdot \phi(z_j) = g(z_i), \tag{3.6}$$

with $i = 1, \cdots, n$. We denote the matrix of the discretized system (3.6) by B_n and view the solution $\phi_1, \cdots, \phi_n$ of (3.6) as an approximation to the solution ϕ of (3.5) at the nodes $z_1, \cdots, z_n$. If (3.5) has a unique solution, then for a wide class of quadrature formulae η_n and sufficiently large n, system (3.6) also has a unique solution. Furthermore, under fairly broad assumptions, the convergence rate of the Nyström algorithm is the same as the convergence rate of the quadrature formula it is based on (see [2]).

Applying the Nyström algorithm based on the n-point trapezoidal quadrature rule to the equation (2.5), we obtain the system of equations

$$\phi_i + \frac{L}{n} \sum_{j=1}^{n} A(z_i, z_j) \cdot \phi_j = \bar{H}(a, z_i), \tag{3.7}$$

with $i=1,\cdots,n$, where ϕ_j is viewed as an approximation to $S(z_j,a)$. The system of equations (3.7) can be written as the linear system

$$B_n\psi=\bar{H}_{n,a}, \tag{3.8}$$

where ψ is viewed as the approximation to one column of the Szegö kernel $(S(z_1,a),\cdots,S(z_n,a))$ and $\bar{H}_{n,a}=(\bar{H}(a,z_1),\cdots,\bar{H}(a,z_n))$.

By the following theorem (see, for example, [2]), the condition number of the matrix B_n is asymptotically bounded.

THEOREM 3.1. *Suppose that $K:[0,L]\times[0,L]\to R^1$ is a c^2-function and that the equation* (3.5) *has a unique solution. Suppose further than the system of linear equations* (3.6) *with corresponding coefficient matrix B_n has been obtained by applying the Nyström algorithm based on the trapezoidal quadrature rule to* (3.5). *Then*

$$\lim_{n\to\infty} k(B_n)=a, \tag{3.9}$$

where $0<a<\infty$ is some real number and $k(B_n)$ denotes the condition number of the matrix B_n.

3.3. Representation of the boundary of Ω. The discretized equation (3.7) assumes that the nodes $z_1,\cdots,z_n$ are equispaced along the boundary $\partial\Omega$. In our implementation, we assume that the boundary of Ω is specified numerically by nodes $w_1,\cdots,w_m$ for some $m>0$. To generate the equispaced nodes, we resample the curve by passing periodic splines under tension through the nodes $w_1,\cdots,w_m$, and generating nodes $z_1,\cdots,z_n$, which are equispaced in arclength along the splines.

3.4. Rapid application of the matrix B_n to arbitrary vectors. In this section we describe the application of Fast Multipole Method (see [10], [3]) to the multiplication of the matrix B_n of (3.8) by the vector ψ in time proportional to n.

For points $z_1,\cdots,z_n\in C^1$, a Hilbert matrix H is defined by the formulae

$$\begin{aligned} H(i,j)&=\frac{1}{z_i-z_j} \quad \text{for } i\neq j,\\ &=0 \quad \text{for } i=j. \end{aligned} \tag{3.10}$$

A description of the Fast Multipole Method (FMM) together with the proof of the following theorem can be found in [3].

THEOREM 3.2. *Let A denote a Hilbert matrix of order n, and let x be an arbitrary vector. The Fast Multipole Method applies the matrix A to vector x in $O(n\cdot\log(1/\varepsilon))$ operations, where ε is the precision of calculations.*

Suppose now that we would like to efficiently apply the matrix B_n of system (3.8) to a vector. Due to (2.2) and (2.6), (3.7) can be rewritten in the following form:

$$\psi_i+\frac{L}{n}\frac{\bar{\gamma}'(z_i)}{2\pi i}\sum_{j\neq i}\frac{1}{(\bar{z}_i-\bar{z}_j)}\cdot\psi_j+\frac{L}{n}\frac{1}{2\pi i}\sum_{j\neq i}\frac{1}{(z_i-z_j)}(\psi_j\cdot\gamma'(z_j))=\bar{H}(a,z_i). \tag{3.11}$$

We denote the Hilbert matrix associated with the points $\bar{z}_1,\cdots,\bar{z}_n$ by U_n, and the Hilbert matrix associated with the points $z_1,\cdots,z_n$ by V_n. We denote the vector $(\psi_1\cdot\gamma'(z_1),\cdots,\psi_n\cdot\gamma'(z_n))$ by χ. It is easy to see from (3.11) that

$$B_n\psi=\psi+c_1U_n\psi+c_2V_n\chi, \tag{3.12}$$

with

$$c_1 = \frac{L}{n}\frac{\bar{\gamma}'(z_i)}{2\pi i} \quad \text{and} \quad c_2 = \frac{L}{n}\frac{1}{2\pi i}. \tag{3.13}$$

From (3.12) it is clear that two applications of a Hilbert matrix to a vector are required to apply the matrix B_n to ψ. Consequently, the matrix B_n can be applied to the vector ψ in order $O(n)$ operations.

3.5. Conjugate residual algorithm. The conjugate residual method (CR) is an iterative method for solving a system of linear equations $Ax = f$ with $x, f \in C^n$ and A a normal $n \times n$-matrix (see [5]). We denote the ith iterate by x_i and the residual at the ith step by $r_i = y - Ax_i$. At the ith step the CR algorithm minimizes the residual $\|r_i\|_2$ over the i-dimensional subspace of C^n spanned by the vectors $r_0, Ar_0, \cdots, A^{i-1}r_0$.

From (3.7), it is clear that the matrix B_n of system (3.8) is normal since it is the sum of the identity matrix and a skew-Hermitian matrix, and hence the conjugate residual algorithm can be used to solve (3.8) (see [5]). Moreover, if $E_2(x_i)$ denotes the L^2-error at the ith iterate, it is known [6] that

$$E_2(x_i) \leqq 2\left[\frac{1 - 1/k(B_n)}{1 + 1/k(B_n)}\right]^i E_2(x_0). \tag{3.14}$$

Combining (3.14) with Theorem 3.1, it is easy to see that the convergence rate of the conjugate residual algorithm applied to (3.8) is independent of n.

4. Description of the algorithm. Following is a description of the algorithm for the numerical evaluation of the mapping $f_a : \partial\Omega \to \partial D$.

- Resample $\partial\Omega$ into equispaced nodes using periodic splines under tension as the basis for interpolation.
- Apply the conjugate residual algorithm to solve the linear system $B_n\psi = \bar{H}_{n,a}$ that arises from the discretized system (3.7).
- In the conjugate residual routine, apply the matrix B_n to the vector using formulae (3.12) and (3.13) where the Fast Multipole Method is used to compute the Hilbert matrix-vector products $U_n\psi$ and $V_n\chi$.
- Use (2.8) to compute f_a' and a combination of formula (2.9) and recursive application of the modified trapezoidal rule (see § 5) to compute f_a.

5. Details of the implementation. For a wide class of regions Ω, the conformal mapping $f_a : \partial\Omega \to \partial D$ compresses much of the boundary of Ω into very small segments of the unit circle. This phenomenon is known as crowding and is an inherent property of the conformal mapping. In areas of crowding, the derivatives f_a' can be very small, which has serious numerical implications. Where crowding is significant, the derivative $1/f_a'$ of the mapping $f_a^{-1} : \partial D \to \partial\Omega$ is sufficiently large that f_a^{-1} cannot be numerically computed from f_a and f_a'.

However, the numerical computation of f_a from f_a' is possible. Once an approximation to the Szegö kernel has been computed, the derivative f_a' and $f_a : \partial\Omega \to \partial D$ can generally be computed using equations (2.8) and (2.9). However, suppose that for some integer $i < n$, $z_i \in \partial\Omega$ is a point in a region of crowding. Then the analytic values of f_a' are very small relative to values of f_a' elsewhere on the boundary (see § 6). Therefore, the numerical approximations of $S(z_i, a)$ and $f_a'(z_i)$ are likely to have a high relative error, rendering formula (2.9) useless (see § 7).

Clearly, an alternate method of computing f_a is integration along the boundary $\partial\Omega$, and we use the modified trapezoidal rule with fourth-order corrections at the endpoints (see [1]) given by the formula

$$\int_{z_1}^{z_n} f(z)\,dz = \sum_{j=1}^{n} f(z_i) - \frac{1}{2}f(z_1) - \frac{1}{24}(f(z_3) - 4f(z_2) + 3f(z_1)) \tag{5.1}$$
$$- \frac{1}{2}f(z_n) - \frac{1}{24}(f(z_{n-2}) - 4f(z_{n-1}) + 3f(z_n)).$$

By separately storing the right endpoint correction and the remaining sum in the formula (5.1), the calculation of the integral from z_1 to z_{k+1} can make use of the calculation of the integral from z_1 to z_k, and therefore the modified trapezoidal rule can be implemented recursively, resulting in an order $O(n)$ procedure.

Other than in areas of crowding, (2.9) can be expected to be more accurate than the trapezoidal rule. We combine the two methods, using (2.9) wherever the derivative is above a threshold value, and recursive integration via the modified trapezoidal rule otherwise.

6. Numerical results. A FORTRAN program has been written implementing the algorithm of this paper. It has been tested on a variety of regions, and in this section, we present the results of six such numerical experiments. Tables 1-6 contain the quantitative information pertaining to the Examples 1-6, respectively. The first column of each table contains the number n of nodes in the discretization of the boundary of the region. The second column contains the CPU time in seconds the algorithm took on a VAX-8600 computer. The third column contains the number of iterations the conjugate residual process took to achieve five-digit precision. The fourth and fifth columns contain the relative accuracies in L^2 and L^∞ norms, respectively. Finally, the sixth column of each table contains the ratios

$$\frac{\min_{z\in\Gamma}|f'(z)|}{\max_{z\in\Gamma}|f'(z)|} \tag{6.1}$$

as determined by the calculation with the corresponding n.

In addition, we present Tables 7 and 8, describing a double precision calculation for Example 4, and illustrating the convergence of the conjugate residual algorithm in this case.

Remark 6.1. Since no exact values are available for many of the conformal mappings constructed in the examples 1-6, evaluating the errors and convergence rates presents a certain difficulty. We have used a standard remedy, and estimated the errors (both L^2 and L^∞) by the difference between two successive discretizations of the same problem.

For each example, we present a figure depicting the equispaced nodes on the boundary of the original region (left column of Figs. 1-6) and their images on the circle of radius one (right column of Figs. 1-6). Following is a more detailed description of each of the examples.

Example 1. An ellipse with the aspect ratio 2:1. The algorithm has been applied to this region with $n = 16, 32, \cdots, 512$. The results for this set of experiments are summarized in Table 1, and the case of $n = 512$ is illustrated in Fig. 1.

Example 2. An ellipse with the aspect ratio 3:1. The algorithm has been applied to this region with $n = 16, 32, \cdots, 512$. The results for this set of experiments are summarized in Table 2, and the case of $n = 512$ is illustrated in Fig. 2.

TABLE 1
A 2:1 ellipse.

nodes	CPU time	iterations	L^2-error	L^∞-error	crowding
16	4.4	4	.59E-3	.17E-2	.089
32	4.6	4	.17E-4	.49E-4	.089
64	5.6	4	.19E-5	.69E-5	.089
128	8.5	4	.64E-6	.11E-5	.089
256	14.6	4	.19E-5	.39E-5	.089
512	22.2	4			.089

TABLE 2
A 3:1 ellipse.

nodes	CPU time	iterations	L^2-error	L^∞-error	crowding
16	11.5	5	.12E-1	.22E-1	.10E-1
32	11.8	5	.60E-3	.10E-2	.97E-2
64	13.2	5	.25E-4	.48E-4	.97E-2
128	17.0	5	.15E-5	.26E-5	.97E-2
256	25.0	5	.59E-6	.12E-5	.97E-2
512	35.6	5			.97E-2

TABLE 3
A 10:1 ellipse.

nodes	CPU time	iterations	L^2-error	L^∞-error	crowding
32	13.4	8	.32E+0	.37E+0	.60E-7
64	15.3	9	.33E-2	.39E-2	.11E-8
128	21.3	9	.20E-3	.20E-3	.10E-10
256	33.5	9	.12E-4	.12E-4	.95E-9
512	57.8	9	.82E-6	.85E-6	.94E-9
1024	121.7	9	.64E-6	.66E-6	.94E-9

TABLE 4
A snake-shaped region.

nodes	CPU time	iterations	L^2-error	L^∞-error	crowding
128	29.6	17	.32E+1	.11E+1	.20E-8
256	51.1	17	.70E+0	.71E+0	.17E-10
512	95.7	17	.10E-1	.10E-1	.30E-14
1024	203.1	17	.51E-3	.51E-3	.15E-15
2048	369.7	17	.54E-4	.54E-4	.98E-16
4096	652.1	17			.36E-15

TABLE 5
A spiral-shaped region.

nodes	CPU time	iterations	L^2-error	L^∞-error	crowding
200	42.9	18	.11E+1	.11E+1	.26E-13
400	81.8	17	.11E+0	.11E+0	.77E-14
800	127.3	16	.19E-2	.19E-2	.21E-14
1600	252.8	16	.12E-3	.12E-3	.13E-14
3200	521.6	16	.68E-4	.68E-4	.12E-14
6400	978.5	16			.11E-15

TABLE 6
A rounded square.

nodes	CPU time	iterations	L^2-error	L^∞-error	crowding
16	6.9	6	.30E-1	.52E-1	.86E-1
32	7.9	6	.83E-2	.14E-1	.47E-1
64	9.5	6	.17E-2	.43E-2	.28E-1
128	13.7	6	.35E-3	.47E-3	.20E-1
256	18.8	6	.12E-4	.45E-4	.21E-1
512	33.0	6			.20E-1

TABLE 7
A snake (double precision).

nodes	iterations	L^2-error	L^∞-error	crowding
128	34	.31E+0	.34E+0	.28E-8
256	32	.13E+0	.13E+0	.26E-10
512	34	.20E-1	.20E-1	.27E-14
1024	34	.42E-3	.42E-3	.16E-21
2048	34	.24E-4	.24E-4	.36E-24
4096	34			.34E-24

TABLE 8
Convergence of the Conjugate Residual Algorithm—A snake (double precision).

iterate	L^2-error	iterate	L^2-error	iterate	L^2-error
1	.79E+00	12	.23E-03	23	.21E-07
2	.54E+00	13	.16E-03	24	.59E-08
3	.30E+00	14	.13E-03	25	.18E-08
4	.17E+00	15	.55E-04	26	.65E-09
5	.82E-01	16	.15E-04	27	.37E-09
6	.45E-01	17	.42E-05	28	.14E-09
7	.25E-01	18	.14E-05	29	.38E-10
8	.11E-01	19	.52E-06	30	.10E-10
9	.45E-02	20	.28E-06	31	.24E-11
10	.13E-02	21	.18E-06	32	.83E-12
11	.48E-03	22	.77E-07		

Example 3. An ellipse with the aspect ratio 10:1. The algorithm has been applied to this region with $n = 32, 64, \cdots, 1{,}024$. The results for this set of experiments are summarized in Table 3, and the case of $n = 1{,}024$ is illustrated in Fig. 3.

Example 4. A snake-shaped region. The number of nodes required to achieve reasonable accuracy is larger than that of the ellipse examples above. Results for the snake-shaped region are presented for $n = 128, 256, \cdots, 4{,}096$. The data for this region are presented in Table 4 and the cases of $n = 256, 4{,}096$ are illustrated in Fig. 4.

Example 5. A spiral-shaped region. As in Example 4, many nodes along the boundary are required to achieve reasonable accuracy. In this example, results are presented for $n = 200, 400, 800, \cdots, 6{,}400$. The data for the experiment are given in Table 5, and the case of $n = 4{,}096$ is illustrated in Fig. 5.

Example 6. A rounded square. The algorithm has been applied to this region with $n = 16, 32, \cdots, 512$. The data for this example are presented in Table 6, and the plot for the case $n = 256$ is presented in Fig. 6.

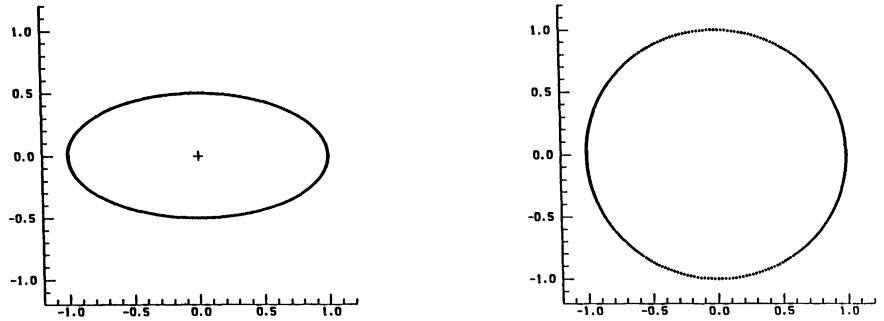


FIG. 1. *A* 2:1 *ellipse*, $n = 512$.

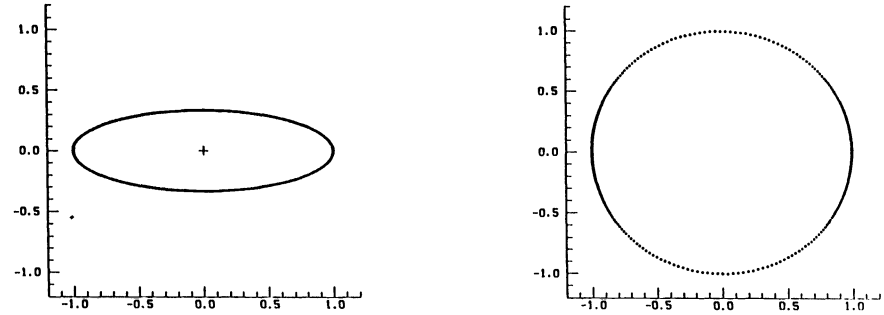


FIG. 2. *A* 3:1 *ellipse*, $n = 512$.

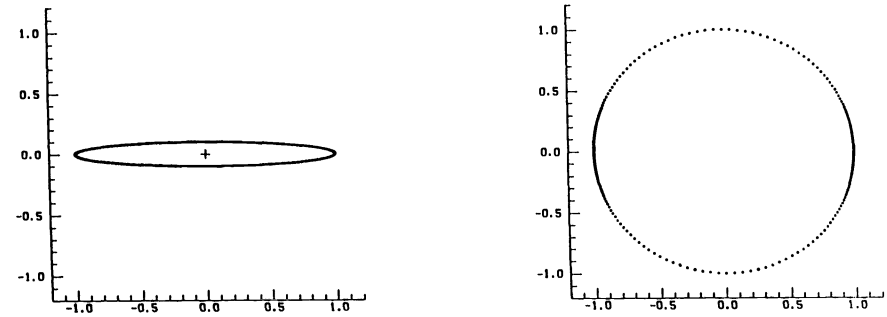


FIG. 3. *A* 10:1 *ellipse*, $n = 1{,}024$.

The following observations can be made from the Tables 1-8 and the Figs. 1-6.

(1) The CPU time requirements of the algorithm grow linearly with n, though the growth is somewhat erratic.

(2) The number of iterations of the conjugate residual process is virtually independent of the number of nodes in the discretization of the boundary of region.

(3) Even for regions of fairly complicated shapes, the number of iterations of the conjugate residual process does not become excessive.

(4) The algorithm does not cause substantial roundoff errors, and single precision arithmetic can be used. This observation is typical for algorithms based on second kind integral equations (see, for example, [2], [17]), and is in agreement with observations made in [22], [14].

(5) Even for fairly large-scale calculations ($n > 4{,}000$), the actual CPU times required by the algorithm are quite acceptable (about 10 minutes on a VAX-8600 for $n = 4{,}098$, and less than one hour for $n = 20{,}000$). Since the boundary of a very wide range of regions can be adequately discretized by 20,000 nodes, the algorithm can be

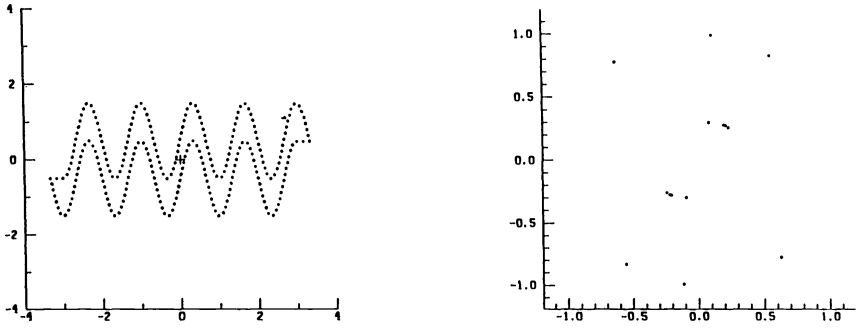


FIG. 4(a). *A snake-shaped region, $n = 256$.*

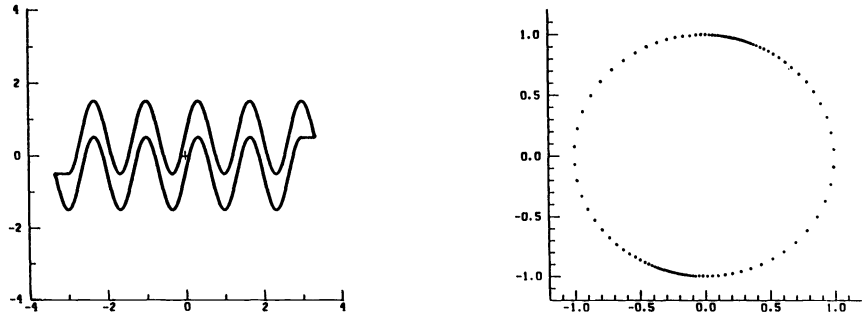


FIG. 4(b). *A snake-shaped region, $n = 4{,}096$.*

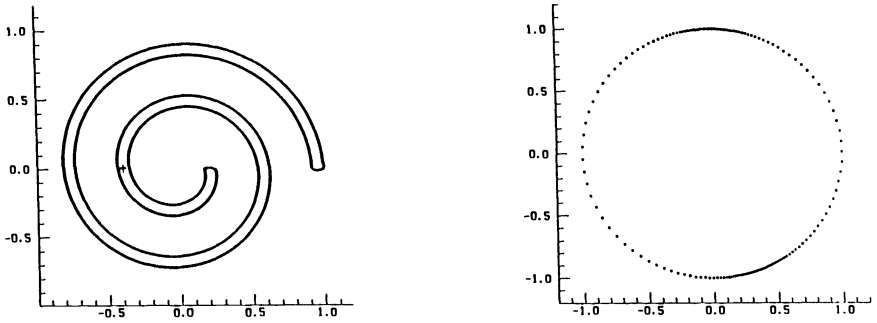


FIG. 5. *A spiral-shaped region, $n = 6{,}400$.*

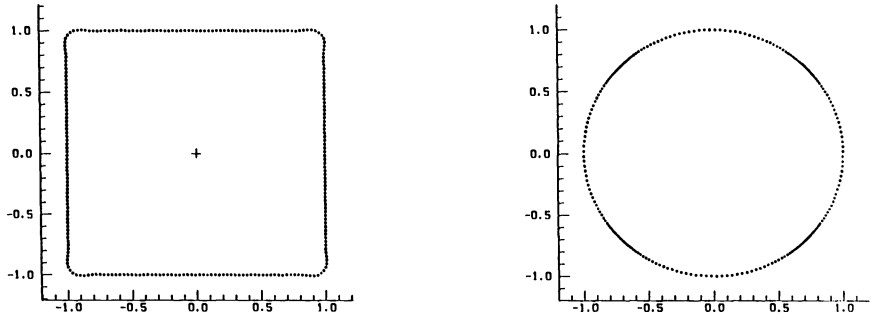


FIG. 6. *A rounded square, $n = 256$.*

viewed as a nearly universal tool for mapping smooth regions in the complex plane onto the unit disk.

7. Crowding and related issues. Inspection of column 6 of the Tables 1–6 yields a disturbing observation. Namely, even for fairly uncomplicated regions (such as an ellipse with an aspect ratio 10:1) the ratio (6.1) tends to be extremely small. The effect is known in the literature under the name of crowding (see [24]), and has received much less attention than the authors feel it deserves. Consider, for example, Figs. 4 and 5. It is obvious that nearly all of the boundary of the "spiral" is compressed onto an extremely small part of the circle. For example, there are 6,400 nodes in the discretization of the boundary of the "spiral" in the left column of Fig. 5, yet only 100 images of these nodes can be discerned in the right column. Similar observations can be made about the "snake" in Fig. 4 and about the ellipse with an aspect ratio 10:1 depicted in Fig. 3. Even for the ellipse with an aspect ratio 3:1 depicted in Fig. 2, the ratio (6.1) is of order 10^{-2}.

One interpretation of this observation is that although the mapping $f:\Omega \to D$ can be constructed numerically for a wide class of regions Ω, the mapping $f^{-1}:D \to \Omega$ is a numerically unmanageable object even for fairly mild Ω. This fact has serious repercussions for attempts to use the conformal mappings as a numerical (as opposed to analytical) tool for the solution of elliptic partial differential equations (PDEs) in general regions in two dimensions. Indeed, a standard prescription for using conformal mappings to solve an elliptic PDE (such as Laplace or Poisson equation) in a two-dimensional region is as follows.

(a) Find a conformal mapping $f^{-1}:D \to \Omega$ and use it to construct an induced PDE on the disk D.

(b) Find the solution ϕ_D of the induced PDE on D via an appropriate numerical technique.

(c) Construct the mapping $f:\Omega \to D$ and evaluate the solution ϕ of the original PDE on Ω via

$$\phi(x) = \phi_D(f(x)). \tag{7.1}$$

Obviously, if the mapping f^{-1} cannot be constructed, step (a) in the above procedure becomes impossible.

Remark 7.1. In fact, it is known that for a region Ω, the ratio (6.1) decays exponentially as a function of the aspect ratio of Ω (see, for example, [24]). It is also known that the mapping f^{-1} viewed as a function of the region Ω is not even continuous (see [18]). Therefore, the poor numerical behavior of f^{-1} is not surprising.

Remark 7.2. Obviously, if the region Ω has an aspect ratio close to one (i.e., it does not deviate from the disk too much), conformal mappings can be an effective numerical device for dealing with elliptic PDEs on Ω. For regions with high aspect ratios, it is natural to attempt to use model regions other than the disk (such as rectangles for which Fast Poisson Solvers and other efficient tools are available). While it is easy to see that no model region can solve the problem of crowding in general, it is clear that certain classes of regions can be handled in this manner.

8. Conclusions. An algorithm is presented for the construction of conformal mappings from complicated regions in the complex plane having smooth boundaries onto the unit disk. The algorithm is based on a combination of the approach of [13], [14], [22] with the FMM described in [17], [10], [3] and has an asymptotic CPU time estimate $O(n)$, where n is the number of nodes in the discretization of the boundary of the region. For regions with relatively small aspect ratios, the schemes of [23], [7], [12] are likely to be considerably more efficient than the present algorithm. When

severe crowding is encountered, the method of this paper appears to be most efficient of presently available numerical tools for construction of conformal mappings.

On the other hand, our experiments strongly indicate that the crowding makes the use of conformal mappings as a numerical tool difficult in some cases and impossible in others. While the existence of this problem has been known for a long time, it has not been treated adequately so far.

Acknowledgments. The authors would like to thank Professor R. R. Coifman and Dr. L. F. Greengard for their interest and help.

REFERENCES

[1] M. ABRAMOWITZ AND I. STEGUN, EDS., *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, U.S. Government Printing Office, National Bureau of Standards Applied Mathematics Series 55, Washington, DC, 1972.

[2] K. E. ATKINSON, *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1976.

[3] J. CARRIER, L. GREENGARD, AND V. ROKHLIN, *A fast adaptive multipole algorithm for particle simulations*, Tech. Report YALEU/DCS/RR-496, Department of Computer Science, Yale University, Cambridge, MA, 1986; SIAM J. Sci. Statist. Comput., 9 (1988), pp. 669-686.

[4] J. B. CONWAY, *Functions of One Complex Variable*, Springer-Verlag, New York, 1973.

[5] 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.

[6] 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. (Also available as Research Report 229.)

[7] B. FORNBERG, *A numerical method for conformal mapping*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 386-400.

[8] D. GAIER, *Konstruktive Methoden der Konformen Abbildung*, Springer-Verlag, Heidelberg, 1964.

[9] L. F. GREENGARD, *The rapid evaluation of potential fields in particle systems*, Ph.D. thesis, Department of Computer Science, Yale University, Cambridge, MA, 1987. (Available in book form from MIT Press, 1988).

[10] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325-348.

[11] P. HENRICI, *Applied and Computational Complex Analysis Vol.* 3, John Wiley, New York, 1986.

[12] O. HÜBNER, *On the Fast Solution of a Linear System Arising in a Numerical Conformal Mapping*, J. Comput. Appl. Math., 16 (1986), pp. 381-386.

[13] N. KERZMAN AND E. M. STEIN, *The Cauchy kernel, the Szegö kernel, and the Riemann mapping function*, Math. Ann., 236 (1978), pp. 85-93.

[14] N. KERZMAN AND M. R. TRUMMER, *Numerical conformal mapping via the Szegö kernel*, J. Comput. Appl. Math., 14 (1986), pp. 111-123.

[15] Z. NEHARI, *Conformal Mapping*, Dover Publications, New York, 1952.

[16] L. REICHEL, *A fast method for solving certain integral equations of the first kind with application to conformal mapping*, J. Comput. Appl. Math., 14 (1986), pp. 125-142.

[17] V. ROKHLIN, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187-207.

[18] S. W. SEMMES, *The Cauchy Integral, Chord-Arc Curves, and Quasiconformal Mappings*, American Mathematical Society, The Bieberbach Conjecture, Proc. Symposium on the Occasion of the Proof, Mathematical Surveys and Monographs, 21, 1986, pp. 167-183.

[19] G. SZEGÖ, *Orthogonal polynomials*, Amer. Math. Soc. Colloq. Publ., 23 (1939).

[20] L. N. TREFETHEN, *Numerical computation of the Schwarz-Christoffel transformation*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 82-102.

[21] ———, ED., *Numerical Conformal Mapping*, Elsevier Science Publishing Co., Inc., New York, 1986. (Reprinted from J. Comput. Appl. Math.)

[22] M. R. TRUMMER, *An efficient implementation of a conformal mapping method based on the Szegö kernel*, SIAM J. Numer. Anal., 23 (1986), pp. 853-872.

[23] R. WEGMANN, *An iterative method for conformal mapping*, Numer. Math., 30 (1978), pp. 453-466.

[24] C. ZEMACH, *A conformal map formula for difficult cases*, J. Comput. Appl. Math., 14 (1984), pp. 207-215.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 488-514, May 1989
© 1989 Society for Industrial and Applied Mathematics
006

LINEAR AND NONLINEAR ACOUSTIC WAVE PROPAGATION IN THE ATMOSPHERE*

S. I. HARIHARAN† AND YU PING†

Abstract. This paper describes the investigation of the acoustic wave-propagation theory and numerical implementation for the situation of an isothermal atmosphere. A one-dimensional model to validate an asymptotic theory and an axisymmetric three-dimensional situation to relate to a realistic situation are considered. In addition, nonlinear wave propagation and the numerical treatment are included. It is known that the gravitational effects play a crucial role in the low-frequency acoustic wave propagation. They propagate large distances and, as such, the numerical treatment of those problems becomes difficult in terms of posing boundary conditions that are valid for all frequencies. Our treatment is discussed in detail. Open questions are posed.

Key words. acoustics, nonlinear waves, hyperbolic systems, absorbing boundary conditions, finite differences

AMS(MOS) subject classifications. primary 35L70, 76Q05

1. Introduction. In this paper we describe procedures for computing linear and nonlinear aspects of acoustic wave propagation in the atmosphere. Acoustic waves originate from a source that has compact support in space, as well as in time. Such a phenomenon occurs in a situation such as a shuttle take-off or any general explosion. The current work concentrates on a situation in which the atmosphere is isothermal. Although this assumption somewhat oversimplifies the physics, the mathematical and computational aspects of the problem remain difficult. The main difficulty arises from the effect of gravity, which is included in this model and plays a crucial role in the nature of the propagation. In the acoustics literature, similar models were used to investigate explosive waves. We cite references [1] and [2] where linear models were considered for such purposes. Our model is an improvement over the model proposed in [1]. Our work appearing in [3] and [4] discusses analysis of the problem in two dimensions. Our work appearing in [5] discusses the first computational effort in the axisymmetric three-dimensional situation but only the linear case is discussed. The work reported in this paper includes numerical aspects of nonlinearities in the axisymmetric three-dimensional situation.

The main goal of this paper is to capture the effects of these atmospheric acoustic waves numerically. While the goal seems to be simple, there are several obstacles in the process of solving the problem under consideration. The nonlinear problem for the acoustic variables cannot be directly solved with a reasonable computational effort unless the problem is somewhat simplified by certain techniques.

In the sequence of work reported in [3] and [4], the nonlinear problem was reduced to a sequence of linear problems using appropriate asymptotic expansions. Then we could show the linearized problems were well posed. There was a question that we could not resolve about the asymptotic expansion. The expansion may not be uniformly valid. Here we provide partial justification by considering an appropriate one-dimensional model. We compute the linearized solutions and show the sum of the first two terms of the expansion indeed yields the solution of the nonlinear problem.

* Received by the editors February 29, 1988; accepted for publication August 23, 1988. This work was supported in part by a grant from National Aeronautics and Space Administration, grant NAG-1-624, and in part by National Science Foundation grant DMS-8604047.

† Department of Mathematical Sciences, University of Akron, Akron, Ohio 44325.

Unfortunately, this can be done only in a one-dimensional situation, as the cost of computing solutions of the nonlinear problem directly is extremely high. This is because the actual nonlinear acoustic field contains the mean flow fluid quantities which are orders of magnitude higher than the acoustic quantities. In particular, the truncation error resulting from the numerical scheme will dominate the acoustic effect unless a very fine grid and extremely small time steps are used. In the calculations reported on the one-dimensional situation, 400 mesh points and about 10,000 time steps are needed to directly compute the nonlinear problem. To carry out this procedure to higher dimensions, it is clear that computational requirements are large. The justification for the validity of the asymptotic expansion that we provide here is computational, and a mathematical proof is a hard task, as pointed out in [3]; therefore, we leave it as an open question.

The plan of the paper is as follows. First, we shall present the governing equations and present the problems that we intend to solve. In these problems, boundary conditions play a crucial role. The boundary conditions that we use here are different from and better than the ones we used in [5]. The derivations are not straightforward. They are presented in the appropriate sections. We devote § 2 of the paper to the discussion of the one-dimensional model where the problem can be reduced to a scalar problem. In § 3, we discuss the sequence of linearized problems, and in § 4, we discuss the nonlinear system in the one-dimensional situation. In § 5, we discuss the axisymmetric three-dimensional problem. We also show the numerical results in §§ 3, 4, and 5 arising from our theory and computations. In the Appendix, we show the numerical schemes that we used for our computations, along with the numerical treatment of our boundary conditions.

1.1. Governing equations. We shall begin with the statement of the fluid flow problem that governs the acoustic phenomena. It is governed by the Euler equations (the equations of continuity, balance of momentum, and energy) and they are

$$\frac{\partial \rho}{\partial t} + \operatorname{div}(\rho \mathbf{q}) = 0, \tag{1.1}$$

$$\frac{\partial \mathbf{q}}{\partial t} + (\mathbf{q} \cdot \nabla)\mathbf{q} = -\frac{1}{\gamma \rho} \nabla \bar{p} - \frac{1}{\gamma} \hat{k}, \tag{1.2}$$

$$\left(\frac{\partial}{\partial t} + \mathbf{q} \cdot \nabla\right) \frac{\bar{p}}{\rho^{\gamma}} = \frac{1}{\rho^{\gamma - 1}} \varepsilon f(x, y, z, t). \tag{1.3}$$

Note that, in (1.2), the forcing term $-1/\gamma \hat{k}$ ($\hat{k}$ is the unit vector in the z direction) arises due to the forcing term per unit mass $-g\hat{k}$ in the original variables which is due to gravity. In (1.3), $f(x, y, z, t)$ dictates the space-time dependency of the source, and ε measures the energy release per unit volume. For the case of an instantaneous energy release, ε is given by

$$\varepsilon = \frac{(\gamma - 1) Q_0}{h^3 P^*} \tag{1.4}$$

where Q_0 is the total energy released at time $t = 0$, P^* is the ambient pressure, and h is the scale height. The initial conditions are

$$\bar{p} = \rho = e^{-z}, \qquad \mathbf{q} = 0 \quad \text{at } t = 0. \tag{1.5}$$

These conditions are obtained from the fact that at $t = 0$ there are no acoustic disturbances and the initial conditions are simply the nondimensional values of pressure,

density, and velocity of the undisturbed atmosphere. The boundary condition at $z=0$ is

$$\mathbf{q}_z = 0, \tag{1.6}$$

which states that the vertical component of the flow is zero at $z=0$.

The acoustic expansion is based on the fact that ε is typically much less than one, and that the flow can be represented as small changes superimposed on the flow of the ambient state. We note that the ambient velocity is zero, but pressure and density have the form e^{-z}. Thus, the expansions are

$$\mathbf{q} = \varepsilon \mathbf{u} + \varepsilon^2 \mathbf{u}_1 + \cdots, \tag{1.7}$$

$$\bar{p} = e^{-z}[1 + \varepsilon p + \varepsilon^2 p_1 + \cdots], \tag{1.8}$$

$$\rho = e^{-z}[1 + \varepsilon\sigma + \varepsilon^2 \sigma_1 + \cdots] \tag{1.9}$$

where $\mathbf{u} = (u, w)$ and $\mathbf{u}_1 = (u_1, w_1)$. Quantities u, u_1 and w, w_1 are the r and z components, respectively, of the acoustic velocities, in cylindrical coordinates. We substitute expansions (1.7)-(1.9) into equations (1.1)-(1.3), initial conditions (1.5), and boundary conditions (1.6) and retain terms of order ε to obtain the field equations. Therefore, the field equations are

$$\sigma_t + u_r + \frac{u}{r} + w_z - w = 0, \tag{1.10}$$

$$u_t + \frac{1}{\gamma} p_r = 0, \tag{1.11}$$

$$w_t + \frac{1}{\gamma} p_z - \frac{(p-\sigma)}{\gamma} = 0, \tag{1.12}$$

$$p_t + \gamma u_r + \gamma w_z + \gamma \frac{u}{r} - w = f(r, z, t), \tag{1.13}$$

where $\gamma = 1.4$ and $f(r, z, t)$ describes the nature of the acoustic source and $r = \sqrt{x^2 + y^2}$.

Boundary conditions are

$$w(r, 0, t) = 0, \qquad t > 0 \tag{1.14}$$

$$u(r, z, t)/r \quad \text{is bounded as } r \to 0. \tag{1.15}$$

Initial conditions are

$$p = \sigma = u = w = 0 \quad \text{for } t = 0. \tag{1.16}$$

In addition, we also need a radiation condition for the problem that describes the behavior of the wave at infinity. In this problem, a radiation condition must be included for the uniqueness of the solution. The condition is that the waves decay at infinity and, when imposed at finite distances, take the form

$$\left(u - \frac{1}{\gamma} p\right)_t = w_z - \frac{1}{\gamma} w + \frac{u}{r} \quad \text{at } r = R, \tag{1.17a}$$

$$\left(w - \frac{1}{\gamma} p\right)_t = \frac{1}{\gamma}(p - \sigma) - \left(\frac{1}{\gamma} w - \frac{u}{r} - u_r\right) \quad \text{at } z = L. \tag{1.17b}$$

The derivation is based on [9] and is given in § 5.

We shall denote the problem formed by equations (1.10)-(1.17) by (P). Our procedure advocated in [3], as well as in [4], is to solve for the next correction and add the correction to the solution of (P) to obtain the solution of the nonlinear problem. In this process, the equations governed by the next order terms contain exactly the same differential operator, except the right-hand side contains the solution of problem (P). We call the resulting problem of the next order to be (P′) and is as follows:

$$\sigma_{1t} + u_{1r} + \frac{u_1}{r} + w_{1z} - w_1 = f_1, \tag{1.18}$$

$$u_{1t} + \frac{1}{\gamma} p_{1r} = f_2, \tag{1.19}$$

$$w_{1t} + \frac{1}{\gamma} p_{1z} - \frac{(p_1 - \sigma_1)}{\gamma} = f_3, \tag{1.20}$$

$$p_{1t} + \gamma u_{1r} + \gamma w_{1z} + \gamma \frac{u_1}{r} - w_1 = f_4, \tag{1.21}$$

where f_1, f_2, f_3, and f_4 contain the solution of the problem (P) and are given by

$$f_1 = -\left[(\sigma u)_r + (\sigma w)_z - \sigma\left(w - \frac{u}{r}\right)\right],$$

$$f_2 = \frac{1}{\gamma}\sigma p_r - (u u_r + w u_z),$$

$$f_3 = -\left[u w_r + w w_z + \frac{1}{\gamma}\sigma(p - \sigma)\right] + \frac{1}{\gamma}\sigma p_z,$$

$$f_4 = \sigma f + p\left(w - \gamma\frac{u}{r}\right) - \gamma p(u_r + w_z) - (u p_r + w p_z).$$

The boundary and initial conditions have the same form as before and are given below.

Boundary conditions are

$$w_1(r, 0, t) = 0, \qquad t > 0, \tag{1.22}$$

$$u_1(r, z, t)/r \quad \text{is bounded as } r \to 0. \tag{1.23}$$

Initial conditions are

$$p_1 = \sigma_1 = u_1 = w_1 = 0 \quad \text{for } t = 0. \tag{1.24}$$

The radiation condition takes the form

$$\left(u_1 - \frac{1}{\gamma} p_1\right)_t = -\left(-w_{1z} + \frac{1}{\gamma} w_1 - \frac{u_1}{r}\right) + \frac{1}{\gamma}\sigma p_r - u u_r$$
$$- w u_z + \frac{1}{\gamma}\left[p\left(w - \gamma\frac{u}{r}\right) - \gamma p(u_r + w_z) - (u p_r + w p_z)\right] \quad \text{at } r = R, \tag{1.25a}$$

$$\left(w_1 - \frac{1}{\gamma} p_1\right)_t = -\frac{1}{\gamma}\left(w_1 - \gamma\frac{u_1}{r} - \gamma u_{1r}\right) + \frac{1}{\gamma}(p_1 - \sigma_1) + \frac{1}{\gamma}\sigma p_z$$
$$- \left[u w_r + w w_z + \frac{1}{\gamma}\sigma(p - \sigma)\right]$$
$$- \frac{1}{\gamma} p\left(w - \gamma\frac{u}{r}\right) + p(u_r + w_z) + \frac{1}{\gamma}(u p_r + w p_z) \quad \text{at } z = L. \tag{1.25b}$$

It is shown in [3] and [5] that these linearized problems are well posed. However, it remains to be determined if the expansions proposed in (1.7)-(1.9) are uniformly valid. If they are, then we have a justification to assume that the full nonlinear problem is well posed in the sense of the results reported in [3] and [5]. To demonstrate the validity of the asymptotic expansion, we consider a sequence of one-dimensional models. We show that the resulting linear problems from the asymptotic expansions indeed lead to the solution of the original nonlinear problem. At the outset, let us emphasize that even the linear one-dimensional problem does not have a closed form solution. For this reason, we solve the problem in two essentially different ways and reach an agreement with the solution that we obtain through numerical methods. In the next section, we reduce the problem to one that is governed by a scalar wave equation. While the field equations become simple, the formulation of the radiation condition becomes a laborious task as we will see in the next section.

2. One-dimensional model problem for the sound propagation in the atmosphere. In this section, we propose a method of solving the one-dimensional model of sound propagation in the atmosphere. As mentioned earlier, obtaining the radiation boundary condition that is suitable for the numerical calculations is a difficult task. We derive the boundary conditions. The propagation is restricted in the vertical direction where the effect of gravity plays a crucial role. For computational purposes, the distance which extends to infinity needs to be truncated. Let us denote this finite interval to be $[0, L]$. The ideas behind deriving the radiation condition here are very similar to the one proposed recently in [6] in which wave propagation in certain types of viscoelastic materials is considered. The key idea here, as in [6], is to construct an interface problem with an interface at $z = L$, where, typically, a well-suited radiation condition that is best for numerical calculations can be posed. For $z > L$, the problem will be solved by an asymptotic procedure that is valid for small times. This procedure is similar to the one reported in [7]. But it turns out that these boundary conditions are not necessarily accurate only for smaller time scales. To present the idea, let us introduce the governing equations obtained from (1.10)-(1.13). The equations are

$$\sigma_t + w_z - w = 0, \tag{2.1}$$

$$w_t + \frac{1}{\gamma} p_z - \frac{1}{\gamma}(p - \sigma) = 0, \tag{2.2}$$

$$p_t + \gamma w_z - w = f. \tag{2.3}$$

These equations are defined over $0 < z < \infty$. Also, these equations, unlike the higher-dimensional equations, can be reduced to a single scalar equation as follows

$$w_{tt} - w_{zz} + w_z = \frac{1}{\gamma}(f - f_z) = G(z, t). \tag{2.4}$$

Boundary conditions are

$$w(0, t) = 0, \tag{2.5}$$

$$\lim_{z \to \infty} w(z, t) \text{ is bounded.} \tag{2.6}$$

Initial conditions are

$$w(z, 0) = 0, \tag{2.7}$$

$$w_t(z, 0) = 0. \tag{2.8}$$

Equation (2.4) can be simplified by the transformation

$$w(z, t) = e^{\lambda z} v(z, t), \tag{2.9}$$

with a choice of $\lambda = \frac{1}{2}$ as follows:

$$v_{tt} - v_{zz} + \tfrac{1}{4}v = e^{-1/2z} G(z, t) = g(z, t). \tag{2.10}$$

Suppose we want to impose a boundary condition, exact or otherwise, at the finite distance L that simulates the behavior at infinity. We claim that these boundary conditions are of the form

$$v_z = Bv + Kg, \tag{2.11}$$

where B and K are operators to be determined. This form has been observed by a number of authors, namely, Engquist and Majda [12]. In the event that g has a compact support in $0 < z < L$, the boundary condition will take the form

$$v_z = Bv. \tag{2.12}$$

2.1. Procedure. Initial conditions suggest that we consider the problem in the Laplace transform domain. Doing so, we obtain

$$s^2 \hat{v} = \hat{v}_{zz} - \tfrac{1}{4}\hat{v} + \hat{g}. \tag{2.13}$$

Here $\hat{v} = \hat{v}(z, s)$ and $\hat{g} = \hat{g}(z, s)$. The main theme here is to obtain asymptotic solutions of (2.13) in the exterior region $z > L$ with a given value of v at $z = L$. To do this we formulate an interface problem in which the boundary $z = L$ is the interface as follows:

$$v_{tt} = v_{zz} - \tfrac{1}{4}v + g(z, t), \qquad 0 < z < L, \tag{2.14}$$

which is the given field equation. Now define $\hat{U}(z, s; t)$ such that

$$s^2 \hat{U} = \hat{U}_{zz} - \tfrac{1}{4}\hat{U} + g(z, t), \qquad z > L, \tag{2.15}$$

with

$$\hat{U}(L, s; t) = v(L, t). \tag{2.16}$$

We remark that here t is considered as a parameter. Now it is easily verified that

$$\hat{v}(z, s) = \int_0^\infty e^{-st} \hat{U}(z, s; t)\, dt. \tag{2.17}$$

With the above construction, it is now possible to solve the problem in the exterior region $z > L$ by asymptotic procedures. We solve (2.15) subject to the interface condition (2.16). In addition, we impose a radiation condition which insures the boundedness of $v(z, t)$ which translates in the transform domain to requiring the boundedness of $\hat{U}(z, s; t)$. As mentioned earlier, our procedure for constructing the solution will be an asymptotic one with s as a large parameter reflecting the fact that solutions will be accurate for smaller times. But, asymptotic methods are known for their power. While the expansion is valid for a small time in the theoretical treatment, in real computational experiments time need not be small. To elaborate this statement, we consider an asymptotic procedure given in [7], which in theory was valid for a parameter α approaching infinity; surprisingly it turned out to be valid for $\alpha = \sqrt{5}$ in numerical experiments. Thus, the results are anticipated to be accurate for more than just small †times.

To solve (2.15), we shall use elementary concepts, along with appropriate asymptotic ideas. We first seek a particular solution. Let $\hat{W}(x, s; t)$ satisfy

$$\hat{W}_{zz} - (s^2 + \tfrac{1}{4})\hat{W} = -g(z, t). \tag{2.18}$$

Then $\hat{W}$ may be sought in the form

$$\hat{W}(z, s; t) = \sum_{k=1}^{\infty} \frac{W_{2k}(z)}{s^{2k}}. \tag{2.19}$$

Substituting the expansion in (2.18), we obtain

$$\left(\frac{W_2''}{s^2} + \frac{W_4''}{s^4} + \frac{W_6''}{s^6} + \cdots\right) - s^2\left(\frac{W_2}{s^2} + \frac{W_4}{s^4} + \frac{W_6}{s^6} + \cdots\right) - \frac{1}{4}\left(\frac{W_2}{s^2} + \frac{W_4}{s^4} + \frac{W_6}{s^6} + \cdots\right) = g(z, t).$$

Equating powers, we obtain the following:

$$\begin{aligned}
&(\text{power } s^0) && W_2 = g,\\
&(\text{power } s^{-2}) && W_2'' - W_4 - \tfrac{1}{4}W_2 = 0, \quad W_4 = g'' - \tfrac{1}{4}g,\\
&(\text{power } s^{-4}) && W_4'' - W_6 - \tfrac{1}{4}W_4 = 0, \quad W_6 = g^{(iv)} - \tfrac{1}{2}g'' + \tfrac{1}{16}g,
\end{aligned}$$

etc.

Remark. We may consider odd powers in the above expansion and verify that all corresponding terms are zero. Thus,

$$\hat{W} = \left(\frac{W_2}{s^2} + \frac{W_4}{s^4} + \frac{W_6}{s^6} + \cdots\right).$$

Let $\hat{R} = \hat{U} - \hat{W}$ (i.e., $\hat{R}$ is the homogeneous solution). Then $\hat{R}$ satisfies

$$\hat{R}_{zz} - (s^2 + \tfrac{1}{4})\hat{R} = 0. \tag{2.20}$$

$\hat{R}$ satisfies the boundary condition that it vanishes at infinity in such a way that the wave is purely outgoing. Now it is possible to construct an expansion for $\hat{R}$ following the geometrical acoustics idea that

$$\hat{R} = e^{-s(z-L)}\left(U_0 + \frac{U_1}{s} + \frac{U_2}{s^2} + \cdots\right). \tag{2.21}$$

Substituting the above expansion in (2.20), we have

$$-2s\left(U_0' + \frac{U_1'}{s} + \frac{U_2'}{s^2} + \cdots\right) + \left(U_0'' + \frac{U_1''}{s} + \frac{U_2''}{s^2} + \cdots\right) - \frac{1}{4}\left(U_0 + \frac{U_1}{s} + \frac{U_2}{s^2} + \cdots\right) = 0.$$

Recall that $\hat{R} = \hat{U} - \hat{W}$. We also have the requirement that $\hat{U}(L, s; t) = v(L, t)$. Thus, $\hat{R}(L, s; t) = v(L, t) - \hat{W}(L, s; t)$, indicating that the first term independent of s on the right of this equation is $v(L, t)$. Using this fact in the previous expansion (2.21) and by equating the s^1 powers, we obtain

$$U_0' = 0; \quad U_0(L, t) = \text{constant}; \quad U_0(L, t) = v(L, t).$$

The next term we obtain in the expansion is

$$-2U_1' + U_0'' - \tfrac{1}{4}U_0 = 0.$$

Again we note that at $z = L$ the right-hand side of $\hat{R} = \hat{U} - \hat{W}$ does not have an inverse power of s, yielding $U_1(L, t) = 0$. Thus, integrating for U_1, we get

$$U_1(z; t) = -\tfrac{1}{8}v(L, t)(z - L).$$

The next term in the expansion has some interesting features. This time, forcing terms will appear in the expansion. When doing the expansion, we find that

$$-2U_2' + U_1'' - \tfrac{1}{4}U_1 = 0.$$

As before, we see from the construction of $\hat{R}$ that at z equal to L,

$$U_2(L, t) = -W_2 = -g(L, t).$$

Thus, integration yields

$$U_2 = \tfrac{1}{128}v(L, t)(z-L)^2 - g(L, t).$$

A similar procedure yields

$$U_3 = \tfrac{1}{128}v(L, t)(z-L) - \tfrac{1}{3182}v(L, t)(z-L)^3 + \tfrac{1}{8}g(L, t)(z-L).$$

We note here that on the boundary $z = L$, $U_3 = 0$. Moreover, the remaining terms in the expansion, when evaluated at $z = L$, yield only terms involving g. Thus summarizing the results, we have

$$\hat{U}(z, s; t) = \frac{g(L, t)}{s^2} + e^{-s(z-L)}\left(U_0 + \frac{U_1}{s} + \frac{U_2}{s^2} + \cdots\right) + \cdots. \tag{2.22}$$

Differentiating with respect to z, we have

$$\hat{U}_z(z, s; t) = e^{-s(z-L)}\left(U_{0,z} + \frac{U_{1,z}}{s} + \frac{U_{2,z}}{s^2} + \cdots\right) - s\,e^{-s(x-L)}\left(U_0 + \frac{U_1}{s} + \frac{U_2}{s^2} + \cdots\right).$$

Using the relations that we obtained for U_0, U_1, and U_2, we obtain

$$\hat{U}_z(L, s; t) = -sv(L, t) - \frac{1}{8}\frac{v(L, t)}{s} + \frac{g(L, t)}{s} + F(g, s), \tag{2.23}$$

where F is a functional of $g(L, t)$ and involves the inverses of s^4 and higher powers. Neglecting F and using the definition (2.17), we have

$$\hat{v}_z(L, s) = -s\hat{v}(L, s) - \frac{1}{8}\frac{\hat{v}(L, s)}{s} + \frac{\hat{g}(L, s)}{s}. \tag{2.24}$$

Inverting the Laplace transform, we obtain our desired boundary condition as follows:

$$v_z(L, t) = -v_t(L, t) - \frac{1}{8}\int_0^t v(L, \tau)\,d\tau + \int_0^t g(L, \tau)\,d\tau. \tag{2.25}$$

We note that this boundary condition is of the form (2.11). Now translating this condition in terms of the original variable w we obtain

$$w_z(L, t) + w_t(L, t) = \frac{1}{2}w(L, \tau) - \frac{1}{8}\int_0^t w(L, \tau)\,d\tau + \int_0^t g(L, \tau)\,e^{L/2}\,d\tau. \tag{2.26}$$

Thus for computational purposes, we shall replace the boundary condition (2.6) by the approximate boundary condition (2.26).

3. Verification of the asymptotic theory and the numerical procedure. As pointed out in the first section of the paper, it is difficult to construct an analytic solution, even for the linear problem. Thus we content ourselves with numerical solutions. To verify the correctness of the linear problem, we compute the results through two different formulations. One is to solve the reduced wave equation (2.4) subject to the initial conditions (2.7) and (2.8), the boundary conditions (2.5) at $z = 0$, and an approximate

radiation condition at (2.26) at $z = L$. Let us emphasize here that we are solving the problem over the interval $[0, L]$ and not over $[0, \infty)$. For the ease of the reader, let us restate the problem consisting of the reduced wave equation which is indicated below and we shall call it (P-W) for future reference.

$$w_{tt} - w_{zz} + w_z = \frac{1}{\gamma}(f - f_z) = G(z, t), \tag{3.1}$$

$$w(z, 0) = 0, \tag{3.2}$$

$$w_t(z, 0) = 0, \tag{3.3}$$

$$w(0, t) = 0, \tag{3.4}$$

$$w_z(L, t) + w_t(L, t) = \frac{1}{2} w(L, \tau) - \frac{1}{8}\int_0^t w(L, \tau)\, d\tau + \int_0^t g(L, \tau)\, e^{L/2}\, d\tau. \tag{3.5}$$

Solving (P-W) numerically is straightforward. It is solved by an explicit finite difference formula. This is a standard discretization. The integrals in the boundary condition (3.5) are evaluated by a rectangular quadrature formula. It could be shown that problem (P-W) is well posed in the sense of a bounded growth of energy. Related ideas can be found in Barry, Bielak, and MacCamy [6]. However, the situation here is different and the related theoretical results will be reported elsewhere.

As mentioned earlier, the comparison could be made by solving the system (2.1)-(2.3) with appropriate initial and boundary conditions as described in the last section. Still the problem is to be considered in the truncated domain $[0, L]$. However, the system involves the primitive variables for the acoustic components p, ρ, and w. The boundary condition (2.26) thus needs to be recast in these variables in an appropriate manner that is suitable for numerical computations. Again, we summarize the equations that we wish to solve:

$$\sigma_t + w_z - w = 0, \tag{3.6}$$

$$w_t + \frac{1}{\gamma} p_z - \frac{1}{\gamma}(p - \sigma) = 0, \tag{3.7}$$

$$p_t + \gamma w_z - w = f. \tag{3.8}$$

The initial conditions are

$$p = \sigma = w = 0 \quad (\text{at } t = 0). \tag{3.9}$$

The boundary conditions are

$$w(0, t) = 0 \tag{3.10}$$

and (3.5). We replace w_z in (3.5) using (3.8) so that the left-side of the boundary operator becomes the rate of change of the incoming Riemann variable. The result is

$$\frac{\partial}{\partial t}(p - \gamma w) = \left(1 - \frac{\gamma}{2}\right) w + f + \frac{\gamma}{8}\int_0^t w(L, \tau)\, d\tau - \gamma \int_0^t g(L, \tau)\, e^{L/2}\, d\tau. \tag{3.11}$$

We shall call the problem (3.6)-(3.11) by (P-L1). We briefly show the calculations that identify the Riemann variables associated with the system. First, we note that the system (3.6)-(3.8) can be written in the form

$$\mathbf{u}_t + \mathbf{A}\mathbf{u}_z = \mathbf{H} \tag{3.12}$$

with

$$\mathbf{u}^T = (\sigma, w, p), \tag{3.13}$$

$$\mathbf{H}^T = \left(w, \frac{1}{\gamma}(p-\sigma), w+f\right), \tag{3.14}$$

and

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1/\gamma \\ 0 & \gamma & 0 \end{bmatrix}. \tag{3.15}$$

It is easily shown that the eigenvalues of $\mathbf{A}$ are $(1, -1, 0)$. Corresponding left eigenvectors are

$$\mathbf{e}_1 = (0, \gamma, 1)^T, \quad \mathbf{e}_2 = (0, -\gamma, 1)^T, \quad \mathbf{e}_3 = (-\gamma, 0, 1)^T.$$

Multiplying on the left of (3.12) by each of these vectors, we obtain

$$R_t + R_z = w + p - \sigma + f, \tag{3.16}$$

$$S_t - S_z = w - p + \sigma + f, \tag{3.17}$$

$$T_t = w - \gamma w + f. \tag{3.18}$$

Here

$$R = p + \gamma w$$

and

$$S = p - \gamma w,$$

are the outgoing and incoming Riemann variables, respectively. $T = p - \gamma\sigma$ carries no information, as it has zero speed. These variables play a crucial role in the numerical implementation of boundary conditions. For example, when we implement the boundary condition (3.11), S is determined at the current time level. However, to obtain the value of p and w, we use (3.16) (discretizations may be seen in the Appendix). This determines R. Once R and S are determined, p is determined by

$$p = \frac{R+S}{2}, \tag{3.19}$$

$$w = \frac{R-S}{2\gamma}. \tag{3.20}$$

A similar consideration is given at $z = 0$, where $w = 0$. To determine p, we can use (3.17) for S. Once S is determined at $z = 0$, we immediately obtain

$$p = S \qquad (z = 0). \tag{3.21}$$

Figures 1 and 2 show a comparison of the results obtained through solving the problems (P-W) and (P-L1) numerically. For the test case presented here, the source function f was chosen to have a compact support in space and time as follows:

$$f(z, t) = \begin{cases} e^{-a(z-z_0)^2} \sin 2\pi t & 0 \leqq t \leqq T_0, \quad |z - z_0| < \varepsilon, \\ 0 & \text{otherwise.} \end{cases} \tag{3.22}$$

Here $a = 40$, $z_0 = 0.25L$, $\varepsilon = 0.025L$, and $T_0 = 1$. The results are the same. The results shown here are after 1000 and 7000 time steps, respectively, and the pressure distributions are shown in Figs. 1 and 2. In Fig. 2, there is a little difference which is due to the

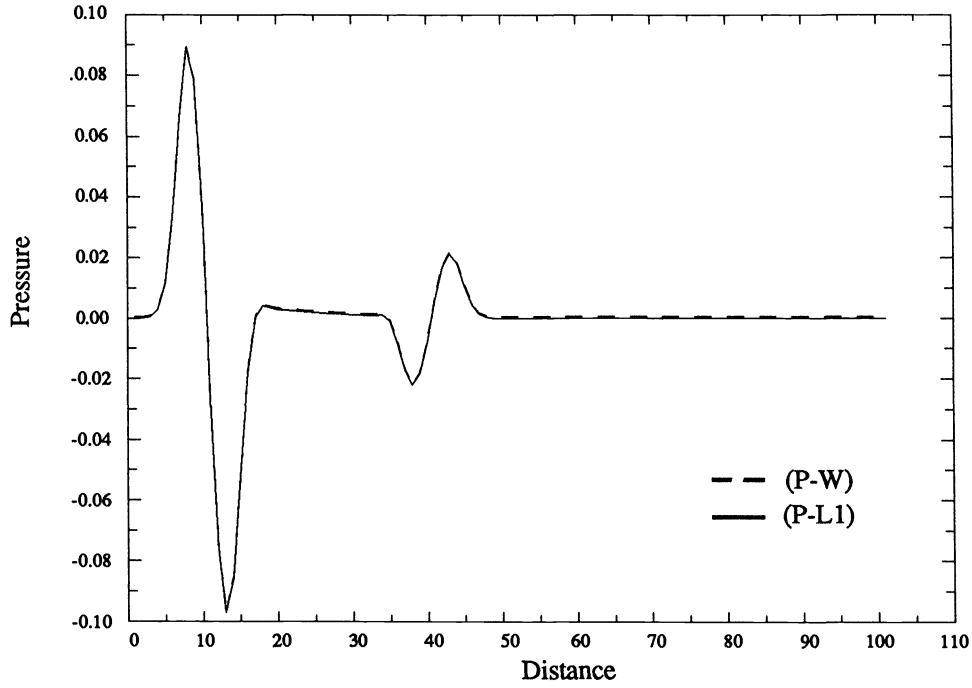


FIG. 1. *Comparison between solutions of* (P-W) *and* (P-L1) *at* 1000 *time steps.*

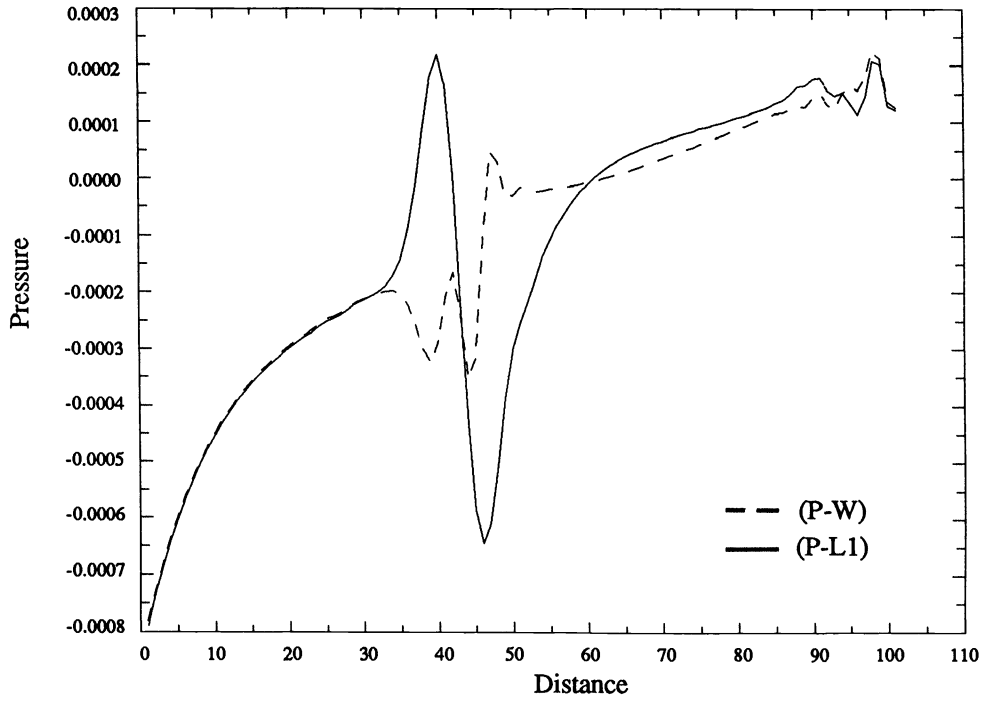


FIG. 2. *Comparison between solutions of* (P-W) *and* (P-L1) *at* 7000 *time steps.*

boundary conditions. However, the scale is small and the difference is below the truncation error. Thus, with this agreement, we contend that the results are nearly equal to the analytic solution of the problem. In Fig. 3, we show the behavior of the pressure wave as a function of both space and time. Time reaches up to 10,000 steps or 10 periods. We see that the wave moves in two directions: one toward the radiation boundary and the other toward the ground. As we should expect, the wave that hits the ground reflects at $z = 0$, and, eventually, decays to zero. In the calculations reported here, $\Delta z = 0.025$, $L = 10$, and the perturbation parameter $\varepsilon = 0.005$. There were 400 grid points used for the computations, indicating a fine grid. Thus, the results that we obtained here had to be thought of as exact.

3.1. Nonlinear correction and the fully nonlinear solution. Now we proceed to investigate the nonlinear effects. First, we propose a second-order correction to the linear problem. This problem results by retaining $O(\varepsilon^2)$ terms as in (1.18)–(1.21). These equations are

$$\sigma_{1t} + w_{1z} - w_1 = -(\sigma w)_z + \sigma w = f_1, \tag{3.23}$$

$$w_{1t} + \frac{1}{\gamma} p_{1z} - \frac{(p_1 - \sigma_1)}{\gamma} = -[\sigma w_t + w w_z] = f_2, \tag{3.24}$$

$$p_{1t} + \gamma w_{1z} - w_1 = \sigma f + \sigma w - \gamma p w_z + \gamma w w_t = f_3. \tag{3.25}$$

Initial conditions are

$$p_1 = \sigma_1 = w_1 = 0 \quad \text{for } t = 0. \tag{3.26}$$

The boundary condition at $z = 0$ is

$$w_1(0, t) = 0, \qquad t > 0. \tag{3.27}$$

The radiation boundary condition at $z = L$ has the same form as before, which is

$$\frac{\partial}{\partial t}(p_1 - \gamma w_1) = \left(1 - \frac{\gamma}{2}\right) w_1 + f_3 + \frac{\gamma}{8} \int_0^t w_1(L, \tau)\, d\tau - \gamma \int_0^t e^{1/2L} H(L, \tau)\, d\tau, \tag{3.28}$$

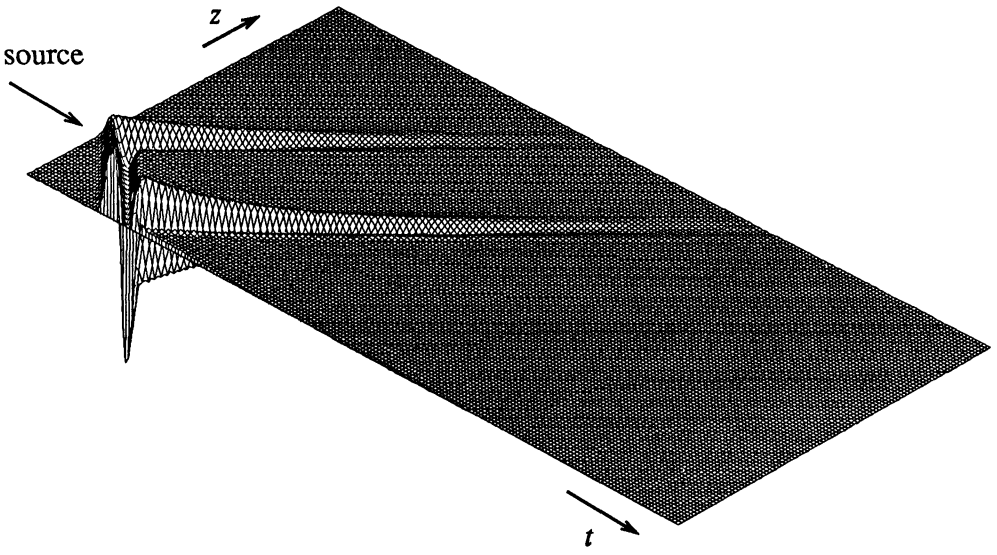


FIG. 3. *Pressure time history in the computational domain for* (P-L1) (10 *periods or* 10,000 *time steps*) $(e^{-z}\varepsilon p)$.

where

$$H(L, \tau) = [(p-\sigma)w_{zz} - 2ww_{zt} - (\gamma+1)w_z w_t]\, e^{1/2L} - \frac{1}{\gamma} f\sigma_z.$$

Let us denote the problem described by (3.23)-(3.28) by (P-L2). This problem has the same form as the problem (P-L1). The same numerical scheme that applies to (P-L1) holds here, including the implementation of the radiation condition.

Figure 4 illustrates the same situation that was considered in Fig. 3. The results shown in Fig. 4 contain the nonlinear correction. In particular, the nonlinear field is computed according to

$$p = \varepsilon p + \varepsilon^2 p_1. \tag{3.29}$$

The results given here have the same pattern as the linear case. However, for the linear case, the pressure profile is smooth. But, in the nonlinear correction case we see the wave steepening as we should expect. To validate this nonlinear solution, we solved the fully nonlinear problem in an independent manner as described in the next section. Indeed, we obtained the nonlinear behavior correctly by combining the two linear solutions obtained from (P-L1) and (P-L2).

4. Fully nonlinear one-dimensional problem. To solve the full nonlinear acoustic equations, we first have to obtain the appropriate governing equations. It is also important to obtain the appropriate radiation condition for the nonlinear system. Let us begin the discussion with the governing equations.

The field equations governed by the fluid flow are

$$\rho_t + \rho q_z + q\rho_z = 0, \tag{4.1}$$

$$q_t + qq_z + \frac{1}{\gamma\rho} p_z = -\frac{1}{\gamma}, \tag{4.2}$$

$$p_t + qp_z + \gamma p q_z = \varepsilon f(z, t)\rho. \tag{4.3}$$

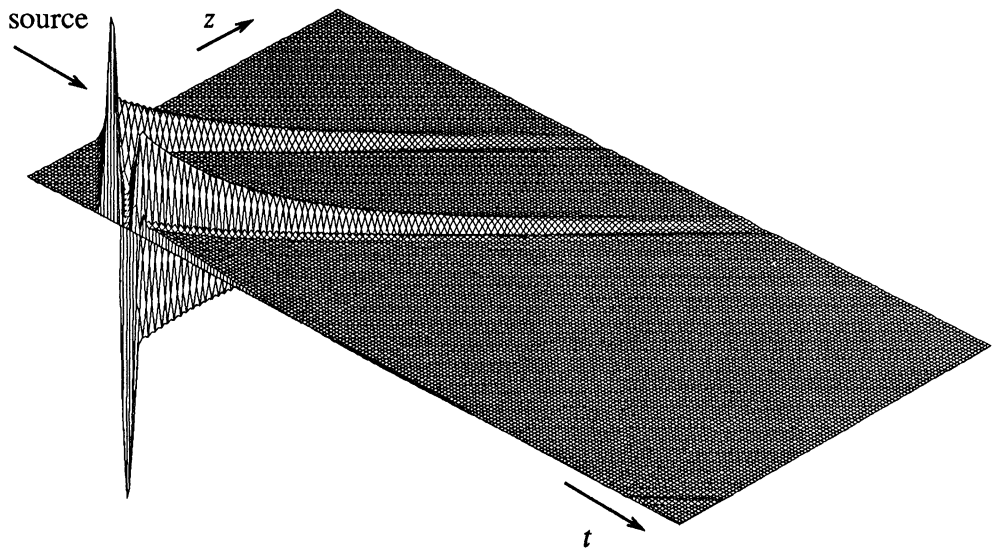


FIG. 4. *Pressure time history in the computational domain for nonlinear correction* (10 *periods or* 10,000 *time steps* ($e^{-z}[\varepsilon p + \varepsilon^2 p_1]$).

When written in the matrix form, we have

$$\mathbf{U}_t + \mathbf{A}\mathbf{U}_z = \mathbf{H}, \tag{4.4}$$

where

$$\mathbf{U} = (\rho, q, p)^T,$$

$$\mathbf{H} = \left(0, \frac{-1}{\gamma}, \varepsilon f \rho\right)^T,$$

and

$$\mathbf{A} = \begin{bmatrix} q & \rho & 0 \\ 0 & q & 1/\gamma\rho \\ 0 & \gamma p & q \end{bmatrix}.$$

The eigenvalues of $\mathbf{A}$ are $\lambda = q + \sqrt{p/\rho}$, $q - \sqrt{p/\rho}$ and q. Left eigenvectors corresponding to these eigenvalues are

$$\mathbf{e}_1 = (0, \gamma\sqrt{p\rho}, 1)^T$$

$$\mathbf{e}_2 = (0, -\gamma\sqrt{p\rho}, 1)^T,$$

$$\mathbf{e}_3 = \left(1, 0, -\frac{\rho}{\gamma p}\right)^T,$$

respectively. Multiplying equation (4.4) by $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$, respectively, we obtain the following equations:

$$(\gamma\sqrt{p\rho}, 1)\begin{pmatrix} q \\ p \end{pmatrix}_t + \left(q + \sqrt{\frac{p}{\rho}}\right)(\gamma\sqrt{p\rho}, 1)\begin{pmatrix} q \\ p \end{pmatrix}_z = (\gamma\sqrt{p\rho}, 1)\begin{pmatrix} -1/\gamma \\ \varepsilon\rho f \end{pmatrix}, \tag{4.5a}$$

$$(-\gamma\sqrt{p\rho}, 1)\begin{pmatrix} q \\ p \end{pmatrix}_t + \left(q - \sqrt{\frac{p}{\rho}}\right)(-\gamma\sqrt{p\rho}, 1)\begin{pmatrix} q \\ p \end{pmatrix}_z = (-\gamma\sqrt{p\rho}, 1)\begin{pmatrix} -1/\gamma \\ \varepsilon\rho f \end{pmatrix}, \tag{4.5b}$$

$$\left(1, -\frac{\rho}{\gamma p}\right)\begin{pmatrix} q \\ p \end{pmatrix}_t + q\left(1, -\frac{\rho}{\gamma p}\right)\begin{pmatrix} q \\ p \end{pmatrix}_z = -\varepsilon\frac{f}{\gamma p}. \tag{4.6}$$

For sources that have a compact support as described in the numerical experiments earlier, (4.3) suggests along a streamline that

$$p = A\rho^\gamma, \tag{4.7}$$

where

$$A = \frac{p_0}{\rho_0^\gamma}.$$

With the above equations we can describe the fully nonlinear acoustic field and the associated radiation conditions. Let

$$p = e^{-z}(1 + p'), \quad \rho = e^{-z}(1 + \rho'), \quad q = u'.$$

Thus, rewriting (4.4) in terms of the acoustic variables and dropping the "prime" notations, we obtain the field equations in the matrix form as follows:

$$\begin{pmatrix} \rho \\ u \\ p \end{pmatrix}_t + \begin{bmatrix} u & (1+\rho) & 0 \\ 0 & u & 1/\gamma(1+\rho) \\ 0 & \gamma(1+p) & u \end{bmatrix}\begin{pmatrix} \rho \\ u \\ p \end{pmatrix}_z = \begin{pmatrix} u(1+\rho) \\ (p-\rho)/\gamma(1+p) \\ u(1+p) + \varepsilon f(1+\rho) \end{pmatrix}. \tag{4.8}$$

These equations are to be solved subject to the initial conditions

$$p = \rho = u = 0 \quad \text{at } t = 0, \tag{4.9}$$

and boundary conditions

$$u = 0 \quad \text{at } t = 0, \tag{4.10}$$

and a radiation condition which we shall derive next at $z = L$.

When the above-mentioned acoustic expansions are used in (4.5a) and (4.5b) together with the isentropic relation (4.7), we obtain the following equations:

$$R_t + (u + (1+p)^{(\gamma-1)/2\gamma}) R_z$$
$$= -\frac{1}{\gamma} + \frac{1}{\gamma}[u + (1+p)^{(\gamma-1)/2\gamma}](1+p)^{(\gamma-1)/2\gamma} + \frac{\varepsilon f}{\gamma}(1+p)^{(\gamma-1)/2\gamma} \tag{4.11}$$

$$S_t + (u - (1+p)^{(\gamma-1)/2\gamma}) S_z$$
$$= -\frac{1}{\gamma} + \frac{1}{\gamma}[u - (1+p)^{(\gamma-1)/2\gamma}](1+p)^{(\gamma-1)/2\gamma} - \frac{\varepsilon f}{\gamma}(1+p)^{(\gamma-1)/2\gamma}, \tag{4.12}$$

and

$$1 + p = (1+\rho)^{\gamma}. \tag{4.13}$$

Here

$$R = u + \frac{2}{\gamma - 1}(1+p)^{(\gamma-1)/2\gamma} \tag{4.14}$$

and

$$S = -u + \frac{2}{\gamma - 1}(1+p)^{(\gamma-1)/2\gamma}. \tag{4.15}$$

These variables, as before in the linear case, play a crucial role on the numerical, as well as physical, boundary conditions. In this nonlinear situation, the boundary condition that we derived for the linear case does not hold. Thus, we settle for a condition that is applicable for nonlinear problems found in Thompson [9]. This is conceptually very simple once the equations are cast in the Riemann variables. The principle here is that at $z = L$ for the waves not to reflect in the computational domain, the speed of the incoming variable must be zero. Thus, applying this condition in (4.12) for the incoming variable, we obtain

$$S_t = -\frac{1}{\gamma} + \frac{1}{\gamma}[u - (1+p)^{(\gamma-1)/2\gamma}](1+p)^{(\gamma-1)/2\gamma} - \frac{\varepsilon f}{\gamma}(1+p)^{(\gamma-1)/2\gamma}. \tag{4.16}$$

This condition will be the replacing radiation condition for the nonlinear situation.

Numerical implementation of the system of equations (4.8) yields considerable problems. Even though this is a hyperbolic system, it is not an integrable system. It cannot be cast in conservation or weak conservation form. For these types of systems, numerical methods are not readily available in the literature. Thus, we had to modify the Lax-Wendroff method described in the Appendix. However, the numerical treatment of the boundary conditions remain the same.

Again, the same test case that was applied to the linear system was simulated here. The source was taken to be the same. The results are given in Fig. 5. Comparing

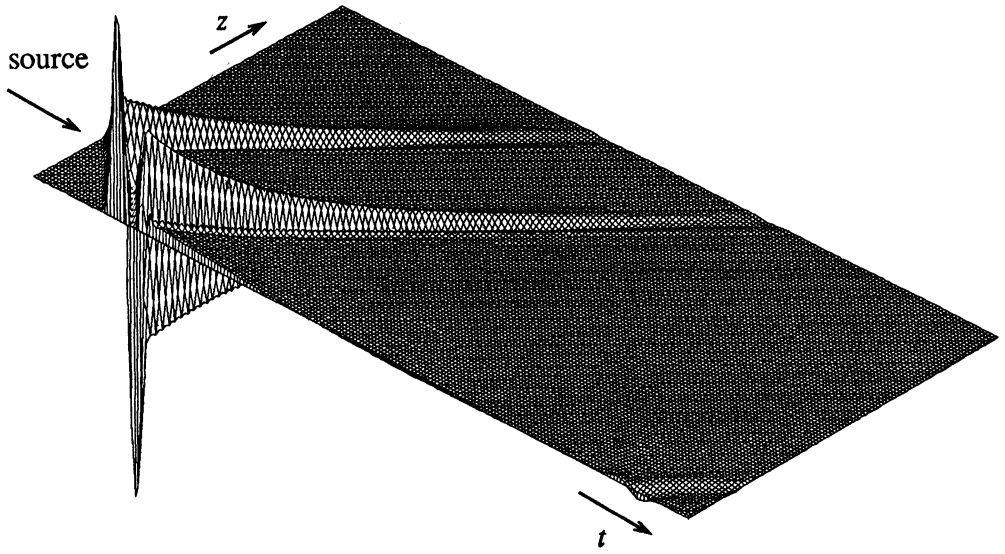


FIG. 5. *Pressure time history in the computational domain for full nonlinear problem* (10 *periods or* 10,000 *time steps*).

the results obtained in Fig. 4, we see an excellent agreement between the combination of the two solutions of the linearized problems and the full nonlinear solution. Again, we have used a surface plot to show the long-time behavior of the solution. We note in Fig. 5 that there is a slight discrepancy in the results after a long time. This discrepancy occurs near the origin and near the radiation boundary, but is of negligible order of magnitude. We attribute this difference to the full nonlinear solution and the quadratic nonlinear solutions (i.e., $O(\varepsilon^2)$ solutions). It should be noted once again that these solutions were obtained from a fine grid (400 grid points) and to obtain the full nonlinear problem in higher dimensions can be tremendously costly. This is where the concept of solving two successive linear problems becomes attractive. Indeed, we followed this principle to obtain the three-dimensional axisymmetric problem which is discussed in the next section.

5. The three-dimensional axisymmetric problem. Here we consider the procedures for solving the three-dimensional problem. In particular, we solve the problems (P) and (P′) as described in § 1. The procedures for obtaining the radiation boundary conditions are similar to those that we considered in the last section on the full one-dimensional nonlinear situation. The difficulty here is that the problem is multi-dimensional. Thus, for the purpose of obtaining radiation conditions, we consider the Riemann variables in the direction in which we wish to impose the boundary condition. Let us consider the computational domain shown in Fig. 6. Suppose on Γ_1, which is located at $r = R$, we want to obtain appropriate radiation conditions. We apply the procedure described in § 4, by considering the system of equations only in the r direction. The first two equations that we obtain are

$$R_t + R_r = -w_z + \frac{1}{\gamma} w - \frac{u}{r}, \tag{5.1}$$

$$S_t - S_r = w_z - \frac{1}{\gamma} w + \frac{u}{r}, \tag{5.2}$$

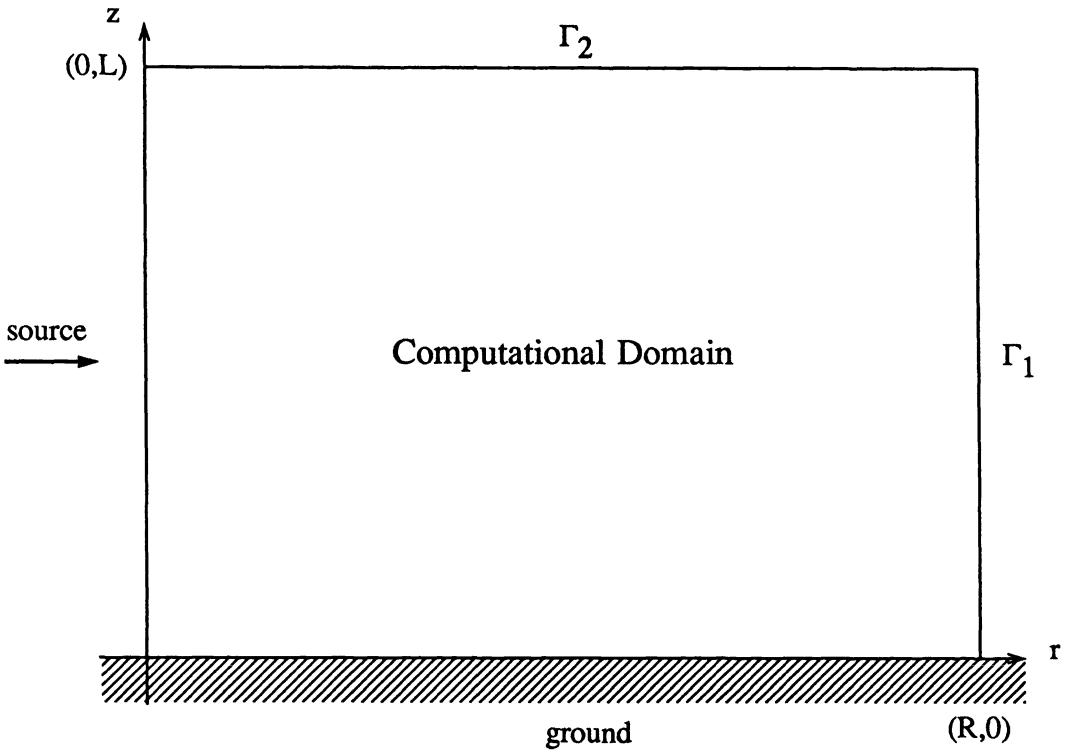


FIG. 6. *Three-dimensional computational domain.*

where

$$R = u + \frac{1}{\gamma} p$$

and

$$S = u - \frac{1}{\gamma} p.$$

Again, by using Thompson's procedure to obtain the radiation condition at $r = R$, we set the speed of the incoming variable S equal to zero, which is equivalent to suppressing the r derivative. This yields

$$S_t = w_z - \frac{1}{\gamma} w + \frac{u}{r}. \tag{5.3}$$

This is exactly the boundary condition given in equation (1.17a). A similar consideration is given in problem (P′) to obtain the boundary condition (1.25a).

To obtain the boundary condition at $z = L$, we rewrite the equations with Riemann variables in the z direction. The first two equations of importance are

$$R_t + R_z = \frac{1}{\gamma}(p - \sigma) + \frac{w}{\gamma} - \frac{u}{r} - u_r, \tag{5.4}$$

$$S_t - S_z = \frac{1}{\gamma}(p - \sigma) - \frac{w}{\gamma} + \frac{u}{r} + u_r. \tag{5.5}$$

Again by the principle used earlier, we obtain the boundary condition

$$S_t = \frac{1}{\gamma}(p-\sigma) - \frac{w}{\gamma} + \frac{u}{r} + u_r. \tag{5.6}$$

A similar consideration yields the boundary condition (1.25b) for problem (P′) at $z = L$.

It is important to compare the difference in the nature of the wave propagation with and without the gravity term. The discussion we have in §§ 3 and 4 gives most of the detail of the problem with the gravity term. Here we want to consider the equations without the gravity term. Also, we want to compare the nonlinear situation that corresponds to the equations with the gravity. Again, formulation of the problem plays a crucial role. However, the principle of deriving a sequence of linear problems remains the same. The expansions that we need here to linearize the governing equations (1.1)-(1.3) are as follows:

$$\rho = 1 + \varepsilon\sigma + \varepsilon^2\sigma_1 + \cdots, \tag{5.7a}$$

$$p = 1 + \varepsilon p + \varepsilon^2 p_1 + \cdots, \tag{5.7b}$$

and

$$\mathbf{q} = \varepsilon\mathbf{u} + \varepsilon^2\mathbf{u}_1 + \cdots. \tag{5.8}$$

It is important to notice that the perturbation parameter ε has a different meaning in this situation than it had in the case of inclusion of gravity. In the gravity case, it was a measure of the entropy production. While the same notion is issued from (1.3), the measure of acoustic pressure can drastically vary. The first resulting linear system from the expansion is

$$\sigma_t + u_r + w_z = -\frac{u}{r}, \tag{5.9}$$

$$u_t + \frac{1}{\gamma} p_r = 0, \tag{5.10}$$

$$w_t + \frac{1}{\gamma} p_z = 0, \tag{5.11}$$

$$p_t + \gamma(u_r + w_z) = -\gamma\frac{u}{r} + f. \tag{5.12}$$

The initial conditions are

$$p = \sigma = u = w = 0 \quad \text{at } t = 0. \tag{5.13}$$

The boundary condition at $z = 0$ is

$$w = 0 \quad \text{for } t > 0. \tag{5.14}$$

The radiation conditions are derived from the principles that we used for the gravity case. They are

$$\left(w - \frac{1}{\gamma} p\right)_t = u_r + \frac{u}{r} \quad \text{at } z = L, \tag{5.15a}$$

$$\left(u - \frac{1}{\gamma} p\right)_t = w_z + \frac{u}{r} \quad \text{at } r = R. \tag{5.15b}$$

Similarly, the problem for the nonlinear correction can be formulated. The equations and boundary conditions are listed below:

$$\sigma_{1t} + u_{1r} + w_{1z} = -\frac{u_1}{r} - \left[(u\sigma)_r + (w\sigma)_z + \sigma \frac{u}{r} \right], \tag{5.16}$$

$$u_{1t} + \frac{1}{\gamma} p_{1r} = -\left[uu_r + wu_z - \frac{1}{\gamma} \sigma p_r \right], \tag{5.17}$$

$$w_{1t} + \frac{1}{\gamma} p_{1z} = -\left[uw_r + ww_z - \frac{1}{\gamma} \sigma p_z \right], \tag{5.18}$$

$$p_{1t} + \gamma(u_{1r} + w_{1z}) = -\gamma \frac{u_1}{r} + \sigma f - \gamma p \left(\frac{u}{r} + u_r + w_z \right) - (up_r + wp_z). \tag{5.19}$$

As before, the initial conditions are zero and the radiation conditions have the same form. At $z = L$,

$$\begin{aligned} \left(w_1 - \frac{1}{\gamma} p_1 \right)_t = &-\left[uw_r + ww_z - \frac{1}{\gamma} wp_z \right] \\ &+ \left[u_{1r} + \frac{u_1}{r} + p(u_r + u_r + w_z) + \frac{1}{\gamma} (up_r + wp_z) \right], \end{aligned} \tag{5.20}$$

and at $r = R$,

$$\left(u_1 - \frac{1}{\gamma} p_1 \right)_t = -\left[uu_r + wu_z - \frac{1}{\gamma} p_r \right] + w_{1z} + \frac{u_1}{r} + p \left(\frac{u}{r} + u_r + w_z \right) + \frac{1}{\gamma} (up_r + wp_z). \tag{5.21}$$

The numerical method and the treatment of boundary conditions for this case are similar to the one employed in the gravity case. Details are found in the next section.

5.1. Numerical results and discussions. Numerical simulations are made by considering a pulse source above the ground which oscillates sinusoidally for a finite time as before. The form is similar to the one that we used for the one-dimensional simulation. It is

$$f(\mathbf{r}, t) = \begin{cases} e^{-a(\mathbf{r}-\mathbf{r}_0)^2} \sin 2\pi t & 0 \leqq t \leqq T, \quad |\mathbf{r} - \mathbf{r}_0| < \varepsilon, \\ 0 & \text{otherwise.} \end{cases} \tag{5.22}$$

Here, $a = 20$, $T = 1$, and $\varepsilon = .005$. Moreover, $\mathbf{r} = (r, z)$ and $\mathbf{r}_0 = (0, 1)$. The time step was chosen to be $\Delta t = .01$. Calculations were carried out up to 1300 steps or 13 periods.

The results reported in this section are three-dimensional plots of pressure distribution over the entire computational domain at different time intervals. The results reported in Figs. 7-11 are the nonlinear solutions of the gravitational field equations. Each figure shows the progress of the wave at the end increment of one period. The results reported in Figs. 12-16 correspond to the same situation without the gravity term. As we see from these figures, the pressure field has higher intensity in the absence of gravity than in the case that includes gravity. However, after a long time the intensity of the gravitational field becomes higher. This is particularly true for large distances. Figure 17 shows the progression of the ratio of the maximum amplitude of the waves in each case. There are two graphs; one is along the ground $z = 0$ (solid line) and the other is at $z = 0.5$ (dashed line). Initially, the ratio of the pressure is higher for the case without gravity; as time progresses, the ratio decreases and eventually the gravitational field begins to dominate. The nondimensional distances used here in the z and

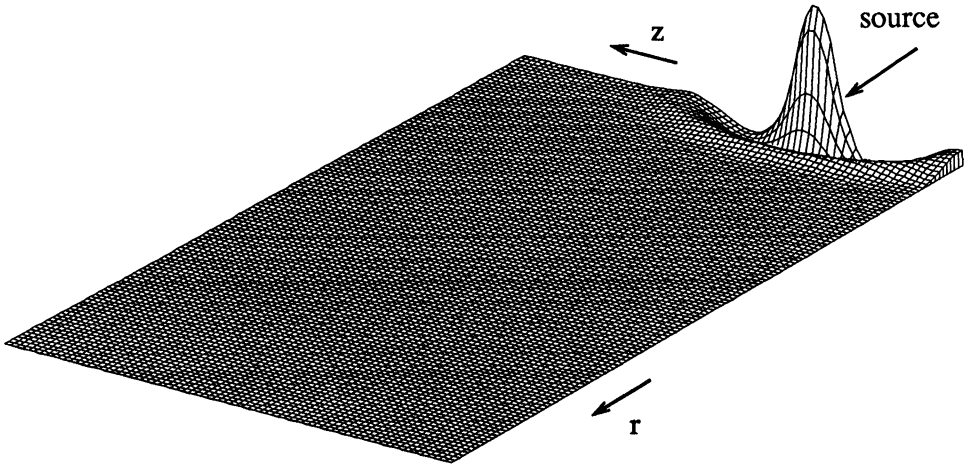


FIG. 7. *Propagation of acoustic gravity waves at time* $t = 1$ *period* (100 *steps*).

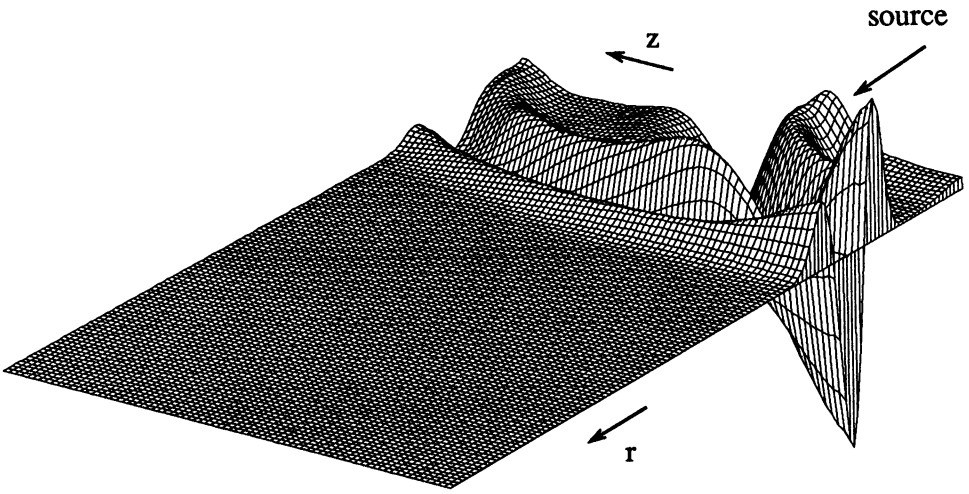


FIG. 8. *Propagation of acoustic gravity waves at time* $t = 3$ *periods* (300 *steps*).

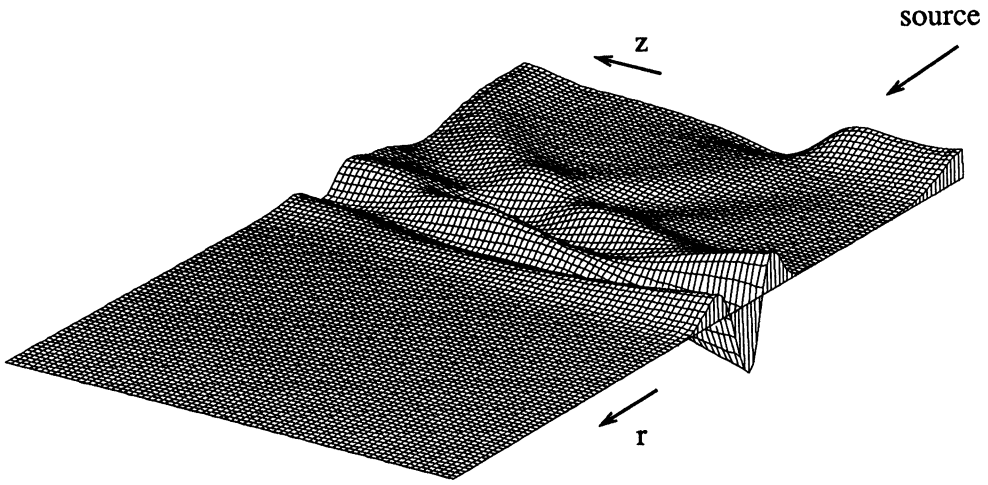


FIG. 9. *Propagation of acoustic gravity waves at time* $t = 5$ *periods* (500 *steps*).

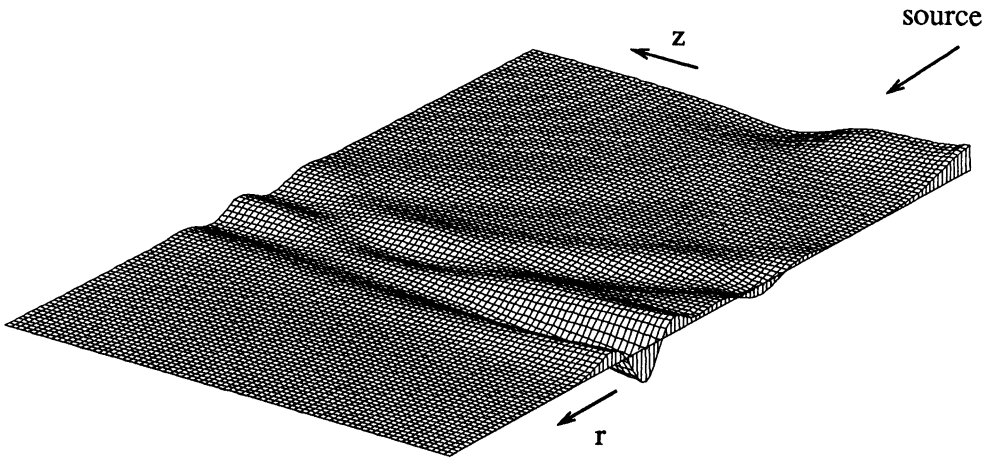


FIG. 10. *Propagation of acoustic gravity waves at time* $t = 7$ *periods* (700 *steps*).

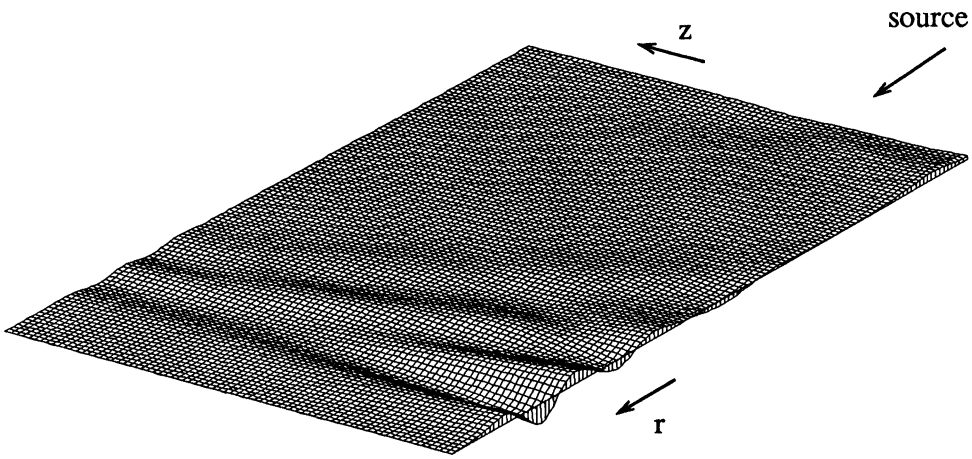


FIG. 11. *Propagation of acoustic gravity waves at time* $t = 9$ *periods* (900 *steps*).

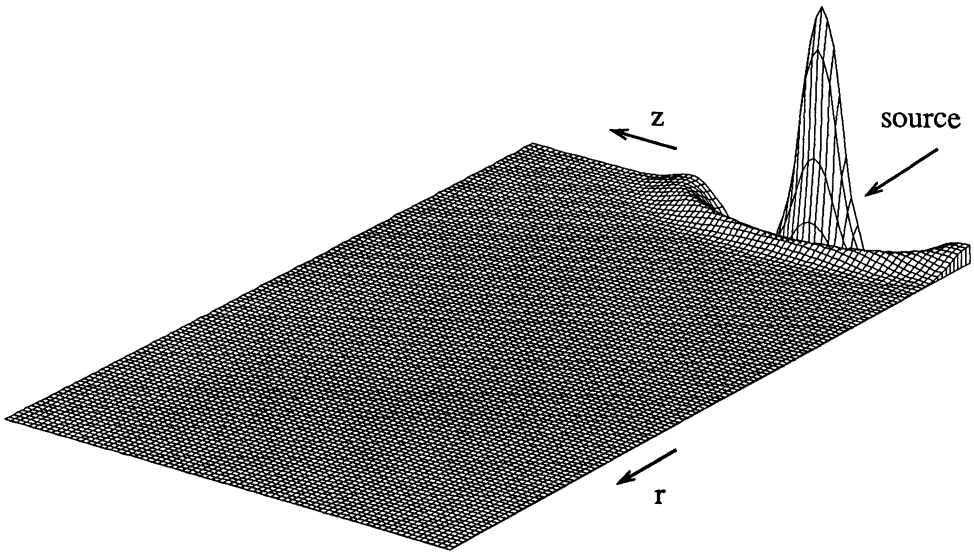


FIG. 12. *Propagation of acoustic waves without gravity at time* $t = 1$ *period* (100 *steps*).

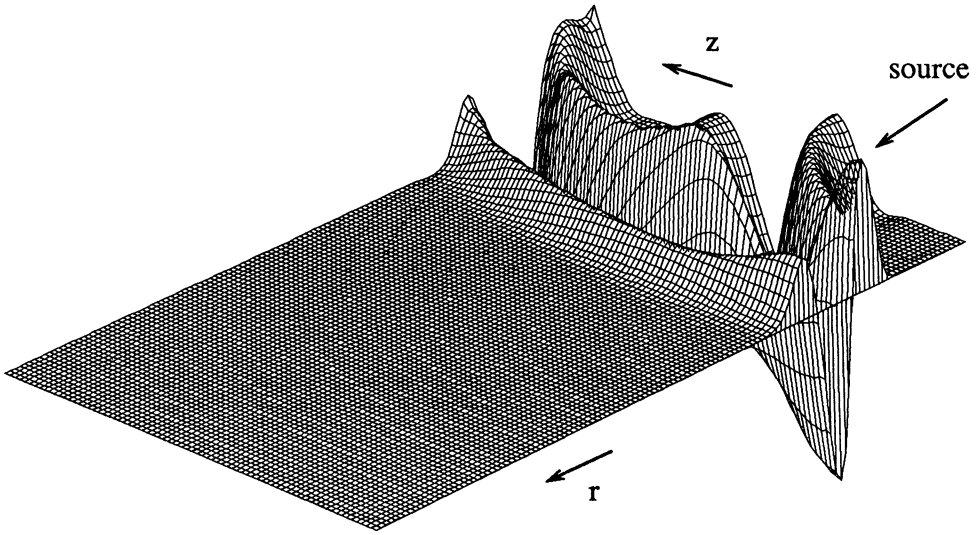


FIG. 13. *Propagation of acoustic waves without gravity at time* $t = 3$ *periods* (300 *steps*).

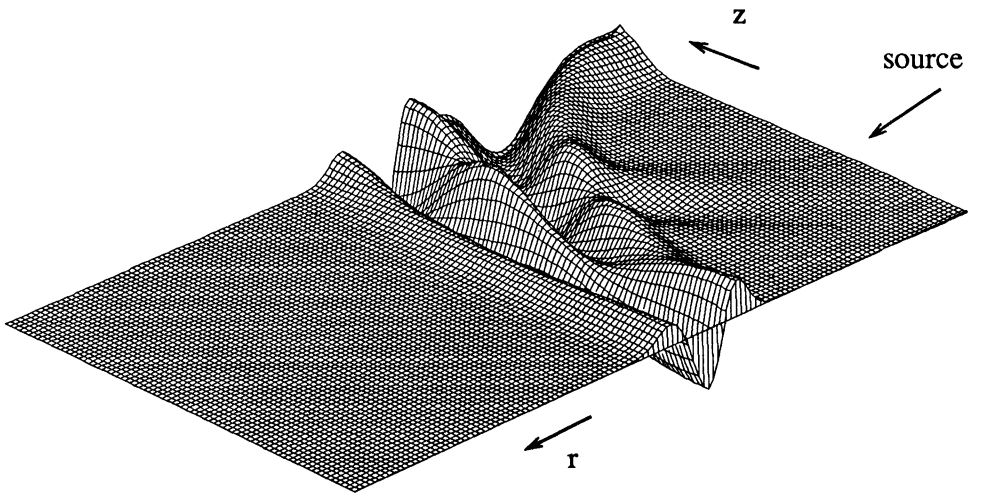


FIG. 14. *Propagation of acoustic waves without gravity at time* $t = 5$ *periods* (500 *steps*).

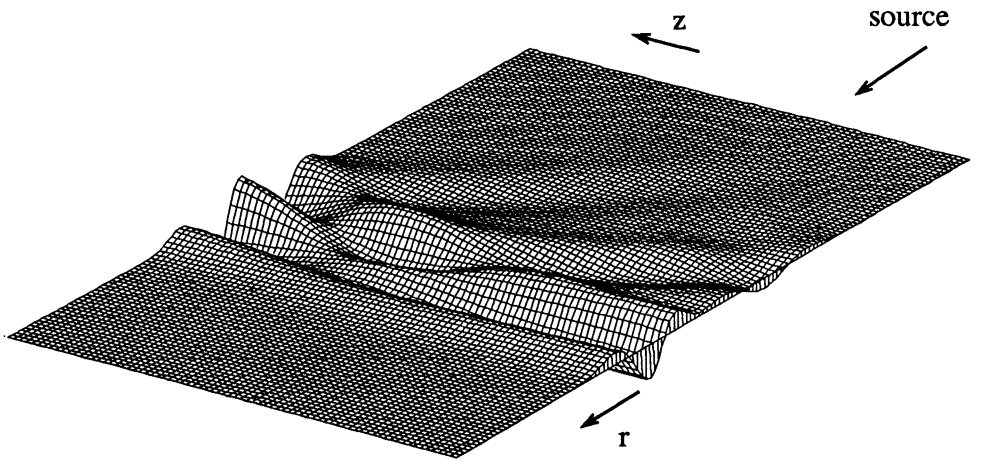


FIG. 15. *Propagation of acoustic waves without gravity at time* $t = 7$ *periods* (700 *steps*).

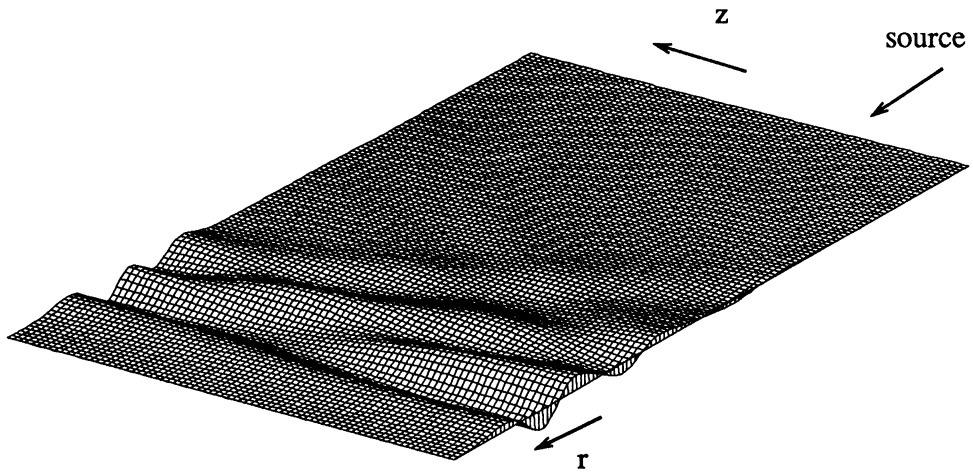


FIG. 16. *Propagation of acoustic waves without gravity at time* $t = 9$ *periods* (900 *steps*).

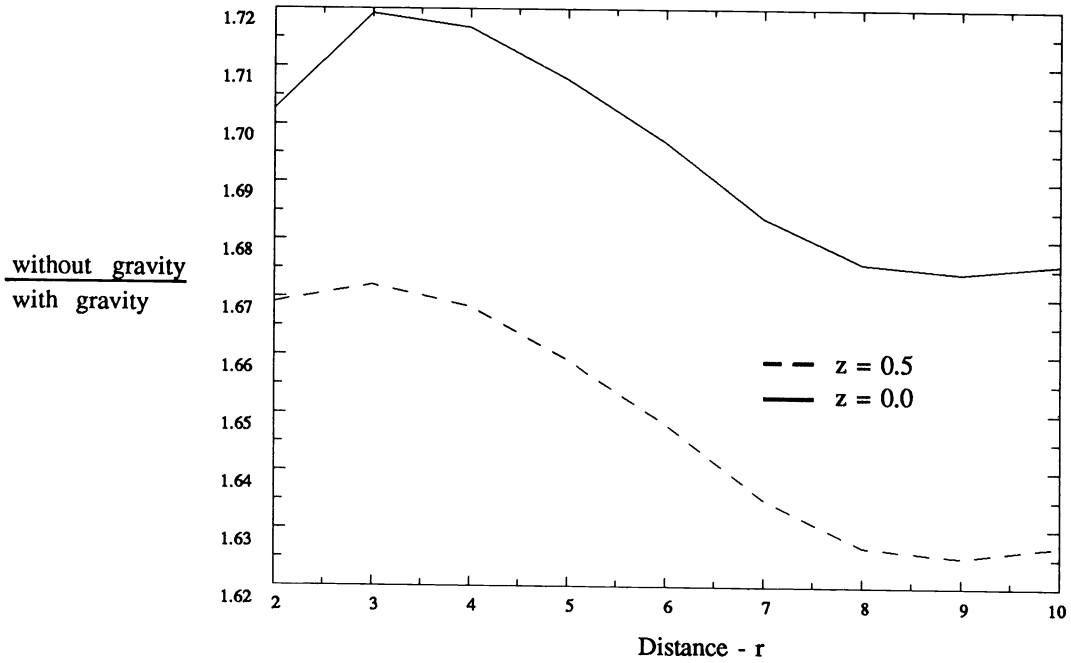


FIG. 17. *Ratio of maximum amplitude* (*without gravity*/*with gravity*).

r directions are 3 and 10, respectively. Since the lengths are nondimensionalized by the scale height, the distance in the r direction is roughly 100 miles. Thus, in the gravitational field, the sound tends to propagate longer distances, as suggested by our theory. Our computations are limited to shorter distances due to the cost in computations. The grid size was 75×250. To do more extensive calculations, such as longer ranges, we require a larger computational facility. The results reported here took roughly 32 hours of computations on a SUN Microsystem Computer (3/260—with floating point accelerator). It took 15 minutes on a CRAY—XMP-48. These considerations give a rough idea of the large computational requirements of the problem.

A more realistic situation that could be considered is to include temperature variation and an improved model of the atmosphere. A better atmospheric model validated with experiments is available (see [8]). We shall include this realistic model in a future work.

In conclusion, we have modeled the nonlinear isothermal atmospheric acoustics problem. A solution procedure consisting of linearizing the nonlinear problem into a sequence of linear problems and then combining the linear solutions to obtain the nonlinear solutions has been proposed and implemented. We have provided partial justification of this linearization principle by considering an appropriate one-dimensional model. Absorbing boundary conditions appropriate to this class of problems have been derived. The model consists of the effect of gravitational body force. We have calculated the nonlinear field with and without gravity. This result has great importance for the experimentalists in this field. As expected by the experimentalists, we have shown that the case with inclusion of gravity carries acoustic signals over a longer range than the case excluding gravity.

6. Appendix. Numerical schemes used in the computations. Here we briefly describe the key numerical methods that we used in our computations. The central method is the well-known Lax-Wendroff method. For one-dimensional conservative systems this method is commonly known. For details, we refer to [10]. For nonconservative systems, we are not aware of any extensions. Our extensions that we used for the computations are presented. Also, the multidimensional extensions that we used here, and are not available in the literature, are given here. We chose the Lax-Wendroff scheme for its effectiveness in treating boundary conditions that are second-order accurate both in space and time. Alternate methods such as spectral methods are under investigation. We begin with the one-dimensional scheme and the numerical evaluation of the Riemann variables used to treat the boundary conditions.

6.1. Scheme for linearized one-dimensional equation. In this case, (3.6)-(3.8) and (3.23)-(3.25) take the following form:

$$\mathbf{u}_t + \mathbf{F}_z = \mathbf{H}(\mathbf{u}, z).$$

In the interior we use the Lax-Wendroff scheme, which is

$$\mathbf{u}_{i+1}^{t+\Delta t/2} = \frac{1}{2}(\mathbf{u}_i^t + \mathbf{u}_{i+1}^t) - \frac{\Delta t}{2\Delta z}(\mathbf{F}(\mathbf{u}_{i+1}^t) - \mathbf{F}(\mathbf{u}_{i+1/2}^t)) + \frac{\Delta t}{2}\mathbf{H}\left(\frac{1}{2}(\mathbf{u}_{i+1}^t + \mathbf{u}_i^t), z_{i+1/2}\right),$$

$$i = 0, 1, \cdots N-1,$$

$$\mathbf{u}_i^{t+\Delta t} = \mathbf{u}_i^t - \frac{\Delta t}{\Delta z}\left(\mathbf{F}(\mathbf{u}_{i+1/2}^{t+\Delta t/2}) - \mathbf{F}(\mathbf{u}_{i-1/2}^{t+\Delta t/2})\right) + \Delta t\mathbf{H}\left(\frac{1}{2}(\mathbf{u}_{i+1/2}^{t+\Delta t/2} + \mathbf{u}_{i-1/2}^{t+\Delta t/2}), z_i\right),$$

$$i = 1, \cdots N-1.$$

At the boundary $z = L$, the Riemann variables satisfy the following conditions:

$$S_t = Q(\mathbf{u}), \qquad R_t + R_z = W(\mathbf{u}).$$

The corresponding discretizations for these equations (see [11]) are

$$S(\mathbf{u}_N^{t+\Delta t}) = S(\mathbf{u}_N^t) + S(\mathbf{u}_{N-1}^t) - S(\mathbf{u}_{N-1}^{t+\Delta t}) + 2\Delta t Q(\mathbf{u}_{N-1/2}^{t+\Delta t/2}),$$

$$\left(1 + \frac{\Delta t}{\Delta z}\right) R(\mathbf{u}_N^{t+\Delta t}) = R(\mathbf{u}_N^t) + R(\mathbf{u}_{N-1}^t) - R(\mathbf{u}_{N-1}^{t+\Delta t}) + 2\Delta t W(\mathbf{u}_{N-1/2}^{t+\Delta t/2})$$

$$- \frac{\Delta t}{\Delta z}(R(\mathbf{u}_N^t) - R(\mathbf{u}_{N-1}^t) - R(\mathbf{u}_{N-1}^{t+\Delta t})).$$

At $z=0$ we have $u=0$, and we use the equation satisfied by S to obtain the numerical boundary condition.

It should be pointed out that in the second-order nonlinear correction, the right-hand side of all equations contains the derivatives of the first-order problem. We evaluate these quantities using the following formulas:

$$u_t^t=(u^t-u^{t-\Delta t})\frac{1}{\Delta t}, \qquad u_t^{t+\Delta t/2}=(u^{t+\Delta t/2}-u^{t-\Delta t/2})\frac{1}{\Delta t},$$

$$u_{iz}^t=(u_{i+1}^t-u_i^t)\frac{1}{\Delta z}, \qquad u_{iz}^{t+\Delta t/2}=(u_{i+1/2}^{t+\Delta t/2}-u_{i-1/2}^{t+\Delta t/2})\frac{1}{\Delta z}.$$

6.2. Scheme for one-dimensional nonlinear equations. In this case, (4.1)-(4.3) take the form

$$\mathbf{u}_t+\mathbf{A}(\mathbf{u})\mathbf{F}_z=\mathbf{H}(\mathbf{u}).$$

As mentioned earlier, we modify the Lax-Wendroff scheme to obtain a numerical scheme. In the event that $\mathbf{A}$ is linear, the scheme coincides with the Lax-Wendroff scheme.

$$\bar{\mathbf{u}}_{i+1}^t=\tfrac{1}{2}(\mathbf{u}_{i+1}^t+\mathbf{u}_i^t), \qquad i=0,\cdots N-1,$$

$$\mathbf{u}_{i+1/2}^{t+\Delta t/2}=\bar{\mathbf{u}}_{i+1}^t-\mathbf{A}(\bar{\mathbf{u}}_{i+1}^t)\frac{\Delta t}{2\Delta z}(\mathbf{F}(\mathbf{u}_{i+1}^t)-\mathbf{F}(\mathbf{u}_i^t))+\frac{\Delta t}{2}\mathbf{H}(\bar{\mathbf{u}}_{i+1}^t, z_{i+1/2}), \qquad i=0,\cdots N-1,$$

$$\bar{\mathbf{u}}^{t+\Delta t/2}=\tfrac{1}{2}(\mathbf{u}_{i+1/2}^{t+\Delta t/2}+\mathbf{u}_{i-1/2}^{t+\Delta t/2}),$$

$$\mathbf{u}_i^{t+\Delta t}=\mathbf{u}_i^t+\mathbf{A}(\bar{\mathbf{u}}_i^{t+\Delta t/2})\frac{\Delta t}{\Delta z}(\mathbf{F}(\mathbf{u}_{i+1/2}^{t+\Delta t/2})-\mathbf{F}(\mathbf{u}_{i-1/2}^{t+\Delta t/2}))+\Delta t\mathbf{H}(\bar{\mathbf{u}}_i^{t+\Delta t/2}, z_i), \qquad i=1,\cdots N-1.$$

6.3. Scheme for linearized three-dimensional equation. In this case the equations have the form

$$\mathbf{u}_t+\mathbf{F}_z+\mathbf{G}_r=\mathbf{H}(\mathbf{u}, r).$$

Here again, we modify the Lax-Wendroff scheme. Our modification is similar to the Richtmyer scheme given in [10]. Moreover, the equations are in the cylindrical coordinate system. Thus, there is a singularity on the axis $r=0$. Our scheme and our treatment of the singularity are given below:

$$\bar{\mathbf{u}}_{i,j}^t=\tfrac{3}{4}\mathbf{u}_{i,j}^t+\tfrac{1}{16}(\mathbf{u}_{i+1,j}^t+\mathbf{u}_{i-1,j}^t+\mathbf{u}_{i,j+1}^t+\mathbf{u}_{i,j-1}^t),$$

$$\bar{\mathbf{u}}_{i,j}^{t+\Delta t}=\bar{\mathbf{u}}_{i,j}^t-\frac{\Delta t}{\Delta z}(\mathbf{F}(\mathbf{u}_{i+1,j}^t)-\mathbf{F}(\mathbf{u}_{i-1,j}^t))-\frac{\Delta t}{\Delta r}(\mathbf{G}(\mathbf{u}_{i,j+1}^t)-\mathbf{G}(\mathbf{u}_{i,j-1}^t))$$

$$+\mathbf{H}(\bar{\mathbf{u}}_{i,j}^t, r_j)\Delta t, \quad i=1,\cdots, N-1, \quad j=1,\cdots M-1,$$

$$\bar{\mathbf{u}}_{i,j}^{t+\Delta t/2}=\tfrac{1}{2}(\mathbf{u}_{i,j}^t+\bar{\mathbf{u}}_{i,j}^{t+\Delta t}),$$

$$\mathbf{u}_{i,j}^{t+\Delta t}=\bar{\mathbf{u}}_{i,j}^{t+\Delta t/2}-\frac{\Delta t}{2\Delta z}(\mathbf{F}(\bar{\mathbf{u}}_{i+1,j}^{t+\Delta t})-\mathbf{F}(\bar{\mathbf{u}}_{i-1,j}^{t+\Delta t}))-\frac{\Delta t}{2\Delta r}(\mathbf{G}(\bar{\mathbf{u}}_{i,j+1}^{t+\Delta t})-\mathbf{G}(\bar{\mathbf{u}}_{i,j-1}^{t+\Delta t}))$$

$$+\mathbf{H}(\bar{\mathbf{u}}_{i,j}^{t+\Delta t}, r_j)\frac{\Delta t}{2}, \quad i=1,\cdots N-1, \quad j=1,\cdots M-1.$$

At $r=0$ there is a singularity in the coordinate system. We evaluate the singular term using l'Hôpital's rule as indicated below:

$$\lim_{r\to 0}\frac{u}{r}=u_r.$$

Then the scheme is modified at the origin by the following scheme.

Predictor step.

$$\bar{\mathbf{u}}^t_{i,0}=\frac{3}{4}\mathbf{u}^t_{i,0}+\frac{1}{16}(\mathbf{u}^t_{i+1,0}+\mathbf{u}^t_{i-1,0}+2\mathbf{u}^t_{i,1}),\qquad u^t_{ri,0}=\frac{2}{\Delta r}u^t_{i,1},$$

$$\mathbf{u}^t_{zi,0}=\frac{0.7}{\Delta z}(\mathbf{u}^t_{i+1,0}-\mathbf{u}^t_{i-1,0})+\frac{0.3}{\Delta z}(\mathbf{u}^t_{i+1,1}-\mathbf{u}^t_{i-1,1}),\qquad \bar{\mathbf{u}}^{t+\Delta t}_{i,0}=\bar{\mathbf{u}}^t_{i,0}+\Delta t\mathbf{X}^t_{i,0}$$

where $\mathbf{X}^t_{i,0}=\mathbf{X}^t_{i,0}(\bar{\mathbf{u}}^t_{i,0},\mathbf{u}^t_{ri,0},\mathbf{u}^t_{zi,0})$.

Corrector step.

$$\bar{\mathbf{u}}^{t+\Delta t}_{i,0}=\frac{1}{2}(\mathbf{u}^t_{i,0}+\bar{\mathbf{u}}^{t+\Delta t}_{i,0}),\qquad \bar{\mathbf{u}}^{t+\Delta t}_{ri,0}=\frac{2}{\Delta r}\bar{\mathbf{u}}^{t+\Delta t}_{i,0},$$

$$\bar{\mathbf{u}}^{t+\Delta t}_{zi,0}=\frac{1}{\Delta z}[0.9(\bar{\mathbf{u}}^{t+\Delta t}_{i+1,0}-\bar{\mathbf{u}}^{t+\Delta t}_{i-1,0})+0.1(\bar{\mathbf{u}}^{t+\Delta t}_{i+1,1}-\bar{\mathbf{u}}^{t+\Delta t}_{i-1,1})],\qquad \mathbf{u}^{t+\Delta t}_{i,0}=\bar{\mathbf{u}}^{t+\Delta t}_{i,0}+\frac{\Delta t}{2}\bar{\mathbf{X}}^{t+\Delta t}_{i,0}$$

where $\bar{\mathbf{X}}^{t+\Delta t}_{i,0}=\bar{\mathbf{X}}^{t+\Delta t}_{i,0}(\bar{\mathbf{u}}^{t+\Delta t}_{i,0},\bar{\mathbf{u}}^{t+\Delta t}_{ri,0},\bar{\mathbf{u}}^{t+\Delta t}_{zi,0})$.

On the boundary $z=L$, we use the Riemann variables R and S together with the nonreflection condition.

$$R_t+R_z=H_1(\mathbf{u}_r,\mathbf{u}),\qquad S_t=H_2(\mathbf{u}_r,\mathbf{u}).$$

The scheme for these is as follows:

$$\bar{\mathbf{u}}^t_{N,j}=0.9\mathbf{u}^t_{N,j}+0.05(\mathbf{u}^t_{N,j-1}+\mathbf{u}^t_{N,j+1}),$$

$$\bar{\mathbf{u}}^t_{N-2,j}=0.9\mathbf{u}^t_{N-2,j}+0.05(\mathbf{u}^t_{N-2,j-1}+\mathbf{u}^t_{N-2,j+1}),$$

$$\bar{\mathbf{u}}^{t+\Delta t}_{N-2,j}=0.9\mathbf{u}^{t+\Delta t}_{N-2,j}+0.05(\mathbf{u}^t_{N-2,j-1}+\mathbf{u}^{t+\Delta t}_{N-2,j+1}),$$

$$\mathbf{u}^{t+\Delta t/2}_r=\frac{1}{2\Delta r}(\mathbf{u}^t_{N-1,j+1}-\mathbf{u}^t_{N-1,j-1})+\frac{1}{2\Delta r}(\mathbf{u}^{t+\Delta t}_{N-1,j+1}-\mathbf{u}^{t+\Delta t}_{N-1,j-1}),$$

$$\left(1+\frac{\Delta t}{\Delta z}\right)R(\mathbf{u}^{t+\Delta t}_{N,j})=R(\bar{\mathbf{u}}^t_{N,j})+R(\bar{\mathbf{u}}^t_{N-2,j})-R(\bar{\mathbf{u}}^{t+\Delta t}_{N-2,j})$$

$$+2\Delta tH_1(\mathbf{u}_r,\mathbf{u})-\frac{\Delta t}{\Delta z}(R(\bar{\mathbf{u}}^t_{N,j})-R(\bar{\mathbf{u}}^t_{N-2,j})-R(\bar{\mathbf{u}}^{t+\Delta t}_{N-2,j})),$$

$$S(\bar{\mathbf{u}}^{t+\Delta t}_{N,j})=S(\bar{\mathbf{u}}^t_{N,j})+S(\bar{\mathbf{u}}^t_{N-2,j})-S(\bar{\mathbf{u}}^{t+\Delta t}_{N-2,j})+2\Delta tH_2(\mathbf{u}^{t+\Delta t/2}_r,\mathbf{u}^{t+\Delta t/2}).$$

The treatment of boundary conditions at $r=R$ boundary is similar to that of $z=L$ above.

Acknowledgment. The authors wish to acknowledge the reviewers for valuable comments and criticisms which improved the style of the manuscript.

REFERENCES

[1] J. D. COLE AND C. GREIFINGER, *Acoustic gravity waves from an energy source at the ground in an isothermal atmosphere*, J. Geophys. Res., 74 (1969), pp. 3693-3703.

[2] A. D. PIERCE, *Propagation of acoustic gravity waves from a small source above the ground in an isothermal atmosphere*, J. Acoust. Soc. Amer., 35 (1963), pp. 1798–1807.

[3] S. I. HARIHARAN, *Nonlinear acoustic wave propagation in atmosphere*, Quart. Appl. Math., XLV (1987), pp. 735–748.

[4] ———, *A model problem for acoustic wave propagation in the atmosphere*, Proc. First IMACS Symposium on Computational Acoustics, D. Lee, R. L. Sternberg, and M. H. Schultz, eds., North-Holland, Amsterdam, Vol. 2, 1988, pp. 65–82.

[5] S. I. HARIHARAN AND P. K. DUTT, *Acoustic gravity waves: a computational approach*, J. Appl. Numer. Math., 4 (1988), pp. 491–506.

[6] A. BARRY, J. BIELAK, AND R. C. MACCAMY, *On absorbing boundary conditions wave propagation problems*, J. Comput. Phys., in press.

[7] S. I. HARIHARAN AND R. C. MACCAMY, *Integral equation procedures for eddy current problems*, J. Comput. Phys., 45 (1982), pp. 80–99.

[8] P. H. ROGERS AND J. H. GARDNER, *Propagation of sonic booms in the thermosphere*, J. Acoust. Soc. Amer., 67 (1980), pp. 78–91.

[9] K. W. THOMPSON, *Time dependent boundary conditions for nonlinear hyperbolic systems*, J. Comput. Phys., 68 (1987), pp. 1–24.

[10] R. D. RICHTMYER AND K. W. MORTON, *Difference Methods for Initial Value Problems*, John Wiley, New York, 1967.

[11] T. HAGSTROM AND S. I. HARIHARAN, *Accurate boundary conditions for exterior problems in gas dynamics*, ICOMP Report 88-2, 1988; Math. Comp., in press.

[12] B. ENGQUIST AND A. MAJDA, *Radiation boundary conditions for acoustic and elastic wave calculations*, Comm. Pure Appl. Math., XXXII (1979), pp. 313–357.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 515-532, May 1989
© 1989 Society for Industrial and Applied Mathematics
007

COMPUTATIONAL BEHAVIOR OF GAUSS-NEWTON METHODS*

C. FRALEY†

Abstract. This paper is concerned with the numerical behavior of Gauss-Newton methods for nonlinear least-squares problems. It is well known that Gauss-Newton methods often cannot be applied successfully without modification. However, no a priori characterization has been given of those problems on which a particular Gauss-Newton method or class of Gauss-Newton methods will or will not work well. The present paper gives some insight into why it is difficult to do so.

Key words. mathematical programming, nonlinear least squares, Gauss-Newton methods, regression, data fitting

AMS(MOS) subject classifications. 62J02, 65D10, 65H10, 65K05, 90C30

1. Introduction.

The nonlinear least-squares problem is given by

$$\min_{x\in\Re^n} \frac{1}{2}\sum_{i=1}^{m} \phi_i^2(x),$$

where $\phi_i(x)$ are real-valued functions, or, equivalently,

$$\min_{x\in\Re^n} \tfrac{1}{2}\|f(x)\|_2^2,$$

where

$$f(x)=\begin{pmatrix}\phi_1(x)\\ \phi_2(x)\\ \vdots\\ \phi_m(x)\end{pmatrix}.$$

We assume that each ϕ_i has continuous second partial derivatives. The function $\frac{1}{2}\|f(x)\|_2^2$ will be called the nonlinear *least-squares objective* function.

The classical approach to nonlinear least squares, called the *Gauss-Newton* method, locally approximates each residual component ϕ_i of f by a linear function, using the relationship

$$f(x+p)\approx f(x)+J(x)p+\mathcal{O}(\|p\|^2), \tag{1.1}$$

where J is the Jacobian matrix of f, that is,

$$J(x)\equiv\nabla f(x)=\begin{pmatrix}\dfrac{\partial\phi_1}{\partial x_1} & \cdots & \dfrac{\partial\phi_1}{\partial x_n}\\ \vdots & \ddots & \vdots\\ \dfrac{\partial\phi_m}{\partial x_1} & \cdots & \dfrac{\partial\phi_m}{\partial x_n}\end{pmatrix}.$$

* Received by the editors August 10, 1987; accepted for publication (in revised form) July 8, 1988. This research was supported by a fellowship from the Xerox Corporation, with additional funding provided by Joseph Oliger under Office of Naval Research contract N00014-82-K-0335, the Systems Optimization Laboratory under National Science Foundation grant CCR-8413211, and Jean-Philippe Vial under Fonds National de la Recherche Scientifique Suisse research grant 1 467-0.86.

† Computer Science Department, Stanford University, Stanford, California 94305. Present address, Université de Genève, COMIN, 1211 Genève 4, Switzerland.

The step to the new iterate from the current point is in the direction of the vector p that solves

$$\min_{p \in \Re^n} \|f + Jp\|_2^2. \tag{1.2}$$

In other words, the change in the nonlinear least-squares objective $\frac{1}{2} f^T f$ is being modeled by the quadratic function

$$g^T p + \tfrac{1}{2} p^T J^T J p, \tag{1.3}$$

where

$$g \equiv \nabla(\tfrac{1}{2} f^T f) = J^T f.$$

Hence the Gauss-Newton method differs from Newton's method in that the Hessian matrix

$$\nabla^2(\tfrac{1}{2} f^T f) = J^T J + \sum_{i=1}^{m} \phi_i \nabla^2 \phi_i$$

is approximated by $J^T J$, a strategy that would seem reasonable when the residuals are small. If x is the current iterate, and p_{GN} a minimizer of (1.3), then the new iterate is $x + \alpha p_{GN}$, where $\alpha > 0$ is a steplength computed by a linesearch procedure relative to the nonlinear least-squares objective.

More recently, the term "Gauss-Newton" has been used for methods that can be applied to a wider class of minimization problems

$$\min_{x \in \Re^n} \rho(f(x)),$$

in which $\rho(f(x))$ is some measure of the size of the vector $f(x)$. Candidates for ρ include, for example, loss functions for computing robust estimates in statistics (see, e.g., Dennis and Welsch [1978] and Watson [1987]). What these methods have in common is that they compute search directions that minimize a quadratic model based on the linearization (1.1). Terms involving the second derivatives of $f(x)$ are ignored in the approximate Hessian matrix. Since it is possible to vary ρ, the merit function, the linesearch procedure, and the method of minimizing the quadratic model in the event of ill-conditioning, Gauss-Newton methods constitute a rather large class of numerical methods. Performance is usually not uniform among Gauss-Newton methods for a given set of residuals and a given starting value, but it is not known—even for specific classes of problems—how to systematically select in advance from among the many variations in order to achieve favorable behavior.

A well-known characteristic of Gauss-Newton methods of all types is their lack of reliability. Typically, a Gauss-Newton method either performs very well, or else it performs very poorly on a particular problem relative to other methods. Moreover, it is not possible to make generalizations about the behavior of any particular Gauss-Newton method or class of Gauss-Newton methods when considering large and varied sets of test problems. The present paper is a detailed numerical study of a class of Gauss-Newton methods that gives insight into why it is difficult to give an a priori characterization of those problems on which the methods will (or will not) work well.

1.1. An overview. This paper is organized as follows. In § 2, we discuss the class of numerical methods for the nonlinear least-squares problem that is defined by the linearization (1.1) of f, and review their motivation from considerations that arise in

unconstrained optimization (see, e.g., Fletcher [1980], Gill, Murray, and Wright [1981], Dennis and Schnabel [1983], or Moré and Sorensen [1984]) and linear least squares (see, e.g., Stewart [1973], Lawson and Hanson [1974], or Golub and Van Loan [1983]). Section 2 also surveys research related to computational aspects of Gauss-Newton methods. In § 3, we give information on how the numerical results for §§ 4 and 5 were obtained. Examples of the performance of some Gauss-Newton methods on problems with ill-conditioned Jacobians are presented in § 4. An example of poor performance of a Gauss-Newton method on a zero-residual problem with a well-conditioned Jacobian is analyzed in § 5.

1.2. Notation. In general subscripts on a function mean that the function is evaluated at the corresponding subscripted variable (for example, $f_k = f(x_k)$). An exception is made for the residual functions ϕ_i, where the subscript is the component index within the vector f.

2. Gauss–Newton methods for nonlinear least squares. As a model for the change in the least-squares objective, (1.3) has the advantage that it involves only first derivatives of the residuals, and that J^TJ is always at least positive semidefinite. For

$$p_k^{\mathrm{GN}} \in \arg\min_{p \in \Re^n} g_k^T p + \tfrac{1}{2} p^T J_k^T J_k p,$$

the sequence $\{x_k + p_k^{\mathrm{GN}}\}$ is locally quadratically convergent to x^* whenever $f(x^*) = 0$ and $J(x^*)^T J(x^*)$ is positive definite, because $J_k^T J_k$ is an $\mathcal{O}(\|x_k - x^*\|)$ approximation to the full Hessian at x^*.

A difficulty with the Gauss-Newton method arises when J^TJ is singular, or, equivalently, when J has linearly dependent columns, because then (1.3) does not have a unique minimizer. The set of vectors that minimize (1.3) is the same as the set of solutions to the linear least-squares problem (1.2). One (theoretically) well-defined alternative that is often approximated computationally is to require the unique solution of minimum l_2 norm:

$$\min_{p \in \mathcal{S}} \|p\|_2, \tag{2.1}$$

where $\mathcal{S}$ is the set of solutions to (1.2). Another possibility is to replace J in (1.2) by a maximal linearly independent subset of its columns. In finite-precision arithmetic, there is often some ambiguity about how to formulate and solve an alternative to (1.2) when the columns of J are "nearly" linearly dependent, because of the necessity of estimating the rank of J. Although it can sometimes be accomplished in specific instances, there is no automatic procedure (such as scaling) by which problems due to ill-conditioning can be avoided. The references cited above for linear least squares discuss at length the difficulties inherent in computing approximate solutions to ill-conditioned least-squares problems. We define the class of Gauss-Newton methods to include all linesearch methods in which the search direction is the result of some well-defined computational method for solving (1.2).

Most often in Gauss-Newton methods the nonlinear least-squares objective is used as a merit function for the linesearch. The vector p_k^{GN} satisfies the equations

$$J_k^T J_k p = -g_k, \tag{2.2}$$

and is therefore a direction of descent for f^Tf at x_k whenever $g_k \neq 0$. To guarantee convergence to a local minimium, the sequence of search directions must also be

bounded away from orthogonality to the gradient of the merit function, a condition that may not be met by successive Gauss-Newton directions relative to $f^T f$ unless the eigenvalues of $J_k^T J_k$ are bounded away from zero. Powell [1970] gives an example of convergence of a Gauss-Newton method with exact linesearch to a nonstationary point in which the search direction becomes orthogonal to a nonzero gradient.

Deuflhard and Apostolescu [1980] suggest selecting a steplength for the Gauss-Newton direction based on decreasing the merit function $\|J_k^\dagger f(x)\|_2^2$ rather than $\|f(x)\|_2^2$ for a class of nonlinear least-squares problems that includes zero-residual problems. The function $J_k^\dagger$ is the *pseudoinverse* of J_k (see, e.g., Golub and Van Loan [1983, Chap. 6]), and $J_k^\dagger f_k$ is the minimum l_2-norm solution to $\|J_k p + f_k\|_2$. Deuflhard and Apostolescu reason that the Gauss-Newton direction is the steepest-descent direction for the function $\|J_k^\dagger f(x)\|_2^2$, so that the geometry of the level surfaces defined by $\|J_k^\dagger f(x)\|_2^2$ is more favorable to avoiding small steps in the linesearch. A shortcoming of this approach (pointed out by Deuflhard and Apostolescu) is that there are no global convergence results. The merit function depends on x_k, so that a different function is being reduced at each step. Another difficulty is that, although the authors state that numerical experience supports selection of a steplength based on $\|J_k^\dagger f(x)\|_2^2$ for ill-conditioned problems, the transformation $J_k^\dagger$ is not numerically well defined under these circumstances. Therefore neither the Gauss-Newton search direction nor the merit function is numerically well defined when the columns of J_k are nearly linearly dependent. Since it is not known how to improve Gauss-Newton methods for general problems through the selection of merit functions for the linesearch, we shall henceforth make the conventional assumption that the linesearch is performed relative to the nonlinear least-squares objective.

There is another reason why it is difficult to say precisely what is meant by a "Gauss-Newton method" for a particular nonlinear least-squares problem. To see this, let $Q(x)$ be an $l \times m$ orthogonal matrix function on $\Re^n$, that is, $Q(x)^T Q(x) = I$ for all x. Then $\|Q(x)f(x)\|_2^2 = \|f(x)\|_2^2$ for all x, and consequently the function $\tilde{f} \equiv Qf$ defines the same nonlinear least-squares problem as f. The Jacobian matrix of $\tilde{f}$ is $\tilde{J} \equiv QJ + (\nabla Q)f$, so that a minimizer of $\|\tilde{J}p + \tilde{f}\|_2$ will ordinarily be different from a minimizer of $\|Jp + f\|_2$, unless $Q(x)$ happens to be a constant transformation. However, if both Q and f have k continuous derivatives, then $\nabla^i \|Q(x)f(x)\|_2^2 = \nabla^i \|f(x)\|_2^2$, for $i = 1, 2, \cdots, k$. Letting $W \equiv (\nabla Q)f$, so that $\tilde{J} = QJ + W$, we have

$$\tilde{J}^T\tilde{J} = J^T J + (J^T Q^T W + W^T Q J) + W^T W,$$

showing that the Gauss-Newton approximation $J^T J$ to the full Hessian matrix is changed when f is transformed by an orthogonal function that varies with x. Thus, there are many Gauss-Newton methods corresponding to a given vector function, although Newton's method remains invariant (see also Nocedal and Overton [1985, p. 826]). In fact, each step of a Gauss-Newton method could be defined by a different transformation of f. Moreover, the conditioning of $\tilde{J}$ may be very different from that of J, so that, for example, the columns of $\tilde{J}$ might be strongly independent, while J is nearly rank deficient. Since the number of rows in Q may be greater than n, it is also possible to imbed the given nonlinear least-squares problem in a larger one.

In practice, Gauss-Newton methods are often among the best available techniques for nonlinear least squares—especially for zero-residual problems—but there are many cases in which they fail or are inefficient (see, e.g., Fraley [1988b]). In fact, most algorithms for nonlinear least-squares problems aim to exploit favorable behavior in

Gauss-Newton methods whenever possible. This aim leaves two issues to be resolved: when to depart from the Gauss-Newton framework, and what to substitute for it.

A large number of alternative strategies has been proposed for the solution of nonlinear least-squares problems (for a recent survey, see Fraley [1988a]). Currently, there are three principal approaches to nonlinear least squares for which software is widely available and has been extensively tested. Each of these methods alters the Gauss-Newton search direction in a different way. Levenberg-Marquardt methods minimize the Gauss-Newton quadratic model subject to a trust-region constraint (see, e.g., Moré [1978]). They replace J^TJ with $J^TJ+\lambda D^TD$ in the quadratic model function, D diagonal and $\lambda \geqq 0$, so that the minimizer may differ from the Gauss-Newton search direction in the range of J. The software package MINPACK (Moré, Garbow, and Hillstrom [1980]) is an implementation of a Levenberg-Marquardt method. Corrected Gauss-Newton methods (Gill and Murray [1976], [1978]) compute a Gauss-Newton direction in a subspace of the range of J^T, and obtain a component in the corresponding null space by a projected Newton method. Versions of this type of algorithm are implemented in the NAG [1984] library nonlinear least-squares software. Special quasi-Newton methods form quasi-Newton approximations to the term $\sum_{i=1}^{m} \phi_i \nabla^2 \phi_i$ in the nonlinear least-squares Hessian, so that the search direction differs from the Gauss-Newton direction in the range of J^T, and usually has a component in the null space of J as well when J is rank deficient. A strategy of this sort is adaptively combined with a Gauss-Newton method and a Levenberg-Marquardt method in the computer algorithm NL2SOL (Dennis, Gay, and Welsch [1981a], [1981b]). Although the techniques mentioned above are more robust than Gauss-Newton methods, they do not perform equally well on all problems, and the question of which method is the "best" for general nonlinear least-squares problems, or even for a given class of problems, is still open (see Fraley [1988b]).

The issue of when to abandon Gauss-Newton has not received adequate treatment in the literature. Because the circumstances under which Gauss-Newton should be abandoned are left vague, and because few detailed numerical studies are available, it is easy to get the impression that the matter is either unimportant or has been well understood for some time. Gauss-Newton methods are not quadratically convergent on large-residual problems, and several hybrid methods use an alternative search direction when the iterates are judged to be close to the solution (e.g., Betts [1976], Fletcher and Xu [1986], and Wedin and Lindström [1987]). A drawback of these approaches is that the decision to make the switch is based on asymptotic relationships for which approximate conditions are not sufficient to guarantee nearness to the solution; often these conditions are merely indicators of inefficiency in the algorithm. The criteria used by Wedin and Lindström have an additional source of uncertainty in that they depend on a rank estimate for the Jacobian. Whether or not the parameters defining the critical conditions can be chosen in such a way as to be suitable over a wide range of problems in their intended domain has yet to be demonstrated. It is largely accepted that Gauss-Newton is to be avoided for problems with nearly rank-deficient Jacobians, but that otherwise Gauss-Newton methods perform well except in the neighborhood of the solution of large-residual problems. The purpose of the present study is to illustrate the difficulties involved in saying precisely when a Gauss-Newton method should or should not be used. In § 4, we show that rapidly convergent Gauss-Newton methods may exist for problems with ill-conditioned Jacobians, but it appears that different rules must be applied to different problems in order to observe good performance. In § 5, we give an example of poor performance of a Gauss-Newton method on a zero-residual problem with well-conditioned Jacobian.

3. Numerical results: methods and presentation. At each iteration in the numerical examples, the LINPACK subroutine DSVDC (Dongarra et al. [1979]) is used to compute the singular value decomposition (SVD) of the Jacobian:

$$J = \begin{cases} U(S \quad 0)V^T & \text{if } m < n, \\ USV^T & \text{if } m = n, \\ U\begin{pmatrix} S \\ 0 \end{pmatrix} V^T & \text{if } m > n \end{cases}$$

where S is diagonal with nonnegative diagonal entries $\sigma_1 \geqq \sigma_2 \geqq \cdots \geqq \sigma_{\min(m,n)}$, and U and V are orthogonal. Let

$$r_{\max} \equiv \max\{i \mid \sigma_i \neq 0\},$$

and

$$p_i \equiv \sum_{j=1}^{i} \tau_j v_j, \quad \tau_j \equiv -\frac{u_j^T f}{\sigma_j}, \quad i = 1, 2, \cdots, r_{\max}, \tag{3.1}$$

where u_j, v_j are the jth columns of U and V, respectively. If the computations were exact, then $r_{\max}$ would be the rank of J, the vector $p_{r_{\max}}$ would be the solution to (2.1), and the columns of V would form an orthonormal basis for $\mathfrak{R}^n$. The quantities τ_j, $j = 1, 2, \cdots, i$, are the components of p_i in terms of this basis, with

$$\|p_i\|_2^2 = \sum_{j=1}^{i} \tau_j^2.$$

When $i < \min\{m, n\}$, p_i has no component in the space spanned by $\{v_{i+1}, v_{i+2}, \cdots, v_{\min\{m,n\}}\}$. Each p_i is either a descent direction for the nonlinear least-squares objective, or it is orthogonal to the gradient $g \equiv J^T f$ (see, e.g., Fraley [1987, Chap. 4]).

In practice the SVD cannot be computed exactly, and the solution to the linear least-squares subproblem (2.1) is taken to be p_r, where $r \leqq r_{\max}$ is an estimate of the rank of J. We define

$$rank(J) \equiv \max\{i \mid \sigma_i > \varepsilon(1 + \sigma_1)\}, \tag{3.2}$$

where ε is a prescribed tolerance. This criterion depends on J only and does not take into account the size of the search direction p, the angle between p and the gradient, or the vector f. In the example of § 5, rank estimation is not an issue because the Jacobian is well conditioned.

The linesearch for the Gauss-Newton examples in § 4, as well as for the results in § 5, is taken from the nonlinear programming package NPSOL (Gill et al. [1979], [1986b]), and requires both function and gradient evaluations. The quantity α_k that appears in the tables is the steplength calculated by this linesearch procedure.

The Gauss-Newton methods in § 4 are compared to results obtained, using the following widely distributed software:

- NPSOL—A linesearch method for unconstrained optimization that uses a quasi-Newton approximation to the Hessian matrix (from the Systems Optimization Laboratory, Stanford University (see Gill et al. [1986]); also available in the Numerical Algorithms Group (NAG) Library.
- LMDER—A Levenberg-Marquardt method for nonlinear least squares (from MINPACK (Moré, Garbow, and Hillstrom [1980]); the algorithm is described in Moré [1978]).

- DN2G/NL2SOL—An adaptive method for nonlinear least squares that combines a special quasi-Newton method with a Gauss-Newton method and a Levenberg-Marquardt method (from the PORT Library [1984]; the algorithm is described in Dennis, Gay, and Welsch [1981a], [1981b], [1983]).

Two sets of data are given for each alternative method on each example, corresponding to two different sets of values for parameters in the termination criteria. These data also appear in Chapters 2 and 5 of Fraley [1987], where details about the choices made for the parameter values can be found.

The following key words, listed under the label "conv." in the tables, are used to describe abnormal termination conditions:

F LIM.—function evaluation limit reached,
LOOP—subroutine appears to loop,
TIME—time limit exceeded.

In the tables, under the label "est. err.," we include the quantity

$$\frac{\|f^*\|_2^2 - \|f_{\text{best}}\|_2^2}{1 + \|f_{\text{best}}\|_2^2},$$

where f^* is the value of f at the point of termination, and $\|f_{\text{best}}\|_2$ is the best available estimate of the norm of the solution, in order to get some idea of the error in $\|f^*\|_2$. For those problems that have nonzero residuals, the value of $\|f_{\text{best}}\|_2$ is given to six figures of accuracy, rounded down.

We use the notation $cond(J)$ for the condition number of the Jacobian (the ratio of the largest singular value to the smallest singular value; see, e.g., Golub and Van Loan [1983]).

All of the programs for the examples were written in FORTRAN double precision and executed on the IBM 3081 and IBM 3033 computers at Stanford Linear Accelerator Center, for which the relative machine precision is $2.22\cdots\times 10^{-16}$.

4. Performance on problems with ill-conditioned Jacobians.

4.1. Chebyquad $n = m = 8$. The first example is related to the problem of locating nodes for Chebyshev quadrature (Fletcher [1963], Moré, Garbow, and Hillstrom [1981], Fraley [1988b, Problem 35a]) and demonstrates that the choice of ε in (3.2) can be critical (see Table 1). The algorithm succeeds in finding an approximate minimum when $\varepsilon = 10^{-14}$, although it fails when $\varepsilon = 10^{-15}$. The problem is rather easily solved by the alternative methods, as shown in Table 2. Tables 3 and 4 trace the progress of the Gauss-Newton methods for $\varepsilon = 10^{-14}$ and $\varepsilon = 10^{-15}$, respectively.

TABLE 1
Gauss-Newton.

	$\varepsilon = 10^{-14}$	$\varepsilon \leqq 10^{-15}$
f, J evals.	147	124
iters.	44	19
$\|x^*\|_2$	1.65	1.63
$\|f^*\|_2$	10^{-2}	10^{-1}
$\|g^*\|_2$	10^{-11}	10^{-1}
est. err.	10^{-9}	10^{-2}

TABLE 2

	f evals.	J evals.	iters.	$\|x^*\|_2$	$\|f^*\|_2$	$\|g^*\|_2$	est. err.
NPSOL	33	33	19	1.65	10^{-1}	10^{-5}	10^{-9}
	35	35	21	1.65	10^{-1}	10^{-7}	10^{-9}
LMDER	40	21	21	1.65	10^{-1}	10^{-5}	10^{-9}
	53	27	27	1.65	10^{-1}	10^{-7}	10^{-9}
DN2G	23	15	14	1.65	10^{-1}	10^{-7}	10^{-9}
	24	16	15	1.65	10^{-1}	10^{-8}	10^{-9}

TABLE 3
Gauss-Newton on Chebyquad ($n = 8$).

$\varepsilon = 10^{-14}$

k	f, J evals.	$\|x_k\|_2$	$\|f_k\|_2$	$\|g_k\|_2$	$\|p_k\|_2$	$g_k^T p_k$	α_k	cond J_k	rank J_k
0	8	2.E+00	2.E−01	8.E−01	2.E+00	−4.E−02	7.3E−02	10^2	8
5	37	2.E+00	1.E−01	3.E−01	2.E−01	−1.E−02	2.2E−01	10^1	8
10	69	2.E+00	1.E−01	2.E−01	1.E+04	−9.E−03	5.1E−11	10^6	8
15	104	2.E+00	1.E−01	2.E−01	1.E+09	−9.E−03	4.7E−21	10^{11}	8
16	111	2.E+00	1.E−01	2.E−01	1.E+10	−9.E−03	4.7E−23	10^{12}	8
17	118	2.E+00	1.E−01	2.E−01	1.E+11	−9.E−03	4.7E−25	10^{13}	8
18	120	2.E+00	1.E−01	2.E−01	8.E−02	−4.E−03	5.7E−01	10^{14}	7
19	123	2.E+00	8.E−02	2.E−01	8.E−03	−8.E−04	2.1E+00	10^{14}	7
20	124	2.E+00	6.E−02	7.E−02	9.E−03	−5.E−04	1.0E+00	10^{14}	7
25	129	2.E+00	6.E−02	8.E−04	6.E−05	−4.E−08	1.0E+00	10^{14}	7
30	134	2.E+00	6.E−02	8.E−06	6.E−07	−4.E−12	1.0E+00	10^{14}	7
31	135	2.E+00	6.E−02	3.E−06	2.E−07	−6.E−13	1.0E+00	10^{14}	7
35	139	2.E+00	6.E−02	8.E−08	6.E−09	−3.E−16	1.0E+00	10^{14}	7
40	144	2.E+00	6.E−02	7.E−10	5.E−11	−3.E−20	1.0E+00	10^{14}	7
41	145	2.E+00	6.E−02	3.E−10	2.E−11	−5.E−21	1.0E+00	10^{14}	7
42	146	2.E+00	6.E−02	1.E−10	8.E−12	−7.E−22	1.0E+00	10^{14}	7
43	147	2.E+00	6.E−02	4.E−11	3.E−12	−1.E−22	1.0E+00	10^{14}	7
		2.E+00	6.E−02	2.E−11					

TABLE 4

$\varepsilon = 10^{-15}$

k	f, J evals.	$\|x_k\|_2$	$\|f_k\|_2$	$\|g_k\|_2$	$\|p_k\|_2$	$g_k^T p_k$	α_k	cond J_k	rank J_k
0	8	2.E+00	2.E−01	8.E−01	2.E+00	−4.E−02	7.3E−02	10^2	8
5	37	2.E+00	1.E−01	3.E−01	2.E−01	−1.E−02	2.2E−01	10^1	8
10	69	2.E+00	1.E−01	2.E−01	1.E+04	−9.E−03	5.1E−11	10^6	8
15	104	2.E+00	1.E−01	2.E−01	1.E+09	−9.E−03	4.7E−21	10^{11}	8
16	111	2.E+00	1.E−01	2.E−01	1.E+10	−9.E−03	4.7E−23	10^{12}	8
17	118	2.E+00	1.E−01	2.E−01	1.E+11	−9.E−03	4.7E−25	10^{13}	8
18	124	2.E+00	1.E−01	2.E−01	1.E+12	−9.E−03	0.0E−01	10^{14}	8

Until iteration 18, the Jacobian has full column rank at each step according to (3.2), and it becomes increasingly ill conditioned as the computation proceeds. The search direction grows very large and approaches orthogonality to the gradient, while the steplength decreases. No significant decrease is observed in either $\|f\|_2$ or $\|g\|_2$ in iterations 6–17. At iteration 18, the two Gauss–Newton methods differ. For $\varepsilon = 10^{-14}$, the estimated rank of the Jacobian is reduced to 7, and a significant decrease in the function is achieved. For $\varepsilon \leqq 10^{-15}$, by (3.2) the Jacobian still has full column rank, and the algorithm terminates because $\alpha_k p_k$ is judged to be negligible relative to $\|x_k\|_2$. Detailed information at the start of iteration 18 for the Gauss–Newton methods is given in Table 5. It seems reasonable to say that $rank(J) = 7$ rather than $rank(J) = 8$ at this point, because $\sigma_8 \ll \sigma_7$, $\|p_8\|_2 \gg \|p_7\|_2$, and $|\cos(g, p_8)| \ll |\cos(g, p_7)|$. Hence it is not surprising that it is the method with $\varepsilon = 10^{-14}$, rather than the one with $\varepsilon = 10^{-15}$, that ultimately makes good progress toward the solution.

TABLE 5

$\varepsilon \leqq 10^{-14}$; iteration 18

r	σ_r	$\lvert\tau_r\rvert$	$\|p_r\|_2$	$\lvert g^T p_r\rvert$	$\lvert\cos(g, p_r)\rvert$
1	10^{1}	10^{-3}	10^{-3}	10^{-4}	10^{0}
2	10^{1}	10^{-16}	10^{-3}	10^{-4}	10^{0}
3	10^{0}	10^{-16}	10^{-3}	10^{-4}	10^{0}
4	10^{0}	10^{-2}	10^{-2}	10^{-3}	10^{0}
5	10^{0}	10^{-15}	10^{-2}	10^{-3}	10^{0}
6	10^{0}	10^{-1}	10^{-1}	10^{-3}	10^{-1}
7	10^{0}	10^{-14}	10^{-1}	10^{-3}	10^{-1}
8	10^{-13}	10^{12}	10^{12}	10^{-2}	10^{-13}

The behavior of the Gauss–Newton methods can be explained by comparing the sequence $\{p_k^*\}$ of steps from the iterates to the minimum of the function, to the sequence $\{p_k\}$ of Gauss–Newton steps. The magnitudes of the components of these vectors in terms of the basis $\{v_j(x_k)\}$, for iterations 6–18, are listed in Tables 6 and 7. The step p_k^* to the minimum approaches orthogonality to $v_8(x_k)$, while the Gauss–Newton search direction becomes dominated by the component in the direction of $v_8(x_k)$ due to the ill-conditioning in the Jacobian. By iteration 18, the vector p_k is almost orthogonal to p_k^*.

This example shows that it may be advantageous to reduce the effective rank in a Gauss–Newton method. But the question of exactly when to reduce the rank is a difficult one whose answer can significantly affect performance. If full column rank is assumed until the search direction becomes numerically orthogonal to the gradient,

TABLE 6

Components $\{\tau_j^*(x_k)\}$ *of* $p_k^* = x^* - x_k$ *in terms of* $\{v_j(x_k)\}$.

k	$\lvert\tau_1^*\rvert$	$\lvert\tau_2^*\rvert$	$\lvert\tau_3^*\rvert$	$\lvert\tau_4^*\rvert$	$\lvert\tau_5^*\rvert$	$\lvert\tau_6^*\rvert$	$\lvert\tau_7^*\rvert$	$\lvert\tau_8^*\rvert$
6	10^{-2}	10^{-9}	10^{-8}	10^{-2}	10^{-9}	10^{-2}	10^{-9}	10^{-3}
9	10^{-2}	10^{-9}	10^{-8}	10^{-2}	10^{-9}	10^{-2}	10^{-9}	10^{-6}
12	10^{-2}	10^{-9}	10^{-8}	10^{-2}	10^{-9}	10^{-2}	10^{-9}	10^{-9}
15	10^{-2}	10^{-9}	10^{-8}	10^{-2}	10^{-9}	10^{-2}	10^{-9}	10^{-12}
18	10^{-2}	10^{-9}	10^{-8}	10^{-2}	10^{-9}	10^{-2}	10^{-9}	10^{-15}

TABLE 7
Components $\{\tau_j(x_k)\}$ *of* p_k *in terms of* $\{v_j(x_k)\}$.

k	$\lvert\tau_1\rvert$	$\lvert\tau_2\rvert$	$\lvert\tau_3\rvert$	$\lvert\tau_4\rvert$	$\lvert\tau_5\rvert$	$\lvert\tau_6\rvert$	$\lvert\tau_7\rvert$	$\lvert\tau_8\rvert$
6	10^{-3}	10^{-17}	10^{-16}	10^{-2}	10^{-14}	10^{-1}	10^{-15}	10^{0}
9	10^{-3}	10^{-16}	10^{-16}	10^{-2}	10^{-15}	10^{-1}	10^{-14}	10^{3}
12	10^{-3}	10^{-17}	10^{-16}	10^{-2}	10^{-15}	10^{-1}	10^{-14}	10^{6}
15	10^{-3}	10^{-16}	10^{-16}	10^{-2}	10^{-15}	10^{-1}	10^{-14}	10^{9}
18	10^{-3}	10^{-16}	10^{-16}	10^{-2}	10^{-15}	10^{-1}	10^{-14}	10^{12}

then the method may become very inefficient (in iterations 6-18, about seven function evaluations are required per iteration). On the other hand, if the step to the minimum has a component in the effective null space *null*(J), underestimating *rank*(J) will inhibit decrease in *null*(J), because the Gauss-Newton search direction will be orthogonal to *null*(J). A scheme for using reduced-rank Gauss-Newton directions is one of the principal features of the nonlinear least-squares algorithm of Wedin and Lindström [1987]. The corrected Gauss-Newton methods of Gill and Murray [1976], [1978] may also compute reduced-rank Gauss-Newton directions, but they add a component in the corresponding null space as well.

4.2. Matrix square root $n = m = 4$. Another instance in which Gauss-Newton methods encounter ill-conditioned Jacobians is the problem of finding the square root of a given (square) matrix (see Fraley [1988b, Problem 36a)]. Although the matrix in question is only of order 2, the problem is a difficult one for the alternative methods, as shown in Table 8. In the Gauss-Newton methods, the Jacobian does become ill conditioned, but unlike the previous problem, a solution is obtained only when the Jacobian is assumed to have full rank at each iteration. A summary of the results for $\varepsilon = 10^{-10}$ and $\varepsilon \leqq 10^{-11}$ is given in Table 9. Tables 10 and 11 trace the iterations of the Gauss-Newton methods for $\varepsilon = 10^{-10}$ and $\varepsilon = 10^{-11}$, respectively.

TABLE 8

	f evals.	J evals.	iters.	$\lVert x^*\rVert_2$	$\lVert f^*\rVert_2$	$\lVert g^*\rVert_2$	est. err.	conv.
NPSOL	786	786	477	9.22	10^{-5}	10^{-5}	10^{-9}	
	2618	2618	1437	10.1	10^{-5}	10^{-7}	10^{-9}	
LMDER	4000	3985	3985	27.9	10^{-7}	10^{-7}	10^{-13}	F LIM.
	4000	3985	3985	27.9	10^{-7}	10^{-7}	10^{-13}	F LIM.
DN2G	4000	3988	3988	27.7	10^{-7}	10^{-7}	10^{-13}	F LIM.
	4000	3988	3988	27.7	10^{-7}	10^{-7}	10^{-13}	F LIM.

TABLE 9
Gauss-Newton.

	$\varepsilon = 10^{-10}$	$\varepsilon \leqq 10^{-11}$
f, J evals.	4004	95
iters.	473	39
$\lVert x^*\rVert_2$	10^{1}	50.0
$\lVert f^*\rVert_2$	10^{-7}	10^{-16}
$\lVert g^*\rVert_2$	10^{-6}	10^{-15}
est. err.	10^{-15}	10^{-33}
conv.	F LIM.	

TABLE 10
Gauss-Newton on the Matrix Square-Root Problem ($n = 4$).

$\varepsilon = 10^{-10}$

k	f, J evals.	$\|x_k\|_2$	$\|f_k\|_2$	$\|g_k\|_2$	$\|p_k\|_2$	$g_k^T p_k$	α_k	cond J_k	rank J_k
0	2	1.E+00	2.E+00	3.E+00	9.E−01	−3.E+00	1.0E+00	10^0	4
5	11	4.E+00	2.E−01	9.E−01	3.E+00	−3.E−02	3.7E−01	10^4	4
10	22	1.E+01	1.E−01	1.E+00	6.E+00	−1.E−02	2.0E−01	10^7	4
15	37	2.E+01	6.E−02	1.E+00	8.E+00	−4.E−03	1.5E−01	10^8	4
20	52	2.E+01	4.E−02	1.E+00	1.E+01	−2.E−03	1.3E−01	10^9	4
25	67	3.E+01	3.E−02	1.E+00	1.E+01	−1.E−03	1.3E−01	10^9	4
30	82	4.E+01	2.E−02	1.E+00	9.E+00	−5.E−04	1.7E−01	10^{10}	4
31	85	4.E+01	2.E−02	1.E+00	9.E+00	−4.E−04	1.9E−01	10^{10}	4
32	86	4.E+01	2.E−02	1.E+00	3.E−04	−3.E−04	1.0E+00	10^{10}	3
33	93	4.E+01	9.E−08	4.E−06	6.E+00	−8.E−15	2.1E−04	10^{10}	4
34	98	4.E+01	9.E−08	4.E−06	6.E+00	−8.E−15	9.9E−05	10^{10}	4
35	103	4.E+01	9.E−08	4.E−06	6.E+00	−8.E−15	9.9E−05	10^{10}	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
470	3986	4.E+01	9.E−08	4.E−06	6.E+00	−8.E−15	2.2E−05	10^{10}	4
471	3995	4.E+01	9.E−08	4.E−06	6.E+00	−8.E−15	2.2E−05	10^{10}	4
472	4004	4.E+01	9.E−08	4.E−06	6.E+00	−8.E−15	2.2E−05	10^{10}	4
		4.E+01	9.E−08	4.E−06					

TABLE 11

$\varepsilon = 10^{-11}$

k	f, J evals.	$\|x_k\|_2$	$\|f_k\|_2$	$\|g_k\|_2$	$\|p_k\|_2$	$g_k^T p_k$	α_k	cond J_k	rank J_k
0	2	1.E+00	2.E+00	3.E+00	9.E−01	−3.E+00	1.0E+00	10^0	4
5	11	4.E+00	2.E−01	9.E−01	3.E+00	−3.E−02	3.7E−01	10^4	4
10	22	1.E+01	1.E−01	1.E+00	6.E+00	−1.E−02	2.0E−01	10^7	4
15	37	2.E+01	6.E−02	1.E+00	8.E+00	−4.E−03	1.5E−01	10^8	4
20	52	2.E+01	4.E−02	1.E+00	1.E+01	−2.E−03	1.3E−01	10^9	4
25	67	3.E+01	3.E−02	1.E+00	1.E+01	−1.E−03	1.3E−01	10^9	4
30	82	4.E+01	2.E−02	1.E+00	9.E+00	−5.E−04	1.7E−01	10^{10}	4
35	92	5.E+01	1.E−02	6.E−01	3.E+00	−1.E−04	1.0E+00	10^{10}	4
36	93	5.E+01	4.E−03	3.E−01	3.E−01	−1.E−05	1.0E+00	10^{11}	4
37	94	5.E+01	1.E−05	7.E−04	6.E−04	−1.E−10	1.0E+00	10^{11}	4
38	95	5.E+01	2.E−11	1.E−09	1.E−09	−4.E−22	1.0E+00	10^{11}	4
		5.E+01	6.E−17	4.E−15					

The first difference between the two methods occurs at iteration 32. Data available from the SVD at the start of the iteration are shown in Table 12. The case for saying that $rank(J) = 3$ appears to be fairly strong. There is a large gap between σ_4 and σ_3, and $|\cos(g, p_4)|$ is significantly smaller than $|\cos(g, p_3)|$. Moreover, it would appear that the step taken when $\varepsilon = 10^{-10}$ and $rank(J) = 3$ is better, in the sense that the reduction in the values of both $\|f\|_2$ and $\|g\|_2$ is appreciably greater than the reduction achieved when $\varepsilon = 10^{-11}$ and $rank(J) = 4$. On the other hand, $|p_4|$ is not especially large for either choice of rank. For $\varepsilon = 10^{-10}$, the algorithm subsequently makes unacceptably slow progress, while for $\varepsilon = 10^{-11}$, quadratic convergence occurs after a few more iterations.

TABLE 12

$\varepsilon \leqq 10^{-10}$; iteration 32

r	σ_r	$\lvert\tau_r\rvert$	$\lVert p_r\rVert_2$	$\lvert g^T p_r\rvert$	$\lvert\cos(g, p_r)\rvert$
1	10^2	10^{-4}	10^{-4}	10^{-4}	10^0
2	10^2	10^{-4}	10^{-4}	10^{-3}	10^0
3	10^{-2}	10^{-15}	10^{-4}	10^{-3}	10^0
4	10^{-8}	10^1	10^1	10^{-3}	10^{-5}

To see why no further progress can be made for $\varepsilon = 10^{-10}$, consider the information on the state of the method at the start of iteration 33 (see Table 13). The singular values are nearly the same as those of the previous iteration, but the change is enough to have $rank(J) = 4$ rather than $rank(J) = 3$ according to (3.2). The value of $\lVert f\rVert_2$ has decreased significantly after iteration 32: $\lvert\tau_1\rvert$ and $\lvert\tau_2\rvert$ are much smaller at the start of iteration 33, although $\lvert\tau_3\rvert$ and $\lvert\tau_4\rvert$ are essentially unchanged. As a consequence, $\lVert p_4\rVert_2$ is now very large relative to $\lVert p_3\rVert_2$, but $\lvert\cos(g, p_4)\rvert$ is small since v_4 is close to being orthogonal to g. In fact, if (3.2) is disregarded and $rank(J)$ forced to be 3, the method will converge to a local minimum in one step.

TABLE 13

$\varepsilon \leqq 10^{-10}$; iteration 33

r	σ_r	$\lvert\tau_r\rvert$	$\lVert p_r\rVert_2$	$\lvert g^T p_r\rvert$	$\lvert\cos(g, p_r)\rvert$
1	10^2	10^{-9}	10^{-9}	10^{-15}	10^0
2	10^2	10^{-9}	10^{-9}	10^{-14}	10^0
3	10^{-2}	10^{-16}	10^{-9}	10^{-14}	10^0
4	10^{-8}	10^1	10^1	10^{-14}	10^{-10}

As in the previous section, we compare the sequence $\{p_k^*\}$ of steps from the iterates to the minimum of the function, to the sequence $\{p_k\}$ of Gauss-Newton steps (see Tables 14 and 15). Taking $rank(J) = 3$ is a bad strategy, in this case, because the solution lies mainly in the direction of $v_4(x_k)$.

It is difficult to say whether situations in which it is better to retain full rank or reduce the effective rank are more likely to be encountered in practice. Fraley [1988b] gives results for two Gauss-Newton methods that differ only in the value of a (fixed) tolerance used to estimate the rank. There are some cases in which the method is successful only for the lower value of the tolerance, and others in which the opposite

TABLE 14

Components $\{\tau_j^*(x_k)\}$ *of* $p_k^* = x^* - x_k$ *in terms of* $\{v_j(x_k)\}$.

k	$\lvert\tau_1^*\rvert$	$\lvert\tau_2^*\rvert$	$\lvert\tau_3^*\rvert$	$\lvert\tau_4^*\rvert$
28	10^{-3}	10^{-3}	10^{-15}	10^1
29	10^{-3}	10^{-3}	10^{-16}	10^1
30	10^{-3}	10^{-3}	10^{-14}	10^1
31	10^{-3}	10^{-3}	10^{-14}	10^1
32	10^{-3}	10^{-3}	10^{-15}	10^1

TABLE 15
Components $\{\tau_j(x_k)\}$ *of* p_k *in terms of* $\{v_j(x_k)\}$.

k	$\lvert\tau_1\rvert$	$\lvert\tau_2\rvert$	$\lvert\tau_3\rvert$	$\lvert\tau_4\rvert$
28	10^{-4}	10^{-4}	10^{-15}	10^{1}
29	10^{-4}	10^{-4}	10^{-15}	10^{1}
30	10^{-4}	10^{-4}	10^{-15}	10^{1}
31	10^{-4}	10^{-4}	10^{-14}	10^{1}
32	10^{-4}	10^{-4}	10^{-15}	10^{1}

is true. Wedin and Lindström [1987] develop a method in which the effective rank is adaptively determined, and state that it is important to allow the rank to increase if possible after a reduced-rank Gauss-Newton direction has been used.

4.3. Watson $n=20$; $m=31$. The final example for this section is a problem that might seem to be very hard for Gauss-Newton methods. In Watson's problem (Brent [1973], Moré, Garbow, and Hillstrom [1981], Fraley [1988b, Problem 20d]), a polynomial of degree n is fitted to approximate the solution of an ordinary differential equation. The Jacobian matrix for $n=20$ has singular values of order 10^{2}, 10^{1}, 10^{1}, 10^{0}, 10^{0}, 10^{0}, 10^{-1}, 10^{-1},10^{-2}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-5}, 10^{6}, 10^{-7}, 10^{-8}, 10^{-9}, 10^{-11}, and 10^{-12} at the origin. Yet there is very little difficulty in obtaining a solution, starting from $x_0=0$, for a wide range of values of ε, as shown in Table 16. Gauss-Newton compares favorably on this problem with results for the alternative methods, which are summarized in Table 17. (The trust-region algorithm DN2G loops for some values of the parameters in the termination criteria.)

TABLE 16
Gauss-Newton.

ε	10^{-8}	10^{-9}	10^{-10}	10^{-11}; 10^{-12}	10^{-13}	$\geqq 10^{-14}$
f, J evals.	6	6	6	6	6	6
iters.	5	5	5	5	5	5
$\lVert x^*\rVert_2$	1.07	1.11	1.55	5.21	29.2	247
$\lVert f^*\rVert_2$	10^{-8}	10^{-8}	10^{-9}	10^{-9}	10^{-10}	10^{-10}
$\lVert g^*\rVert_2$	10^{-14}	10^{-14}	10^{-14}	10^{-12}	10^{-14}	10^{-12}

TABLE 17

	f evals.	J evals.	iters.	$\lVert x^*\rVert_2$	$\lVert f^*\rVert_2$	$\lVert g^*\rVert_2$	est. err.	conv.
NPSOL	76	76	38	1.06	10^{-4}	10^{-5}	10^{-8}	
	200	200	99	1.06	10^{-5}	10^{-8}	10^{-11}	
LMDER	18	14	14	247	10^{-10}	10^{-12}	10^{-24}	
	23	15	15	247	10^{-10}	10^{-12}	10^{-25}	
DN2G	8	7	6	1.11	10^{-8}	10^{-13}	10^{-16}	
	(471)	(138)	(137)	1.18	10^{-8}	10^{-14}	10^{-16}	LOOP

Watson's problem has a number of local minima, so that the value of the Gauss-Newton solution is dependent on the tolerance ε. Nothing can be said concerning which of the local minima is the "better" one without knowing how the solution is

going to be used. For the larger values of ε, solutions are obtained that are small in magnitude and hence closer to the starting value, because lower values of the rank restrict the size of the search directions. On the other hand, the final value of the sum of squares is smaller for smaller values of ε, because the objective function is being decreased in a larger subspace at each step. Details of the Gauss-Newton iterations are given in Table 18.

TABLE 18
Gauss-Newton on Watson ($n = 20$).

k	f, J evals.	$\|x_k\|_2$	$\|f_k\|_2$	$\|g_k\|_2$	$\|p_k\|_2$	$g_k^T p_k$	α_k	cond J_k	rank J_k
				$\varepsilon = 10^{-8}$					
0	2	0.E+00	5.E+00	2.E+02	1.E+00	−3.E+01	1.0E+00	10^{14}	15
1	3	1.E+00	3.E+00	1.E+02	4.E−01	−6.E+00	1.0E+00	10^{13}	15
2	4	1.E+00	4.E−01	2.E+01	5.E−02	−2.E−01	1.0E+00	10^{13}	15
3	5	1.E+00	2.E−03	1.E−01	5.E−02	−4.E−06	1.0E+00	10^{13}	15
4	6	1.E+00	3.E−08	5.E−07	3.E−05	−2.E−16	1.0E+00	10^{13}	15
		1.E+00	3.E−08	2.E−14					
				$\varepsilon = 10^{-11}$					
0	2	0.E+00	5.E+00	2.E+02	1.E+00	−3.E+01	1.0E+00	10^{14}	18
1	3	1.E+00	3.E+00	1.E+02	4.E−01	−6.E+00	1.0E+00	10^{13}	18
2	4	1.E+00	4.E−01	2.E+01	2.E+00	−2.E−01	1.0E+00	10^{13}	18
3	5	2.E+00	2.E−03	1.E−01	3.E+00	−4.E−06	1.0E+00	10^{13}	18
4	6	5.E+00	1.E−08	5.E−07	4.E−03	−2.E−16	1.0E+00	10^{13}	18
		5.E+00	1.E−09	6.E−13					
				$\varepsilon = 10^{-14}$					
0	2	0.E+00	5.E+00	2.E+02	1.E+00	−3.E+01	1.0E+00	10^{14}	20
1	3	1.E+00	3.E+00	1.E+02	4.E−01	−6.E+00	1.0E+00	10^{13}	20
2	4	1.E+00	4.E−01	2.E+01	8.E+01	−2.E−01	1.0E+00	10^{13}	20
3	5	8.E+01	2.E−03	1.E−01	2.E+02	−4.E−06	1.0E+00	10^{13}	20
4	6	2.E+02	1.E−08	5.E−07	3.E−01	−2.E−16	1.0E+00	10^{13}	20
		2.E+02	2.E−10	4.E−12					

The condition number of the Jacobian remains very large throughout, yet the search direction is never especially large regardless of the choice of rank, because the sequence $\{|u_i^T f|\}$ is monotonically decreasing at about the same rate as the singular values (see (3.1)). The unit step gives sufficient decrease in every instance, on account of the many local minima. Moreover, there is superlinear convergence for each value of ε, despite the fact that p becomes very close to being orthogonal to the gradient, with $|\cos(g, p)|$ ranging from 10^{-5} for $\varepsilon = 10^{-8}$, to 10^{-9} for $\varepsilon \geqq 10^{-14}$, in the final step.

This example illustrates a situation in which it is virtually impossible to give a precise definition of rank. The rank estimate does affect the value of the solution obtained by the procedure, but is not in this case critical to the success or failure of the Gauss-Newton method. The Watson example appears to be a rather unusual one, and it seems more typical in Gauss-Newton methods to encounter problems of the type illustrated in the first two examples of this section (see Fraley [1988b]).

5. An example of poor performance on a well-conditioned zero-residual problem. Most statements about the inadequacy of Gauss-Newton methods concern their local behavior. The focus has been on the fact that Gauss-Newton methods are only linearly convergent on zero- or small-residual problems with rank-deficient Jacobians at the solution, and on large residual problems. It is widely believed that Gauss-Newton methods will work well on zero- or small residual problems in which the Jacobian is never ill conditioned. In this section we exhibit a zero-residual problem on which the Gauss-Newton method performs poorly in contrast to Newton's method—although the Gauss-Newton method is locally quadratically convergent and $cond(J_k)$ never exceeds 5×10^3. Thus the omission of the second-order part of the Hessian can adversely affect the global behavior of the Gauss-Newton method. The example is the following modification of Rosenbrock's function (e.g., Moré, Garbow, and Hillstrom [1981, p. 21]).

Modified Rosenbrock Function ($n = m = 2$).

$$\phi_1(x) = 100(x_2 - x_1^2),$$

$$\phi_2(x) = 1 - x_1,$$

$$x_0 = (0, 0),$$

$$f(x) = \begin{pmatrix} \phi_1(x) \\ \phi_2(x) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \text{at } (1, 1).$$

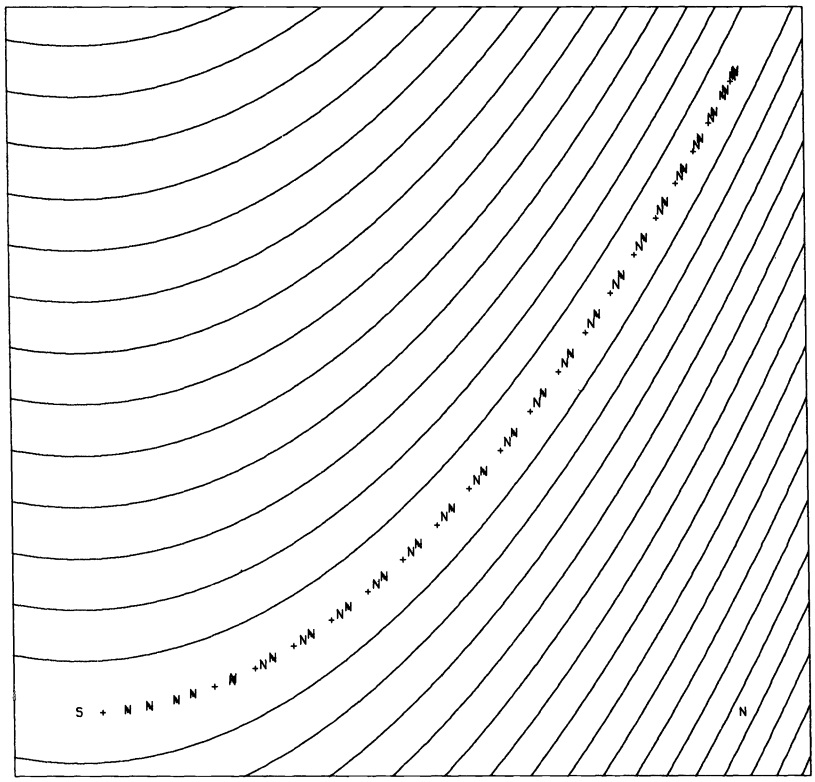


FIG. 1. *Newton's method on the modified Rosenbrock function. Starting value*: $x_0 = (0, 0)$; *solution*: $x^* = (1, 1)$. S—x_0; N—$x_k + p_k$; +—$x_k + \alpha_k p_k$.

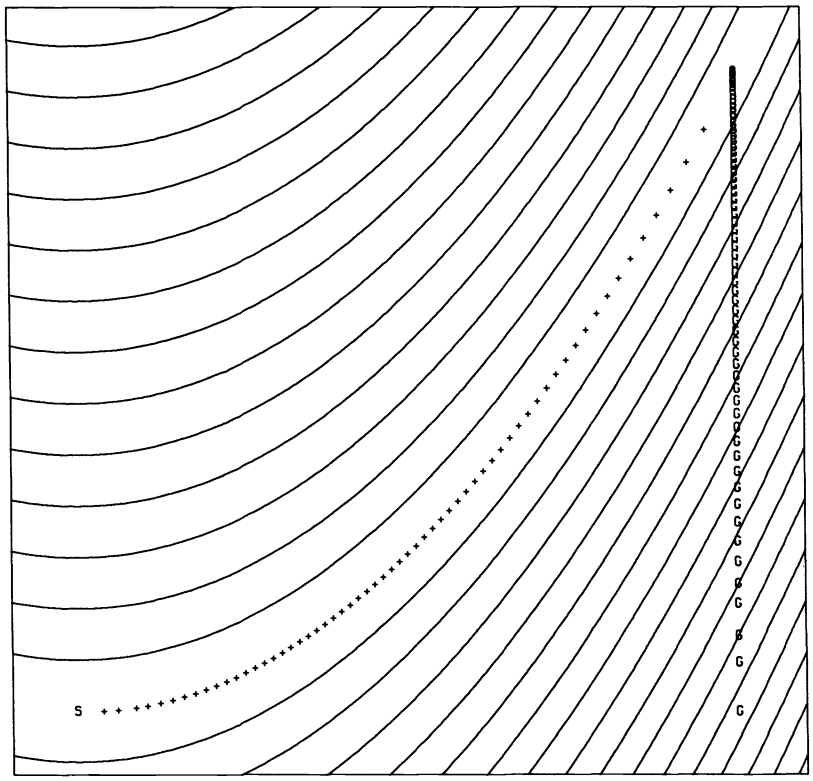


FIG. 2. *Gauss-Newton method on the modified Rosenbrock function. Starting value*: $x_0 = (0, 0)$; *solution*: $x^* = (1, 1)$. S—x_0; G—$x_k + p_k$; +—$x_k + \alpha_k p_k$.

The starting point (0, 0) lies at the bottom of a curved steep-sided valley in which the solution (1, 1) also lies. The modification to the multiplier in the first residual causes this valley to be steeper than it would have been in the original problem. Table 19 gives the results for both Gauss-Newton and Newton's method on this problem.

TABLE 19
Modified Rosenbrock ($n = m = 2$); $x_0 = (0, 0)$.

	Gauss-Newton	Newton's method
f, J evals.	467	77
iters.	100	50
$\|x^*\|_2$	1.41	1.41
$\|f^*\|_2$	10^{-15}	10^{-13}
$\|g^*\|_2$	10^{-13}	10^{-12}
est. err.	10^{-30}	10^{-26}

The same linesearch is used here for both methods (see § 3). It happens that Newton's method can be applied without modification, since the Hessian, as well as the Jacobian, is well conditioned. In this case, Gauss-Newton is Newton's method for nonlinear equations, because $n = m$. Contour plots of the progress of the two methods are displayed in Figs. 1 and 2.

The minimum of the Gauss-Newton model (1.3) lies well outside the valley in which the starting value and minimum are located, at least until the iterates are very

close to the solution. The merit function $\Phi(\alpha) = \|f(x_k + \alpha p_k)\|_2^2$ attains its maximum over the interval $[0, 1]$ at $\alpha = 1$, rather than a minimum as predicted by the quadratic model. Moreover, Φ increases very sharply from the valley floor ($\alpha = 0$) to the maximum. Hence a significant number of function evaluations is required in the linesearch in order to minimize $\Phi(\alpha)$, and, initially, rather small steps are taken along the search directions. By contrast, the relatively efficient performance of Newton's method can be explained by the fact that the minimum of the Newton quadratic model falls very near the curve along the valley floor connecting $(0, 0)$ to $(1, 1)$ (which is followed by the iterates of both methods) at all iterations except the first one.

There are several immediate improvements that can be made for this particular problem. Since the starting point is at the bottom of a valley in which the solution lies, any linesearch method that uses the nonlinear least-squares objective as a merit function is forced to follow this valley until it reaches the solution. Certainly it would be better to start from a point that does not have this property. However, for general problems, there is no known way to prevent a method from producing an iterate that lies at the bottom of such a valley. For the Rosenbrock example, given the knowledge that the residuals are zero at the solution, an alternative merit function could be used that assigns different weights to the residuals and thus allows the iterates to lie outside the valley. The efficiency of the Gauss-Newton method used in these examples could also be improved by including a strategy for decreasing the maximum allowed steplength, and by relaxing the parameter η that controls the accuracy of the linesearch (see, e.g., Gill, Murray, and Wright [1981]). What is not clear is how to formulate an adaptive strategy within the Gauss-Newton framework to prevent such inefficiencies from arising for general problems. No scheme for abandoning Gauss-Newton based on asymptotic behavior or on handling ill-conditioning in the Jacobian will offer improvement in this instance. On the other hand, using a trust-region as in Levenberg-Marquardt methods makes sense, because minimization of the Gauss-Newton quadratic model takes place only over a restricted region around the valley floor. A method such as NL2SOL is an even better option for a problem of this type, because it combines a trust-region strategy with the ability to select an alternative local approximation when the Gauss-Newton quadratic model is inadequate.

Acknowledgments. I thank Philip Gill, Walter Murray, Joseph Oliger, and Margaret Wright for discussion and comments on earlier versions of this paper. Special thanks goes to Margaret Wright for all kinds of help with the present version, and for encouraging me to submit it for publication. I would also like to thank the editor and referees for their comments, which have led to a number of improvements. Finally, I am indebted to the Stanford Linear Accelerator Center for the use of their computer facilities.

REFERENCES

J. T. BETTS [1976], *Solving the nonlinear least square problem: Application of a general method*, J. Optim. Theory Appl., 18, pp. 469-484.

R. P. BRENT [1973], *Algorithms for Minimization Without Derivatives*, Prentice-Hall, Englewood Cliffs, NJ.

J. E. DENNIS AND R. E. WELSCH [1978], *Techniques for nonlinear least squares and robust regression*, Communications in Statistics—Simulation and Computation B7, pp. 345-359.

J. E. DENNIS, D. M. GAY, AND R. E. WELSCH [1981a], *An adaptive nonlinear least-squares algorithm*, ACM Trans. Math. Software, 7, pp. 348-368.

———, [1981b], ALGORITHM 573 NL2SOL: *an adaptive nonlinear least-squares algorithm*, ACM Trans. Math. Software, 7, pp. 369-383.

J. E. DENNIS AND R. B. SCHNABEL [1983], *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ.

J. E. DENNIS, D. M. GAY, AND R. E. WELSCH [1983], *Remark on* ALGORITHM 573, ACM Trans. Math. Software, 9, p. 139.

P. DEUFLHARD AND V. APOSTOLESCU [1980], *A study of the Gauss-Newton algorithm for the solution of nonlinear least squares problems*, in Special Topics in Applied Mathematics, North-Holland, Amsterdam, pp. 129-150.

J. J. DONGARRA, J. R. BUNCH, C. B. MOLER, AND G. W. STEWART [1979], LINPACK *Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA.

R. FLETCHER [1963], *Function minimization without evaluating derivatives—a review*, Computer J., 8, pp. 163-168.

———, [1980], *Practical Methods of Optimization: Volume* I, *Unconstrained Optimization*, John Wiley, New York.

R. FLETCHER AND C. XU [1986], *Hybrid methods for nonlinear least squares*, Numerical Analysis Report NA/92, Department of Mathematical Sciences, University of Dundee, Dundee, (December 1985; revised July 1986); IMA J. Numer. Anal., to appear.

C. FRALEY [1987], *Solution of nonlinear least-squares problems* [1987], Tech. Report STAN-CS-87-1165, Ph.D. thesis, Department of Computer Science, Stanford University, and Report CLaSSiC-87-04, Center for Large Scale Scientific Computation, Stanford University, Stanford, CA, July.

———, [1988a], *Algorithms for nonlinear least-squares problems*, Tech. Report JUL 88-16, Systems Optimization Laboratory, Department of Operations Research, Stanford University; Center for Large Scale Scientific Computation, Stanford University; Université de Genève, Dept. SES-COMIN.

———, [1988b], *Software performance on nonlinear least-squares problems*, Tech. Report JUL 88-17, Systems Optimization Laboratory, Department of Operations Research, Stanford University; Center for Large Scale Scientific Computation, Stanford University; Université de Genève, Dept. SES-COMIN.

P. E. GILL AND W. MURRAY [1976], *Nonlinear least squares and nonlinearly constrained optimization*, in Numerical Analysis—Proc. Dundee 1975, Lecture Notes in Mathematics 506, Springer-Verlag, Berlin, New York, pp. 134-147.

———, [1978], *Algorithms for the solution of the nonlinear least-squares problem*, SIAM J. Numer. Anal., 15, pp. 977-992.

P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND M. H. WRIGHT [1986], *Users' guide for* NPSOL (*Version* 4.0): *A Fortran package for nonlinear programming*, Tech. Report SOL 86-2, Systems Optimization Laboratory, Department of Operations Research, Stanford University, Stanford, CA, January (revised version 4.2: 1987).

P. E. GILL, W. MURRAY, AND M. H. WRIGHT [1981], *Practical Optimization*, Academic Press, New York.

G. H. GOLUB AND C. L. VAN LOAN [1983], *Matrix Computations*, The Johns-Hopkins University Press, Baltimore, MD.

C. L. LAWSON AND R. J. HANSON [1974], *Solving least squares problems*, Prentice-Hall, Englewood Cliffs, NJ.

J. J. MORÉ [1978], *The Levenberg-Marquardt algorithm: Implementation and theory*, in Numerical Analysis—Proc. Dundee 1977, Lecture Notes in Mathematics 630, Springer-Verlag, Berlin, New York, pp. 105-116.

J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM [1980], *User Guide for* MINPACK-1, Tech. Report ANL-80-74, Applied Mathematics Division, Argonne National Laboratory, Argonne, IL, August.

J. J. MORÉ AND D. C. SORENSEN, *Newton's Method*, in Studies in Numerical Analysis, MAA Studies in Mathematics, Vol. 24, G. H. Golub, ed., The Mathematical Association of America, 1984, pp. 29-82.

——— [1981], *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7, pp. 17-41.

NAG *Fortran Library Manual, Vol.* 3 [1984], Numerical Algorithms Group, January.

J. NOCEDAL AND M. L. OVERTON [1985], *Projected Hessian updating algorithms for nonlinearly constrained optimization*, SIAM J. Numer. Anal., 22, pp. 821-850.

PORT *Mathematical Subroutine Library Manual* [1984], AT&T Bell Laboratories, May 1984.

M. J. D. POWELL [1970], *A hybrid method for nonlinear equations*, in Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowitz, ed., Gordon and Breach, New York, pp. 87-114.

G. W. STEWART [1973], *Introduction to Matrix Computations*, Academic Press, New York.

G. A. WATSON [1987], *Methods for best approximation and regression*, in The State of the Art in Numerical Analysis, A. Iserles and M. J. D. Powell, eds., Clarendon Press, Oxford, pp. 139-164.

P.-Å. WEDIN AND P. LINDSTRÖM [1987], *Methods and software for nonlinear least squares problems*, Report UMINF-133.87, ISSN-0348-0542, Institute of Information Processing, University of Umeå, Sweden, (revised July 1988).

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 533-549, May 1989
© 1989 Society for Industrial and Applied Mathematics
008

ON COLLOCATION IMPLEMENTATION FOR SINGULARLY PERTURBED TWO-POINT PROBLEMS*

URI ASCHER† AND SIMON JACOBS‡

Abstract. The numerical solution of singularly perturbed two-point boundary value problems in ordinary differential equations is considered. Implementation methods for general-purpose solvers of first-order linear systems are examined, with the basic difference scheme being *collocation* at Gaussian points. Adaptive mesh selection is based on localized error estimates at the *collocation points*. These methods are implemented as modifications to the successful collocation code COLSYS, which was originally designed for mildly stiff problems only. Efficient high-order approximations to extremely stiff problems are obtained, and comparisons to COLSYS show that the modifications work relatively much better as the singular perturbation parameter gets small (i.e., the problem gets stiff), for both boundary-layer and turning-point problems.

Key words. collocation, two-point problems, singular perturbation, symmetric schemes

AMS(MOS) subject classifications. 65L10, 65L20

1. Introduction. During the last decade a considerable amount of effort has been devoted to the numerical solution of singular perturbation problems in ordinary differential equations. The most prominent feature of such problems is one or more thin transition layers in which the profile of the analytic solution contains very steep gradients. Corner, shock, and boundary-layer-type features are commonly observed. These problems can pose significant numerical difficulty, yet they arise frequently in many practical applications, such as semiconductor theory, fluid dynamics, seismology, and nonlinear mechanics. Often in such problems we may expect some higher-order derivatives to be multiplied by a small parameter, which we will call ε.

When discretizing a singularly perturbed problem, the large local variation of the derivatives usually suggests a highly nonuniform mesh that is fine where the solution varies rapidly, and coarse where the solution is smooth. If we call h the largest mesh stepsize used over the entire domain, then $\varepsilon \ll h$, i.e., h cannot, for reasons of efficiency, be chosen small with respect to the problem coefficient ε. This raises theoretical difficulties because the usual asymptotic theory, which is valid when h can be chosen "sufficiently small," no longer holds in practice. Hence we are confronted with two major difficulties: The choice of a suitable discretization method and a theory to support its use; and the construction of an appropriate nonuniform mesh.

In this paper we restrict our attention to boundary value problems (BVPs) of the singularly perturbed type. We are interested in the solution of linear first-order systems and concentrate our efforts on implementation methods for general-purpose codes.

One basic difficulty with designing suitable discretization methods for this class of problems is that the differential operators considered give rise to various modes with very different behaviour types: We may have rapidly increasing and rapidly decreasing fundamental solutions, as well as "harmless" slow ones. (Another problematic mode type, that of rapidly oscillatory, nondecaying solutions, is excluded from consideration in this work.)

* Received by the editors December 8, 1986; accepted for publication (in revised form) July 29, 1988.

† Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada V6T 1WS. This research was supported in part by the National Science and Engineering Research Council of Canada under operating grant 4306.

‡ Department of National Defense, Operational Research Analysis Establishment, 101 Colonel Vy Drive, Ottawa K1AOK2, Canada.

There are two general approaches to the numerical solution of problems that contain markedly different mode types mixed together. One approach is to first apply a transformation to *decouple* the component types from one another. Following such a decoupling, separate discretization schemes, suitable for the different solution types, can be applied. For instance, decoupling "differential" and "algebraic" solution components in a system of differential/algebraic equations (DAEs) may allow the use of separate *nonstiff* discretizations for the numerical solution of such a problem (see, e.g., Marz [23]). Another instance is the decoupling of rapidly oscillatory solution components from other components in a stiff BVP allowing recovery of a smooth solution (e.g., Kreiss [21]) or again treating different component types separately (e.g., Ascher and Spudich [8]).

A third instance is the decoupling of rapidly increasing and rapidly decreasing modes for a BVP such as those under consideration here. This yields stiff initial value problems (IVPs) (in appropriate directions) only, and subsequently allows the use of (upwinded) one-sided difference schemes (such as the Backward Differentiation Formulas (BDFs) of Gear [17]) that produce approximate modes that preserve the decay of the analytical ones. A number of methods for performing such a decoupling, followed by an appropriate one-sided integration, have been recently proposed (e.g., see Kreiss, Nichols, and Brown [22], Brown and Lorenz [15], Dieci and Russell [16], Meyer [24]).

The second general approach to solving problems with mixed solution types is to attempt to find numerical schemes that are capable, at least to some acceptable degree, of simultaneously handling the various solution types, thus eliminating the need for an explicit decoupling transformation. Since an explicit decoupling transformation is generally expensive to calculate and at times tricky to find (especially for nonlinear problems; see the discussion in Ascher and Weiss [11]), a scheme that automatically deals with the problem is rather desirable, although not always attainable. For so-called transferrable initial value DAEs, the BDF schemes successfully fulfill the role (Gear and Petzold [18]); on the other hand, we are not aware of any corresponding general-purpose scheme for highly oscillatory BVPs.

For BVPs under consideration in this paper, piecewise polynomial collocation at (symmetric) Gaussian points is an instance of the second general approach, providing a family of schemes that have been extensively analyzed, implemented, and tested (de Boor and Swartz [13], Ascher, Christiansen, and Russell [5], Ascher and Weiss [9]–[11], Weiss [28], Ascher [1], [3], Ascher and Bader [4]). These schemes do not need a preliminary decoupling transformation, so they can be applied efficiently, and they have proved to work very well in many practical applications. The general-purpose code COLSYS (Ascher, Christiansen, and Russell [6]) implements these schemes. At the same time, they are theoretically less satisfactory when applied to a stiff IVP than are stiffly stable one-sided schemes (such as BDF, or collocation at Radau points). This is typical for all symmetric difference schemes. It is therefore not surprising to observe that many theoretical papers for stiff BVPs deal with (upwinded) one-sided schemes, while most practical calculations use symmetric ones.

In this paper we propose an improved algorithm for adaptive mesh selection using collocation at Gaussian points. To help illustrate our point, we examine a simple example first.

Example 1. Consider the stable IVP for a single equation

$$\varepsilon y' = -y + q(x), \qquad x > 0 \tag{1.1a}$$

$$y(0) = \alpha \tag{1.1b}$$

where $0 < \varepsilon \ll 1$ and the inhomogeneity $q(x)$ is smooth and bounded by a constant of

order one. The *reduced solution* for this problem, obtained by setting $\varepsilon = 0$ in (1.1a), is

$$\bar{y}(x) = q(x).$$

A well-scaled fundamental solution is $\exp(-x/\varepsilon)$, and the solution y of (1.1a,b) can be written as

$$y(x) = (\alpha - q(0)) \exp(-x/\varepsilon) + \bar{y}(x) + O(\varepsilon).$$

Thus, for $x \geqq \varepsilon|\ln \varepsilon|$, $y(x) = \bar{y}(x) + O(\varepsilon)$.

For this problem, two difference approximations are considered on a mesh

$$0 = x_1 < x_2 < \cdots, \qquad h_i := x_{i+1} - x_i.$$

(i) The *backward Euler scheme* is the first-order BDF and also the collocation scheme at one Radau point. It gives

$$\frac{\varepsilon}{h_i}(y_{i+1} - y_i) = -y_{i+1} + q(x_{i+1}), \qquad i = 1, 2, \cdots$$

or

$$y_{i+1} = \frac{\varepsilon/h_i}{\varepsilon/h_i + 1} y_i + \frac{1}{\varepsilon/h_i + 1} q(x_{i+1}).$$

We therefore observe that for $\varepsilon \ll h_i$, the approximation for the fundamental solution satisfies

$$\frac{\varepsilon/h_i}{\varepsilon/h_i + 1} \approx 0 \approx \exp\left(-\frac{h_i}{\varepsilon}\right)$$

and

$$y_{i+1} = q(x_{i+1}) + O\left(\frac{\varepsilon}{h_i}\right).$$

The approximate solution improves as $\varepsilon \to 0$ and the reduced solution is retrieved in the limit.

(ii) The *midpoint scheme* is obtained by collocation at one Gauss point. It gives

$$\frac{\varepsilon}{h_i}(y_{i+1} - y_i) = -\frac{1}{2}(y_i + y_{i+1}) + q(x_{i+1/2}), \quad x_{i+1/2} := x_i + \frac{1}{2}h_i, \quad i = 1, 2, \cdots \tag{1.2}$$

or

$$y_{i+1} = \frac{\varepsilon/h_i - \frac{1}{2}}{\varepsilon/h_i + \frac{1}{2}} y_i + \frac{1}{\varepsilon/h_i + \frac{1}{2}} q(x_{i+1/2}).$$

We therefore observe that the approximation for the fundamental solution, while satisfying

$$\left|\frac{\varepsilon/h_i - \frac{1}{2}}{\varepsilon/h_i + \frac{1}{2}}\right| \leqq 1$$

(and, more generally, A-stability; see, e.g., [17]), also satisfies for $\varepsilon \ll h_i$

$$\frac{\varepsilon/h_i - \frac{1}{2}}{\varepsilon/h_i + \frac{1}{2}} \approx -1 \neq 0,$$

and for $1 \leqq l \leqq i$,

$$y_{i+1} \approx -y_i + 2q(x_{i+1/2}) \approx (-1)^{i-l} y_l + 2 \sum_{j=l}^{i} (-1)^{i-j} q(x_{j+1/2}).$$

The reduced solution is not retrieved in the limit $\varepsilon \to 0$ at meshpoints. Moreover, when the local truncation error

$$\tau_j = \tfrac{1}{8} h_j^2 y''(x_{j+1/2}) + O(h_j^3)$$

is substituted for $q(x_{j+1/2})$ and when the error δ emanating from the boundary layer is substituted for y_l (this can be made small using a fine mesh in the narrow layer $0 < x \lesssim \varepsilon |\ln \varepsilon|$), the above equation gives an error estimate that is *nonlocal* and is of a possibly lower order than in the nonstiff case.

The situation is better, however, at the collocation point $x_{i+1/2}$. For $y_{i+1/2} = \frac{1}{2}(y_i + y_{i+1})$, which is the collocation approximation at $x_{i+1/2}$, we obtain from (1.2) (using the stability, albeit marginal, of the scheme),

$$y_{i+1/2} = q(x_{i+1/2}) + O(\varepsilon / h_i).$$

This gives a localized error estimate, and the reduced solution is obtained in the limit $\varepsilon \to 0$, as for the backward Euler scheme.

In particular, it is important to note the effect of the layer error δ *away* from the layer, because this error component may not be small before we obtain a good fine mesh at the layer: While at meshpoints away from the layer the error has an $O(\delta)$ component, at midpoints the corresponding error component is $O(\delta \varepsilon h_i^{-1})$ only. □

This simple example displays the basic observation underlying this paper. *When using collocation at Gaussian points for a stiff* BVP, *the error when* $\varepsilon \ll h_i$ *is better localized at collocation points than elsewhere* (*including meshpoints*). The evaluation of the approximate solution at collocation points cannot, of course, be used to improve the stability of the scheme that generates it, but it may possibly be used to help to generate a better mesh in an adaptive mesh-selection algorithm such as the one used in COLSYS.

In § 2 we discuss error estimates for collocation at Gaussian points. As it turns out, the mesh-selection algorithm in COLSYS, which was designed on the basis of nonstiff theory, essentially attempts to do the right thing even in the very stiff case. However, the evaluation of the solution derivative required in this mesh-selection algorithm is better achieved using solution values at collocation points, as discussed in § 3. An experimental implementation in the context of linear first-order BVPs has been carried out, and in § 4 we demonstrate its potentially remarkable improvement over the performance of COLSYS.

2. Collocation at Gaussian points. We consider a collocation method for the BVP of order n:

$$\mathbf{u}' = A(x)\mathbf{u} + \mathbf{q}(x), \qquad a < x < b, \tag{2.1a}$$

$$B_a \mathbf{u}(a) + B_b \mathbf{u}(b) = \beta. \tag{2.1b}$$

First select a set of k *canonical collocation points* ρ,

$$0 \leqq \rho_1 < \rho_2 < \cdots < \rho_k \leqq 1,$$

which are later chosen to be Gauss points (i.e., zeros of the Legendre polynomial) on $[0, 1]$. Now, given the mesh

$$\pi\colon a = x_1 < x_2 \cdots < x_N < x_{N+1} = b, \tag{2.2}$$
$$h_i := x_{i+1} - x_i, \qquad h := \max_{1 \leqq i \leqq N} h_i$$

and the points ρ, a piecewise polynomial *collocation solution* $\mathbf{u}_\pi(x)$ of (2.1) is determined such that

$$\mathbf{u}_\pi(x) \in C[a, b] \cap P_{k+1,\pi}, \tag{2.3a}$$

$$\mathbf{u}'_\pi(x_{ij}) = A(x_{ij})\mathbf{u}_\pi(x_{ij}) + \mathbf{q}(x_{ij}), \qquad 1 \leqq i \leqq N, \quad 1 \leqq j \leqq k, \tag{2.3b}$$

$$B_a(a)\mathbf{u}_\pi(a) + B_b(b)\mathbf{u}_\pi(b) = \beta \tag{2.3c}$$

where

$$x_{ij} := x_i + h_i\rho_j, \qquad 1 \leqq i \leqq N, \quad 1 \leqq j \leqq k,$$

i.e., the solution is continuous on $[a, b]$, satisfies the boundary conditions, and is on each interval of π a polynomial of degree less than $k+1$ that satisfies the differential equation at the k *collocation points* x_{ij}. It can be shown that the requirements (2.3) uniquely determine the approximate solution $\mathbf{u}_\pi(x)$, and are equivalent to a Runge-Kutta formulation (see, e.g., [10]).

Now, consider collocating a smooth, *nonstiff* BVP (2.1). The following convergence results are known to hold [2].

THEOREM 2.4. *Assume that the* BVP (2.1) *is well posed in the sense that the conditioning constant* κ *(i.e., a bound on Green's function) is of moderate size, and denote its unique solution by* $\mathbf{u}$. *Assume further that* $A, \mathbf{q} \in C^{2k}[a, b]$, *and that collocation at* k *Gaussian points is used.*

Then, for h *sufficiently small we have the following*:

(a) *The collocation method has a unique solution* $\mathbf{u}_\pi(x)$;

(b) *There exists an implementation that is stable, with condition number (for the resulting algebraic system)* $\sim \kappa N$;

(c) *At mesh points there is superconvergence, i.e.,*

$$\mathbf{u}(x_i) - \mathbf{u}_\pi(x_i) = O(h^{2k}), \qquad 1 \leqq i \leqq N; \tag{2.4c}$$

(d) *Other points*

$$\mathbf{u}^{(j)}(x) - \mathbf{u}_\pi^{(j)}(x) = p^{(j)}\left(\frac{x - x_i}{h_i}\right)\mathbf{u}^{(k+1)}(x_i)h_i^{k+1-j} + O(h_i^{k+2-j}) + O(h^{2k}), \tag{2.4d}$$

$$x_i < x < x_{i+1}, \qquad 1 \leqq i \leqq N, \quad 0 \leqq j \leqq k$$

$p(t)$ a known smooth function. □

For adaptive mesh selection in COLSYS the estimate (2.4d) is used because of the localization of its leading term for $k > 1$. Further, at subinterval midpoints,

$$\mathbf{u}^{(k)}(x_{i+1/2}) - \mathbf{u}_\pi^{(k)}(x_{i+1/2}) = O(h_i^2) \tag{2.5}$$

so an $O(h)$ estimate of $\mathbf{u}^{(k+1)}$ is obtained by differentiating a linear interpolant of the values $\mathbf{u}_\pi^{(k)}(x_{i+1/2})$, $1 \leqq i \leqq N$. Thus, having solved on a current mesh, the code attempts to refine it by equidistributing an approximation of the leading error term in (2.4d).

Another way to obtain an $O(h)$ estimate of $\mathbf{u}^{(k+1)}$ when $k > 1$ is as follows. From (2.4d) we can write

$$\mathbf{u}^{(k-1)}(x_{i+1/2}) = \mathbf{u}_\pi^{(k-1)}(x_{i+1/2}) + p_i h_i^2 + O(h_i^3) + O(h^{2k}), \qquad 1 \leqq i \leqq N \tag{2.6a}$$

with the constants p_i satisfying

$$p_{i+1} - p_i = O(h_i), \qquad 1 \leqq i < N. \tag{2.6b}$$

Thus,

$$\mathbf{u}_\pi^{(k-1)}(x_{i+3/2}) - \mathbf{u}_\pi^{(k-1)}(x_{i+1/2}) = \mathbf{u}^{(k-1)}(x_{i+3/2}) - \mathbf{u}^{(k-1)}(x_{i+1/2}) + p_i(h_{i+1}^2 - h_i^2) + O(h_{i+1}h_i^2) + O(h_i^3) + O(h_{i+1}^3) + O(h^{2k}).$$

If the mesh is *locally almost uniform*, i.e., for $1 \leqq i \leqq N$

$$h_{i+1} = h_i(1 + O(h_i)) \tag{2.7}$$

then we have for the first divided differences of $\mathbf{u}_\pi^{(k-1)}$ and $\mathbf{u}^{(k-1)}$

$$\mathbf{u}_\pi^{(k-1)}[x_{i+1/2}, x_{i+3/2}] = \mathbf{u}^{(k-1)}[x_{i+1/2}, x_{i+3/2}] + O(h_i^2) + O(h^{2k}h_i^{-1}).$$

So, letting $\mathbf{u}_Q(x)$ be the quadratic interpolant of $\mathbf{u}_\pi^{(k-1)}(x_{i-1/2})$, $\mathbf{u}_\pi^{(k-1)}(x_{i+1/2})$ and $\mathbf{u}_\pi^{(k-1)}(x_{i+3/2})$, and differentiating it twice, we find

$$\begin{aligned}\mathbf{u}_Q'' &= 2\frac{\mathbf{u}_\pi^{(k-1)}[x_{i+1/2}, x_{i+3/2}] - \mathbf{u}_\pi^{(k-1)}[x_{i-1/2}, x_{i+1/2}]}{x_{i+3/2} - x_{i-1/2}} \\ &= \mathbf{u}^{(k+1)}(x_i) + O(h_i) + O(h^{2k}h_i^{-2}).\end{aligned}$$

This estimation procedure is obviously more cumbersome and restricted than the one used in COLSYS; but as it turns out it can be extended for stiff problems, whereas the former cannot.

The basic difficulty with mesh selection for (very) *stiff* BVPs is that neither Theorem 2.4 nor (2.5) hold (cf. [9]-[11], [1]). First, conclusion (b) in Theorem 2.4 is not guaranteed, i.e., the stability of the collocation method does not automatically follow from well-posedness of the BVP (2.1). This must be assumed in addition, although the assumption rarely poses a practical restriction. Second, and more importantly in practice, the error is no longer localized, as indicated in Example 1. Third, the superconvergence order (2.4c) drops [10].

To be more specific, consider a segment $[x_{\underline{i}}, x_{\bar{i}}]$ of $[a, b]$ where the solution is known to be smooth (i.e., it and some of its derivatives are bounded). We make the dependence on a small parameter $0 < \varepsilon \ll 1$ explicit and assume the ODE (2.1a) to be in the form

$$\begin{aligned}\varepsilon\mathbf{y}' &= A_{11}(x;\varepsilon)\mathbf{y} + A_{12}(x;\varepsilon)\mathbf{z} + \mathbf{q}_1(x;\varepsilon), \\ \mathbf{z}' &= A_{21}(x;\varepsilon)\mathbf{y} + A_{22}(x;\varepsilon)\mathbf{z} + \mathbf{q}_2(x;\varepsilon),\end{aligned} \tag{2.8}$$

i.e.,

$$A(x;\varepsilon) = \begin{bmatrix} \varepsilon^{-1}A_{11}(x;\varepsilon) & \varepsilon^{-1}A_{12}(x;\varepsilon) \\ A_{21}(x;\varepsilon) & A_{22}(x;\varepsilon) \end{bmatrix},$$

$$\mathbf{u}(x;\varepsilon) = \begin{pmatrix} \mathbf{y}(x;\varepsilon) \\ \mathbf{z}(x;\varepsilon) \end{pmatrix}, \qquad \mathbf{q}(x;\varepsilon) = \begin{pmatrix} \varepsilon^{-1}\mathbf{q}_1(x;\varepsilon) \\ \mathbf{q}_2(x;\varepsilon) \end{pmatrix}$$

where A_{lm}, $\mathbf{q}_l$ are smooth and bounded, $1 \leqq l, m \leqq 2$, and $A_{11}(x;0)$ has a hyperbolic splitting. Thus, there is an appropriate transformation $T(x)$ satisfying

$$T^{-1}(x)A_{11}(x;0)T(x) = \begin{pmatrix} \Lambda^-(x) & \\ & \Lambda^+(x) \end{pmatrix}, \qquad x_{\underline{i}} \leqq x \leqq x_{\bar{i}} \tag{2.9}$$

where $\Lambda^-(x)$ is an upper triangular matrix with diagonal elements satisfying

$$\operatorname{Re}(\lambda_j(x)) \leqq \lambda_- < 0, \qquad 1 \leqq j \leqq n_-,$$

$\Lambda^+(x)$ is an upper triangular matrix satisfying

$$\mathrm{Re}\,(\lambda_j(x)) \geqq \lambda_+ > 0, \qquad n_- + 1 \leqq j \leqq n,$$

and

$$|\mathrm{Im}\,(\lambda_j(x))| \leqq \mathrm{const}\,|\mathrm{Re}\,(\lambda_j(x))|, \qquad 1 \leqq j \leqq n,$$

for all $x_{\underline{i}} \leqq x \leqq x_{\bar{i}}$.

Defining

$$\mathbf{w}(x) := T^{-1}(x)\mathbf{y}(x), \tag{2.10a}$$

$$\mathbf{w}(x) \equiv \begin{pmatrix} \mathbf{w}^-(x) \\ \mathbf{w}^+(x) \end{pmatrix}, \qquad \mathbf{w}^- \in R^{n_-} \tag{2.10b}$$

we assume that the BVP consisting of (2.8) subject to

$$\mathbf{w}^-(x_{\underline{i}}), \qquad \mathbf{z}(x_{\underline{i}}), \qquad \mathbf{w}^+(x_{\bar{i}}) \quad \text{specified} \tag{2.11}$$

is well conditioned. The analysis in [10] then yields the following theorem (using single bars to denote maximum vector norms).

THEOREM 2.12. *Let the assumptions above hold for* (2.8). *Define* $\mathbf{w}_\pi(x)$ *as a transformation of the collocation solution* $\mathbf{y}_\pi(x)$ *using* (2.10). *If* $\varepsilon \ll \min_{\underline{i} \leqq j < \bar{i}} h_j^2$ *and*

$$|\mathbf{w}^-(x_{\underline{i}}) - \mathbf{w}_\pi^-(x_{\underline{i}})|, \qquad |\mathbf{w}^+(x_{\bar{i}}) - \mathbf{w}_\pi^+(x_{\bar{i}})| \leqq \delta, \tag{2.12a}$$

then the error satisfies

$$|\mathbf{u}(x_i) - \mathbf{u}_\pi(x_i)| \leqq c_1 \Big| \sum_{j=\underline{i}}^{\bar{i}-1} \gamma_j \|\mathbf{u}^{(k+1)}\|_j h_j^{k+1} \Big| + c_2\delta + c_3 h^{2k}, \qquad \underline{i} \leqq i \leqq \bar{i} \tag{2.12b}$$

where c_l *are constants of moderate size,* $\|\phi\|_j := \max_{x_j \leqq x \leqq x_{j+1}} |\phi(x)|$, *and*

$$\gamma_j = (-1)^{kj} + O(\varepsilon h_j^{-1}). \tag{2.12c}$$

□

The estimate (2.12b) for the error at meshpoints is generally not better than the error at other points x, $x_{\underline{i}} \leqq x \leqq x_{\bar{i}}$.

From (2.12b) we see that if the layer regions for a given BVP are adequately resolved by a fine mesh so that δ in (2.12a) is small, then the leading term of the error in regions where the mesh is sparse is

$$c \sum_{j=\underline{i}}^{\bar{i}-1} \gamma_j |\mathbf{u}^{(k+1)}(x_j)| h_j^{k+1}.$$

This indicates that the superconvergence order (2.4c) drops and the error is no longer localized. However, COLSYS still attempts to equidistribute a relevant quantity!

Better results than those of (2.12b) are obtained at collocation points, as in Example 1. Consider the reduced equations at x_{ij}, $1 \leqq j \leqq k$, when $\varepsilon \ll h_i$. This gives

$$\mathbf{y}_\pi(x_{ij}) \approx -A_{11}^{-1}(x_{ij})[A_{12}(x_{ij})\mathbf{z}_\pi(x_{ij}) + \mathbf{q}_1(x_{ij})], \tag{2.13}$$

which may be substituted to eliminate $\mathbf{y}_\pi(x_{ij})$ from the nonstiff equations for $\mathbf{z}_\pi(x_{ij})$. The localized estimate (such as (2.4d)) that holds for the error in the slow variables $\mathbf{z}_\pi(x_{ij})$ also gives a localized bound for the error in the fast variables $\mathbf{y}_\pi(x_{ij})$ through (2.13). This heuristic argument can be made rigorous following the analysis in [10], obtaining the following theorem.

THEOREM 2.14. *Under the assumptions of Theorem* 2.12,

$$\begin{aligned} &\mathbf{u}(x_{ij}) - \mathbf{u}_\pi(x_{ij}) = \varphi_{ij}\|\mathbf{u}^{(k+1)}\|_i h_i^{k+1} + O(h_i^{k+2}) + O(\delta\varepsilon h_i^{-1}) + O(h^{2k}) \\ &\varphi_{i+1,j} - \varphi_{ij} = O(h_i), \qquad \underline{i} \leqq i \leqq \bar{i} - 1, \quad 1 \leqq j \leqq k. \end{aligned} \tag{2.14}$$

□

3. Mesh-selection implementation. The results in the previous section indicate that even when $\varepsilon \ll h_i$, an attempt to select a mesh that equidistributes the leading error term in (2.4d) is reasonable. A good mesh on the entire interval must still be fine in layer regions, i.e., $h_i \lesssim \varepsilon$ there, so (2.4d) does hold in some regions; however, the practical transition between this asymptotic form and (2.12b) is hard to identify. For these reasons we have kept the basic mesh-selection strategy in COLSYS, as described in [5]. It remains to discuss the choice of constant in the error expression and the approximation of $\mathbf{u}^{(k+1)}(x_i)$ appearing there.

The equidistribution of a mesh for a given mesh size is not affected by the particular choice of the error constant, whose primary use is to help to intelligently estimate the mesh size needed for a given error tolerance. Since little is known about such a choice in the stiff case we use the constant for the nonstiff case, as implemented in COLSYS [5].

The estimation of $\mathbf{u}^{(k+1)}$ is much more critical, and care must be taken to find a "good" approximation. If the values of $\mathbf{u}_\pi$ used for this purpose are heavily polluted by nonlocal effects (such as may emanate from a layer region with a poorly chosen mesh) then the approximation to the $(k+1)$st derivative may be extremely poor. We take advantage of the localized error estimate *at collocation points* (2.14) when $\varepsilon \ll h_i$, assuming that (2.7) holds for $\underline{i} \leqq i < \bar{i}$.

We first estimate $\mathbf{u}^{(k-1)}$ on each interval. Let $L_1(t), \cdots, L_k(t)$ be the Lagrange interpolating polynomials of the k canonical points ρ on $[0, 1]$. Then

$$\mathbf{u}_L(x) := \sum_{j=1}^{k} L_j\left(\frac{x-x_i}{h_i}\right)\mathbf{u}_\pi(x_{ij}), \qquad x_i \leqq x < x_{i+1}, \quad \underline{i} \leqq i \leqq \bar{i}-1$$

is a piecewise polynomial of order k that interpolates $\mathbf{u}_\pi(x)$ at the k collocation points on $[x_i, x_{i+1}]$ for each i. Using (2.14), we have

$$\begin{aligned}\mathbf{u}(x) = \mathbf{u}_L(x) + \sum_{j=1}^{k} L_j\left(\frac{x-x_i}{h_i}\right)\{O(\delta\varepsilon h_i^{-1}) + \varphi_{ij}\|\mathbf{u}^{(k+1)}\|_i h_i^{k+1} + O(h_i^{k+2}) + O(h^{2k})\} \\ + R_i(x), \\ x_i \leqq x < x_{i+1}, \qquad \underline{i} \leqq i \leqq \bar{i}-1\end{aligned} \tag{3.1}$$

with the remainder written using divided differences as follows:

$$R_i(x) = \mathbf{u}[x_{i1}, \cdots, x_{ik}, x] \prod_{j=1}^{k} (x - x_{ij}).$$

In general, of course, $R_i = O(h_i^k)$, so $R_i^{(k-1)} = O(h_i)$, but for symmetric schemes it is easy to see that

$$R_i^{(k-1)}(x_{i+1/2}) = \bar{R}_i h_i^2$$

where $\bar{R}_i$ are constants satisfying $\bar{R}_{i+1} = \bar{R}_i + O(h_i)$. (Also, $u_\pi^{(k-1)}(x_{i+1/2}) = u_L^{(k-1)}(x_{i+1/2})$.) Differentiation of (3.1) then gives

$$\begin{aligned}\mathbf{u}^{(k-1)}(x_{i+1/2}) = \mathbf{u}_L^{(k-1)}(x_{i+1/2}) + \tilde{R}_i h_i^2 + O(h_i^3) + O(\delta\varepsilon h_i^{-k}) + O(h_i^{1-k}h^{2k}), \\ \underline{i} \leqq i \leqq \bar{i}-1\end{aligned} \tag{3.2}$$

where the constants $\tilde{R}_i$ satisfy $\tilde{R}_{i+1} = \tilde{R}_i + O(h_i)$. In (3.2), we have a relation similar to (2.6). Thus, for each interval i, $\underline{i}+1 \leqq i \leqq \bar{i}-2$, by interpolating $\mathbf{u}_L^{(k-1)}(x_{i-1/2})$,

$\mathbf{u}_L^{(k-1)}(x_{i+1/2})$, and $\mathbf{u}_L^{(k-1)}(x_{i+3/2})$ quadratically and then differentiating twice, we obtain the piecewise constant approximation:

$$\begin{aligned}\mathbf{u}^{(k+1)}(x) = &\frac{2}{(x_{i-1/2}-x_{i+1/2})(x_{i-1/2}-x_{i+3/2})}\mathbf{u}_L^{(k-1)}(x_{i-1/2})\\ &+\frac{2}{(x_{i+1/2}-x_{i-1/2})(x_{i+1/2}-x_{i+3/2})}\mathbf{u}_L^{(k-1)}(x_{i+1/2})\\ &+\frac{2}{(x_{i+3/2}-x_{i-1/2})(x_{i+3/2}-x_{i+1/2})}\mathbf{u}_L^{(k-1)}(x_{i+3/2})+\hat{R}_i,\\ &x_i \leqq x \leqq x_{i+1}, \qquad \underline{i}+1 \leqq i \leqq \bar{i}-2\end{aligned} \tag{3.3}$$

where, assuming that the rightmost two terms in (3.2) can be neglected and taking into account the "smoothness" of $\tilde{R}_i$,

$$\hat{R}_i = O(h_i). \tag{3.4}$$

The bound (3.4) justifies, when $k>1$ and (2.7) holds, taking the computable expression obtained when ignoring $\hat{R}_i$ in (3.3) as an approximation of $|\mathbf{u}^{(k+1)}(x_i)|$. The approximation to $\mathbf{u}^{(k+1)}$ and the error coefficient are both piecewise constant, and hence are easily used to generate new equidistributed meshes.

More generally, if we attempt to apply this error estimate everywhere then the assumption of local almost uniformity appears at first to be intolerable, as it implies global quasiuniformity of the mesh: Since in layer regions we must have $h_i = O(\varepsilon)$, we get $h = O(\varepsilon)$ as well. However, this assumption can be relaxed to hold only segmentwise. Thus, consider a segmentation

$$a = t_1 < t_2 < \cdots < t_M < t_{M+1} = b$$

such that M is fixed (independent of ε) and on each segment $[t_j, t_{j+1}]$ the solution either has a layer or is smooth (see [22], [7]). We may assume a locally almost uniform mesh only on each segment, if we avoid interpolating across the points t_j, interpreting them as endpoints. Then we may take $h_i \gg \varepsilon$ on smooth solution segments.

In practice, (3.3) usually works well. However, difficulty is encountered when the quadratic interpolation is applied for a mesh interval where h_{i-1}, h_i, h_{i+1} differ by many orders of magnitude. This indicates an interpolation across a segment endpoint. Usually, when this problem occurs we simply apply the interpolation in a one-sided way (as at the endpoints, normally). Difficulty arises, however, when no three consecutive intervals of similar magnitude are available to make even the one-sided estimation. If there are only two intervals of comparable magnitude we can, similarly to (3.3), estimate $\mathbf{u}^{(k)}$ on two intervals and then linearly interpolate. The necessary $(k+1)$st collocation point is taken from the adjacent interval.

When only one interval is available (i.e., when the ratios of h_{i-1} to h_i and h_i to h_{i+1} vary considerably or a boundary interferes), no reasonable approximation is at hand, and we cannot resolve the problem. To prevent this situation, an additional meshpoint is placed in such a case at the midpoint of $[x_i, x_{i+1}]$ when the mesh is created in order that a sufficient number of intervals will always exist for further redistribution.

Other approaches for error equidistribution have also been examined. One idea is to equidistribute a lower derivative, say $\mathbf{u}^{(k)}$, instead of $\mathbf{u}^{(k+1)}$ (Russell and Christiansen [27, § 3]), for which an $O(h^2)$ approximation exists ($O(h)$ for $k=1$). However, experience shows that this is inferior in practice because the code does not detect and respond to trouble areas as well as with the above method. Another idea is to estimate

$\mathbf{u}^{(k+1)}$ directly, by interpolating from k points on one interval and two points from adjacent intervals, and then differentiating. The problem with this idea is that an $O(1)$ factor of the error in estimating $\mathbf{u}^{(k+1)}$ behaves instead as does $O(h_{i+1}^k/h_i^k)$ (ignoring the contribution from global terms), and creates a much more severe problem for adjacent intervals of differing size than with the method used.

Note that the considerations based on (2.14) hold only for $\varepsilon \ll h$. The main advantage of the proposed algorithm is in the improved error estimate and mesh selection that it offers away from layers, where we expect the assumption $\varepsilon \ll h$ to hold. But if we now assume $h \lesssim \varepsilon$, we get the nonstiff error estimates at collocation points. Therefore, as the algorithm succeeds in "regularizing" the problem in a transition layer, it still equidistributes the mesh in an appropriate way. Observe, however, that the equidistribution when $h \ll \varepsilon$ is inferior to that of COLSYS, which equidistributes on the basis of a nonstiff problem.

4. Numerical examples. The a posteriori mesh-selection considerations of the previous sections have been incorporated into an experimental adaptive collocation code, which we refer to as RKNEW. The program integrates first-order linear systems with separated two-point boundary conditions by implementing a symmetric Runge-Kutta solution representation (see Bader and Ascher [12, § 4], Jacobs [20]). RKNEW has been tested on a large number of very stiff problems, with both boundary layers and turning points, and we now examine three representative examples, all of the singularly perturbed type. In some instances, comparisons are made with the most recent version of COLSYS, called COLNEW [12].

Differential systems are converted to first order before integration (even for COLNEW, which can integrate higher-order equations directly), a limit of 500 mesh intervals is placed on any mesh generated by either code, and the error tolerance is imposed on all components of the first-order system. Both codes were run in double precision ($\approx$15 decimal digits) on a VAX 750, using the Fortran 77 compiler of BSD 4.2 UNIX. Run-time comparisons are not meaningful given the preliminary state of RKNEW, however, the mesh-selection procedure of RKNEW is slightly more costly because, unlike COLNEW, the solution must be evaluated at the k collocation points on each interval of the mesh every time a mesh redistribution is possible. Since the cost to find an approximation on a mesh of size N is $O(N)$, the best indication of the relative work associated with finding a solution for a given ε is to compare the total number of mesh intervals required in all meshes generated by each code. The largest N encountered serves as a relative measure for storage.

The following notation will be used:

u_l—lth component of the exact solution,

$E(u_l)$—maximum of error in u_l calculated at x_i, $x_{i+1/3}$, $x_{i+2/3}$ for each mesh interval i, $1 \leqq i \leqq N$,

δ—mixed absolute and relative error tolerance,

k—number of collocation points per mesh interval,

N—number of mesh intervals,

Mesh sequence—sizes of successive meshes generated to meet tolerances,

N_{tot}—total number of mesh intervals in the mesh sequence,

$a \pm b$—$a \cdot 10^{\pm b}$,

*—an extra point has been added to make the mesh sufficiently locally uniform.

Example 2. (A Turning Point, Hemker [19, Ex. 3.7.4], [5, Ex. 1].) The BVP

$$\varepsilon y'' + xy' = -\varepsilon\pi^2 \cos(\pi x) - \pi x \sin(\pi x), \qquad -1 < x < 1,$$
$$y(-1) = -2, \qquad y(1) = 0$$

is converted to first order by $u_1 := y$, $u_2 := y'$, with solution

$$u_1(x) = \cos(\pi x) + \operatorname{erf}\left(\frac{x}{(2\varepsilon)^{1/2}}\right) \Big/ \operatorname{erf}\left(\frac{1}{(2\varepsilon)^{1/2}}\right).$$

As $\varepsilon \to 0$ the solution develops a shock layer in the turning point region near $x = 0$ and is smooth elsewhere (Fig. 1). To find a cheap, accurate approximation to this problem, each code must recognize the shock and place a sufficiently dense mesh

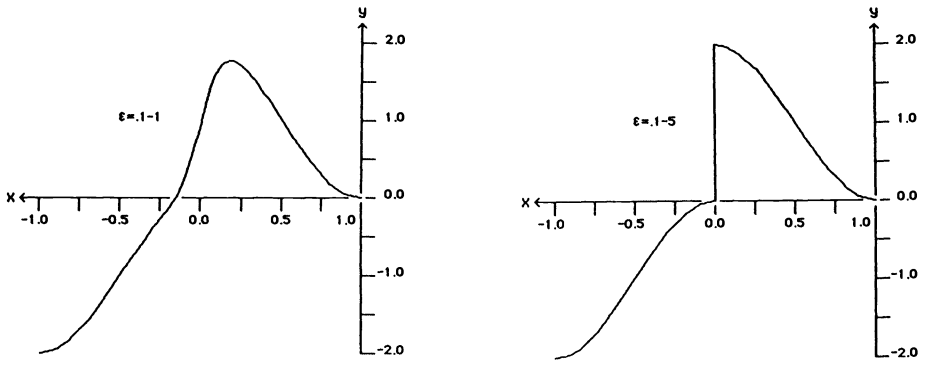


FIG. 1. *The solution for Example* 2.

TABLE 1
Results for Example 2 *with a uniform initial mesh.*

			COLNEW				RKNEW			
ε	k	δ	$E(u_1)$	$E(u_2)$	N_{tot}	Mesh sequence	$E(u_1)$	$E(u_2)$	N_{tot}	Mesh sequence
.1−1	4	.1−1	.52−4	.79−3	24	8, 16	.52−4	.79−3	24	8, 16
.1−1	4	.1−5	.89−7	.59−6	119	8, 16, 32, 21, 42	.15−6	.37−6	132	8, 16, 16, 32, 20, 40
.1−3	4	.1−5	.12−7	.49−6	426	8, 16, 32, 64, 51, 102, 51, 102	.58−7	.37−6	312	8, 16, 16, 16, 32, 32, 64, 128
.1−5	4	.1−5	.30−9	.27−7	1352	8, 16, 16, 32, 32, 32, 64, 64, 64, 128, 128, 128, 256, 128, 256	.83−7	.37−6	474	8, 16, 16, 16, 32, 32, 32, 64, 43, 86, 43, 86
.1−6	4	.1−5				8, 16, 16, 32, 32, 32, 64, 64, 64, 128, 128, 128, 256, 256, 256, >500	.15−6	.24−5	406	8, 16, 16, 16, 32, 32, 32, 64, 64, 42, 84
.1−8	4	.1−5					.60−7	.77−7	942	8, 16, 16, 16, 32, 32, 32, 64, 60, 120, 111, 222, 111, 222
.1−11	4	.1−5					.59−7	.24−6	1263	8, 16, 16, 16, 32, 32, 32, 64, 39, 78, 53, 106, 85, 170, 86, 172, 86, 172

around it, using as few mesh intervals as possible (i.e., N small). Results for COLNEW and RKNEW are given in Table 1 for an initially uniform mesh.

As ε gets smaller, the problem gets more difficult to solve and more meshpoints are required. It is clear that for large ε there is little difference in the performance, but as the problem gets harder, RKNEW is far superior, finding adequate meshes of relatively small size for extremely small values of ε. The size of the final mesh is important because it determines the amount of storage used, and it also reflects how well the code has detected and responded to difficulties early in the mesh-selection process; once a particular size, $\hat{N}$ say, has been reached, further meshes are always of size $\geqq \hat{N}/2$. Note how both codes redistribute the mesh several times early in each mesh sequence, indicating how quickly the difficulty has been detected. For the final mesh of RKNEW at $\varepsilon = .1-11$, 144 of 172 mesh intervals are in $(-.1-4, .1-4)$, $\max_{1\leqq i\leqq 172} h_i = .69-1$, $\min_{1\leqq i\leqq 172} h_i = .15-6$, and $h_{159}/h_{158} = .85+5$. RKNEW does not have difficulty dealing with the locally nonuniform mesh interval sizes. □

Example 3. (A Boundary Layer [19, Ex. 1.3.2].) The BVP

$$\varepsilon y'' + y' = 0, \qquad 0 < x < \frac{1}{4},$$

$$y(0) = 1, \qquad y\left(\frac{1}{4}\right) = \exp\left(-\frac{1}{4\varepsilon}\right)$$

is converted to first order by $u_1 := y$, $u_2 := y'$, with solution

$$u_1(x) = \exp\left(-\frac{x}{\varepsilon}\right).$$

As $\varepsilon \to 0$ the solution forms a steep boundary layer of width $O(\varepsilon)$ at $x = 0$ and is smooth elsewhere; there are no turning points. Results for an initially uniform mesh are given in Table 2. There is no evidence to suggest that one code is superior to the other. Note that for $\varepsilon = .1-5$, both codes fail to locate the problem and keep doubling past the arbitrary 500 meshpoint limit. In fact, as ε gets smaller, more consecutive doublings of the initial mesh are required before equidistribution is even attempted. The difficulty is that as $\varepsilon \to 0$ the layer gets thinner, and while coarse meshes "see" the effect as a large global error, they are unable to pinpoint the location of the disturbance; the layer is simply "skipped over."

This is easy to understand for the algorithm proposed here. As in Example 1, when $\varepsilon \ll h_i$ the approximate solution values at collocation points are essentially equal to the reduced solution values, and the latter do not vary faster in the layer region than away from it. Hence the code has "no reason" to concentrate mesh points in the layer region. (The situation is similar to that for one-sided schemes; cf. [22].) We emphasize that the advantage offered by our algorithm as compared to that in COLSYS is in exploiting the accurate solution values at collocation points *away* from layers. Note also that the error estimates in the code are calculated at points other than collocation points.

An alternative is to use the knowledge that a boundary layer at $x = 0$ actually exists (from a solution for a larger ε, or by some analytic means), and to choose the initial mesh so that the codes can detect the layer. A surprisingly simple way is to choose a new initial mesh, $\pi_\alpha := \{0.0, \alpha, 2\alpha, 3\alpha, 4\alpha, 0.25\}$, where α is some small parameter that controls the initial concentration of meshpoints around the layer. Results for smaller ε with this nonuniform initial mesh are given in Table 3. Various values of α were tried for the initial COLNEW mesh at $\varepsilon = .1-5$, but none was found to generate a solution in under 500 mesh intervals. In all instances the initial mesh was

TABLE 2
Results for Example 3 with a uniform initial mesh.

			COLNEW				RKNEW			
ε	k	δ	$E(u_1)$	$E(u_2)$	N_{tot}	Mesh sequence	$E(u_1)$	$E(u_2)$	N_{tot}	Mesh sequence
.1−1	5	.1−1	.10−3	.88−3	14	5, 3, 6	.12−3	.11−2	14	5, 3, 6
.1−1	5	.1−3	.80−6	.27−5	33	5, 4, 8, 16	.25−5	.96−5	26	5, 3, 6, 12
.1−1	5	.1−5	.12−6	.34−6	44	5, 5, 10, 8, 16	.52−8	.13−7	69	5, 5, 10, 7, 14, 28
.1−3	5	.1−5	.97−9	.27−7	279	5, 10, 20, 40, 34, 68, 34, 68	.41−8	.25−4	295	5, 10, 20, 40, 40, 20, 40, 20, 40, 20, 40
.1−4	5	.1−5	.26−10	.20−9	955	5, 10, 20, 40, 80, 160, 160, 80, 160, 80, 160	.11−11	.17−9	1206	5, 10, 20, 40, 80, 160, 99, 198, 99, 198, 99, 198
.5−5	5	.1−5	.53−9	.12−8	1949	5, 10, 20, 40, 80, 160, 320, 219, 438, 219, 438	.61−7	.17−6	1595	5, 10, 20, 40, 80, 160, 320, 160, 320, 160, 320
.1−5	5	.1−5				5, 10, 20, 40, 80, 160, 320, >500				5, 10, 20, 40, 80, 160, 320, >500

adjusted a little for small N and then simply doubled past 500 intervals. With RKNEW, on the final mesh at $\varepsilon = .1-11$, 174 out of 176 intervals were in (0.0, .1−9), and $h_{175}/h_{174} = .62+11$. It is interesting to note that after the first mesh of 176 intervals, the maximum error is still $\sim .1+9$, and that after only two mesh redistributions, it is reduced to $\sim .1-6$. This quick improvement is a common feature of both codes, but is more pronounced for RKNEW.

Another option is to use the exponential meshes of [9], [10], [1] as initial meshes. We have experimented with these, but found that while their performance is far superior to the initially uniform meshes, it is also rather inferior to the performance of the very crude nonuniform meshes, π_α. To understand this, note first that at the initial stages of our adaptive mesh-selection algorithm the recognition of a fast solution variation in a layer region is more important than making the error size very small everywhere. Second, COLSYS and its derivatives tested here measure a global error, while the exponential mesh of [9] relies on superconvergence at meshpoints. □

Example 4. A Turning Point and Boundary Layer (Brown and Lorenz [15, Ex. 2]) reads

$$-\varepsilon y'' - \frac{x}{2} y' + \frac{x}{2} z' + z = \varepsilon\pi^2 \cos(\pi x) + \frac{\pi}{2} x \sin(\pi x) =: g(x), \qquad -1 < x < 1,$$

$$-\varepsilon z'' + z = 0,$$

$$y(-1) = -1, \quad z(-1) = 1, \quad y(1) = z(1) = \exp\left(-\frac{2}{\varepsilon^{1/2}}\right).$$

TABLE 3
Results for Example 3 with a simple nonuniform initial mesh.

				COLNEW				RKNEW			
ε	α	k	δ	$E(u_1)$	$E(u_2)$	N_{tot}	Mesh sequence	$E(u_1)$	$E(u_2)$	N_{tot}	Mesh sequence
.1-3	.1-1	5	.1-1	.29-8	.93-6	258	5, 5, 10, 20, 40, 40, 23, 46, 23, 46	.11-6	.37-6	156	12, 24, 20, 40, 20, 40
.1-4	.1-2	5	.1-3	.17-9	.39-9	1220	5, 5, 10, 20, 40, 80, 160, 150, 300, 150, 300				
.1-5	.1-1	5	.1-5				5, 10, 20, 40, 80, 160, 320, >500	.26-10	.40-7	654	6*, 12, 24, 48, 96, 52, 104, 52, 104, 52, 104
.1-7	.1-4	5	.1-5					.26-10	.41-7	762	6*, 12, 24, 48, 96, 64, 128, 64, 128, 64, 128
.1-9	.1-6	5	.1-5					.27-10	.42-7	870	6*, 12, 24, 48, 96, 76, 152, 76, 152, 76, 152
.1-11	.1-8	5	.1-5					.27-10	.43-7	978	6*, 12, 24, 48, 96, 88, 176, 88, 176, 88, 176

This is converted to first order by introducing the variables w and v such that

$$\varepsilon z' = v, \qquad w' = \tfrac{1}{2}y + \tfrac{1}{2}xv + z - g,$$

and $u_1 := y$, $u_2 := v$, $u_3 := w$, $u_4 := z$. On integrating we get

$$\varepsilon u_1' = -\frac{x}{2}u_1 + \frac{1}{2}(\varepsilon - 1)u_2 + u_3 + (1-\varepsilon)\frac{x}{2}u_4,$$

$$u_2' = u_4, \quad u_3' = \frac{1}{2}u_1 + \frac{x}{2}u_2 + u_4 - g, \quad \varepsilon u_4' = u_2$$

under appropriate boundary conditions. The exact solution is

$$u_1(x) = \frac{\operatorname{erf}(x/2\varepsilon^{1/2})}{\operatorname{erf}(1/2\varepsilon^{1/2})} + u_4 + \cos(\pi x), \qquad u_4(x) = \exp\left(-\frac{(x+1)}{\varepsilon^{1/2}}\right).$$

Under the given boundary conditions, the solution for z has a boundary layer at $x = -1$. The differential equation for y is formed essentially by adding terms involving z and its derivative to the problem of Example 2. Hence, the solution for y contains a boundary layer similar to z, plus a shock layer at the turning point $x = 0$ similar to the solution of Example 2 (Fig. 2). The turning point is the more difficult feature to resolve. Results for an initially uniform mesh are given in Table 4.

Again, RKNEW performs better. The turning point is noticed first, and for instance with $\varepsilon = .1-7$, it is not until the second mesh of size 126 that any attempt is made to resolve the boundary layer. For the final mesh at $\varepsilon = .1-7$, 73 points are in $(-1, -.99)$ and 79 points are in $(-.1-2, .1-2)$; the rest of the mesh is a smooth transition between these regions and the boundaries (unlike the previous two examples).

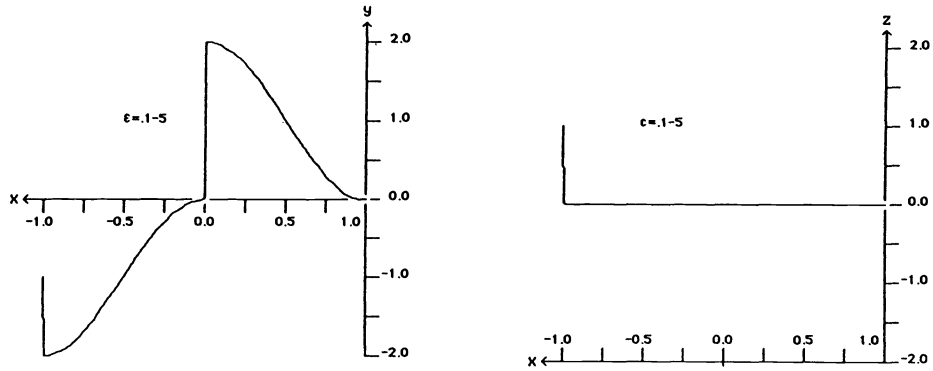


FIG. 2. *The solution for Example* 4.

TABLE 4
Results for Example 4 *with a uniform initial mesh.*

			COLNEW				RKNEW			
ε	k	δ	$E(u_1)$	$E(u_4)$	N_{tot}	Mesh sequence	$E(u_1)$	$E(u_4)$	N_{tot}	Mesh sequence
.1−1	4	.1−1	.11−3	.24−3	15	5, 10	.11−3	.24−3	15	5, 10
.1−1	4	.1−5	.27−6	.35−6	75	5, 10, 20, 40	.27−6	.35−6	75	5, 10, 20, 40
.1−2	4	.1−5	.79−7	.96−7	175	5, 10, 20, 20, 40, 80	.59−7	.13−7	233	5, 10, 20, 17, 34, 21, 42, 84
.1−3	4	.1−5	.81−6	.29−7	390	5, 10, 20, 40, 35, 70, 35, 70, 35, 70	.14−6	.31−7	290	5, 10, 20, 20, 20, 40, 25, 50, 100
.1−5	4	.1−5	.44−6	.11−7	817	5, 10, 10, 10, 20, 20, 40, 40, 40, 80, 80, 77, 154, 77, 154	.14−6	.78−8	635	5, 10, 20, 40, 40, 40, 80, 40, 80, 40, 80, 160
.1−6	4	.1−5				5, 10, 10, 10, 20, 20, 20, 40, 40, 40, 80, 80, 80, 160, 160, 160, 320, >500				
.1−7	4	.1−5					.27−5	.24−6	1343	5, 10, 20, 20, 40, 40, 40, 80, 80, 63, 126, 63, 126, 63, 126, 63, 126, 252
.1−8	4	.1−5								5, 10, 20, 20, 40, 40, 40, 80, 80, 80, 160, 160, 320, >500

The results in Table 4 can be extended using a simple continuation of meshes: The final mesh at one value of ε is used as the initial mesh for a smaller value of ε. This is most natural to do in the context of nonlinear problems (see, e.g., [6], [11]). In practice we use the second last mesh (of which the last is a doubling) to convey the character of the final mesh with as few points as possible. Results are given in Table 5, where the first initial mesh is taken from the smallest value of ε in Table 4,

TABLE 5
Results for Example 4 using continuation of meshes.

			COLNEW				RKNEW			
ε	k	δ	$E(u_1)$	$E(u_4)$	N_{tot}	Mesh sequence	$E(u_1)$	$E(u_4)$	N_{tot}	Mesh sequence
.1−7	4	.1−5	.28−5	.27−8	740	77, 51, 102, 51, 102, 51, 102, 204				
.1−9	4	.1−5				102, 61, 122, 61, 122, 65, 130, 65, 130, 122, 244, 244, 244, 488, 309, >500	.24−6	.14−6	567	126, 63, 126, 252
.1−11	4	.1−5					.56−4	36−7	1701	126, 63, 126, 63, 126, 63, 126, 252, 126, 252, 126, 252

and subsequent initial meshes are taken from the previous value of ε in the table. It is interesting to note that if we decrease ε by a factor of .1 instead of .1−1, then the step from .1−8 to .1−9 for RKNEW requires >500 meshpoints. This indicates the sensitivity of the process. □

5. Conclusions. In this paper we have discussed an a posteriori mesh-selection procedure for collocation at Gaussian points based on localized error estimates at *collocation points.* This procedure can be used to give efficient high-order approximations to extremely stiff problems. These features have been incorporated within the framework of COLSYS to give an experimental, general-purpose code, RKNEW, for linear first-order problems.

RKNEW was compared to COLNEW (the most recent version of COLSYS) for numerous linear examples and the results are promising. As the perturbation parameter, ε, gets small, RKNEW usually outperforms COLNEW, with respect to both the total work required (based on the total number of meshpoints in the mesh sequence), and the size of largest mesh required. Furthermore, RKNEW recognizes and responds to stiff problems better and is more methodical in the way it chooses meshes. The results are quite dramatic for turning points, and to a lesser degree, for boundary layers. RKNEW does especially well when meshes with very large local nonuniformities of interval sizes are present, a situation that causes COLNEW difficulty. COLNEW does work better for some nonstiff and mildly stiff problems; however, RKNEW usually still performs adequately in such cases. The creation of a new mesh is slightly more expensive in RKNEW.

The performance of RKNEW may be critically affected in extreme cases by the choice of initial mesh. If ε is extremely small, a coarse initial mesh may actually "miss" the disturbance, and RKNEW will require very large meshes to recover. Simple nonuniform meshes, continuation of meshes, and the exponential meshes of [9] are all reasonable alternatives, if used intelligently.

Extension of the ideas presented here to nonlinear problems and their full general-purpose implementation are under investigation and will be reported elsewhere.

REFERENCES

[1] U. ASCHER, *On some difference schemes for singular singulary-perturbed boundary value problems,* Numer. Math., 46 (1985), pp. 1-30.

[2] U. ASCHER, *Collocation for two-point boundary value problems revisited*, SIAM J. Numer. Anal., 23 (1986), pp. 596-609.

[3] ———, *Two families of symmetric difference schemes for singular perturbation problems*, in Proc. Workshop on Numerical Boundary Value ODEs, U. Ascher and R. D. Russell, eds., Birkhäuser, Boston, 1985.

[4] U. ASCHER AND G. BADER, *Stability of collocation at Gaussian points*, SIAM J. Numer. Anal., 23 (1986), pp. 412-422.

[5] 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.

[6] ———, *Collocation software for boundary value* ODEs, ACM Trans. Math. Software, 7 (1981), pp. 209-222.

[7] U. ASCHER AND R. MATTHEIJ, *General framework, stability and error analysis for numerical stiff boundary value methods*, Numer. Math., 1989, to appear.

[8] U. ASCHER AND P. SPUDICH, *A hybrid collocation method for calculating complete theoretical seismograms in vertically varying media*, Geophys. J. Roy. Astrophys. Soc., 86 (1986), pp. 19-40.

[9] U. ASCHER AND R. WEISS, *Collocation for singular perturbation problems* I: *First order systems with constant coefficients*, SIAM J. Numer. Anal., 15 (1983), pp. 537-557.

[10] ———, *Collocation for singular perturbation problems* II: *Linear first order systems without turning points*, Math. Comp., 43 (1984), pp. 157-187.

[11] ———, *Collocation for singular perturbation problems* III: *Nonlinear problems without turning points*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 811-829.

[12] 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.

[13] C. DE BOOR AND B. SWARTZ, *Collocation at Gaussian points*, SIAM J. Numer. Anal., 10 (1973), pp. 582-606.

[14] C. DE BOOR AND R. WEISS, SOLVEBLOK: *A package for solving almost block diagonal linear systems*, ACM Trans. Math. Software, 6 (1980), pp. 80-87.

[15] D. L. BROWN AND J. LORENZ, *A high-order method for stiff boundary-value problems with turning points*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 790-805.

[16] L. DIECI AND R. D. RUSSELL, *Riccati and other methods for singularly perturbed* BVPs, in Proc. BAILI Conference, J. Miller, ed., Boole Press, Dublin, 1986.

[17] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[18] C. W. GEAR AND L. PETZOLD, ODE *methods for the solution of differential/algebraic systems*, SIAM J. Numer. Anal., 21 (1984), pp. 716-728.

[19] P. W. HEMKER, *A numerical study of stiff two-point boundary value problems*, Tech. Report 80, Math. Centrum, Amsterdam, 1977.

[20] S. JACOBS, *Implementation methods for singularly perturbed two-point boundary value problems*, M.Sc. thesis, Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada, 1986.

[21] H.-O. KREISS, *Problems with different time scales for ordinary differential equations*, SIAM J. Numer. Anal., 16 (1979), pp. 980-998.

[22] H.-O. KREISS, N. NICHOLS AND D. BROWN, *Numerical methods for stiff two-point boundary value problems*, SIAM J. Numer. Anal., 23 (1986), pp. 325-368.

[23] R. MARZ, *On boundary value problems in differential-algebraic equations*, preprint 124, Humboldt University, Berlin, 1986.

[24] G. MEYER, *Continuous orthonormalization for boundary value problems*, J. Comput. Phys., 62 (1986), pp. 248-262.

[25] V. PEREYRA AND E. G. SEWELL, *Mesh selection for discrete solution of boundary value problems in ordinary differential equations*, Numer. Math., 23 (1975), pp. 261-268.

[26] R. D. RUSSELL, *Mesh Selection Methods*, Springer Lecture Notes in Computer Science 76, B. Childs, M. Scott, J. W. Daniel, E. Denman, and P. Nelson, eds., Springer-Verlag, New York, 1979.

[27] R. D. RUSSELL AND J. CHRISTIANSEN, *Adaptive mesh selection strategies for solving boundary value problems*, SIAM J. Numer. Anal., 15 (1978), pp. 59-80.

[28] R. WEISS, *An analysis of the box and trapezoidal schemes for linear singularly perturbed boundary value problems*, Math. Comp., 42 (1984), pp. 41-67.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 550-561, May 1989
© 1989 Society for Industrial and Applied Mathematics
009

ON GENERATING DISCRETE LINEAR l_∞ TEST PROBLEMS*

RICHARD BARTELS† AND BARRY JOE‡

Abstract. A method of generating discrete, linear, l_∞ approximation problems with known solutions and properties is outlined. The method is general enough to produce rank-deficient problems with linear constraints and to produce problems that closely mimic those arising from the area of data fitting.

Key words. *l*-infinity approximation, problem generator, data fitting, constraints

AMS(MOS) subject classifications. 65F99, 65K99, 65D99

1. Introduction. The discrete layer l_∞ (Chebyshev) approximation problem involves solving an overdetermined system of linear equations by minimizing the l_∞ norm of the residual vector. Many algorithms for solving unconstrained and constrained versions of this problem have been developed. Those published after 1970 include [1]-[7], [9]-[14], [18], [19], [22], and [23].

The corresponding period has not seen a single offering in methods for testing such algorithms. In the optimization literature we find some papers on testing or problem generation [8], [15], [20], [21], [25]-[28]. Of these, only [15] and [21], deal with linear problems, the former with general linear programming problems and the latter with unconstrained l_1 data-fitting problems. The remaining references are concerned with the testing of nonlinear optimization problems. We hope to fill the gap by presenting an approach to generating linear, constrained and unconstrained, full rank or rank-deficient, l_∞ data-fitting problems.

Our methods draw somewhat on those of Charnes, Raike, Stutz, and Walters [15], who generated linear programming problems with known solutions, and on the work of Hoffmann and Shier [21], who provided a test problem generator for discrete linear l_1 approximation problems. Charnes and his colleagues did not consider any special matrix structure in their proposals, and they produced full-rank problems. Hoffman and Shier introduced the l_1 structure into their problems. Their approach, like ours, is based on the formulation of the problem as primal and dual linear programming problems and, like ours, is capable of generating rank-deficient problems. Our approach, aside from being for the l_∞ problem, differs from theirs by providing problems with linear equality and inequality constraints. We have used our approach to compare three algorithms [23].

The problem of interest is

$$\begin{aligned} &\text{minimize} && \|c - A^T y\|_\infty \\ &\text{subject to} && D^T y = f \\ &\text{and} && G^T y \geqq h \end{aligned} \tag{1.1}$$

where A is an $m \times n$ matrix, D is an $m \times p$ matrix, G is an $m \times q$ matrix, c, f, and h are vectors of dimension n, p, and q, respectively, and $y = [y_1, \cdots, y_m]^T$ is a vector

* Received by the editors August 20, 1987; accepted for publication (in revised form) August 22, 1988. This research was supported under grant A4076 by the Natural Sciences and Engineering Research Council of Canada.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

‡ Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2G1.

of unknown parameters. All vectors and matrices are real. None of the matrices need be of full rank, and there is no requirement that the constraints need be imposed (i.e., D and/or G may be vacuous), or if imposed, that they form a consistent system.

We wish to construct

$$A, D, G, c, f, h, \text{ and } y \tag{1.2}$$

so that any of a variety of conditions may be met, for example,

(1) No feasible y exists;
(2) A unique, prescribed y solves (1.1);
(3) A prescribed y represents one of a number of possible solutions to (1.1);
(4) A prescribed y represents a degenerate point for (1.1);
(5) The solution to (1.1) is defined in an ill-conditioned fashion;
(6) Any of the arrays A, D, G can be deficient in rank;
(7) A has a form of the kind that arises from data-fitting problems;
(8) Prescribed residual magnitudes $|c_i - a_i^T y|$ are maximal at a solution to (1.1) and/or prescribed inequality constraints $g_i^T y \geqq h_i$ are satisfied with equality.

Section 2 covers the connection between (1.1) and a linear programming problem. Most of the conditions we wish to impose upon the arrays (1.2) can be induced using the dual of this linear programming problem. In § 3 we present a scheme for constructing the arrays (1.2) so as to satisfy conditions from the above list. In § 4 we pay closer attention to item (7) from the above list. In § 5 we look at some issues concerning the use of random numbers.

2. Linear programming formulation. We will construct problems of the form (1.1) using an equivalent linear programming problem, which will be discussed in a rank-deficient format. This is slightly more general than the format usually encountered in the reference works (e.g., Chvátal [17]), but the extension only requires that we loosen our concept of a basis matrix slightly. The construction is based on the fundamental optimality conditions for linear programming. These conditions extend verbatim to the case in which the canonical form of a linear programming problem (i.e., with equality constraints plus nonnegativity constraints on the individual variables) has a constraint matrix $\bar{A}$ with less than full rank. To make the extension, a basis matrix is understood to be any submatrix consisting of a linearly independent, spanning subset of columns from $\bar{A}$, and its inverse is understood in the generalized (Moore-Penrose) sense. Linear programming problems in this format cannot be feasible unless the right-hand side vector of the equality constraints lies in the column space of $\bar{A}$, which is a condition that one is not usually accustomed to verifying. A possible reformulation of (1.1) is

$$\begin{aligned} &\text{minimize} \quad [1 \quad 0^T] \begin{bmatrix} y_0 \\ y \end{bmatrix} \\ &\text{subject to} \quad \begin{bmatrix} e & A^T \\ e & -A^T \\ 0 & D^T \\ 0 & -D^T \\ 0 & G^T \end{bmatrix} \begin{bmatrix} y_0 \\ y \end{bmatrix} \geqq \begin{bmatrix} c \\ -c \\ f \\ -f \\ h \end{bmatrix} \end{aligned} \tag{2.1}$$

where y_0 is a variable representing $\|c - A^T y\|_\infty$ and e is the n-vector containing a one in all components. The constraints of this problem guarantee that y_0 will be nonnegative.

The dual of (2.1) is

$$\text{maximize} \quad [c^T - c^T \quad f^T - f^T \quad h^T] \begin{bmatrix} t \\ u \\ v \\ w \\ z \end{bmatrix}$$

(2.2)

$$\text{subject to} \quad \begin{bmatrix} e^T & e^T & 0^T & 0^T & 0^T \\ A & -A & D & -D & G \end{bmatrix} \begin{bmatrix} t \\ u \\ v \\ w \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\text{and} \quad t \geqq 0, \quad u \geqq 0, \quad v \geqq 0, \quad w \geqq 0, \quad z \geqq 0,$$

where t and u are n-vectors, v and w are p-vectors, and z is a q-vector.

A basis matrix, B, for (2.2), under suitable permutation, would generally have the following form:

$$B = \begin{bmatrix} 1 & & 1 & 0 & & 0 & 0 & & 0 \\ & \cdots & & & \cdots & & & \cdots & \\ \sigma_{i_1} a_{i_1} & & \sigma_{i_{nb}} a_{i_{nb}} & \sigma_{j_1} d_{j_1} & & \sigma_{j_{pb}} d_{j_{pb}} & g_{k_1} & & g_{k_{qb}} \end{bmatrix}. \tag{2.3}$$

Each σ appearing above represents a $+1$ or a -1. The columns containing $+a_i$ are associated with the variables t_i; those containing $-a_i$ are associated with u_i; those containing $+d_j$ are associated with v_j; those containing $-d_j$ are associated with w_j, and those containing g_k are associated with z_k. Any segment of columns depicted above, of course, might be absent in a particular basis matrix. The single feature of B that must not be absent is the fact that its columns are linearly independent and span the column space of the constraint matrix in (2.2).

The variables y of the original problem, together with the value y_0 introduced by the reformulation (2.1), are the dual variables of (2.2). We have

$$B^T \begin{bmatrix} y_0 \\ y \end{bmatrix} = (\text{basic costs}), \tag{2.4}$$

where "basic costs" stands for the vector of those components of coefficient array $[c^T - c^T f^T - f^T h^T]$ that correspond to B. The basic variables, that is, the t, u, v, w, and z components corresponding to the columns of B, will be designated by $\lambda_B = [\lambda_1, \cdots, \lambda_{nb+pb+qb}]^T$ and the nonbasic variables will be designated by $\lambda_N = [\lambda_{nb+pb+qb+1}, \cdots, \lambda_{2n+2p+q}]^T$. We have

$$B\lambda_B = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.5}$$

and $\lambda_B \geqq 0$. If $\lambda = [\lambda_B^T, \lambda_N^T]^T = [\lambda_B^T, 0^T]^T$ is optimal for (2.2), then the values of y_0 and y corresponding to B are optimal for (1.1).

Briefly, a test problem for (1.1), with a known solution y and a known value for $\|c - A^T y\|_\infty$ can be generated by

(a) selecting suitable columns of A, D, and G, together with suitable signs σ, to form the basis matrix B;
(b) selecting additional columns of A, D, and G to complete the matrices of (1.1);
(c) setting the basic variables λ_B in such a way that B is feasible for (2.2);
(d) selecting a desired solution $[y_0, y_1, \cdots, y_m]^T$;
(e) and, finally, determining c, f, and h so that B is forced to be optimal for (2.2).

As a start, consider (1.1) in its simplest form, together with the correspondingly simplified version of (2.2), where no constraints are imposed, A has full rank, and $n \geqq m+1$. For B to be feasible, $\lambda_i \geqq 0$, $\sigma_i \in \{-1, +1\}$, and a_i must be chosen so that (2.5) is satisfied. We can begin by choosing m linearly independent vectors a_i together with m signs $\sigma_i = \pm 1$. Let

$$[1, \sigma_i a_i^T]^T \quad i = 1, \cdots, m,$$

be the first m columns of B. Select

$$\lambda_{m+1} > 0, \quad \sigma_{m+1} = \pm 1, \quad \text{and } \lambda_1, \cdots, \lambda_m \geqq 0 \quad \text{(not all zero)},$$

and normalize $\lambda_1, \cdots, \lambda_{m+1}$ so that

$$\sum_{i=1}^{m+1} \lambda_i = 1. \tag{2.6}$$

Set

$$a_{m+1} = \frac{-1}{\lambda_{m+1}\sigma_{m+1}} \sum_{i=1}^{m} \lambda_i \sigma_i a_i,$$

and let $[1, \sigma_{m+1} a_{m+1}^T]^T$ be the final colmun of B. It is easy to see that all of the columns of B are linearly independent by this definition, and that

$$\sum_{i=1}^{m+1} \lambda_i (\sigma_i a_i) = 0, \tag{2.7}$$

so that (2.5) holds. Any additional vectors $a_{m+2}, \cdots, a_n$ may now be chosen to complete the matrix A.

In order that any given $y_0 \geqq 0$ and $y = [y_1, \cdots, y_m]^T$ represent consistent dual variable values for B and λ as constructed above, it is necessary that the vector c be chosen such that the basic costs, $c_B = [\sigma_1 c_1, \cdots, \sigma_{m+1} c_{m+1}]^T$, satisfy:

$$B^T \begin{bmatrix} y_0 \\ y \end{bmatrix} = c_B,$$

that is,

$$c_j = a_j^T y + \sigma_j y_0 \quad \text{for } j = 1, \cdots, m+1. \tag{2.8}$$

In order for B to be optimal for (2.2) with such a set of values for y_0, y, and $c_1, \cdots, c_{m+1}$, it is further necessary that the reduced costs associated with the nonbasic portion of (2.2) be nonpositive, that is;

$$-c_j - [y_0, y^T] \begin{bmatrix} 1 \\ -a_j \end{bmatrix} \leqq 0, \qquad c_j - [y_0, y^T] \begin{bmatrix} 1 \\ a_j \end{bmatrix} \leqq 0,$$

or more succinctly,

$$a_j^T y + y_0 \geqq c_j \geqq a_j^T y - y_0 \quad \text{for } j = m+2, \cdots, n. \tag{2.9}$$

Thus, we can choose quite arbitrary values for $y_0 \geqq 0$, $y_1, \cdots, y_m$ to be optimal for (1.1) so long as the vector c is subsequently chosen in accord with (2.8) and (2.9). To

make matters more interesting, of course, we could now permute the columns of A, and the components of c correspondingly, before handing the problem over to an algorithm to be solved.

Equations (2.6)-(2.9) agree with the conditions given in [16] for y to minimize $y_0 = \|c - A^T y\|_\infty$.

Even in the simplified construction process above, there is the opportunity to specify all save one of the n-vectors a_i arbitrarily (so long as m of them are linearly independent). We can specify the solution $y = [y_1, \cdots, y_m]^T$ and the minimum norm y_0 arbitrarily. There is considerable latitude in specifying which $m+1$ residuals $c_i - a_i^T y$ should have maximal magnitude, and we can specify what their signs should be. We are restricted primarily in our choice of the components of c, but even here there is flexibility. If $y_0 > 0$, the components of $c_{m+2}, \cdots, c_n$ are restricted only to an interval.

Solutions y that are determined in an ill-conditioned way can be constructed by selecting $a_1, \cdots, a_m$ to be nearly linearly dependent. Finally, y is guaranteed to be unique if the inequalities of (2.9) are all strict and $\lambda_i > 0$ for $i = 1, \cdots, m+1$.

There is room to experiment with various forms of nonuniqueness and/or degeneracy as follows. Suppose that $\lambda_i = 0$ for values of $i \in \{1, \cdots, m\}$; then the basis matrix B and the nonbasic portion of the constraint matrix in (2.2) may not be clearly separated. Columns of one can often be exchanged with columns of the other, with no change in λ, but with different y_0, y for each different choice of B. On the other hand, suppose that c_j has been chosen to satisfy not only (2.8) for the values of j indicated, but also $c_j = a_j^T y \pm y_0$ for one or more values of $j \in \{m+2, \cdots, n\}$. Then an excess of constraints are active at $[y_0, y^T]^T$ with regard to (2.1), in the simplified form being considered. Such a point corresponds, potentially, to alternative points λ in (2.2). For technical reasons, many algorithms used for solving (1.1) have difficulties if one or the other of these situations prevail.

In the following section, we present a plan for constructing problems (1.1) in more generality based on extensions to the above observations.

3. Constraints and rank. The following method, which is based on the discussion in the previous section, can be used to construct problems (1.1) with known solutions y and known $y_0 = \|c - A^T y\|_\infty$, even in cases where the matrices A, D, and G are allowed to be deficient in rank. Let m, n, p, q, nb, pb, and qb have the following meanings:

m is the number of unknowns (rows in $[A\ D\ G]$);
n is the number of equations (columns in A);
p is the number of equality constraints (columns in D);
q is the number of inequality constraints (columns in G);
nb is the number of columns of A to be used in the optimal basis;
pb is the number of columns of D to be used in the optimal basis;
and qb is the number of columns of G to be used in the optimal basis.

The construction will be carried out with the following inequalities in mind:

$$1 \leqq m < n + p + q, \quad 0 \leqq p, \quad 0 \leqq q,$$

$$1 \leqq nb \leqq n,$$

$$1 \leqq pb \leqq p \quad \text{if } 1 \leqq p, \quad pb = \operatorname{rank}(D),$$

$$0 \leqq qb \leqq q, \quad \text{and}$$

$$2 \leqq nb + pb + qb = r + 1 \leqq m + 1,$$

where r is the rank of matrix $[A\ D\ G]$.

If any process described in the construction below contradicts one of these inequalities for particular values of m, n, etc., then that process should be ignored (e.g., the second part of Step 2 disappears if $qb = 0$). In the construction, $\bar{a}_i$, $\bar{d}_i$, and $\bar{g}_i$ denote the first r components of a_i, d_i, and g_i, respectively.

A fundamental process required below has the form,

"choose vectors $x_\rho, \cdots, x_\pi$ of a specified dimension to be such that $x_1, \cdots, x_{\rho-1}, x_\rho, \cdots, x_\pi$ are linearly independent."

To obtain x_i for $\rho \leqq i \leqq \pi$, it is useful to assume that the matrix $[x_1, \cdots, x_{i-1}]$ has been decomposed into QR form. The selection process can make use of this as follows. Let

$$[W \quad Z]\begin{bmatrix} R \\ 0 \end{bmatrix}$$

represent the decomposition, where R is a right-triangular matrix of order $i-1$, W represents the left-hand $i-1$ columns of the orthogonal matrix Q, and Z represents the remaining columns. Obtain a candidate vector $\hat{x} \neq 0$. If $\|Z^T\hat{x}\|_2/\|\hat{x}\|_2$ is smaller than a given tolerance, $\hat{x}$ is to be rejected, a new candidate is to be obtained, and the testing is repeated. Otherwise, $\hat{x}$ is accepted as x_i, the QR decomposition is updated to reflect this addition, and the next value of i in sequence is taken. (In generating random matrices by this process, we have observed that rejection rarely occurs. It appears usual for random matrices to have full rank.) For a discussion of QR decompositions of matrices and of their updating, as well as an indication of why the above ratio constitutes a numerical test for linear dependence, see Lawson and Hanson [24].

CONSTRUCTION

Step 1. Choose $\bar{a}_1, \cdots, \bar{a}_{nb-1}$ to be vectors of dimension r that are linearly independent.

Step 2. Choose $\bar{d}_1, \cdots, \bar{d}_{pb}$ to be vectors of dimension r such that $\bar{a}_1, \cdots, \bar{a}_{nb-1}$, $\bar{d}_1, \cdots, \bar{d}_{pb}$ are linearly independent.
Choose $\bar{g}_1, \cdots, \bar{g}_{qb}$ to be vectors of dimension r such that $\bar{a}_1, \cdots, \bar{a}_{nb-1}$, $\bar{d}_1, \cdots, \bar{d}_{pb}$, $\bar{g}_1, \cdots, \bar{g}_{qb}$ are linearly independent.

Step 3. Choose $\lambda_{nb} > 0$ and $\lambda_1, \cdots, \lambda_{nb-1}, \lambda_{nb+1}, \cdots, \lambda_{nb+pb+qb} \geqq 0$ (not all zero).
Normalize $\lambda_1, \cdots, \lambda_{nb}$ so that $\sum_{i=1}^{nb} \lambda_i = 1$.
Choose $\sigma_1, \cdots, \sigma_{nb+pb} \in \{-1, +1\}$.
Set

$$\bar{a}_{nb} = \frac{-1}{\lambda_{nb}\sigma_{nb}}\left\{\sum_{i=1}^{nb-1} \lambda_i\sigma_i\bar{a}_i + \sum_{i=1}^{pb} \lambda_{nb+i}\sigma_{nb+i}\bar{d}_i + \sum_{i=1}^{qb} \lambda_{nb+pb+i}\bar{g}_i\right\}.$$

Step 4. Let $\bar{a}_{nb+1}, \cdots, \bar{a}_n$ be linear combinations of $\bar{a}_1, \cdots, \bar{a}_{nb-1}$, $\bar{d}_1, \cdots, \bar{d}_{pb}$, $\bar{g}_1, \cdots, \bar{g}_{qb}$.

Step 5. Let $\bar{d}_{pb+1}, \cdots, \bar{d}_p$ be linear combinations of $\bar{d}_1, \cdots, \bar{d}_{pb}$.
Let $\bar{g}_{qb+1}, \cdots, \bar{g}_q$ be linear combinations of $\bar{a}_1, \cdots, \bar{a}_{nb-1}$, $\bar{d}_1, \cdots, \bar{d}_{pb}$, $\bar{g}_1, \cdots, g_{qb}$.

Step 6. Let rows $r+1, \cdots, m$ of

$$[A \; D \; G] = [a_1, \cdots, a_n, d_1, \cdots, d_p, g_1, \cdots, g_q]$$

be linear combinations of the r rows of

$$[\bar{a}_1, \cdots, \bar{a}_n, \bar{d}_1, \cdots, \bar{d}_p, \bar{g}_1, \cdots, \bar{g}_q].$$

Let $[1, \sigma_j a_j^T]^T$, for $j = 1, \cdots, nb$, be the first nb columns of B.

Let $[0, \sigma_{nb+j} d_j^T]^T$, for $j = 1, \cdots, pb$, be the next pb columns of B.
Let $[0, g_j^T]^T$, for $j = 1, \cdots, qb$, be the next qb columns of B.

Step 7. Select desired values for $y_0 \geqq 0$, $y_1, \cdots, y_m$.
For $j = 1, \cdots, nb$ set $c_j = a_j^T y + \sigma_j y_0$.
For $j = nb+1, \cdots, n$ set $c_j = a_j^T y + u_j y_0$ where u_j is a number satisfying $u_j \in [-1, 1]$.
For $j = 1, \cdots, p$ set $f_j = d_j^T y$.
For $j = 1, \cdots, qb$ set $h_j = g_j^T y$.
For $j = qb+1, \cdots, q$ set $h_j = g_j^T y - v_j$ where v_j is a number satisfying $v_j \geqq 0$.
Let $c = [c_1, \cdots, c_n]$, $f = [f_1, \cdots, f_p]$, and $h = [h_1, \cdots, h_q]$.

The above process has generated $nb + pb + qb$ linearly independent columns of a basis matrix B. It has also established values for basic primal variables $\lambda_i \geqq 0$, $i = 1, \cdots, nb + pb + qb$. (By the construction of column nb of B, we are assured that the vector $[1, 0^T]^T$ of dimension $m+1$ lies in the space spanned by the $nb + pb + qb$ columns of B that have been chosen.) B now has the form given in (2.3) and satisfies (2.5).

The generated vector $[y_0, y^T]^T$ will be an optimal solution to problem (1.1) as it applies to these arrays A, D, G, c, f, and h. B will be an optimal basis for problem (2.2), and λ_B will constitute optimal primal basic variables for problem (2.2). Residuals $(c_j - a_j^T y)$ will have maximal magnitude y_0 and signs σ_j for $j = 1, \cdots, nb$ and for any $j \in \{nb+1, \cdots, n\}$ for which $u_j = \pm 1$. Furthermore, at the point y, inequality constraints indexed by $j = 1, \cdots, qb$ will be active (satisfied with equality), along with any indexed by $j \in \{qb+1, \cdots, q\}$ for which $v_j = 0$.

The optimal solutions λ_B and $[y_0, y^T]^T$ as constructed above should be obtained by algorithms for discrete linear l_∞ approximation if the solutions are unique. This will be guaranteed if $[A\ D\ G]$ and D have full rank, which can be arranged by setting $nb + pb + qb = m+1$ and $pb = p$; if degeneracy is avoided, which can be arranged by making $\lambda_B > 0$; and if the reduced costs are strictly separated from their critical values, which means that $y_0 > 0$, $-1 < u_j < +1$ for all u_j, and $v_j > 0$ for all v_j chosen in Step 7.

The simplest way to construct problems that are potentially degenerate or non-unique, in imitation of what was proposed for the simplified construction in the previous section, involves: (1) setting some (but not all) of $\lambda_1, \cdots, \lambda_{nb-1}, \lambda_{nb+1}, \cdots, \lambda_{nb+pb+qb}$ to zero; (2) constructing redundant constraints, e.g., by setting $p > pb$ to produce an excess of equality constraints and/or by setting $v_j = 0$ for some of the v_j generated in Step 7, to produce an excess of active inequality constraints; (3) allowing $u_j = \pm 1$ for some of the quantities u_j chosen in Step 7; and (4) setting $nb + pb + qb < m+1$ so as to force $[A\ D\ G]$ to have less than full rank, and hence, a nontrivial null space.

Finally, infeasible problems can be constructed by arranging that the arrays D, f, G, and h represent inconsistent equalities and inequalities. For example, inconsistent equalities can be arranged by setting $d_j = d_i$ and $f_i \neq f_j$ for some i, j and inconsistent inequalities can be arranged by setting $g_j = -g_i$ and $h_i > -h_j$ for some i, j.

Naturally, after the construction phase is complete, the columns of

$$\begin{bmatrix} c^T \\ A \end{bmatrix}, \quad \begin{bmatrix} f^T \\ D \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} h^T \\ G \end{bmatrix}$$

can be permuted.

In the following section we suggest how the process of constructing the matrix A can be modified to give it the structure associated with data-fitting problems.

4. Data-fitting structure. The problem of finding an approximating function $L(y) = \sum_{i=1}^m y_i \psi_i(t)$ to n data points (t_j, c_j) leads to a system of linear equations $c = A^T y$,

where

$$A = \begin{bmatrix} \psi_1(t_1) & \psi_1(t_2) & \cdots & \psi_1(t_n) \\ \psi_2(t_1) & \psi_2(t_2) & \cdots & \psi_2(t_n) \\ \vdots & \vdots & & \vdots \\ \psi_m(t_1) & \psi_m(t_2) & \cdots & \psi_m(t_n) \end{bmatrix} \quad \text{and} \quad c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}. \tag{4.1}$$

A problem of the form (1.1) arises in such a context if $n > m$. Linear constraints on this problem may come from sources and considerations quite independent of those that produce A and c, so the structure of A does not necessarily show up again in the matrices D or G.

Matrices having a structure that mimics the one shown for A can be formed by choosing functions $\psi_1, \cdots, \psi_m$ and evaluating them at distinct points t_j on an interval $[\alpha, \beta]$. Some obvious choices, with which we have experimented, are

(4.2)

(a) $\psi_i(t) = t^{i-1}$ for $1 \leqq i \leqq m$, and any interval $[\alpha, \beta]$.

(b) $\psi_1(t) = 1$, $\psi_2(t) = t$, $\psi_i(t) = 2t\psi_{i-1}(t) - \psi_{i-2}(t)$ for $3 \leqq i \leqq m$ where t is in the interval $[\alpha, \beta] = [-1, +1]$.

(c) $\psi_1(t) = 1$, $\psi_i(t) = \begin{cases} \cos\left(\frac{i}{2} t\right) & \text{if } i \text{ is even} \\ \sin\left(\frac{i-1}{2} t\right) & \text{if } i \text{ is odd} \end{cases}$ for $2 \leqq i \leqq m$

where t is in the interval $[\alpha, \beta] = [0, 2\pi)$,

but this list is not meant to be exhaustive.

With this setting, the rank of A will not necessarily be under our control. We begin by forming a pool of nonzero vectors

$$a_j = [\psi_1(t_j), \cdots, \psi_m(t_j)]^T$$

by selecting distinct numbers $t_j \in [\alpha, \beta]$ for $j = 1, \cdots, n$. These vectors can be filtered using the QR techniques of § 3 to determine their rank, r (viewed as the columns of a matrix), and to order them so that $[\psi_1(t_j), \cdots, \psi_m(t_j)]$ are linearly independent for $j = 1, \cdots, r$. This involves beginning with any one of the vectors and regarding it to be a matrix, Ψ. Recursively, given Ψ, its QR decomposition is formed, where $Q = [W Z]$ and Z spans the null space of Ψ^T, and the vectors a_j not included in Ψ are tested by forming $\|Z^T a_j\| / \|a_j\|$. The a_j corresponding to the largest of these ratios is added as the next column of Ψ, if it is larger than a chosen tolerance. This process continues until no remaining a_j forms a ratio larger than the tolerance. At this point Ψ will have r linearly independent columns. A similar process can now be used on the rows of Ψ to rearrange them so that the first r are linearly independent. Having done this, we may assume that the values t_j and the functions ψ_i have been reordered so that $[\psi_1(t_j), \cdots, \psi_m(t_j)]$ are linearly independent for $j = 1, \cdots, r$, and that the subvectors $[\psi_j(t_j), \cdots, \psi_r(t_j)]$ are also independent.

The testing and rearranging can be ignored if it happens that the functions $\psi_1, \cdots, \psi_m$ form a Chebyshev system on the interval $[\alpha, \beta]$. Then, for any chosen, distinct values $\alpha \leqq t_1, \cdots, t_n \leqq \beta$, $n \geqq m$, the matrix A shown in (4.1) will have rank $r = m$. More strongly, any selection of m of its columns will be linearly independent. If matrices of some given rank $r < m$ are desired with this structure, we may let $\psi_1, \cdots, \psi_r$ form a Chebyshev system, and define ψ_i for $i = r+1, \cdots, m$ to be linear combinations of $\psi_1, \cdots, \psi_r$.

Assuming that r has been determined and integers $nb+pb+qb=r+1$ have been chosen, we propose modifying Steps 1, 3, and 4 of the construction that was given in the previous section as follows:

Step 1′. Using the ψ_i and t_j just described, let $\bar{a}_j=[\psi_1(t_j),\cdots,\psi_r(t_j)]^T$, for $j=1,\cdots,nb-1$.

Step 3′. Let $\bar{a}_{nb}=[\psi_1(t_{nb}),\cdots,\psi_r(t_{nb})]^T$.
This vector will be a linear combination of the vectors $\bar{a}_1,\cdots,\bar{a}_{nb-1}$, $\bar{d}_1,\cdots,\bar{d}_{pb}$, $\bar{g}_1,\cdots,\bar{g}_{qb}$.
Determine $\lambda_1,\cdots,\lambda_{nb-1},\lambda_{nb+1},\cdots,\lambda_{nb+pb+qb}$ by first solving

$$[\bar{a}_1,\cdots,\bar{a}_{nb-1},\bar{d}_1,\cdots,\bar{d}_{pb},\bar{g}_1,\cdots,\bar{g}_{qb}]\begin{bmatrix}\gamma_1\\ \vdots\\ \gamma_{nb-1}\\ \gamma_{nb+1}\\ \vdots\\ \gamma_{nb+pb+qb}\end{bmatrix}=\bar{a}_{nb},$$

and then setting $\sigma_j=\operatorname{sgn}(\gamma_j)$ and $\lambda_j=|\gamma_j|$, for $j=1,\cdots,nb-1,nb+1,\cdots,nb+pb+qb$, and finally setting $\lambda_{nb}=1$ and $\sigma_{nb}=-1$.
Normalize $\lambda_1,\cdots,\lambda_{nb}$ so that $\sum_{i=1}^{nb}\lambda_i=1$.

Step 4′. Let $\bar{a}_j=[\psi_1(t_j),\cdots,\psi_r(t_j)]^T$ for $j=nb+1,\cdots,n$.

We note that matrices A generated from (4.2a) may be ill-conditioned, since they could contain Vandermonde submatrices. Using (4.2a), the conditioning of the problem can be varied by choosing the t_j values from a uniform distribution in intervals of different sizes. A smaller interval size implies that the t_j values are closer to each other, which implies that the condition numbers of the Vandermonde submatrices (and the problem) are larger. In the case of an ill-conditioned problem (2.2) with a unique solution, the optimal solutions λ_B and $[y_0,y^T]^T$ obtained by a discrete linear l_∞ approximation algorithm may not be very accurate when compared with that generated by the above construction.

5. Choice of distribution for random numbers. In the preceding sections we have left the process of choosing numbers and vectors unspecified. It is a common practice to draw numbers from a distribution such as the uniform or normal. VanDam, Frenk, and Telgen [29] point out that polytopes constructed from randomly generated linear constraints may bias test results for mathematical programming problems. Specifically, they prove that if the entries of the columns of a mattrix are drawn identically and independently from any distribution, the cosines of the angles between the column tends to a limiting value as the dimension of the columns increases, and the variance of the cosines tends to zero.

To see whether this property is relevant for l_∞ approximation problems, we compared the frequency distribution of the cosine of the angles between the column vectors of matrices A generated from uniformly distributed random numbers in $[-1,1]$ and from the three systems of functions given in (4.2) where the t_j are randomly generated from the uniform distribution in $[-1,1]$ for (a) and (b) and $[0,2\pi]$ for (c). We generated matrices A of size m by n where $m=11$, 51, and 101, and $n=200$, so the number of cosine of angles computed for each matrix is $n(n-1)/2=19{,}900$.

For each type and size of matrix, the frequency distribution of cosine of angles is approximately the same for different matrices, so we report the results for only one matrix of each type and size. Tables 1 to 4 contain the relative frequency distribution

TABLE 1
Relative frequency distribution of cosine of angles for random matrices.

Interval	$m = 11$	$m = 51$	$m = 101$
[−.9, −.8)	.00116		
[−.8, −.7)	.00593		
[−.7, −.6)	.01513		
[−.6, −.5)	.02879	.00015	
[−.5, −.4)	.04859	.00161	
[−.4, −.3)	.07151	.01588	.00090
[−.3, −.2)	.09055	.06095	.02116
[−.2, −.1)	.11226	.16487	.14146
[−.1, 0.0)	.12422	.25930	.33854
[0.0, 0.1)	.12271	.25970	.34020
[0.1, 0.2)	.11518	.16106	.13789
[0.2, 0.3)	.09533	.06116	.01849
[0.3, 0.4)	.07010	.01377	.00136
[0.4, 0.5)	.05015	.00151	
[0.5, 0.6)	.02829	.00005	
[0.6, 0.7)	.01392		
[0.7, 0.8)	.00538		
[0.8, 0.9)	.00075		
[0.9, 1.0]	.00005		

TABLE 2
Relative frequency distribution of cosine of angles for matrices generated from (4.2a).

Interval	$m = 11$	$m = 51$	$m = 101$
[0.0, 0.1)	.00467	.01608	.02291
[0.1, 0.2)	.04085	.05065	.05427
[0.2, 0.3)	.06312	.06990	.06668
[0.3, 0.4)	.07528	.07573	.07492
[0.4, 0.5)	.08015	.08156	.08116
[0.5, 0.6)	.08719	.08482	.08839
[0.6, 0.7)	.09377	.09176	0.9352
[0.7, 0.8)	.10568	.10704	.10136
[0.8, 0.9)	.13583	.13065	.12638
[0.9, 1.0]	.31347	.29181	.29040

in intervals of size 0.1 from $[-1, -0.9)$ to $[0.9, 1]$ for random matrices and matrices generated from (4.2a), (4.2b), and (4.2c), respectively. The frequencies are scaled so that they sum to one, and intervals with a freqeuncy of zero are omitted from the tables.

We see the behavior that VanDam, Frenk, and Telgen discovered very clearly for Table 1, which reports on matrices A of the type constructed in the straightforward fashion of § 3. It is gratifying to observe that the proposals of § 4, however, clearly have the power to "counteract" the bias of random vector generation. By selecting evaluation points t_j randomly, and constructing the columns of A from vectors with components of the form $\psi_i(t_j)$, the frequency distribution of angle cosines can be greatly affected. We see from Table 2, in particular, that the frequency distribution for the matrices generated from (4.2a) stay approximately the same as the dimension m

TABLE 3
Relative frequency distribution of cosine of angles for matrices generated from (4.2b).

Interval	$m = 11$	$m = 51$	$m = 101$
[−.3, −.2)		.00196	.00281
[−.2, −.1)	.05101	.02618	.01372
[−.1, 0.0)	.14839	.21663	.23533
[0.0, 0.1)	.29246	.68809	.71628
[0.1, 0.2)	.25352	.02764	.01347
[0.2, 0.3)	.06894	.00342	.00166
[0.3, 0.4)	.01889	.00367	.00146
[0.4, 0.5)	.01889	.00327	.00111
[0.5, 0.6)	.01789	.00357	.00196
[0.6, 0.7)	.01899	.00367	.00231
[0.7, 0.8)	.02166	.00467	.00246
[0.8, 0.9)	.02829	.00558	.00231
[0.9, 1.0]	.06106	.01166	.00513

TABLE 4
Relative frequency distribution of cosine of angles for matrices generated from (4.2c).

Interval	$m = 11$	$m = 51$	$m = 101$
[−.3, −.2)			.00246
[−.2, −.1)	.05467	.02412	.01090
[−.1, 0.0)	.14533	.21603	.22905
[0.0, 0.1)	.29663	.70191	.72955
[0.1, 0.2)	.29447	.02462	.01131
[0.2, 0.3)	.06412	.00337	.00176
[0.3, 0.4)	.01452	.00307	.00221
[0.4, 0.5)	.01528	.00342	.00106
[0.5, 0.6)	.01598	.00332	.00126
[0.6, 0.7)	.01347	.00397	.00166
[0.7, 0.8)	.01759	.00281	.00161
[0.8, 0.9)	.02161	.00432	.00281
[0.9, 1.0]	.04633	.00905	.00437

increases, while the angles tend to be equal as m increases for the other three types of matrices.

Acknowledgment. The authors would like to thank the referees for their useful comments.

REFERENCES

[1] N. N. ABDELMALEK, *The discrete linear one-sided Chebyshev approximation*, J. Inst. Math. Appl., 18 (1976), pp. 888-999.

[2] ———, *The discrete linear restricted Chebyshev approximation*, BIT, 17 (1977), pp. 249-261.

[3] R. D. ARMSTRONG, E. L. FROME, AND D. S. KUNG, *A revised simplex algorithm for the absolute deviation curve fitting problem*, Commun. Statist. B, 8 (1979), pp. 175-190.

[4] R. D. ARMSTRONG AND D. S. KUNG, *A dual method for discrete Chebyshev curve fitting*, Math. Programming, 19 (1980), pp. 186-199.

[5] I. BARRODALE AND C. PHILLIPS, *An improved algorithm for discrete Chebyshev linear approximation*, Proc. Fourth Manitoba Conference on Numerical Mathematics, University of Manitoba, 1974, pp. 177-190.

[6] ———, *Algorithm* 495, *solution of an overdetermined system of linear equations in the Chebyshev norm*, ACM Trans. Math. Software, 1 (1975), pp. 264-270.

[7] R. H. BARTELS, A. R. CONN, AND C. CHARALAMBOUS, *On Cline's direct method for solving overdetermined linear systems in the L-infinity sense*, SIAM J. Numer. Anal., 15 (1978), pp. 255-270.

[8] R. H. BARTELS AND N. MAHDAVI-AMIRI, *On generating test problems for nonlinear programming algorithms*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 769-798.

[9] R. H. BARTELS, A. R. CONN, AND Y.-Y. LI, *Primal methods are better than dual methods for solving overdetermined linear systems in the l_∞ sense*? CS-87-12, University of Waterloo, 1987. (SIAM J. Numer. Anal., to appear.)

[10] P. BLOOMFIELD AND W. L. STEIGER, *Least absolute deviations curve-fitting*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 290-301.

[11] ———, *Least Absolute Deviations*, Birkhäuser, Boston, 1983.

[12] M. BRANNIGAN, *The Strict Chebyshev Solution of Overdetermined Systems of Linear Equations with Rank Deficient Matrix*, CSIR, South Africa, 1980.

[13] J. A. CADZOW, *A finite algorithm for the minimum l_∞ solution to a system of consistent linear equations*, SIAM J. Numer. Anal., 10 (1973), pp. 607-617.

[14] ———, *An efficient algorithmic procedure for obtaining a minimum l_∞ norm solution to a system of consistent linear equations*, SIAM J. Numer. Anal., 11 (1974), pp. 1151-1165.

[15] A. CHARNES, W. M. RAIKE, J. D. STUTZ, AND A. S. WALTERS, *On generation of test problems for linear programming codes*, Commun. ACM, 17 (1974), pp. 583-586.

[16] E. W. CHENEY, *Introduction to Approximation Theory*, McGraw-Hill, New York, 1966.

[17] V. CHVÁTAL, *Linear Programming*, W. H. Freeman, New York, 1980.

[18] A. K. CLINE, *A descent method for the uniform solution to overdetermined systems of linear equations*, SIAM J. Numer. Anal., 13 (1976), pp. 293-309.

[19] C. S. DURIS AND M. G. TEMPLE, *A finite step algorithm for determining the "strict" Chebyshev solution to $Ax = b$*, SIAM J. Numer. Anal., 10 (1973), pp. 690-699.

[20] W. HOCK AND K. SCHITTKOWSKI, *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economic and Mathematical Systems 187, Springer-Verlag, Berlin, New York, 1981.

[21] K. L. HOFFMAN AND D. R. SHIER, *A test problem generator for discrete linear l_1 approximation problems*, ACM Trans. Math. Software, 6 (1980), pp. 587-593.

[22] M. J. HOPPER AND M. J. D. POWELL, *A technique that gains speed and accuracy in the minimax solution of overdetermined linear equations*, in Mathematical Software III, Academic Press, New York, 1977.

[23] B. JOE AND R. H. BARTELS, *An exact penalty method for constrained, discrete linear l_∞ data fitting*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 69-84.

[24] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[25] W. M. MICHAELS AND R. P. O'NEILL, *A mathematical program generator* MPGENR, ACM Trans. Math. Software, 6 (1980), pp. 31-44.

[26] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM, *Algorithm* 566: FORTRAN *subroutines for testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 136-140.

[27] ———, *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 17-41.

[28] K. SCHITTKOWSKI, *Randomly generated nonlinear programming test problems*, 9th IFIP Conference on Optimization Techniques, 1979.

[29] W. B. VANDAM, J. B. G. FRENK, AND J. TELGEN, *Randomly generated polytopes for testing mathematical programming algorithms*, Math. Programming, 26 (1983), pp. 172-181.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 562-580, May 1989
© 1989 Society for Industrial and Applied Mathematics
010

AN EFFICIENT TRUST REGION ALGORITHM FOR MINIMIZING NONDIFFERENTIABLE COMPOSITE FUNCTIONS*

EIKI YAMAKAWA†, MASAO FUKUSHIMA†, AND TOSHIHIDE IBARAKI†

Abstract. This paper presents a trust region algorithm for solving the following problem. Minimize $\phi(x)=f(x)+h(c(x))$ over $x\in R^n$, where f and c are smooth functions and h is a polyhedral convex function. Problems of this form include various important applications such as min-max optimization, Chebyshev approximation, and minimization of exact penalty functions in nonlinear programming. The algorithm is an adaptation of a recently proposed successive quadratic programming method for nonlinear programming and makes use of the second-order approximations to both f and c in order to avoid the Maratos effect. It is proved under appropriate assumptions that the algorithm is globally and quadratically convergent to a solution of the problem. Some numerical results exhibiting the effectiveness of the algorithm are also reported.

Key words. nondifferentiable optimization, trust region, Maratos effect, global convergence, second-order convergence

AMS(MOS) subject classifications. 90C30, 65K05, 65D10

1. Introduction.

We consider the following problem:

$$\text{minimize } \phi(x)\triangleq f(x)+h(c(x)), \tag{1.1}$$

where $f: R^n\to R$ and $c: R^n\to R^M$ are twice continuously differentiable, and $h: R^M\to R$ is a polyhedral convex function defined by

$$h(c)=\max_{1\leqq i\leqq m}\{c^Th_i+b_i\} \tag{1.2}$$

for given vectors h_i and constants b_i, $i=1,\cdots,m$.

It is well known [6] that first-order necessary conditions for x^* to be a local minimum of (1.1) are stated as follows. There exists a vector λ^* such that

$$\nabla f(x^*)+\nabla c(x^*)\lambda^*=0, \tag{1.3}$$

$$\lambda^*\in\partial h(c(x^*)), \tag{1.4}$$

where $\partial h(c(x^*))$ denotes the subdifferential of h at $c^*=c(x^*)$. The vector λ^* is called the optimal Lagrange multiplier for (1.1). Since h is defined by (1.2), the subdifferential ∂h is given as follows:

$$\partial h(c)=\left\{\sum_{i=1}^m \mu_i h_i \,\middle|\, \mu_i(c^Th_i+b_i-h(c))=0,\ \mu_i\geqq 0,\ i=1,\cdots,m,\ \sum_{i=1}^m\mu_i=1\right\}. \tag{1.5}$$

Problems of the form (1.1) include various important applications such as min-max optimization, Chebyshev approximation and, among others, minimization of exact penalty functions in nonlinear programming. For example, let us consider the following general nonlinear programming problem:

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && c_i(x)=0, \qquad i=1,2,\cdots,M', \\ &&& c_i(x)\leqq 0, \qquad i=M'+1,\cdots,M. \end{aligned} \tag{1.6}$$

* Received by the editors April 30, 1986; accepted for publication (in revised form) June 14, 1988.

† Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto 606, Japan.

Then, an exact penalty function for problem (1.6) is given by

$$\Phi_t(x) = f(x) + t\left[\sum_{i=1}^{M'} |c_i(x)| + \sum_{i=M'+1}^{M} \max\{0, c_i(x)\}\right], \tag{1.7}$$

where t is a positive penalty parameter. It is well known [15], [17] that, under appropriate assumptions, unconstrained minimizers of $\Phi_t(x)$ yield optimal solutions for problem (1.6), provided that the penalty parameter t is chosen sufficiently large.

For the solution of problem (1.1), two types of approaches have recently attracted much attention. The first approach has primarily been developed for the case in which problem (1.1) actually amounts to minimizing an exact penalty function associated with the nonlinear programming problem (1.6). The underlying idea consists of constructing a quadratic model of the problem (1.6) successively, and hence methods of this type are commonly referred to as successive quadratic programming (SQP) methods [1], [6], [11], [12], [19]-[21]. (See also [2], [13], [14] for similar ideas.) When this quadratic model is appropriate, the methods usually exhibit rapid local convergence. Moreover, global convergence is ensured by incorporating line searches along directions obtained from the quadratic model. A possible difficulty in making use of line searches is, however, that the rapid local convergence property might be hampered by the nondifferentiability of the objective function [3], [21]. This phenomenon is often called the Maratos effect, and a number of attempts have recently been made to devise algorithms that possess both global and rapid local convergence properties [3]-[5], [10], [18].

The second approach also constructs a quadratic model of the problem (1.1) and uses a step restriction or trust region technique to guarantee the validity of the quadratic model as an approximation to the original problem. Fletcher [6]-[8] presents a method based on this idea, and proves that it possesses a global convergence property if the radius of the trust region is adjusted adaptively in every iteration. It is also pointed out in [6] that the quadratic rate of convergence of the method can be guaranteed, if the exact second derivatives of the problem functions are used and the trust region constraint is assumed to become inactive ultimately. Unfortunately, it is possible that the step obtained from the quadratic model fails to reduce the objective value of (1.1) in any neighborhood of the optimal solution, and hence the radius of the region must become arbitrarily small. In such a case, the convergence of the generated sequence becomes slow and the Maratos effect may result. To overcome this difficulty, Fletcher [9] has modified the method by incorporating a procedure called the projection step, and has demonstrated that, under appropriate assumptions, it recovers rapid local convergence, while inheriting a global convergence property from the original method. Various properties of this type of algorithm are also considered in [22]-[25].

In this paper, we present a new trust region algorithm for solving problem (1.1) which possesses both global and superlinear convergence properties. This algorithm first attempts to determine a step using the ordinary quadratic model with trust region in the same manner as in Fletcher [9]. If the step gives a sufficient reduction in the objective function, then it is accepted and is used to determine the next iterate. Otherwise, based on the information obtained from this step, another quadratic model approximating the problem more accurately is constructed and solved under the trust region constraint in order to determine an improved step. In any event, the size of the trust region is adjusted according to the ratio of the actual and predicted reductions caused by the step chosen to determine the next iterate. Note that the quadratic model used to obtain the improved step plays a role similar to the one used in Fletcher's

algorithm [9]. It may be said, however, that the former model depends on the second-order approximation to the original problem more directly than the latter. The idea of utilizing two quadratic models as above is adopted in the approach recently proposed by Fukushima [10] in the context of SQP methods for nonlinear programming.

In the next section, we describe how the new quadratic model is constructed from the second-order information on the original problem. In § 3, we state the algorithm in detail. In § 4, we establish the global and superlinear convergence results. In § 5, we discuss a quasi-Newton modification of the algorithm. Finally, in § 6, we report some numerical results for several test problems in order to demonstrate the effectiveness of the algorithm.

2. Quadratic models. The model algorithm of Fletcher [6]–[8] forms the basis of the algorithm presented in this paper. Let δ^k be the radius of the trust region at the kth iteration, and let x^k and λ^k be estimates of the optimal solution x^* of problem (1.1) and the associated Lagrange multipliers λ^*, respectively. The model algorithm of [6]–[8] solves the following subproblem:

$$\begin{aligned} &\text{minimize} && \psi^k(d) \triangleq q^k(d) + h(l^k(d)) \\ &\text{subject to} && \|d\| \leqq \delta^k, \end{aligned} \tag{2.1}$$

where the function h is defined by (1.2) and

$$q^k(d) = f(x^k) + \nabla f(x^k)^T d + \tfrac{1}{2} d^T W^k d,$$

$$l^k(d) = c(x^k) + \nabla c(x^k)^T d,$$

$$W^k = \nabla^2 f(x^k) + \sum_{i=1}^{M} \lambda_i^k \nabla^2 c_i(x^k).$$

Note that the quadratic function q^k contains information about curvature of both f and c, implying that the function ψ^k is a second-order approximation of the function ϕ around x^k. We hope that δ^k is selected in such a way that $\psi^k(d)$ is a reasonable approximation to $\phi(x^k + d)$ within the constraint of (2.1). The norm $\|\cdot\|$ in (2.1) is arbitrary, but the L_∞ norm is convenient in practice, because subproblem (2.1) may then be transformed into a quadratic programming problem.

Now, let $\tilde{d}^k$ be a solution of (2.1). If $\tilde{d}^k$ reduces the objective function ϕ of (1.1), then the next iterate x^{k+1} is chosen to be $x^k + \tilde{d}^k$ and λ^{k+1} is set to the optimal Lagrange multipliers $\tilde{\lambda}^k$ of (2.1), i.e., $\tilde{\lambda}^k \in \partial h(l^k(\tilde{d}^k))$ (cf. (1.5)). Otherwise, the radius of the trust region must be controlled to enforce the reduction of the function ϕ. In practice, the parameter δ^k is adjusted on the basis of the ratio of the actual and predicted reductions

$$\tilde{\rho}^k = \frac{\phi(x^k) - \phi(x^k + \tilde{d}^k)}{\phi(x^k) - \psi^k(\tilde{d}^k)}. \tag{2.2}$$

Then it can be shown under mild conditions that the generated sequence $\{x^k\}$ is globally convergent to a point at which first-order optimality conditions hold [8, Thm. 3.1]. Moreover, if the trust region constraint in (2.1) happens to be inactive at $\tilde{d}^k$ for all k sufficiently large, the convergence rate is shown to be superlinear under suitable assumptions including linear independence, strict complementarity and second-order sufficient conditions at the solution of (1.1). The last statement is a consequence of the fact that, if x^k is sufficiently close to x^* and the trust region constraint is actually

inactive, then solving (2.1) is essentially equivalent to Newton iteration and we have

$$\frac{\|x^k+\tilde{d}^k-x^*\|}{\|x^k-x^*\|}=O(\max\{\|x^k-x^*\|,\|\lambda^k-\lambda^*\|\}) \tag{2.3}$$

for each k [6]-[8]. It then follows from (2.3) that the convergence rate is superlinear. In addition, if

$$\|\lambda^k-\lambda^*\|=O(\|x^k-x^*\|)$$

holds, the sequence $\{x^k\}$ converges to x^* at a quadratic rate.

It should be noted that the above arguments rely entirely on the assumption that the step $\tilde{d}^k$ is always acceptable because it yields a sufficient decrease in the objective function ϕ. Unfortunately, it may happen that, for some x^k in a neighborhood of x^*, $\tilde{d}^k$ fails to reduce the function ϕ even though the trust region is inactive at $\tilde{d}^k$. This unfavorable phenomenon, which corresponds to the Maratos effect, is due to the fact that the nondifferentiable term $h(l^k(d))$ in the function ψ^k only accounts for the first-order approximation to the function c, implying that

$$h(c(x^k+\tilde{d}^k))-h(l^k(\tilde{d}^k))=O(\|\tilde{d}^k\|^2).$$

(See [9] for more details.)

Consequently, we can overcome this difficulty if we are able to construct another quadratic model:

$$\begin{aligned}&\text{minimize} && \hat{\psi}^k(d)\triangleq\hat{q}^k(d)+h(\hat{l}^k(d))\\ &\text{subject to} && \|d\|\leqq\delta^k,\end{aligned} \tag{2.4}$$

where $\hat{q}^k$ and $\hat{l}^k$ are quadratic and linear functions, respectively, in such a way that a solution $\hat{d}^k$ of (2.4) satisfies

$$h(c(x^k+\hat{d}^k))-h(\hat{l}^k(\hat{d}^k))=o(\|\hat{d}^k\|^2) \tag{2.5}$$

$$\frac{\|x^k+\hat{d}^k-x^*\|}{\|x^k-x^*\|}=O(\max\{\|x^k-x^*\|,\|\lambda^k-\lambda^*\|\}). \tag{2.6}$$

In fact, (2.5) indicates that the predicted reduction $\phi(x^k)-\psi^k(\tilde{d}^k)$ approximates the actual reduction $\phi(x^k)-\phi(x^k+\hat{d}^k)$ to second order. Hence, if there exists a constant $\beta>0$ such that $\phi(x^k)-\psi^k(\tilde{d}^k)\geqq\beta\|\tilde{d}^k\|^2$ for k large enough (this is true when the second-order sufficient condition holds), then it is possible to show that

$$\lim_{k\to\infty}\hat{\rho}^k=1, \tag{2.7}$$

where $\hat{\rho}^k$ is the ratio defined by

$$\hat{\rho}^k=\frac{\phi(x^k)-\phi(x^k+\hat{d}^k)}{\phi(x^k)-\psi^k(\tilde{d}^k)}. \tag{2.8}$$

Since (2.7) ensures that the step $\hat{d}^k$ gives a sufficient decrease in the function ϕ for all k large enough, the next iterate x^{k+1} may be chosen as $x^k+\hat{d}^k$. Further, the rapid local convergence of $\{x^k\}$ will be preserved by virtue of (2.6).

Now, we proceed to define $\hat{q}^k(d)$ and $\hat{l}^k(d)$ in the new quadratic model (2.4) for which (2.5) and (2.6) are satisfied. Given the solution $\tilde{d}^k$ of (2.1) and corresponding

Lagrange multipliers $\tilde{\lambda}^k$, let us define

$$p^k = \nabla f(x^k) - \sum_{i=1}^{M} \tilde{\lambda}_i^k r_i^k, \tag{2.9}$$

$$a_i^k = \nabla c_i(x^k) + r_i^k, \qquad i = 1, \cdots, M, \tag{2.10}$$

where the vectors r_i^k are given by

$$r_i^k = \tfrac{1}{2}\nabla^2 c_i(x^k)\tilde{d}^k, \qquad i = 1, \cdots, M. \tag{2.11}$$

Note that calculation of the above vectors does not require the extra work of evaluating the second-order derivatives $\nabla^2 c_i(x^k)$, because they have already been used to evaluate the matrix W^k in problem (2.1). Next we define the functions $\hat{q}^k(d)$ and $\hat{l}^k(d)$, which appear in (2.4) as follows:

$$\hat{q}^k(d) = f(x^k) + (p^k)^T d + \tfrac{1}{2} d^T W^k d, \tag{2.12}$$

$$\hat{l}^k(d) = c(x^k) + (A^k)^T d, \tag{2.13}$$

where A^k denotes the $n \times M$ matrix with columns $a_1^k, \cdots, a_M^k$. Then, it follows from (2.10), (2.11), and (2.13) that

$$\hat{l}_i^k(d) = c_i(x^k) + \nabla c_i(x^k)^T d + \tfrac{1}{2}(\tilde{d}^k)^T \nabla^2 c_i(x^k) d. \tag{2.14}$$

Since the multipliers $\tilde{\lambda}^k \in \partial h(l^k(\tilde{d}^k))$ are bounded (cf. (1.5)), we can expect from (2.9)-(2.11) that the solution $\hat{d}^k$ of (2.4) is close to the solution $\tilde{d}^k$ of (2.1), when $\|\tilde{d}^k\|$ is sufficiently small. Thus (2.14) will ensure that $\hat{l}^k(\hat{d}^k)$ approximates $c(x^k + \hat{d}^k)$ to second order, implying that (2.5) is satisfied. Also, if $\hat{d}^k$ is sufficiently close to $\tilde{d}^k$, then (2.6) is expected to hold from (2.3). Rigorous proofs of these conjectures will be given in § 4.

Incidentally, it is worth pointing out that a condition similar to (2.7) is also satisfied by the second-order correction algorithm of Fletcher [9], which makes use of a projection step obtained from a subproblem involving $c(x^k + \tilde{d}^k)$ and $\nabla c(x^k)$. In fact, the second subproblem of Fletcher's algorithm may be rewritten as follows:

$$\begin{aligned} &\text{minimize} && q^k(d) + h(\bar{c}^k + \nabla c(x^k)^T d) \\ &\text{subject to} && \|d\| \leqq \delta^k, \end{aligned} \tag{2.15}$$

where $\bar{c}^k = c(x^k + \tilde{d}^k) - \nabla c(x^k)^T \tilde{d}^k$. Since the relation

$$\bar{c}_i^k + \nabla c_i(x^k)^T d = c_i(x^k) + \nabla c_i(x^k)^T d + \tfrac{1}{2}(\tilde{d}^k)^T \nabla^2 c_i(x^k)\tilde{d}^k + o(\|\tilde{d}^k\|^2)$$

holds, we may also expect that $\bar{c}^k + \nabla c(x^k)^T d$ agrees with $c(x^k + d)$ to second order when evaluated at a solution of (2.15). Fletcher's algorithm requires an extra function value $c(x^k + \tilde{d}^k)$ to construct subproblem (2.15), while no extra function values or gradient values are needed in (2.12) and (2.13). However, when the second derivatives are unavailable or very expensive to evaluate, (2.12) and (2.13) should be modified by replacing the definition (2.11) of r_i^k with the one that uses only first derivatives of c_i at $x^k + \tilde{d}^k$ (see (5.1) below). This additional gradient evaluation may still be expensive, especially when n and M are large.

3. Algorithm. In this section, we present a trust region algorithm that is based on not only the ordinary quadratic model (2.1) but also on the modified quadratic model

(2.4) using the approximations (2.12) and (2.13). The algorithm requires the solution $\tilde{d}^k$ of subproblem (2.1) at every iteration, while subproblem (2.4) is solved only when we believe $\tilde{d}^k$ may cause the Maratos effect. By the arguments in the previous section, it would be helpful to calculate a solution $\hat{d}^k$ of (2.4) only if the following conditions (i)-(iii) are met simultaneously: (i) A sufficient decrease in ϕ is not obtained by $\tilde{d}^k$; (ii) x^k is close to the surface of discontinuity of $\nabla\phi$; and (iii) the error $h(c(x^k+\tilde{d}^k))-h(\hat{l}^k(\tilde{d}^k))$ is significantly smaller than the error $h(c(x^k+\tilde{d}^k))-h(l^k(\tilde{d}^k))$. A rationale of condition (iii) may be explained as follows. Let $\hat{d}^k$ be calculated. Then, since we have

$$\hat{d}^k-\tilde{d}^k=O(\|\tilde{d}^k\|^2) \quad \text{(see (4.18) below)},$$

$$h(c(x^k+\tilde{d}^k))-h(l^k(\tilde{d}^k))=O(\|\tilde{d}^k\|^2),$$

$$h(c(x^k+\hat{d}^k))-h(\hat{l}^k(\hat{d}^k))=h(c(x^k+\tilde{d}^k))-h(\hat{l}^k(\tilde{d}^k))+O(\|\tilde{d}^k\|)(\hat{d}^k-\tilde{d}^k)+o(\|\tilde{d}^k\|^2),$$

condition (iii) would imply that the error $h(c(x^k+\hat{d}^k))-h(\hat{l}^k(\hat{d}^k))$ is significantly smaller than $h(c(x^k+\tilde{d}^k))-h(l^k(\tilde{d}^k))$, when $\tilde{d}^k$ is small enough, so that $h(\hat{l}^k(\hat{d}^k))$ is a good approximation to $h(c(x^k+\hat{d}^k))$ (cf. (2.5)).

Condition (i) can be checked from the value of $\tilde{\rho}^k$ given by (2.2). Condition (ii) may be verified by examining the index set defined by

$$J_\varepsilon(x^k)=\{i \mid h(c(x^k))-\varepsilon \leqq c(x^k)^T h_i+b_i\}, \tag{3.1}$$

where $\varepsilon>0$ is a given constant. Indeed, if $J_\varepsilon(x^k)$ consists of more than one element, then x^k is certainly considered to lie near the surface of discontinuity of $\nabla\phi$. (Note that $J_\varepsilon(x^k)$ is always nonempty.) Finally, condition (iii) is judged to hold if the inequality

$$\Delta_2^k \leqq \nu\Delta_1^k \tag{3.2}$$

is satisfied, where $\nu\in(0,1)$ is a given constant, and Δ_i^k, $i=1,2$, are defined by

$$\Delta_1^k=|h(c(x^k+\tilde{d}^k))-h(l^k(\tilde{d}^k))|, \tag{3.3}$$

$$\Delta_2^k=|h(c(x^k+\tilde{d}^k))-h(\hat{l}^k(\tilde{d}^k))|. \tag{3.4}$$

Note that calculation of these quantities requires additional evaluation of the function c.

We now explicitly state the algorithm

ALGORITHM.

```
begin
    choose x^0, λ^0, δ^0 > 0, ε > 0, 0 < ν < 1, k := 0;
first:  stage := 1;
    solve (2.1) to obtain d̃^k and λ̃^k;
    if d̃^k = 0 then stop else
      begin
        evaluate ρ̃^k by (2.2);
        if ρ̃^k > 0.75 then
          begin
            if ‖d̃^k‖ < δ^k then index := 2, go to update;
            if ‖d̃^k‖ = δ^k and ρ̃^k ≦ 0.9 then index := 3, go to update;
            if ‖d̃^k‖ = δ^k and ρ̃^k > 0.9 then index := 4, go to update
          end
```

```
        elseif J_ε(x^k) given by (3.1) is a singleton then
          begin
            if ρ̃^k ≦ 0 then index := 0, go to update;
            if 0 < ρ̃^k ≦ 0.25 then index := 1, go to update;
            if ρ̃^k > 0.25 then index := 2, go to update
          end
        else
          begin
            compute p^k and A^k by (2.9) and (2.10);
            evaluate Δ_1^k and Δ_2^k by (3.3) and (3.4);
            if ρ̃^k ≦ 0 and (3.2) does not hold then
              index := 0, go to update;
            if 0 < ρ̃^k ≦ 0.25 and (3.2) does not hold then
              index := 1, go to update;
            if ρ̃^k > 0.25 and (3.2) does not hold then
              index := 2, go to update;
            if ρ̃^k > 0.25 and (3.2) holds then
              index := 3, go to update;
            if ρ̃^k ≦ 0.25 and (3.2) holds then
              go to second
          end
      end
second: stage := 2;
      solve (2.4) to obtain d̂^k and λ̂^k;
      evaluate ρ̂^k by (2.8);
      if ρ̂^k > 0.75 then
        begin
          if ‖d̂^k‖ < δ^k then index := 2, go to update;
          if ‖d̂^k‖ = δ^k and ρ̂^k ≦ 0.9 then index := 3, go to update;
          if ‖d̂^k‖ = δ^k and ρ̂^k > 0.9 then index := 4, go to update
        end
      else
        begin
          if ρ̂^k ≦ 0 then index := 0, go to update;
          if 0 < ρ̂^k ≦ 0.25 then index := 1, go to update;
          if ρ̂^k > 0.25 then index := 2, go to update
        end
update: begin
        if stage = 1 then d̄^k := d̃^k, λ̄^k := λ̃^k;
        if stage = 2 then d̄^k := d̂^k, λ̄^k := λ̂^k;
        if index = 0 then x^{k+1} := x^k, λ^{k+1} := λ^k, δ^{k+1} := α^k‖d̄^k‖;
        if index = 1 then x^{k+1} := x^k + d̄^k, λ^{k+1} := λ̄^k, δ^{k+1} := α^k‖d̄^k‖,
        (comment: α^k is an approximate minimizer of φ(x^k + αd̄^k)
          in the interval [0.1, 0.5].)
        if index = 2 then x^{k+1} := x^k + d̄^k, λ^{k+1} := λ̄^k, δ^{k+1} := δ^k;
        if index = 3 then x^{k+1} := x^k + d̄^k, λ^{k+1} := λ̄^k, δ^{k+1} := 2δ^k;
        if index = 4 then x^{k+1} := x^k + d̄^k, λ^{k+1} := λ̄^k, δ^{k+1} := 4δ^k;
        k := k + 1, go to first
      end
  end
```

It is noted that the algorithm sets $x^{k+1} := x^k + \tilde{d}^k$ or $x^{k+1} := x^k + \hat{d}^k$ only when the reduction in the value of the objective function ϕ is obtained by doing so, and sets $x^{k+1} := x^k$ otherwise. Therefore, the sequence $\{\phi(x^k)\}$ is monotonically nonincreasing.

In adjusting the radius δ^k of the trust region, the algorithm mostly follows the idea of Fletcher [9]. There are a few differences between the two methods, however. For example, in addition to two measures corresponding to $\bar{\rho}^k$ and $\hat{\rho}^k$, Fletcher uses another measure to foresee whether good agreement is obtained by the projection step. In the above algorithm, such information is gained by evaluating the function c at $x^k + \tilde{d}^k$ to obtain Δ_1^k and Δ_2^k. Another difference consists in the treatment of the case where $\tilde{d}^k$ gives an average agreement $\tilde{\rho}^k \in [0.25, 0.75]$. In such cases, Fletcher's algorithm requires the solution of the second subproblem, which is used not to determine the next iterate x^{k+1} but merely to predict the effect of expanding the trust region in the next iteration. The proposed algorithm actually predicts such an effect by simply examining the condition (3.2) without proceeding to the second subproblem.

We now make a few additional comments on the algorithm. The notation $\bar{d}^k$ is used to denote either $\tilde{d}^k$ or $\hat{d}^k$, and likewise $\bar{\lambda}^k$. This is only for simplicity of presentation and no extra storage is needed for those quantities. As mentioned in the comment in the algorithm, α^k is determined as an appropriate minimizer of $\phi(x^k + \alpha \bar{d}^k)$ in the interval $[0.1, 0.5]$. A suitable technique for locating α^k is suggested in [7].

When the objective function ϕ is, in particular, the L_1 exact penalty function (1.7), it may be convenient to replace the definitions (3.3)-(3.4) of Δ_1^k and Δ_2^k with

$$\Delta_1^k = \max \{|c_i(x^k + \tilde{d}^k) - l_i^k(\tilde{d}^k)|\}, \tag{3.5}$$

$$\Delta_2^k = \max \{|c_i(x^k + \tilde{d}^k) - \hat{l}_i^k(\tilde{d}^k)|\}, \tag{3.6}$$

respectively, where max is taken only over ε-active constraints in problem (1.6). The numerical experiments to be reported later actually make use of (3.5)-(3.6) rather than (3.3)-(3.4), since the objective functions treated there are all of the form (1.7).

4. Convergence. In this section, we show that the sequence generated by the algorithm presented in the previous section has an accumulation point at which the first-order optimality conditions (1.3)-(1.4) for problem (1.1) hold. Also we prove that, under appropriate assumptions, the convergence rate of the algorithm is second order.

First, we establish a global convergence theorem. The key to proving it is to note that the global convergence property relies on the fact that $\{\tilde{d}^k\}$ and $\{\tilde{\rho}^k\}$, which are obtained by solving subproblem (2.1), actually satisfy the conditions used in the proof of the global convergence theorem [8, Thm. 3.1] for Fletcher's model algorithm (without second-order corrections).

THEOREM 1. *Let $\{x^k\}$ be a sequence generated by the algorithm. If $\{x^k\}$ is bounded, then it has an accumulation point x^∞ at which the first-order optimality conditions* (1.3)-(1.4) *hold.*

Proof. There exists a subsequence $K \subset \{1, 2, 3, \cdots\}$ such that $\{x^k\}_{k \in K}$ converges to x^∞ and either (i) $\{\delta^k\}_{k \in K} \to 0$ or (ii) $\inf_{k \in K} \delta^k > 0$ holds.

In case (i), there is no loss of generality in assuming that $\delta^{k+1} < \delta^k$ for all $k \in K$. Then, since δ^k is decreased only if index $= 0$ or 1, we have $\tilde{\rho}^k \leqq 0.25$ for all $k \in K$, whether or not the step $\hat{d}^k$ is calculated. This fact indicates that case (i) reduces to that of the proof of [8, Thm. 3.1].

In case (ii), taking into consideration the manner in which δ^k is adjusted, we may assume without loss of generality that $\delta^{k+1} \geqq \delta^k$ for every $k \in K$. Then it follows that

either

$$\tilde{\rho}^k \geqq 0.25 \quad \text{or} \quad \hat{\rho}^k \geqq 0.25 \tag{4.1}$$

holds for each $k \in K$. Since $x^{k+1} = x^k + \tilde{d}^k$ if $\tilde{\rho}^k \geqq 0.25$, and $x^{k+1} = x^k + \hat{d}^k$ if $\tilde{\rho}^k < 0.25$ and $\hat{\rho}^k \geqq 0.25$, it follows from (4.1) that

$$\frac{\phi(x^k) - \phi(x^{k+1})}{\phi(x^k) - \psi^k(\tilde{d}^k)} \geqq 0.25 \tag{4.2}$$

for all $k \in K$. On the other hand, the boundedness of $\{x^k\}$ and the descent property of $\{\phi(x^k)\}$ imply

$$\phi(x^k) - \phi(x^{k+1}) \to 0. \tag{4.3}$$

Consequently, it follows from (4.2) and (4.3) that the sequence $\{\phi(x^k) - \psi^k(\tilde{d}^k)\}_{k \in K}$ tends to zero. Since $\tilde{d}^k$ are the solutions of subproblems (2.1) that are equivalent to the subproblems solved in Fletcher's model algorithm [8], the remainder of the proof proceeds in a manner similar to that of [8, Thm. 3.1]. (Note that these arguments are essentially independent of the second subproblem (2.4)). This situation is similar to that of the global convergence theorem for the second-order correction algorithm of Fletcher [9].) □

Remark. It is easy to verify that if the algorithm terminates because $\tilde{d}^k = 0$ for some k, then (x^k, λ^k) actually satisfies (1.3)-(1.4).

Now we consider the convergence rate of the algorithm. To prove this, we assume as usual that the sequence $\{(x^k, \lambda^k)\}$ generated by the algorithm tends to (x^*, λ^*), satisfying the first-order optimality conditions (1.3)-(1.4). Note that, by (1.5), conditions (1.3)-(1.4) may be rewritten as follows:

$$\begin{aligned} &\nabla f(x^*) + \sum_{i=1}^{m} \mu_i^* \nabla c(x^*) h_i = 0, \qquad \sum_{i=1}^{m} \mu_i^* = 1, \\ &c(x^*)^T h_i + b_i \leqq v^*, \qquad \mu_i^* \geqq 0, \\ &\mu_i^* (c(x^*)^T h_i + b_i - v^*) = 0, \qquad i = 1, \cdots, m, \end{aligned} \tag{4.4}$$

where $v^* = h(c(x^*))$, and (1.4) gives the relation

$$\lambda^* = \sum_{i=1}^{m} \mu_i^* h_i. \tag{4.5}$$

In the following, we assume, without loss of generality, that

$$\begin{aligned} &c(x^*)^T h_i + b_i = v^* \quad \text{for } i = 1, 2, \cdots, m', \\ &c(x^*)^T h_i + b_i < v^* \quad \text{for } i = m'+1, \cdots, m. \end{aligned}$$

Then, the strict complementarity condition can be written as

$$\mu_i^* > 0, \qquad i = 1, \cdots, m'. \tag{4.6}$$

If (4.6) is satisfied, then conditions (4.4) reduce to the equalities

$$\nabla f(x^*) + \nabla c(x^*)(\Xi \mu^* + h_{m'}) = 0, \qquad \Xi^T c(x^*) + \eta = 0, \tag{4.7}$$

where

$$\begin{aligned} &\mu^* = (\mu_1^*, \cdots, \mu_{m'-1}^*)^T, \qquad \eta = (b_1 - b_{m'}, \cdots, b_{m'-1} - b_{m'})^T, \\ &\Xi = [\xi_1, \cdots, \xi_{m'-1}], \qquad \xi_i = h_i - h_{m'}, \qquad i = 1, \cdots, m'-1. \end{aligned}$$

Note that, using $\sum_{i=1}^{m} \mu_i^* = 1$ and (4.6), $\mu_{m'}^*$ has been eliminated in (4.7), and the relation (4.5) may be rewritten as

$$\lambda^* = \Xi\mu^* + h_{m'}. \tag{4.8}$$

Moreover, under the strict complementarity assumption (4.6), second-order sufficient conditions [6, Thm. 14.2.3] for problem (1.1) may be written as

$$s^T W^* s > 0 \quad {}^\forall s \neq 0: (\nabla c(x^*)\Xi)^T s = 0, \quad s \in R^n, \tag{4.9}$$

where

$$W^* = \nabla^2 f(x^*) + \sum_{i=1}^{M} \lambda_i^* \nabla^2 c_i(x^*).$$

In addition, if the linear independence condition

$$\operatorname{rank}(\nabla c(x^*)\Xi) = m' - 1 \tag{4.10}$$

holds, then the next matrix is nonsingular [22]:

$$\begin{pmatrix} W^* & \nabla c(x^*)\Xi \\ (\nabla c(x^*)\Xi)^T & 0 \end{pmatrix}. \tag{4.11}$$

The assumptions that will be imposed in the rest of this section are summarized as follows:

(a) (x^k, λ^k) tends to (x^*, λ^*), satisfying (1.3)-(1.4) as $k \to \infty$.

(b) The linear independence condition (4.10), strict complementarity condition (4.6), and second-order sufficient conditions (4.9) hold at (x^*, λ^*).

(c) The trust region constraints are inactive for k sufficiently large.

Note that assumption (c) is also made by Fletcher [9] in the analysis of the second-order correction algorithm. Interesting discussions about this assumption may be found in Yuan [23], [25]. In particular, Yuan [25] proves that Fletcher's algorithm [9] satisfies (c) under some conditions, and hence superlinear convergence is obtained.

In the next lemma, we first establish that the step $\hat{d}^k$ obtained from subproblem (2.4) satisfies condition (2.6), which ensures rapid convergence of the algorithm.

LEMMA 1. *Under assumptions* (a)-(c), $\hat{d}^k$ *satisfies* (2.6).

Proof. Under the given assumptions, we may suppose that the active indices $i = 1, 2, \cdots, m'$ are identified by solving subproblems (2.1) and (2.4) for all k sufficiently large. This indicates that first-order optimality conditions for (2.1) and (2.4) are reduced to the equations

$$\begin{aligned} W^k \tilde{d}^k + \nabla c(x^k)\Xi\tilde{\mu}^k &= -\nabla f(x^k) - \nabla c(x^k) h_{m'}, \\ (\nabla c(x^k)\Xi)^T \tilde{d}^k &= -\Xi^T c(x^k) - \eta \end{aligned} \tag{4.12}$$

and

$$\begin{aligned} W^k \hat{d}^k + A^k\Xi\hat{\mu}^k &= -p^k - A^k h_{m'}, \\ (A^k\Xi)^T \hat{d}^k &= -\Xi^T c(x^k) - \eta, \end{aligned} \tag{4.13}$$

respectively, where $\tilde{\mu}^k$ and $\hat{\mu}^k$ satisfy the relations

$$\tilde{\lambda}^k = \Xi\tilde{\mu}^k + h_{m'}, \tag{4.14}$$

$$\hat{\lambda}^k = \Xi\hat{\mu}^k + h_{m'}, \tag{4.15}$$

which correspond to (4.8). Then we can rewrite (4.13) as

$$\begin{pmatrix} W^k & A^k\Xi \\ (A^k\Xi)^T & 0 \end{pmatrix}\begin{pmatrix} \hat{d}^k - \tilde{d}^k \\ \hat{\mu}^k - \tilde{\mu}^k \end{pmatrix} = -\begin{pmatrix} p^k + A^k\tilde{\lambda}^k + W^k\tilde{d}^k \\ \Xi^T c(x^k) + \eta + (A^k\Xi)^T\tilde{d}^k \end{pmatrix}. \tag{4.16}$$

However, since (2.9)-(2.11), (4.12), and (4.14) imply

$$p^k + A^k\tilde{\lambda}^k + W^k\tilde{d}^k = \nabla f(x^k) + \nabla c(x^k)\tilde{\lambda}^k + W^k\tilde{d}^k = 0,$$
$$\Xi^T c(x^k) + \eta + (A^k\Xi)^T\tilde{d}^k = [\{A^k - \nabla c(x^k)\}\Xi]^T\tilde{d}^k = O(\|\tilde{d}^k\|^2),$$

it follows from (4.16) that

$$\begin{pmatrix} W^k & A^k\Xi \\ (A^k\Xi)^T & 0 \end{pmatrix}\begin{pmatrix} \hat{d}^k - \tilde{d}^k \\ \hat{\mu}^k - \tilde{\mu}^k \end{pmatrix} = \begin{pmatrix} 0 \\ O(\|\tilde{d}^k\|^2) \end{pmatrix}. \tag{4.17}$$

Note that assumptions (a) and (c) imply (2.3), and hence $\|\tilde{d}^k\| \to 0$. (See the comment after this proof.) Also it follows that W^k and A^k tend to W^* and $\nabla c(x^*)$, respectively. The coefficient matrix of (4.17) is therefore uniformly nonsingular, since the matrix defined by (4.11) is nonsingular under assumption (b). Thus (4.17) gives the estimate

$$\|\hat{d}^k - \tilde{d}^k\| = O(\|\tilde{d}^k\|^2), \tag{4.18}$$

$$\|\hat{\mu}^k - \tilde{\mu}^k\| = O(\|\tilde{d}^k\|^2). \tag{4.19}$$

Moreover, since Ξ is of full rank by (4.10), it follows from (4.14), (4.15), and (4.19) that

$$\|\hat{\lambda}^k - \tilde{\lambda}^k\| = O(\|\tilde{d}^k\|^2). \tag{4.20}$$

On the other hand, we can easily verify that

$$\frac{\|x^k + \hat{d}^k - x^*\|}{\|x^k - x^*\|} \leqq \frac{\|x^k + \tilde{d}^k - x^*\|}{\|x^k - x^*\|} + \frac{\|\hat{d}^k - \tilde{d}^k\|}{\|\tilde{d}^k\|} \cdot \frac{\|\tilde{d}^k\|}{\|x^k - x^*\|}. \tag{4.21}$$

Since (2.3) implies

$$\frac{\|\tilde{d}^k\|}{\|x^k - x^*\|} = 1 + O(\max\{\|x^k - x^*\|, \|\lambda^k - \lambda^*\|\}), \tag{4.22}$$

(4.18) yields

$$\frac{\|\hat{d}^k - \tilde{d}^k\|}{\|\tilde{d}^k\|} = O(\|\tilde{d}^k\|) = O(\|x^k - x^*\|). \tag{4.23}$$

Therefore, it follows from (4.21)-(4.23) and (2.3) that the estimate (2.6) holds. □

In the above proof, we have used the fact that assumptions (a) and (c) imply (2.3). This is valid because (2.3) is only concerned with the property enjoyed by the solution of the first subproblem (2.1), which is equivalent to the basic step subproblem of Fletcher [9], and (2.3) is independent of other aspects such as the second subproblem (2.4). In fact, it is pointed out by Fletcher [9, Eq. (2.9)] that the solution $\tilde{d}^k$ of the first subproblem (2.1) satisfies (2.3) under the given assumptions. (Note that $\tilde{d}^k$ in (2.3) corresponds to $\hat{x} - x^*$ in [9].)

We now proceed to show that $\hat{d}^k$ satisfies condition (2.7), which implies that $\hat{d}^k$ gives a sufficient decrease in ϕ for all k large enough.

LEMMA 2. *Under assumptions* (a)-(c), *we have for sufficiently large k that*

$$h(l^k(\tilde{d}^k)) - \lambda^T l^k(\tilde{d}^k) = h(\hat{l}^k(\hat{d}^k)) - \lambda^T\hat{l}^k(\hat{d}^k)$$

for any $\lambda \in \partial h(\hat{l}^k(\hat{d}^k))$.

Proof. As mentioned at the beginning of the proof of Lemma 1, assumptions (a)-(c) imply that the active indices $i = 1, 2, \cdots, m'$ are identified by solving subproblems (2.1) and (2.4) for all k large enough, and hence

$$\partial h(c(x^*)) = \partial h(l^k(\tilde{d}^k)) = \partial h(\hat{l}^k(\hat{d}^k))$$
$$= \left\{ \sum_{i=1}^{m'} \mu_i h_i \;\middle|\; \sum_{i=1}^{m'} \mu_i = 1, \mu_i \geqq 0 \right\}.$$

(cf. (1.5)). Now let us choose $\lambda \in \partial h(\hat{l}^k(\hat{d}^k))$ arbitrarily. Then there exist $\mu_i \geqq 0$, $i = 1, 2, \cdots, m'$, such that $\sum_{i=1}^{m'} \mu_i = 1$ and $\lambda = \sum_{i=1}^{m'} \mu_i h_i$. Since

$$h(l^k(\tilde{d}^k)) = l^k(\tilde{d}^k)^T h_i + b_i, \qquad i = 1, 2, \cdots, m',$$

and

$$h(\hat{l}^k(\hat{d}^k)) = \hat{l}^k(\hat{d}^k)^T h_i + b_i, \qquad i = 1, 2, \cdots, m',$$

multiplying these equalities by μ_i and summing over $i = 1, 2, \cdots, m'$ yield

$$h(l^k(\tilde{d}^k)) = l^k(\tilde{d}^k)^T \lambda + \sum_{i=1}^{m'} \mu_i b_i$$

and

$$h(\hat{l}^k(\hat{d}^k)) = \hat{l}^k(\hat{d}^k)^T \lambda + \sum_{i=1}^{m'} \mu_i b_i.$$

This completes the proof. □

LEMMA 3. *Under assumptions* (a)-(c), $\hat{d}^k$ *satisfies* (2.7).

Proof. First note that (2.14) and (4.18) yield

$$c(x^k + \hat{d}^k) = \hat{l}^k(\hat{d}^k) + o(\|\tilde{d}^k\|^2). \tag{4.24}$$

Since $\hat{\lambda}^k \in \partial h(\hat{l}^k(\hat{d}^k))$, it follows from Lemma 2, (2.10), (2.11), and (2.13) that

$$\begin{aligned} h(\hat{l}^k(\hat{d}^k)) &= h(l^k(\tilde{d}^k)) + (\hat{\lambda}^k)^T \{\hat{l}^k(\hat{d}^k) - l^k(\tilde{d}^k)\} \\ &= h(l^k(\tilde{d}^k)) + (A^k \hat{\lambda}^k)^T (\hat{d}^k - \tilde{d}^k) + \frac{1}{2} \sum_{i=1}^{M} \hat{\lambda}_i^k (\tilde{d}^k)^T \nabla^2 c_i(x^k) \tilde{d}^k. \end{aligned} \tag{4.25}$$

Thus, by (4.18), (4.24), (4.25), and the definition of W^k, we have

$$\begin{aligned} \psi^k(\tilde{d}^k) - \phi(x^k + \hat{d}^k) &= (\nabla f(x^k) + A^k \hat{\lambda}^k)^T (\tilde{d}^k - \hat{d}^k) \\ &\quad + \frac{1}{2} \sum_{i=1}^{M} (\lambda_i^k - \hat{\lambda}_i^k)(\tilde{d}^k)^T \nabla^2 c_i(x^k) \tilde{d}^k + o(\|\tilde{d}^k\|^2). \end{aligned} \tag{4.26}$$

Using (4.13) and (4.15), and then (2.9) and (2.11), we have

$$\nabla f(x^k) + A^k \hat{\lambda}^k = \nabla f(x^k) - p^k - W^k \hat{d}^k$$
$$= \frac{1}{2} \sum_{i=1}^{m} \tilde{\lambda}_i^k \nabla^2 c_i(x^k) \tilde{d}^k - W^k \hat{d}^k.$$

Thus it follows from (4.18) and the boundedness of $\{\lambda^k\}$ that the first term on the right-hand side of (4.26) is of magnitude $o(\|\tilde{d}^k\|^2)$, and hence

$$\psi^k(\tilde{d}^k) - \phi(x^k + \hat{d}^k) = \|\hat{\lambda}^k - \lambda^k\| O(\|\tilde{d}^k\|^2) + o(\|\tilde{d}^k\|^2). \tag{4.27}$$

Since assumptions (a)-(c) ensure that

$$\|\tilde{d}^k\| = \|x^k - x^*\| + o(\|x^k - x^*\|),$$
$$\|\tilde{\lambda}^k - \lambda^k\| = \|\lambda^k - \lambda^*\| + o(\|x^k - x^*\|)$$

hold [9], (4.20), and (4.27) yield

$$\psi^k(\tilde{d}^k) - \phi(x^k + \hat{d}^k) = o(\|x^k - x^*\|^2). \tag{4.28}$$

On the other hand, as shown in Lemmas 2.2 and 2.5 of [9], there exists a constant $\beta > 0$ such that

$$\phi(x^k) - \psi^k(\tilde{d}^k) \geqq \beta\|x^k - x^*\|^2 + o(\|x^k - x^*\|^2). \tag{4.29}$$

Since $\hat{\rho}^k$ may be written as

$$\hat{\rho}^k = 1 + \frac{\psi^k(\tilde{d}^k) - \phi(x^k + \hat{d}^k)}{\phi(x^k) - \psi^k(\tilde{d}^k)},$$

the desired result (2.7) follows from (4.28) and (4.29). □

In the above proof, we derived (4.29) by combining Lemmas 2.2 and 2.5 of Fletcher [9]. We need to comment on this because those lemmas might seem to depend on the results concerning Fletcher's second-order correction algorithm. First, Lemma 2.2 is merely related to the property of the composite (objective) function itself. Also, Lemma 2.5 concerns the solution of the basic step subproblem, i.e., the first subproblem (2.1) only. (The proof of Lemma 2.5 refers to Lemmas 2.3 and 2.4 in [9], which contain results on the solution of the projection step subproblem as well. But it can be seen that the proof of Lemma 2.5 actually does not use any result related to the projection step subproblem.) Thus, (4.29) follows from Lemmas 2.2 and 2.5 in [9], independently of the points in which the proposed algorithm differs from Fletcher's second-order correction algorithm.

Combining Lemmas 1 and 3, we are now able to demonstrate the rate of convergence results as follows.

THEOREM 2. *Let assumptions* (a)-(c) *be satisfied. Then the sequence* $\{x^k\}$ *generated by the algorithm converges to* x^* *superlinearly. Moreover, if* $\|\lambda^k - \lambda^*\| = O(\|x^k - x^*\|)$, *then* $\{x^k\}$ *is quadratically convergent.*

Proof. To prove the first half of the theorem, it suffices to show that

$$\frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|} = O(\max\{\|x^k - x^*\|, \|\lambda^k - \lambda^*\|\}). \tag{4.30}$$

If $\tilde{\rho} \geqq 0.25$, then we have

$$x^{k+1} = x^k + \tilde{d}^k$$

and (4.30) follows from (2.3). Even if $\tilde{\rho}^k < 0.25$, Lemma 3 ensures that the step $\hat{d}^k$ is accepted and

$$x^{k+1} = x^k + \hat{d}^k,$$

provided that k is sufficiently large. Then Lemma 1 indicates that (4.30) still holds.

The second half is a direct consequence of (4.30). □

5. Quasi-Newton modifications. The algorithm described in § 3 uses the Hessian matrix W^k to constrict the quadratic models (2.1) and (2.4). From a practical viewpoint, however, it may be very expensive to evaluate the second derivatives on each iteration. In such cases, it would be convenient to adopt quasi-Newton methods in which W^k is approximated by a symmetric matrix B^k using gradient information only.

Now, remember that the definitions (2.9)–(2.10) of p^k and a_i^k contain via (2.11) the second derivatives $\nabla^2 c_i(x^k)$. Since second-order information is no longer used in quasi-Newton update formulas for B^k, it may be necessary to modify the definitions of p^k and a_i^k so as to involve first-order information only. This can actually be accomplished by replacing (2.11) with

$$r_i^k = \tfrac{1}{2}(\nabla c_i(x^k + \tilde{d}^k) - \nabla c_i(x^k)), \qquad i = 1, \cdots, M, \tag{5.1}$$

and then defining p^k and a_i^k again by (2.9) and (2.10), respectively. Since (5.1) implies

$$r_i^k = \tfrac{1}{2}\nabla^2 c_i(x^k)\tilde{d}^k + o(\|\tilde{d}^k\|),$$

we have

$$\hat{l}_i^k(d) = c_i(x^k) + \nabla c_i(x^k)^T d + \tfrac{1}{2}(\tilde{d}^k)^T \nabla^2 c_i(x^k) d + \|d\| o(\|\tilde{d}^k\|),$$

which corresponds to (2.14) and suggests that $\hat{l}^k(d)$ agrees with $c(x^k + d)$ to second order when evaluated at $\hat{d}^k$.

6. Numerical results. To test the behavior of the proposed algorithm and its quasi-Newton modification, numerical experiments have been carried out for several problems. The composite functions $\phi(x)$ used in the experiments are the L_1 exact penalty functions $\Phi_t(x)$ defined by (1.7) for the following nonlinear programming problems.

PROBLEM 1 [3], [21].

$$\begin{aligned} &\text{minimize} && -x_1 + 10(x_1^2 + x_2^2 - 1) \\ &\text{subject to} && x_1^2 + x_2^2 - 1 = 0, \\ & && x^* = (1, 0), \qquad \lambda^* = -\tfrac{19}{2}. \end{aligned}$$

PROBLEM 2 [16].

$$\begin{aligned} &\text{minimize} && (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^4 + (x_4 - x_5)^4 \\ &\text{subject to} && x_1 + x_2^2 + x_3^3 - 2 - 3\sqrt{2} = 0, \\ & && x_2 - x_3^2 + x_4 + 2 - 2\sqrt{2} = 0, \\ & && x_1 x_5 - 2 = 0, \\ & && x^* = (1.1911, 1.3626, 1.4728, 1.6350, 1.6791), \\ & && \lambda^* = (-0.03882, -0.01674, -0.0002873). \end{aligned}$$

PROBLEM 3 [16].

$$\begin{aligned} &\text{minimize} && x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4 \\ &\text{subject to} && x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_1 - x_2 + x_3 - x_4 - 8 \leqq 0, \\ & && x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 - 10 \leqq 0, \\ & && 2x_1^2 + x_2^2 + x_3^2 + 2x_1 - x_2 - x_4 - 5 \leqq 0, \\ & && x^* = (0, 1, 2, -1), \qquad \lambda^* = (1, 0, 2). \end{aligned}$$

Problem 1 is one of the typical examples for which the Maratos effect occurs in the ordinary SQP methods [3], [21]. Problems 2 and 3 are taken from [16], where they are Problems 79 and 43, respectively. In particular, Problem 3 is also known as the Rosen–Suzuki test problem.

Recall that, for nonlinear programming, the Maratos effect may occur only near the curved constraint boundary. So each problem has been solved using two starting points. Since Problems 1 and 2 contain equality constraints only, any feasible point may be considered to lie on the constraint boundary. Thus, one of the starting points has been chosen as feasible and the other as infeasible. For Problem 3, the starting points have been selected from both the interior and the boundary of the feasible region.

The initial estimate of Lagrange multipliers and the radius of the initial trust region are set to zero and one, respectively, for all problems. Also we set parameters ε and ν, which appear in (3.1) and (3.2), to be 1.0 and 0.1, respectively. It has been observed, however, that the results are not very sensitive to the choice of ε and ν.

In the implementation of the algorithm, we have used L_∞ norm in the trust region constraints of subproblems (2.1) and (2.4), so that those subproblems are transformed into quadratic programming problems. In the quasi-Newton modification of the algorithm, we have employed the Powell-symmetric-Broyden (PSB) formula [11], [25], one of the most effective update formulas.

For the purpose of comparison, we have also solved each problem by the model algorithm and second-order correction algorithm due to Fletcher [6]–[9]. All programs were coded by the authors in FORTRAN 77. The computational results are summarized in Tables 1–6, where

A1: algorithm described in § 3,

TABLE 1
Results for Problem 1: *Feasible starting point* $x^0 = (0.8, 0.6)$.

	$t = 10$				$t = 100$				$t = 1000$			
	A1	A2	SC	MA	A1	A2	SC	MA	A1	A2	SC	MA
N_i	4	3	7	155	4	3	18	803	4	3	34	5052
N_1	0	0	2	147	0	0	2	769	0	0	0	4991
N_2	4	3	5	-	4	3	12	-	4	3	33	-
$\bar{\delta}$	1.0	1.0	1.536	$<10^{-5}$	1.0	1.0	0.080	$<10^{-5}$	1.0	1.0	0.019	$<10^{-5}$
N_f	9	7	13	156	9	7	34	770	9	7	68	5053
N_g	5	7	8	148	5	7	15	770	5	7	34	4992
N_H	5	-	8	148	5	-	15	770	5	-	34	4992
N_Q	9	7	14	156	9	7	36	804	9	7	69	5053

TABLE 2
Results for Problem 1: *Infeasible starting point* $x^0 = (-1, -1)$.

	$t = 10$				$t = 100$				$t = 1000$			
	A1	A2	SC	MA	A1	A2	SC	MA	A1	A2	SC	MA
N_i	9	10	9	142	11	10	41	837	37	9	78	7192
N_1	4	1	4	129	4	2	4	819	3	6	4	6995
N_2	3	7	3	-	6	6	26	-	27	2	60	-
$\bar{\delta}$	0.32	0.44	0.32	$<10^{-5}$	0.2	0.31	0.071	$<10^{-5}$	0.084	0.60	0.058	$<10^{-5}$
N_f	15	20	13	143	19	19	79	838	72	13	153	7193
N_g	8	18	8	130	11	17	31	820	31	12	65	6976
N_H	8	-	8	130	11	-	31	820	31	-	65	6976
N_Q	15	20	15	143	19	19	80	838	72	13	154	7193

TABLE 3
Results for Problem 2: Feasible starting point $x^0 = (\sqrt{2}, \sqrt{2}, \sqrt{2}, \sqrt{2}, \sqrt{2})$.

	$t = 10$				$t = 100$				$t = 1000$			
	A1	A2	SC	MA	A1	A2	SC	MA	A1	A2	SC	MA
N_i	3	7	5	13	9	8	20	365	12	30	38	3278
N_1	2	0	2	12	1	0	1	348	1	0	0	3150
N_2	1	6	2	-	6	7	11	-	10	21	29	-
$\bar{\delta}$	1.0	0.69	0.25	0.013	0.041	0.067	0.030	$<10^{-5}$	0.053	0.007	0.008	$<10^{-5}$
N_f	5	15	9	14	18	17	40	366	24	61	77	3279
N_g	4	14	5	13	8	16	13	349	12	52	30	3151
N_H	4	-	5	13	8	-	13	349	12	-	30	3151
N_Q	5	15	10	18	18	17	40	366	24	61	77	3279

TABLE 4
Results for Problem 2: Infeasible starting point $x^0 = (2, 2, 2, 2, 2)$.

	$t = 10$				$t = 100$				$t = 1000$			
	A1	A2	SC	MA	A1	A2	SC	MA	A1	A2	SC	MA
N_i	4	6	4	4	4	6	4	4	4	6	4	4
N_1	4	6	4	4	4	6	4	4	4	6	4	4
N_2	0	0	0	-	0	0	0	-	0	0	0	-
$\bar{\delta}$	1.0	2.0	1.0	1.0	1.0	2.0	1.0	1.0	1.0	2.0	1.0	1.0
N_f	5	7	5	5	5	7	5	5	5	7	5	5
N_g	5	7	5	5	5	7	5	5	5	7	5	5
N_H	5	-	5	5	5	-	5	5	5	-	5	5
N_Q	5	7	5	5	5	7	5	5	5	7	5	5

TABLE 5
Results for Problem 3: Boundary starting point $x^0 = (0, -2, 0, 1)$.

	$t = 10$				$t = 100$				$t = 1000$			
	A1	A2	SC	MA	A1	A2	SC	MA	A1	A2	SC	MA
N_i	8	14	8	6	6	18	10	227	6	32	13	1998
N_1	5	4	6	6	1	4	3	211	1	2	1	1934
N_2	2	7	1	-	5	9	5	-	5	20	9	-
$\bar{\delta}$	1.72	0.50	2.08	0.5	8.0	1.50	2.11	$<10^{-5}$	0.83	0.028	0.11	$<10^{-5}$
N_f	11	24	10	7	12	33	16	228	12	63	24	1999
N_g	8	21	8	7	7	28	9	212	7	53	11	1935
N_H	8	-	8	7	7	-	9	212	7	-	11	1935
N_Q	11	24	14	7	12	33	18	228	12	63	26	1999

TABLE 6
Results for Problem 3: *Interior starting point* $x^0 = (0, 0, 0, 0)$.

	$t = 10$				$t = 100$				$t = 1000$			
	A1	A2	SC	MA	A1	A2	SC	MA	A1	A2	SC	MA
N_i	6	7	8	8	18	19	17	238	46	56	36	2115
N_1	5	2	4	7	3	2	2	233	6	6	3	2055
N_2	1	5	3	-	9	11	10	-	26	31	25	-
$\bar{\delta}$	2.0	1.0	1.6	0.22	0.28	0.32	0.056	$<10^{-5}$	0.022	0.030	0.013	$<10^{-5}$
N_f	8	13	12	9	33	36	32	239	83	103	70	2116
N_g	7	13	8	8	13	30	13	224	33	84	29	2056
N_H	7	-	8	8	13	-	13	224	33	-	29	2056
N_Q	8	13	15	9	33	36	34	239	83	103	70	2116

A2: quasi-Newton modification of A1,
SC: Fletcher's second-order correction algorithm,
MA: Fletcher's model algorithm
and

t = value of penalty parameter in $\Phi_t(x)$,
N_i = total number of iterations,
N_1 = number of iterations that accepted $\tilde{d}^k$,
N_2 = number of iterations that accepted $\hat{d}^k$,
$\bar{\delta}$ = radius of the trust region on termination of the algorithm,
N_f = total number of function evaluations,
N_g = total number of gradient evaluations,
N_H = total number of Hessian evaluations,
N_Q = total number of quadratic programming subproblems actually solved.

Note that Tables 1, 3, and 5, in particular, correspond to the case where the starting point is selected on the constraint boundary. The tables show that, for such starting points, the proposed algorithm and its quasi-Newton modification give an accurate solution very quickly for any choice of the parameter value. This fact indicates that the proposed algorithm is effective against the Maratos effect.

A few more comments on the choice of the penalty parameter are in order. If the penalty value is not large enough, then it is likely that the generated iterates eventually lie outside the feasible region of the original problem. On the other hand, when the penalty value is unduly large, standard methods such as the model algorithm, MA, will usually encounter serious inefficiency in the speed of convergence. Therefore it is desirable for an exact penalty function method to contain a sophisticated strategy for adjusting the penalty parameter to a suitable value automatically. However, this is by no means an easy problem. In particular, it seems difficult to systematically reduce the penalty value when it becomes too large. So it would be preferable if an algorithm had the property that its efficiency did not deteriorate for large parameter values. In fact, if this is the case, we may even fix the penalty parameter at a fairly large value that is considered certainly safe from convergence to an infeasible solution. Thus it is important to examine how an algorithm behaves for various values of the penalty parameter. This is the reason we solved the test problems by choosing extremely large penalty values. The results shown in Tables 1-6 suggest that the proposed algorithm can successfully cope with the difficulty caused by increasing the penalty value, and hence has the desirable feature mentioned above.

7. Conclusion. We have presented a trust region algorithm for minimizing the nondifferentiable composite functions ϕ of the form (1.1). This algorithm is similar in nature to the SQP method presented in [10] for constrained nonlinear optimization problems. We have shown that the sequence generated by the algorithm has an accumulation point satisfying the first-order optimality conditions (1.3)-(1.4) for problem (1.1). Moreover, superlinear convergence is established under the linear independence, strict complementarity, and second-order sufficient conditions if the trust region constraints become inactive eventually. In fact, the numerical results shown in § 6 indicate that the proposed algorithm is free of the Maratos effect.

Acknowledgment. The authors are grateful to the referees for careful reading and many helpful comments. They are also indebted to Dr. M. H. Wright for her comments and suggestions, which were very helpful in revising the manuscript.

REFERENCES

[1] P. T. BOGGS, J. W. TOLLE, AND P. WANG, *On the local convergence of quasi-Newton methods for constrained optimization*, SIAM J. Control Optim., 20 (1982), pp. 161-171.

[2] J. V. BURKE, *Descent methods for composite nondifferentiable optimization problems*, Math. Programming, 33 (1985), pp. 260-279.

[3] R. M. CHAMBERLAIN, M. J. D. POWELL, C. LEMARECHAL, AND H. C. PEDERSEN, *The watchdog technique for forcing convergence in algorithms for constrained optimization*, Math. Programming Study, 16 (1982), pp. 1-17.

[4] T. F. COLEMAN AND A. R. CONN, *Nonlinear programming via an exact penalty function: Asymptotic analysis*, Math. Programming, 24 (1982), pp. 123-136.

[5] I. D. COOPE, *The Maratos effect in sequential quadratic programming algorithms using the L_1 exact penalty function*, Tech. Report CS-85-32, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1985.

[6] R. FLETCHER, *Practical Methods of Optimization, Volume 2: Constrained Optimization*, John Wiley, Chichester, 1981.

[7] ———, *Numerical experiments with an exact L_1 penalty function method*, in Nonlinear Programming 4, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, eds., Academic Press, New York, 1981, pp. 99-129.

[8] ———, *A model algorithm for composite nondifferentiable optimization problems*, Math. Programming Study, 17 (1982), pp. 67-76.

[9] ———, *Second order corrections for non-differentiable optimization*, in Numerical Analysis, Dundee 1981, G. A. Watson, ed., Springer-Verlag, Berlin, New York, 1982, pp. 85-114.

[10] M. FUKUSHIMA, *A successive quadratic programming algorithm with global and superlinear convergence properties*, Math. Programming, 35 (1986), pp. 253-264.

[11] S. -P. HAN, *Superlinearly convergent variable metric algorithms for general nonlinear programming problems*, Math. Programming, 11 (1976), pp. 263-282.

[12] ———, *A globally convergent method for nonlinear programming*, J. Optim. Theory Appl., 22 (1977), pp. 297-309.

[13] ———, *Superlinear convergence of a minimax method*, Tech. Report TR78-336, Department of Computer Science, Cornell University, Ithaca, NY, 1978.

[14] ———, *Variable metric methods for minimizing a class of nondifferentiable functions*, Math. Programming, 20 (1981), pp. 1-13.

[15] S -P. HAN AND O. L. MANGASARIAN, *Exact penalty functions in nonlinear programming*, Math. Programming, 17 (1979), pp. 251-269.

[16] W. HOCK AND K. SCHITTKOWSKI, *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems 187, Springer-Verlag, Berlin, New York, 1981.

[17] O. L. MANGASARIAN, *Sufficiency of exact penalty minimization*, SIAM J. Control Optim., 23 (1985), pp. 30-37.

[18] D. Q. MAYNE AND E. POLAK, *A superlinearly convergent algorithm for constrained optimization problems*, Math. Programming Study, 16 (1982), pp. 45-61.

[19] M. J. D. POWELL, *A fast algorithm for nonlinearly constrained optimization calculations*, in Numerical Analysis, Dundee 1977, G. A. Watson, ed., Springer-Verlag, Berlin, New York, 1978, pp. 144-157.

[20] M. J. D. POWELL, *The convergence of variable metric methods for nonlinearly constrained optimization calculations*, in Nonlinear Programming 3, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, eds., Academic Press, New York, 1978, pp. 27-63.

[21] ———, *Variable metric methods for constrained optimization*, in Mathematical Programming, The State of the Art, Bonn, 1982, A. Bachem, M. Grötschel, and B. Korte, eds., Springer-Verlag, Berlin, 1983, pp. 288-311.

[22] R. S. WOMERSLEY, *Local properties of algorithms for minimizing nonsmooth composite functions*, Math. Programming, 32 (1985), pp. 69-89.

[23] Y. YUAN, *An example of only linear convergence of trust region algorithms for non-smooth optimization*, IMA J. Numer. Anal., 4 (1984), pp. 327-335.

[24] ———, *Conditions for convergence of trust region algorithms for nonsmooth optimization*, Math. Programming, 31 (1985), pp. 220-228.

[25] ———, *On the superlinear convergence of a trust region algorithm for nonsmooth optimization*, Math. Programming, 31 (1985), pp. 269-285.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 3, pp. 581-605, May 1989

© 1989 Society for Industrial and Applied Mathematics
011

APPROXIMATE SCHUR COMPLEMENT PRECONDITIONERS ON SERIAL AND PARALLEL COMPUTERS*

HOWARD C. ELMAN†

Abstract. A class of preconditioning techniques for sparse matrices is considered, based on computing an approximation of the Schur complement of a (suitably ordered) matrix. The techniques generalize the reduced system methodology for 2-cyclic matrices to non-2-cyclic matrices, and in addition, they are well suited to parallel architectures. Their effectiveness with numerical experiments on a nine-point finite-difference operator is demonstrated, and an analysis showing that they can be implemented efficiently on multiprocessors is presented.

Key words. sparse iterative methods, preconditioners, parallel linear algebra

AMS(MOS) subject classifications. primary, 65F10, 65N20; secondary, 15A06

1. Introduction. We consider preconditioning techniques for use with iterative methods for solving sparse linear systems of equations

$$Ax = f, \tag{1}$$

of the type arising from the discretization of elliptic and parabolic differential equations. The methods are approximate block cyclic reduction methods of the type studied for five-point matrices by Axelsson [2], Eisenstat [12], Kershaw [26], and Rodrigue and Wolitzer [36]. For two-dimensional problems, they make use of a line red-black ordering of gridpoints, and they depend on choosing approximations to the set of Schur complements computed by the *direct* block cyclic reduction method [5]. In this paper, we show that the same class of methods can be applied to non-2-cyclic operators such as the two-dimensional nine-point operator, and that they can be implemented efficiently on multiprocessor architectures, with a detailed analysis for the hypercube.

Our motivation for examining these methods stems from our experience with block 2-cyclic red-black ordered problems [13], [15]. We have found that the combination of decoupling of the red and black points by block elimination together with preconditioning of the resulting reduced system by incomplete factorizations [10], [19], [29] forms the basis of very fast and robust iterative solvers. Here, conjugate gradient-like (CG-like) methods [14] are used to solve the preconditioned equations. In particular, omission of either the decoupling step or the preconditioning hurts overall performance more than does weakening of the CG-like solver.

However, both the decoupling (or preprocessing) and preconditioning methodologies have limitations for present-day computing. The preprocessing step is only useful if the new reduced matrix is also sparse; this is true in the 2-cyclic case but not in general for discretized partial differential equations (see § 2). As for preconditioning, methods based on incomplete factorizations are often difficult to implement on parallel architectures because the required sparse triangular solves involve the solution of clustered recurrences that cannot be separated into independent tasks. This is the case, for example, if incomplete factorizations are used to precondition the

* Received by the editors October 20, 1986; accepted for publication (in revised form) June 16, 1988. The work presented in this paper was supported by the National Science Foundation under grants DCR-82-19507 and DMS-8607478.

† Computer Science Department and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland 20742.

reduced matrix in the block 2-cyclic case, or if they are applied directly to discretized partial differential equations that use the natural ordering of gridpoints (cf. [21], [39] for parallel implementations of the latter example based on pipelining). One way to alleviate this difficulty is to use "multicolor orderings" for the coefficient matrix [1], [32], [33], [41]. The resulting sparse preconditioners can then be implemented in m fully parallel steps for some small m, although the effect of such reorderings on overall performance (i.e., iteration counts) is not well understood. Our aim in this paper is to present and analyze a class of methods that simultaneously generalizes the reduced system methodology to non-2-cyclic matrices, and to develop a preconditioning (for use with the reduced system) that can be implemented efficiently on parallel architectures.

The methods that we consider make use of a set of block matrices of the form

$$\begin{pmatrix} D & C \\ E & F \end{pmatrix}, \tag{2}$$

and the associated Schur complement $F - ED^{-1}C$. The block diagonals D and F each consist of a set of uncoupled tridiagonal matrices corresponding to nearest-neighbor connections along the lines of the grid, and parallelism is achieved by handling these matrices independently. We remark that the class of methods known as *domain decomposition* also uses the block form (2) and its Schur complement (see, e.g., [8], [11]). For those methods, D corresponds to a set of independent *two-dimensional* discrete operators, and parallelism (at least at the highest level) can also be achieved by treating these two-dimensional solves independently. See [27] for an analysis of parallel domain decomposition techniques and for a more complete set of references for domain decomposition.

In § 2, we describe the approximate Schur complement methods, and in § 3, we outline their operation counts for serial implementation. In § 4, we present the results of numerical experiments, demonstrating that the methods compare favorably with standard techniques on serial computers. In § 5, we show that the methods can also be implemented efficiently on parallel architectures, and in § 6, we draw conclusions.

2. Approximmate Schur complement preconditioners. In this section, we describe the approximate Schur complement preconditioners and show how they can be applied to nine-point finite difference operators for two-dimensional partial differential equations. By way of introduction, let (1) represent a sparse linear system arising from the standard five-point or seven-point finite difference operators for two-dimensional (respectively, three-dimensional) elliptic or parabolic problems [18]. Then the coefficient matrix A is 2-cyclic and there is a permutation matrix P_1 that permutes (1) into the red-black ordered system

$$[P_1 A P_1^{-1}][P_1 x] = \begin{pmatrix} D_1 & C_1 \\ E_1 & F_1 \end{pmatrix} \begin{pmatrix} x^{(r)} \\ x^{(b)} \end{pmatrix} = \begin{pmatrix} f^{(r)} \\ f^{(b)} \end{pmatrix} = P_1 f, \tag{3}$$

in which D_1 and F_1 are diagonal matrices [45]. (See Fig. 1 below for the point red-black grid ordering.) The block unknowns $x^{(r)}$ and $x^{(b)}$ can be decoupled [34] by premultiplying and postmultiplying the coefficient matrix of (3) by V_1 and W_1, respectively, where

$$V_1 = \begin{pmatrix} G_1^{-1} & 0 \\ -E_1 D_1^{-1} & I \end{pmatrix}, \qquad W_1 = \begin{pmatrix} H_1^{-1} & -D_1^{-1} C_1 \\ 0 & I \end{pmatrix},$$

and $D_1 = G_1 H_1$ is a factorization. The resulting system is

$$\begin{pmatrix} I & 0 \\ 0 & F_1 - E_1 G_1^{-1} C_1 \end{pmatrix} \begin{pmatrix} \hat{x}^{(r)} \\ x^{(b)} \end{pmatrix} = \begin{pmatrix} G_1^{-1} f^{(r)} \\ f^{(b)} - E_1 D_1^{-1} f^{(r)} \end{pmatrix}, \tag{4}$$

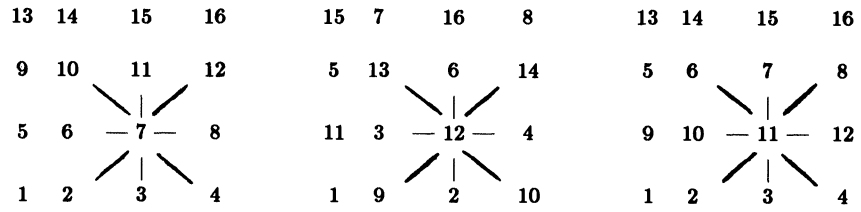


FIG. 1. *A nine-point operator on a* 4×4 *grid using the natural ordering (left), point red-black ordering (center) and line red-black ordering (right).*

where $\hat{x}^{(r)} = H_1 x^{(r)} + G_1^{-1} C_1 x^{(b)}$. Let S_1 denote the Schur complement $F_1 - E_1 D_1^{-1} C_1$, and let $s_1 = f^{(b)} - E_1 D_1^{-1} f^{(r)}$. The lower block equation of (4) then has the form

$$S_1 x^{(b)} = s_1, \tag{5}$$

which we refer to as the "reduced system." For these finite-difference examples, S_1 is also a sparse matrix: the two-dimensional version has roughly nine nonzeros per row, and the three-dimensional version has roughly nineteen nonzeros per row. Hence, standard preconditioned iterative methods can be used to solve the reduced system. An immediate advantage is that the order of the reduced matrix is roughly half that of the original matrix so the iterative solver applied to (5) requires less work per step. In our experience, solving the reduced system using CG-like iterative methods and incomplete factorizations of S_1 as preconditioners is both faster and far more robust than either solving the original system (1) using the same class of methods or solving the reduced system without preconditioning [13], [15].

We now discuss a generalization of this procedure that can also be used in cases where A is a sparse matrix that can be permuted into block form (3), but for which the Schur complement S_1 is *not* sparse. We focus on a nine-point operator on a two-dimensional rectangular $n \times n$ grid [18]; each gridpoint is connected to its eight nearest horizontal, vertical and diagonal neighbors (see Fig. 1). (In § 6, we briefly describe how the same class of methods can be applied to other operators.) If the natural ordering of gridpoints is used, then A is a block tridiagonal matrix of order n^2, where each block is tridiagonal of order n.

First, recall that CG-like methods only refer to the coefficient matrix via a matrix-vector product. Even if S_1 is dense, the product $S_1 v$ could be evaluated implicitly by computing $u = C_1 v$, using a sparse matrix solver to compute $D_1^{-1} u$, multiplying the result by E_1 and then subtracting that result from the product $F_1 v$. This procedure will be inexpensive, provided that the application of D_1^{-1} is inexpensive. If, as above, the point red-black ordering of gridpoints is used to generate the permutation for (2), then D_1 is a banded matrix with bandwidth approximately n, so that $D_1^{-1} u$ will be an expensive computation. For an alternative, let P_1 correspond to a *line* red-black ordering of the gridpoints (see Fig. 1, right). Then D_1 and F_1 are block diagonal matrices of block order $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$, respectively, with each block a tridiagonal matrix of order n. Hence, computing $D_1^{-1} u$ requires a set of $\lceil n/2 \rceil$ independent tridiagonal solves, C_1 and E_1 are block bidiagonal with the blocks consisting of tridiagonal matrices, and $S_1 v$ can be performed inexpensively.[1]

[1] The implicit matrix-vector product can also be used for the point red-black reduced matrix in the 2-cyclic case. Indeed, on a uniform three-dimensional $n \times n \times n$ grid, the explicit reduced matrix-vector product requires approximately $19n^3/2$ floating point computations, whereas the implicit version needs just $7n^3$, so that the implicit version is actually more efficient. The costs for a uniform two-dimensional grid are $9n^2/2$ for the explicit and $5n^2$ for the implicit versions.

We also want a *preconditioner* for S_1. The individual blocks of D_1^{-1} are dense, so that S_1 is a dense block tridiagonal matrix. However, the off-diagonal entries are bounded in an exponentially decaying manner [9]. In [7], this fact is exploited to derive effective block preconditioners. Here, we propose using a sparse approximation of D_1^{-1} to produce an approximation A_1 of S_1 such that A_1, like A, is a block tridiagonal matrix with tridiagonal blocks. For the preconditioner, we could use an incomplete factorization L_1U_1 of A_1. Thus, an algorithm for solving the reduced system for the nine-point operator is a CG-like method that performs the matrix-vector product S_1v using the individual components of S_1, and uses as preconditioner an incomplete factorization of a sparse approximation A_1 of (the dense banded matrix) S_1. We will consider the symmetrizable version of the preconditioning, i.e., where the preconditioned problem is $[L_1^{-1}S_1U_1^{-1}][U_1x^{(b)}] = L_1^{-1}f^{(b)}$.

Although this technique is suitable for serial computation, it suffers from essentially the same difficulties as standard preconditioners on multiprocessors. An alternative that has better properties for parallel architectures is to apply the reduction procedure in a recursive manner. Note that the sparse matrix A_1 available prior to computing the incomplete factorization preconditioner can be thought of as a difference operator on an $n \times \lfloor n/2 \rfloor$ grid. Therefore, the same type of reduction can be applied to A_1: let P_2 be a permutation matrix such that

$$P_2 A_1 P_2^{-1} = \begin{pmatrix} D_2 & C_2 \\ E_2 & F_2 \end{pmatrix}, \tag{6}$$

and given a factorization $D_2 = G_2H_2$, let

$$V_2 = \begin{pmatrix} G_2^{-1} & 0 \\ -E_2D_2^{-1} & I \end{pmatrix}, \qquad W_2 = \begin{pmatrix} H_2^{-1} & -D_2^{-1}C_2 \\ 0 & I \end{pmatrix}.$$

Let $S_2 = F_2 - E_2D_2^{-1}C_2$. Then (5) is equivalent to the linear system

$$[V_2P_2S_1P_2^{-1}W_2][W_2^{-1}P_2x^{(b)}] = [V_2P_2s_1], \tag{7}$$

and

$$V_2P_2S_1P_2^{-1}W_2 \approx V_2P_2A_1P_2^{-1}W_2 = \begin{pmatrix} I & 0 \\ 0 & S_2 \end{pmatrix}. \tag{8}$$

Thus, premultiplying (5) by V_2P_2 and postmultiplying by $P_2^{-1}W_2$ can be viewed as an *approximate* decoupling of the unknowns of $x^{(b)}$. Proceeding as above, we could then compute $A_2 \approx S_2$, and to stop the reduction here, an incomplete factorization L_2U_2 of A_2. The block factored matrix

$$\mathcal{L}_2\mathcal{U}_2 = \begin{pmatrix} I & 0 \\ 0 & L_2 \end{pmatrix}\begin{pmatrix} I & 0 \\ 0 & U_2 \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & L_2U_2 \end{pmatrix}$$

is then a preconditioner for (7).

Alternatively, the approximate reduction can be continued further. We outline one more stage of reduction. Instead of computing L_2 and U_2, let $\tilde{P}_3$ be a permutation matrix such that

$$\tilde{P}_3A_2\tilde{P}_3^{-1} = \begin{pmatrix} D_3 & C_3 \\ E_3 & F_3 \end{pmatrix},$$

let $D_3 = G_3H_3$, and let the reduction operator be defined as

$$\tilde{V}_3 = \begin{pmatrix} G_3^{-1} & 0 \\ -E_3D_3^{-1} & I \end{pmatrix}, \qquad \tilde{W}_3 = \begin{pmatrix} H_3^{-1} & -D_3^{-1}C_3 \\ 0 & I \end{pmatrix},$$

so that

$$\tilde{V}_3\tilde{P}_3A_2\tilde{P}_3^{-1}\tilde{W}_3=\begin{pmatrix}I & 0\\ 0 & S_3\end{pmatrix}.$$

Then with

$$P_3=\begin{pmatrix}I & 0\\ 0 & \tilde{P}_3\end{pmatrix},\qquad V_3=\begin{pmatrix}I & 0\\ 0 & \tilde{V}_3\end{pmatrix},\qquad W_3=\begin{pmatrix}I & 0\\ 0 & \tilde{W}_3\end{pmatrix}, \tag{9}$$

the three-stage reduced equation is

$$[V_3P_3V_2P_2S_1P_2^{-1}W_2P_3^{-1}W_3][W_3^{-1}P_3W_2^{-1}P_2x^{(b)}]=[V_3P_3V_2P_2s_1], \tag{10}$$

and as in (8), the coefficient matrix of (10) is approximated by

$$\begin{pmatrix}I & 0\\ 0 & \begin{pmatrix}I & 0\\ 0 & S_3\end{pmatrix}\end{pmatrix}.$$

The choices $A_3\approx S_3$ and $L_3U_3\approx A_3$ result in the preconditioner

$$\mathscr{L}_3\mathscr{U}_3=\begin{pmatrix}I & 0\\ 0 & \begin{pmatrix}I & 0\\ 0 & L_3\end{pmatrix}\end{pmatrix}\begin{pmatrix}I & 0\\ 0 & \begin{pmatrix}I & 0\\ 0 & U_3\end{pmatrix}\end{pmatrix}=\begin{pmatrix}I & 0\\ 0 & \begin{pmatrix}I & 0\\ 0 & L_3U_3\end{pmatrix}\end{pmatrix} \tag{11}$$

for (10). In general, an m-stage approximate reduction and symmetrically applied preconditioning produces the linear system

$$[\mathscr{L}_m^{-1}\mathscr{V}_mS_1\mathscr{W}_m\mathscr{U}_m^{-1}][\mathscr{U}_m\mathscr{W}_m^{-1}x^{(b)}]=[\mathscr{L}_m^{-1}\mathscr{V}_ms_1], \tag{12}$$

where

$$\mathscr{V}_m=\prod_{j=m}^{2}V_jP_j,\qquad \mathscr{W}_m=\prod_{j=2}^{m}P_j^{-1}W_j,$$

V_j, W_j, and P_j are defined by analogy with (9), and $\mathscr{L}_j$ and $\mathscr{U}_j$ are as in (11).

Note that the first reduction, based on the permutation P_1 and reduction operators V_1 and W_1, results in the reduced system (5), and no explicit reference to P_1, V_1, or W_1 is needed during the iteration. All subsequent reductions make use of the approximate Schur complements $A_j\approx S_j$, so they are not true decouplings. As a result, for $j\geqq 2$, the permutation matrices P_j and reduction operators V_j, W_j figure explicitly in the representation (12), and these operators must be used in the CG-like procedure. We will discuss the ramifications of this observation in terms of serial and parallel implementations and operation counts in §§ 3 and 5.

The sparse approximate Schur complement A_1 contains $n/2$ block rows, each consisting of tridiagonal matrices of order n (we omit the floor function for simplicity). As we have observed, A_1 corresponds to a difference operator on an $n\times(n/2)$ grid, where every other line in the x-direction is connected. Now consider the permutation of A_1 given by (6). If P_2 is such that the individual blocks are again of order n, then the second approximate Schur complement corresponds to an operator on an $n\times(n/4)$ grid (where every fourth line in the x-direction is connected). Alternatively, P_2 could rearrange the rows and columns so that the blocks of $P_2A_1P_2^{-1}$ are of order $n/2$. This corresponds to a red-black vertical line ordering. The second approximate Schur complement is then a difference operator on an $(n/2)\times(n/2)$ grid, connecting alternate gridpoints in both directions. In general, the set of approximate Schur complements

is associated with a hierarchical set of grids, each one coarser than its predecessor, and for each reduction, there is a choice of directionality. In the following, we will consider two strategies for an m-stage reduction: *one-way*, in which all reductions are made in the same direction; and *alternating*, in which the direction of reduction is changed at every other stage (i.e., the roles of horizontal and vertical are interchanged prior to every even numbered reduction). For the one-way scheme, the jth stage produces an operator on an $n \times (n/2^j)$ grid; for the alternating scheme, at every even numbered stage the number of gridpoints is the same in both directions.

We now discuss some strategies for computing the approximate Schur complements $A_j \approx S_j = F_j - E_j D_j^{-1} C_j$. For simplicity of notation, we drop the subscript j in this discussion and take the matrices to be of block order $n/2$ (for even n), with blocks of order n (i.e., as when $j = 1$). The product $ED^{-1}C$ has block form

$$\begin{pmatrix} \mathscr{E}_{11} & \mathscr{E}_{12} & & & \\ & \mathscr{E}_{22} & \mathscr{E}_{23} & & \\ & & \ddots & \mathscr{E}_{(n/2)-1,n/2} & \\ & & & \mathscr{E}_{n/2,n/2} & \end{pmatrix} \begin{pmatrix} \mathscr{D}_1^{-1} & & & \\ & \mathscr{D}_2^{-1} & & \\ & & \ddots & \\ & & & \mathscr{D}_{n/2}^{-1} \end{pmatrix}$$
$$\times \begin{pmatrix} \mathscr{C}_{11} & & & \\ \mathscr{C}_{21} & \mathscr{C}_{22} & & \\ & \ddots & & \\ & & \mathscr{C}_{n/2,(n/2)-1} & \mathscr{C}_{n/2,n/2} \end{pmatrix}.$$

The resulting matrix is block tridiagonal with typical block row given by

$$\begin{bmatrix} \mathscr{E}_{ii}\mathscr{D}_i^{-1}\mathscr{C}_{i,i-1} & \mathscr{E}_{ii}\mathscr{D}_i^{-1}\mathscr{C}_{ii} + \mathscr{E}_{i,i+1}\mathscr{D}_{i+1}^{-1}\mathscr{C}_{i+1,i} & \mathscr{E}_{i,i+1}\mathscr{D}_{i+1}^{-1}\mathscr{C}_{i+1,i+1} \end{bmatrix}.$$

We are seeking approximations for the dense individual blocks $\mathscr{E}\mathscr{D}^{-1}\mathscr{C}$.

A possible strategy for approximating the Schur complement S is to construct $ED^{-1}C$ exactly and then discard unwanted nonzero values to obtain a sparse approximation. This is an $O(n^3)$ process which, especially for the early reductions, we regard as too expensive. As in [12], [36], our basic plan is to replace each of the blocks with tridiagonal approximations, so that the resulting approximation is block tridiagonal with tridiagonal blocks. We use the notation sp $(ED^{-1}C)$ to denote the sparse approximation, and $A_s = F - \text{sp}\,(ED^{-1}C)$ for the approximate Schur complement. We consider three options for computing A_s:

(1) Replace $\mathscr{E}\mathscr{D}^{-1}\mathscr{C}$ with its tridiagonal part.

(2) *Diagonal row sum modification.* Let $\tilde{A}_s$ denote the result of the approximation using option (1). Then let A_s denote the matrix obtained by modifying the diagonal of $\tilde{A}_s$ so that the row sum of A_s is the same as the row sum of S. The required computation is

$$\text{diag}\,(A_s) \leftarrow \text{diag}\,(\tilde{A}_s) + \text{Rowsum}\,(S) - \text{Rowsum}\,(\tilde{A}_s),$$

where Rowsum (M) is the diagonal matrix whose ith entry is the row sum of the ith row of M.

(3) *Block row sum modification.* Given $\tilde{A}_s$ as in (2), let A_s denote the matrix obtained by modifying the diagonal of each block so that its row sum is the same as that of the corresponding block of S.

We show in § 3 that all these computations can be done with $O(n^2)$ floating point operations. Other heuristics of this type are discussed in [4], [7], [16].

After the last approximate Schur complement A_m is computed, an incomplete factorization L_mU_m of A_m is required. We examine the standard ILU [29] and MILU [10], [19] incomplete factorizations, and the line Jacobi preconditioning [20], in which A_m is approximated by its block diagonal (consisting of a set of tridiagonal matrices).

The following result shows that the property of diagonally dominant M-matrix is preserved by the unmodified approximate reduction. The proof is essentially the same as those of Lemma 1 of both [4] and [7].

THEOREM 1. *If A is a diagonally dominant M-matrix, then each of the approximate reduced Schur complements $\{A_j\}$ constructed using option* (1) *is also a diagonally dominant* M-*matrix.*

Proof. The first Schur complement $S_1 = F_1 - E_1D_1^{-1}C_1$ is the matrix obtained after $n/2$ (the order of D_1) columns of A are eliminated by Gaussian elimination. Hence, S_1 is also a diagonally dominant M-matrix [17]. Also, every diagonal block of A is an M-matrix, so that $D_1^{-1} \geqq 0$. But since $C_1 \leqq 0$ and $E_1 \leqq 0$, we have

$$0 \leqq \text{tri}\,(E_1\ \text{penta}\,(D_1^{-1}C_1)) \leqq E_1D_1^{-1}C_1.$$

(Here tri (M) and penta (M) denote the approximations to M in which blocks of order n are replaced by their tridiagonal and pentadiagonal parts, respectively.) Hence, the offdiagonal elements of $A_1 = F_1 - \text{sp}\,(E_1D_1^{-1}C_1)$ are also nonpositive, and A_1 is more diagonally dominant than S_1. The result for later stages follows by induction. □

We remark that the methods presented here are symmetrizable variants of techniques presented in [16]. We prefer these versions because of their superior numerical performance (see § 4).

3. Serial operation counts. In this section, we compute the serial operation counts for using the approximate Schur complement preconditioning with a CG-like iterative method. We count floating point multiplications and divisions only, although we point out some extra overhead involving just additions and subtractions for some of the methods. We distinguish two steps:

(1) preprocessing: computing the approximate Schur complements and incomplete factorization;

(2) the work per iteration, including the matrix-vector product, preconditioning operation, and the inner products and scalar-vector products performed by the CG-like method Orthomin(1), [14].

As above, we use the nine-point finite-difference matrix on an $n \times n$ grid. We also present the costs of using the ILU preconditioning with no extra fill for the original system (step 1).

3.1. Preprocessing. We describe the preprocessing costs in detail. They are dominated by the cost of computing sp $(E_jD_j^{-1}C_j)$ plus the cost of the row sum modifications, for j varying between 1 and m. We will need the following result. The proof given is due to Meyerink [30]; see also [4] and [7].

LEMMA 1. *Let $\mathcal{D}$ denote a nonsingular tridiagonal matrix of order n with nonzero off-diagonal bands. Then for odd k, the k interior bands of $\mathcal{D}^{-1}$ can be computed with fewer than $(k+5)n$ multiplications.*

Proof. Let $\mathcal{D} = (I - \mathcal{L})\Delta^{-1}(I - \mathcal{U})$ denote an LDU factorization, which requires $4n$ multiplications to compute. Let $\{\delta_i\}$ denote the diagonal entries of Δ. The needed elements $[s_{ij}]$ of $\mathcal{S} = \mathcal{D}^{-1}$ can be computed by the following algorithm:

$s_{nn} = \delta_n, \qquad s_{ii} = \delta_i + u_{i,i+1}s_{i+1,i+1}l_{i+1,i}, \qquad i = n-1, \cdots, 1.$
For $j = 1$ to $(k-1)/2$
  For $i = j+1$ to n
    $s_{i,i-j} = s_{i,i-j+1}l_{i-j+1,i-j}$
    $s_{i-j,i} = u_{i-j,i-j+1}s_{i-j+1,i}$
  End
End

The costs are $2n$ for the diagonal elements and at most $(k-1)n$ for the off-diagonals. the fact that $\mathcal{S} = \mathcal{D}^{-1}$ can be verified by expressing $(I-\mathcal{L})^{-1}$ and $(I-\mathcal{U})^{-1}$ in terms of elementary triangular factors and then computing $(I-\mathcal{U})^{-1}\Delta(I-\mathcal{L})^{-1}$. □

First consider the *unmodified* factorization, option 1, with the one-way reductions. The tridiagonal part of $\mathcal{E}\mathcal{D}^{-1}\mathcal{C}$ can be computed from $\mathcal{E}$ and the pentadiagonal part of $\mathcal{D}^{-1}\mathcal{C}$, which in turn requires seven diagonals of $\mathcal{D}^{-1}$. The costs for the jth step are shown in Table 1, where the first line comes from Lemma 1 with $k = 7$. Summing over the m reductions gives the preprocessing costs for this strategy as $144n^2(1-(1/2^m))$.

TABLE 1
Costs of the unmodified one-way reductions.

Compute 7-diagonal approximation to $\mathcal{D}^{-1}$	$12n$
Compute block of penta $(\mathcal{D}^{-1}\mathcal{C})$	$15n$
Compute block of tri ($\mathcal{E}$ penta $(\mathcal{D}^{-1}\mathcal{C})$)	$9n$
Total per block	$36n$
$\times 4$ (= no. of block products per block row)	$144n$
$\times n/2^j$ (= no. of block rows)	$144n^2/2^j$

The row sum of S_j can be obtained from the matrix-vector product $S_j\mathbf{1}$, where $\mathbf{1}$ is the vector all of whose entries are 1. This computation requires $18n^2/2^j$ multiplications. Hence, option (2) can be implemented at an extra cost of $18n^2(1-(1/2^m))$. Some savings can be achieved by summing the elements of C_j rather than multiplying by $\mathbf{1}$. Note that we are not counting the additions and subtractions needed to compute the row sum of (the unmodified matrix) $\tilde{A}_s$ or to perform the diagonal modification. The block row sum of $\mathcal{E}\mathcal{D}^{-1}\mathcal{C}$ can be computed in the same way at a cost of $9n$ per block, for four blocks per block row and $n/2^j$ block rows, making a total of $36n^2/2^j$ multiplications. Thus, the extra cost for option (3) is $36n^2(1-(1/2^m))$. Again, some savings can be achieved by summing the appropriate entries of C instead of multiplying by $\mathbf{1}$, and we have not counted the costs of summing the blocks of $\tilde{A}_s$ and performing the modifications.

The operation counts for the alternating reductions are identical. To see this, note that the counts at each stage all have the form $(c_1 n_{bs})(c_2 n_{br})$, where n_{bs} is the blocksize, n_{br} is the number of block rows, and c_1 and c_2 are independent of these values. For the one-way schemes, n_{br} is reduced by a factor of two at each stage while n_{bs} is fixed at n. For the alternating schemes, one or the other of these quantities is reduced by a factor of two at each stage, so the product is the same as for the one-way schemes. (If n is not a power of two, then the repeated use of the floor function makes the one-way reduction slightly less expensive.) We will see in § 5 that this equivalence does *not* hold for parallel implementations

We also require that the preconditioners be *symmetrizable*, i.e., that they have the property that when they are applied to a symmetric positive-definite matrix, the preconditioned operator of (12) is also symmetric positive-definite. Thus, we compute

the triangular factorization $D_j = G_j H_j$ so that diag (G_j) = diag (H_j) for $j \geqq 2$. This can be done after all approximate Schur complements are computed. In the notation of Lemma 1, we compute $\Delta^{1/2}$ for stages 2 through m, and subsequently represent the factorization as $\mathcal{D} = [(I - \mathcal{L})\Delta^{-1/2}][\Delta^{-1/2}(I - \mathcal{U})]$. For simplicity, we take the square root to be as costly as a multiplication, making the extra cost $n^2/2(1-(1/2^m))$.

Finally, the operation counts for the ILU factorization of the nine-point matrix on an $n \times n$ grid is $22n^2$, and for MILU it is $26n^2$. The same result shows that the incomplete factorization of the last approximate Schur complement is $22n^2/2^m$ for ILU and $26n^2/2^m$ for MILU. These costs are also for symmetrizable operators; the costs are slightly lower without this property. The operation counts for preprocessing are summarized in Table 2.

3.2. Work per step of preconditioned Orthomin(1). We compute the costs of solving both the preconditioned reduced system and the ILU-preconditioned full system using Orthomin(1) as the iterative solver. This method is equivalent to the conjugate residual method when the coefficient method is symmetric positive-definite; see e.g., [14], [20], [38] for properties of this and other nonsymmetric CG-like methods. For a linear system of order N, Orthomin(1) performs $7N$ operations plus one (preconditioned) matrix-vector product per step. For the reduced system, $N = n^2/2$, and the preconditioned matrix product consists of the operation

$$w \leftarrow [\mathcal{L}_m^{-1} \mathcal{V}_m S_1 \mathcal{W}_m \mathcal{U}_m^{-1}] v.$$

A straightforward analysis shows that the matrix-vector product $w \leftarrow S_1 v$ requires $9n^2$ operations. By an analysis similar to that just given for the preprocessing step, we can show that the reduction operations $w \leftarrow \mathcal{V}_m v$ and $w \leftarrow \mathcal{W}_m v$ each require

$$\sum_{j=2}^{m} 10n^2/2^j = 5n^2(1 - 1/2^{m-1})$$

arithmetic operations. Here, the individual summands are the cost of multiplying by V_j or W_j. The triangular solves $w \leftarrow L_m^{-1} v$ and $w \leftarrow U_m^{-1} v$ each require $5n^2/2^m$ computations. Adding these terms gives the cost of the preconditioning as $10n^2(1-1/2^m)$. Consequently, the work per step for using Orthomin(1) to solve the preconditioned reduced system is $22.5n^2 - 10n^2/2^m$. The permutations $w \leftarrow P_j v$ can be implemented using permutation vectors $\{p_j\}$; the ith component satisfies $w(i) = v(p_j(i))$; the inverse operation $w \leftarrow P_j^{-1} v$ is $w(p_j(i)) = v(i)$. Although the overhead for such indirect addressing may not be negligible, for simplicity we ignore it in our operation counts.

For the original system, $N = n^2$, so that each step of Orthomin(1) requires $7n^2$ operations, plus $9n^2$ for the matrix-vector product, and $10n^2$ for preconditioning by ILU. We summarize the costs per iteration as follows:

TABLE 2
Preprocessing operation counts.

Method	Costs
No Modification	$144.5n^2 - 123n^2/2^m$
Diagonal row	$162.5n^2 - 141n^2/2^m$
Block row	$180.5n^2 - 159n^2/2^m$
ILU/Full	$22n^2$
MILU/Full	$26n^2$

THEOREM 2. *If Orthomin(1) is used to solve the linear system arising from the nine-point finite-difference discretization of a two-dimensional elliptic differential equation, the cost per step is*

$$\begin{cases} 22.5n^2 - 10n^2/2^m & \text{for the reduced system with an } m\text{-stage approximate Schur complement preconditioning,} \\ 26n^2 & \text{for the full system with ILU or MILU preconditioning.} \end{cases}$$

We remark that the relative costs of the reduced system techniques would be lower for Orthomin(k) with $k > 1$, and that a savings of n^2 is possible for both the reduced and full systems if the symmetrizable forms are not used. The storage requirements are roughly $9n^2/2^j$ for the jth approximate Schur complement, $j = 1, \cdots, m$. The sparse factorization can overwrite A_m. Thus, the total (ignoring pointer storage) is $9n^2(1 - (1/2^m))$, compared to $9n^2$ for the full system incomplete factorization.

4. Numerical experiments with nine-point operators. In this section, we discuss the results of numerical experiments in which the approximate Schur complement preconditioners were used to solve a nine-point finite difference discretization of the two-dimensional elliptic equation

$$-\Delta u + 2\alpha u_x + 2\beta u_y = g \tag{13}$$

on the unit square $0 \leqq x, y \leqq 1$ with homogeneous Dirichlet boundary conditions. We discretized (13) on a uniform $n \times n$ grid, approximating the Laplacian by

$$\Delta_9 = \tfrac{2}{3}\Delta_5 + \tfrac{1}{3}\Delta_\times .$$

Here

$$(\Delta_5 u)_{ij} = -[4u_{ij} - u_{i-1,j} - u_{i-1,j} - u_{i,j-1} - u_{i,j+1}]/h^2,$$

$$(\Delta_\times u)_{ij} = -[4u_{ij} - u_{i-1,j-1} - u_{i+1,j-1} - u_{i-1,j+1} - u_{i+1,j+1}]/(2h^2),$$

where $h = 1/n+1$ and $u_{ij} \approx u(ih, jh)$. The first derivatives were approximated by centered differences:

$$u_x(ih, jh) \approx (u_{i+1,j} - u_{i-1,j})/(2h), \qquad u_y(ih, jh) \approx (u_{i,j+1} - u_{i,j-1})/(2h).$$

(The resulting scheme is therefore still second order.) Rather than specify the right-hand side g for (13), for the discrete problem (1) we used $f \equiv 0$. We considered three sets of values for the coefficients of (13):

(1) $\alpha = 1$, $\beta = 2$
(2) $\alpha = 25$, $\beta = 50$
(3) $\alpha = \beta = 0$.

The resulting matrices are nonsymmetric for problems 1 and 2, and symmetric positive definite for problem 3. They are diagonally dominant if and only if $|\alpha h| \leqq 2/3$ and $|\beta h| \leqq 2/3$.

The experiments were run on a VAX-8600 in double precision Fortran (55-bit mantissa). For all tests, we chose the initial guess u_0 to be a vector of random numbers in the interval $(-1, 1)$. The stopping criterion was $\|r_i\|/\|r_0\| \leqq 10^{-6}$, where $r_i = f - Au_i$ is the residual at the ith iteration and $\|v\|$ is the Euclidian norm $(v^T v)^{1/2}$. We used Orthomin(1) for the CG-like solver, which is equivalent to the conjugate residual method for problem 3.

Iteration counts for a mesh size of $h = \frac{1}{64}$ ($n = 63$), are shown in Tables 3, 4, and 5. These tables include results for the three options for approximating $\mathscr{E}\mathscr{D}^{-1}\mathscr{C}$ presented in § 2, where the ILU incomplete factorization is used for the last approximate Schur

TABLE 3
Iteration counts for problem 1 ($\alpha = 1$, $\beta = 2$), *with* $h = \frac{1}{64}$.

# Reductions	No modifications one-way alternating		Diagonal row one-way alternating		Block row one-way alternating	
1	55		43		45	
2	51	45	28	25	32	25
3	49	43	17	19	22	19
4	49	43	24	13	16	12
5	49	43	*	14	17	11
6	*	43	*	14	*	11
7	*	43	*	15	*	12
0	ILY/Full: 50		MILU/Full: 20			

TABLE 4
Iteration counts for problem 2 ($\alpha = 25$, $\beta = 50$), *with* $h = \frac{1}{64}$.

# Reductions	No modifications one-way alternating		Diagonal row one-way alternating		Block row one-way alternating	
1	19		9		11	
2	23	24	13	11	18	10
3	23	25	21	21	22	18
4	23	25	31	23	21	19
5	23	25	*	24	21	19
6	*	25	*	27	*	20
7	*	25	*	28	*	20
0	ILU/Full: 14		MILU/Full: 6			

TABLE 5
Iteration counts for problem 3 ($\alpha = 0$, $\beta = 0$), *with* $h = \frac{1}{64}$.

# Reductions	No modifications one-way alternating		Diagonal row one-way alternating		Block row one-way alternating	
1	28		23		23	
2	26	24	17	15	19	15
3	26	24	16	13	16	13
4	25	24	18	12	16	10
5	25	24	*	13	16	11
6	*	24	*	13	*	11
7	*	24	*	14	*	11
0	ILU/Full: 25		MILU/Full: 20			

complement. An asterisk indicates that the preconditioning is not well defined, either because of a mesh size of 0 or (for the diagonal row modified one-way scheme with five reductions) a zero diagonal entry in the last diagonal block D_j. Each of the tables also shows the iteration counts for solving the preconditioned full system $[L^{-1}AU^{-1}] \times [Ux] = L^{-1}f$ (corresponding to *no* reductions), using both the ILU and MILU factorizations for the preconditioner LU.

The total cost (in operation counts) of solving a problem is

$$\text{Preprocessing cost} + (\#\text{ of Steps}) \times (\text{Work per step}).$$

In Table 6, we compare these costs for the most effective choice of Schur complement method, based on the alternating scheme with option 3 (block row modification), to those of the full system MILU preconditioner. The entries in the table are the ratios of the costs of Schur complement method to the costs of solving the full system with the MILU preconditioning. The last line compares the cost of using the ILU preconditioning for the full system to that of MILU; dividing the other entries in a column by these numbers gives a comparison of the Schur complement method to ILU. The preprocessing costs and costs per step are taken from Table 2 and Theorem 2, for $n = 63$.

In Table 7, we compare the effect of the incomplete factorization of the last approximate Schur complement A_m. In addition to the ILU and MILU factorizations, we examined the line Jacobi factorization, which uses the block diagonal of A_m (a tridiagonal matrix) as the incomplete factorization [20]. This data is for problem 1, for the alternating scheme and the diagonal row sum modification (option 2). The qualitative results for the other problems and approximate Schur complement strategies are similar.

Finally, in Table 8, we show how iteration counts vary with mesh size for all three problems, for the alternating scheme with four reductions and ILU applied to the last

TABLE 6

Ratio of costs of approximate Schur complement preconditioners (alternating scheme with option 3), to those of the full system MILU *preconditioner.*

# Reductions	Problem 1	Problem 2	Problem 3
1	1.63	1.61	.92
2	1.17	1.87	.81
3	1.03	2.98	.80
4	.79	3.22	.71
5	.77	3.28	.77
6	.78	3.43	.78
7	.82	3.45	.78
ILU/Full	2.42	2.12	1.23

TABLE 7

Effect of the incomplete factorization for the last Schur complement. Problem 1 *with the alternating scheme and diagonal row sum modification (option* 2).

# Reductions	ILU	MILU	Line Jacobi
1	43	42	95
2	25	19	104
3	19	20	53
4	13	16	42
5	14	17	22
6	14	17	18
7	15	16	16

TABLE 8
Iteration counts for different mesh sizes, using the alternating scheme with four reductions.

	No modifications			Diagonal row			Block row		
Problem	1	2	3	1	2	3	1	2	3
$h = 1/16$	12	9	8	8	15	8	7	12	7
$h = 1/32$	23	15	13	10	20	10	9	16	9
$h = 1/64$	43	25	24	13	23	12	12	19	10
$h = 1/128$	78	48	38	23	18	15	22	16	14

approximate Schur complement. Experimental results for the one-way scheme showed similar asymptotic behavior.

We make the following observations on these results:

(1) In nearly all cases, the alternating scheme has better performance than the one-way scheme. The strategies based on row sum modifications are more effective than the unmodified versions, and the block row sum modification is superior to the diagonal row sum modification. In most cases, performance is not dramatically affected by the number of reductions.

(2) For two of the three problems, the best Schur complement preconditioner is more effective than the MILU point preconditioner. For the other problem (number 2), the Schur technique is slower by about a factor of three.

(3) For the incomplete factorization of A_m, the Jacobi method is less effective if a small number of reductions is used, but it is competitive for a large number of reductions. There appears to be no advantage to using MILU for this factorization.

(4) For the discrete Poisson equation (problem 3), the iteration counts appear to be proportional to $1/\sqrt{h}$ for the modified Schur techniques, and to $1/h$ for the unmodified methods. The pattern is less clear for the nonsymmetric problems. For the mildly nonsymmetric problem 1, the counts are roughly proportional to $1/h$ for all variants. Problem 2 is highly nonsymmetric and not diagonally dominant except on the fine mesh $h = \frac{1}{128}$, and the iteration counts actually decrease when h is changed from $\frac{1}{64}$ to this value.

We remark that these results are dramatically superior to the results for the *one-sided* approximate Schur complement factorizations presented in [16].

5. Parallel implementation and costs. In this section we describe and analyze the costs of parallel implementation of the approximate Schur complement preconditioners. We focus on the case of n processors used to solve a nine-point problem on an $n \times n$ grid. For simplicity, we take n to be a power of two. We first determine the complexity of parallel arithmetic *alone* by considering an idealized shared-memory model in which communication is free; we show that the $O(n^2)$ serial cost (at each iteration) can be reduced to $O(n)$ in parallel. Then we factor in communication costs on a hypercube with message passing. We show that on a $(\log_2 n)$-dimensional cube, the one-way scheme needs $O(mn)$ communication and the alternating scheme requires just $O(n)$. For the methods under consideration, computations and communications are intermeshed in a natural way and we do not attempt to overlap them. Communication costs are in units of "data-items sent in parallel," where distinct processors can send and receive simultaneously. Since there is usually some overhead associated with initiating a communication, we also specify the number of *startups* needed, under the assumption that passing a block of data of arbitrary size requires one startup.

5.1. Preliminaries. We first establish some conventions of notation. Recall that the original matrix A and the approximate Schur complements are associated with a set of hierarchical grids, where the first grid is of size $n \times n$ and each successive one is constructed from its predecessor by removing every other line in one of the directions. We use subscripts to represent vectors defined on these grids, i.e., for $1 \leqq j \leqq m$, v_j is a vector defined on the jth grid. Any such v_j can be considered as being divided approximately equally into alternating "red" lines $v_j^{(r)}$ and "black" lines $v_j^{(b)}$. For example, in Fig. 1, right, $v_1^{(r)}$ corresponds to the lower-numbered points and $v_1^{(b)}$ corresponds to the higher-numbered points. Note that $v_j^{(b)} = v_{j+1}$. To distinguish among individual lines of a grid, $v_j^{(ri)}$ ($v_j^{(bi)}$) refers to the ith red (respectively, black) line of the jth grid. As in § 2, the rows of the matrices are divided analogously, with D_j and C_j associated with red lines and E_j and F_j with black lines. (One slight ambiguity here is that the jth Schur complement S_j and approximation A_j are actually defined on the $(j+1)$st grid. However, the notation is such that subscripts agree in all formulas for matrix computations presented below.) Finally, the individual processors of a k-processor machine are denoted by $\{\mathrm{PE}_j\}_{j=1}^{k}$.

Before examining the Schur complement preconditioners, we briefly consider the reduced matrix-vector product and the sparse solves for the ILU preconditioner. (The latter operations are used for both the full system and as part of the Schur complement methods.) See [1], [37], [43] for parallel implementations of the other operations of CG-like methods.

For n processors, we assume that the unknowns corresponding to the jth line of the full mesh are handled by the jth processor and that the matrix entries are processed by block row in an analogous manner. For message-passing systems, we assume that unknowns and matrix entries are distributed among local memories by blocks in the same way, and that the data corresponding to neighboring grid lines are stored in neighboring processors. An example for $n = 6$ showing how this distribution of data is handled for a line red-black ordered system is as follows:

$$
(14)\qquad
\begin{array}{l}
\mathrm{PE}_1: \\ \mathrm{PE}_3: \\ \mathrm{PE}_5: \\ \mathrm{PE}_2: \\ \mathrm{PE}_4: \\ \mathrm{PE}_6:
\end{array}
\begin{pmatrix}
D_1^{(1)} & & & C_1^{(11)} & & \\
 & D_1^{(2)} & & C_1^{(21)} & C_1^{(22)} & \\
 & & D_1^{(3)} & & C_1^{(32)} & C_1^{(33)} \\
E_1^{(11)} & E_1^{(12)} & & F_1^{(1)} & & \\
 & E_1^{(22)} & E_1^{(23)} & & F_1^{(2)} & \\
 & & E_1^{(33)} & & & F_1^{(3)}
\end{pmatrix}
\begin{pmatrix} v_1^{(r1)} \\ v_1^{(r2)} \\ v_1^{(r3)} \\ v_1^{(b1)} \\ v_1^{(b2)} \\ v_1^{(b3)} \end{pmatrix}
$$

Data corresponding to red lines, D_1, C_1, and $v_1^{(r)}$, are stored in odd-numbered processors, and data corresponding to black lines, E_1, F_1, and $v_1^{(b)}$, are in even-numbered processors. For message passing, there are connections (not shown) between processors numbered j and those numbered $j+1$. (The matrix superscripts are the indices within the blocks D_1, C_1, E_1, and F_1.)

The reduced matrix-vector product $w_1^{(b)} \leftarrow S_1 v_1^{(b)} = F_1 v_1^{(b)} - E_1(D_1^{-1}(C_1 v_1^{(b)}))$ then consists of the following three steps:

(1) $s_1^{(r)} \leftarrow C_1 v_1^{(b)}$, $s_1^{(b)} \leftarrow F_1 v_1^{(b)}$	$6n$ multiplications (for $C_1 v_1^{(b)}$)
(2) $t_1^{(r)} \leftarrow D_1^{-1} s_1^{(r)}$	$3n$ multiplications
(3) $w_1^{(b)} \leftarrow E_1 t_1^{(r)} + s_1^{(b)}$	$6n$ multiplications.

These must be performed in the sequential order given (except the multiplication $F_1 v_1^{(b)}$, which could be postponed to step 2). The parallel costs are shown on the right.

The key to parallel implementation is that the individual block rows of each of the matrices D_1, C_1, E_1, and F_1 can be handled independently. For example, parallel multiplication by C_1 requires just $6n$ multiplications, the cost of multiplying by each of the block rows. Thus, the $9n^2$ serial cost of arithmetic is reduced to $15n$ on n processors. On message-passing systems, the computation $C_1 v_1^{(b)}$ must be preceded by communication of the blocks of $v_1^{(b)}$ in two directions, from even-numbered processors to neighboring odd-numbered processors (except at the boundary, where a shift is performed in just one direction). For (14), $v_1^{(b1)}$ is moved from PE_2 to PE_1 and PE_3, $v_1^{(b2)}$ is moved from PE_4 to PE_3 and PE_5, and $v_1^{(b3)}$ is moved from PE_6 to PE_5. Simultaneous parallel multiplication by C_1 and F_1 is done on the odd- and even-numbered processors, respectively, followed by parallel execution of the tridiagonal solves required for $D_1^{-1} s_1^{(r)}$ on the odd-numbered processors. The blocks of $t_1^{(r)}$ must then be moved from the odd processors to neighboring even processors, where the computation is completed. Thus, all data movement is local and in blocks of length n. For simplicity of analysis of message passing, we adopt the conventions that these bidirectional shifts of data can be performed simultaneously and that processors can simultaneously send to and receive data from neighbors.

Although the triangular solves required by sparse ILU preconditioners for naturally ordered discretized PDEs (partial differential equations) are commonly viewed as the bottleneck for parallel iterative methods, considerable parallelism is possible. The following technique is a straightforward generalization to the nine-point operator of five-point methods introduced in [21] and analyzed in [39]. Figure 2 shows the stencil for the lower triangular solve and a depiction of a parallel implementation on a 6×6 grid. The unknown "*" depends on the four points below it and to its left; hence a parallel solve can move in a front across the grid, from lower left to upper right. The integers in the grid on the right side of the figure correspond to the time steps at which the grid values can be computed. In the following, we will be concerned with an $n_1 \times n_2$ grid on n_1 processors, where $n_1 \geqq n_2$. Ignoring communication costs, the sweep can be performed in $n_1 - 1 + 2n_2 - 1 = n_1 + 2n_2 - 2$ arithmetic steps, where each gridpoint requires roughly five multiplications. The cost for the upper-triangular solve is the same, so the total cost of arithmetic is $10(n_1 + 2n_2)$.

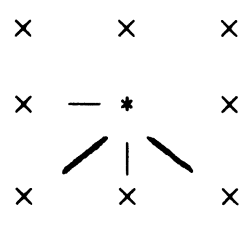
On message-passing systems, it is possible to optimize costs as a function of arithmetic speed, startup cost and bandwidth (cf. [39]). For example, let α denote the cost (in time) of a floating point operation, and let $\sigma + \tau l$ denote the cost of sending l items of data to nearest neighbors. Suppose the n_1 lines of the grid are distributed among processors so that neighboring lines are in neighboring processors; this is possible, for example on a linear array of processors. The lower-triangular solve can be performed by computing l contiguous values in a line and then passing the results to a neighbor in blocks of size l. Assuming that arithmetic and communication are not overlapped, the cost is

$$\left(\frac{n_1}{l} + 2(n_2 - 1)\right) 5\alpha l + \left(\frac{n_1}{l} + 2(n_2 - 2)\right)(\sigma + \tau l). \tag{15}$$

(This can be seen from Fig. 2, where the integers now represent horizontal blocks of size l, i.e., $n_1/l = 6$ and $n_2 = 6$. The communication cost is lower because data on the topmost line need not be moved.) As a function of l, (15) is minimized when

$$l^2 = \frac{n_1 \sigma}{10\alpha(n_2 - 1) + 2(n_2 - 2)\tau} \approx \frac{n_1}{n_2} \frac{\sigma}{10\alpha + 2\tau}.$$

Using the approximation on the right, we find that the time spent on arithmetic (i.e.,



PE_6:	11	12	13	14	15	16
PE_5:	9	10	11	12	13	14
PE_4:	7	8	9	10	11	12
PE_3:	5	6	7	8	9	10
PE_2:	3	4	5	6	7	8
PE_1:	1	2	3	4	5	6

FIG. 2. *Stencil and pipelined implementation of the lower-triangular solve.*

the coefficient of α in (15)) is $5n_1+10\sqrt{n_1n_2}(\sigma/(10\alpha+2\tau))^{1/2}$, the time spent sending data (coefficient of τ) is $n_1+2\sqrt{n_1n_2}(\sigma/(10\alpha+2\tau))^{1/2}$, and the time for startups is $2n_2+\sqrt{n_1n_2}((10\alpha+2\tau)/\sigma)^{1/2}$.[2] Note that startup cost is large if $n_2=n$ as for the full system, but it will be small for small n_1 or n_2, as when a large number of reductions is used for the approximate Schur complement factorization.

The approximate Schur complement preconditioner performs three types of computations at each step of a CG-like iteration:

(1) left reductions: $w \leftarrow \mathcal{V}_m v = \prod_{j=m}^{2} V_j P_j v$,
(2) right reductions: $w \leftarrow \mathcal{W}_m v = \prod_{j=2}^{m} P_j^{-1} W_j v$,
(3) sparse triangular solves: $w \leftarrow \mathcal{L}_m^{-1} v$, $w \leftarrow \mathcal{U}_m^{-1} v$.

In § 5.2, we examine the costs of these computations in an idealized setting in which communication is free, and in § 5.3, we present the communication costs on a hypercube. It is easily verified that the left and right reductions have identical costs. In the following, we restrict our analysis to the left reductions.

5.2. Idealized shared memory architectures. Consider the arithmetic operations performed by the left reduction $w \leftarrow \mathcal{V}_m v$. The jth step $w_j \leftarrow V_j v_j$ has the form

$$w_j^{(r)} \leftarrow G_j^{-1} v_j^{(r)}, \qquad w_j^{(b)} \leftarrow v_j^{(b)} - E_j H_j^{-1} w_j^{(r)}. \tag{16}$$

As in the matrix-vector product, a parallel implementation handles the individual sub-blocks of G_j and H_j and the block rows of E_j independently. For the one-way reduction scheme, each step requires approximately $10n$ operations, so the cost of the full left reduction is $10n(m-1)$. For the alternating scheme, the size of the blocks goes down with increasing j according to the sequence

$$\begin{array}{lll} j=2\text{: } n/2, & j=4\text{: } n/4, & j=6\text{: } n/8, \\ j=3\text{: } n/2, & j=5\text{: } n/4, & j=7\text{: } n/8, \end{array} \quad \ldots.$$

The cost of the left reduction is, therefore,

$$\begin{cases} 2\sum_{k=1}^{m/2-1} 10\times n/2^k + 10n/2^{m/2} = 20n - 30m/2^{m/2} & \text{for even } m \\ 2\sum_{k=1}^{(m-1)/2} 10\times n/2^k = 20n - 20n/2^{(m-1)/2} & \text{for odd } m. \end{cases}$$

In the following we restrict our attention to even m.

Under the assumption that communication through the shared memory is free, the permutations performed by the left reductions and inverse permutations of the right reductions can be implemented with indirect addressing as in the serial case, and we again neglect their cost.

[2] Here α, σ, and τ are taken to be dimensionless, so it makes sense for them to appear in these expressions. It is also possible to overlap communication and arithmetic to advantage, although the asymptotic costs are of the same order of magnitude; e.g., let $\alpha=0$.

We must also include the cost of the sparse solves $\mathcal{L}_m^{-1}v_m$ and $\mathcal{U}_m^{-1}v_m$. For the one-way scheme, these computations are done on an $n \times n/2^m$ grid, and for the alternating scheme, on an $n/2^{m/2} \times n/2^{m/2}$ grid. For both the ILU and MILU factorizations, as outlined above, the costs are $10(n+(2n/2^m))$ for the one-way scheme and $30n/2^{m/2}$ for the alternating scheme. For the Jacobi factorization, the costs are $10n$ and $10n/2m^{m/2}$, respectively.

Summing the costs of the reductions and sparse solves (and using the fact that the left and right reductions have the same cost), we have the following result:

THEOREM 3. *On an idealized shared memory system where communication is free, the costs per iteration of the m-stage approximate Schur complement preconditioners are bounded by*

$$\begin{cases} 20nm - 10n + 20n/2^m & \textit{for the one-scheme,} \\ 40n - 30n/2^{m/2} & \textit{for the alternating scheme (with m even).} \end{cases}$$

These operation counts reflect the cost for using the ILU and MILU sparse solves; with the Jacobi factorization, the costs are slightly lower. Note that in contrast to the serial case, the two schemes have different parallel costs. For most m, the alternating scheme is less expensive. The reasons are that, as we have observed, the size of the sub-blocks is decreasing at every other stage, whereas the size is fixed at n for the one-way scheme; and, the number of independent tasks at each stage is greater. For the one-way scheme, this number decreases by a factor of two at each stage, but for the alternating scheme it decreases only at every other stage.

Finally, the $O(n^2)$ serial preprocessing costs (see § 3.1) are also reduced to $O(nm)$ for the one-way scheme and $O(n)$ for the alternating scheme. For example, for the unmodified one-way scheme, the next to last line of Table 1 shows the cost of computing a block row of A_j. Since each of these block rows can be processed in parallel, these are precisely the parallel costs. Thus, an m-stage unmodified one-way preprocessing step requires $144nm$ computations. Similar considerations hold for the modifications and the alternating scheme, see [16].

5.3. Costs on a hypercube with message passing. In § 5.1, we showed that data movement for the reduced matrix-vector product can be implemented efficiently if contiguous processors contain block rows of the coefficient matrix corresponding to contiguous lines of the grid. This distribution of data can be made on a linear array of processors. The preconditioning operations consist of matrix computations similar to those of the matrix-vector product, but on a set of hierarchical grids. In this section we present an analysis showing that the communication required by the preconditioners can be done efficiently on a hypercube.

Assume that each approximate Schur complement A_j is distributed by block row among processors in a manner analogous to the storage of the original matrix, i.e., that any block row is stored in the memory of one processor. (We will specify the exact distribution below.) The diagonal blocks of $D_j = G_j H_j$ can be stored in factored form. Three types of communication are required. Consider the left reduction $w \leftarrow \mathcal{V}_m v$, whose jth step $w_j \leftarrow V_j P_j v_j$ consists of a permutation followed by a reduction step. The first type of communication is for the permutation, in which data is moved between grid levels. That is, P_j moves $v_j (= v_{j-1}^{(b)})$ from processors using it as the black points on the $(j-1)$st grid to those that reference it as $(v_j^{(r)}, v_j^{(b)})^T$ on the jth grid. The second type occurs during the reduction step, in which data is moved between processors containing neighboring lines of a given grid. These movements mirror those of the matrix-vector product. In (16), they occur between the application of the operators H_j^{-1} and E_j. Thus, for both these data movements to be inexpensive, successive grids

in the hierarchy should be stored in nearby processors, and neighboring lines of each grid should be stored near one another. The third type of communication is for the sparse triangular solves, which are performed on the coarsest grid; the costs will be determined from the analysis of § 5.1.

Let the processors in the hypercube be numbered by a binary reflected Gray code [35]. That is, the n-processor cube is identified with the unit cube of real $(\log_2 n)$-space, with processors associated with bit sequences given by Cartesian coordinates. The numbering is determined inductively as follows. If $c_1, c_2, \cdots, c_{n/2}$ is the coordinate ordering for the $n/2$-processor of next lower dimension, then $0c_1, 0c_2, \cdots, 0c_{n/2}$, $1c_{n/2}, \cdots, 1c_2, 1c_1$ is the order for the n-processor cube. The inductive definition begins with the coordinate ordering $\{0, 1\}$ for $\{PE_1, PE_2\}$ of the two-processor (one-dimensional) cube. The ordering for the three-dimensional cube is shown in Fig. 3. One consequence of this ordering is that the highest-numbered processor of one subcube is connected to the lowest numbered of the other.

The analysis of communication on the hypercube depends on the following results.

LEMMA 2. *Given a $(\log_2 n)$-dimensional hypercube H with vertices $\{1, \cdots, n\}$ ordered by binary reflected Gray code, let H_j denote the set of vertices in H whose indices are a multiple of 2^j. Then for $1 \leqq j \leqq \log_2 n - 1$, H_j is a $(\log_2 n - j)$-dimensional hypercube in which neighbors are connected by two disjoint paths of length 2 in H.*

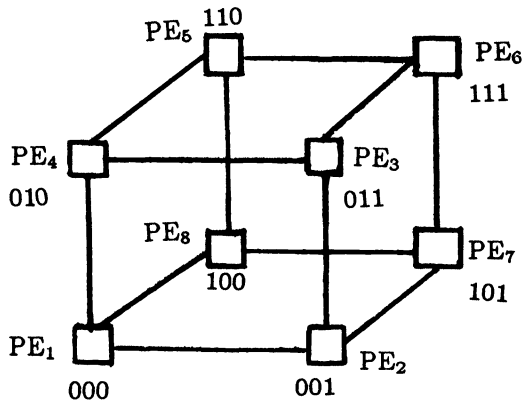


FIG. 3. *A three-dimensional hypercube ordered with binary reflected Gray code.*

Proof. Let $d = \log_2 n$. The set of pairs $\{1, 2\}, \{3, 4\}, \cdots, \{n-1, n\}$ can be thought of as a $(d-1)$-dimensional hypercube, $\hat{H}_1$. Geometrically, this corresponds to "flattening" H along one of its dimensions to form $\hat{H}_1$. Similarly (by repeating this process with $\hat{H}_1$), the set of 4-tuples $\{1, 2, 3, 4\}, \{5, 6, 7, 8\}, \cdots$, is a $(d-2)$-dimensional cube, and in general, the set of consecutively ordered 2^j-tuples is a $(d-j)$-dimensional cube, $\hat{H}_j$. The largest value of each 2^j-tuple is a multiple of 2^j, so that H_j can be identified with $\hat{H}_j$. Moreover, each 2^j-tuple is itself a j-dimensional hypercube.

It remains to determine the paths in H between neighbors of H_j. Given two such neighbors, let T_1 and T_2 denote the 2^j-tuples/j-cubes to which they correspond in $\hat{H}_j$. By construction, every member of T_1 is connected in H to a member of T_2. In particular, the Gray coding dictates that the highest value of T_1 is connected to the lowest value of T_2. Similarly, lowest and highest values within T_2 are neighbors in H. This determines one path of distance two. The alternative is to move first within T_1 (high to low) and then to T_2 (low to high). □

For example, processors 4 and 8 form a one-dimensional subcode of the three-dimensional cube. The connecting paths are 4——5——8 and 4——1——8. Essentially the same argument shows that for each j, the set of nodes whose indices agree mod 2^j forms a hypercube. See, e.g., [23], [40] for other results on paths and distances in hypercubes.

LEMMA 3. *Let a k-processor hypercube contain sk items of data in each processor. The data can be redistributed such that each processor sends s items to every other processor at approximate cost*

$$\begin{cases} sk/2 & \text{if each processor can communicate with all its neighbors simultaneously} \\ sk/2 \log_2 k & \text{if each processor can communicate with just one neighbor at a time.} \end{cases}$$

In both cases, $\log_2 k$ *startups are needed.*

A proof of this result is given in [25]; we omit the proof. When $s = 1$, the action is equivalent to redistributing a matrix from row-oriented to column-oriented storage.

The storage schemes and data movements used on the hypercube are dictated by Lemmas 2 and 3. For the one-way scheme, the processors used for all data associated with the $(j+1)$st grid are those numbered by a multiple of 2^j, with the ith line in processor $i2^j$. As a result, no communication is needed for the permutations. By Lemma 2, neighboring lines in every subgrid are stored in processors of distance two from one another in the hypercube. Hence the data movement between neighboring lines on any subgrid is a shift of distance two, costing $2n$ with two startups. (See [6], [40] for mappings of higher-dimensional grids onto hypercubes.)

Communication for the alternating scheme is more complicated because the directionality of the grid lines changes between every odd and even stage. Figure 4

		PE_2		PE_4		PE_6		PE_8		
		$v_2^{(r1)}$	$v_2^{(b1)}$	$v_2^{(r2)}$	$v_2^{(b2)}$	$v_2^{(r3)}$	$v_2^{(b3)}$	$v_2^{(r4)}$	$v_2^{(b4)}$	Grid 2
	PE_8 : $v_1^{(b4)}$	2	3	2	3	2	3	2	3	
	PE_7 : $v_1^{(r4)}$	1	1	1	1	1	1	1	1	
	PE_6 : $v_1^{(b3)}$	2	3	2	3	2	3	2	3	
Grid 1:	PE_5 : $v_1^{(r3)}$	1	1	1	1	1	1	1	1	
	PE_4 : $v_1^{(b2)}$	2	3	2	3	2	3	2	3	
	PE_3 : $v_1^{(r2)}$	1	1	1	1	1	1	1	1	
	PE_2 : $v_1^{(b1)}$	2	3	2	3	2	3	2	3	
	PE_1 : $v_1^{(r1)}$	1	1	1	1	1	1	1	1	
		$v_3^{(r1)}$		$v_3^{(b1)}$		$v_3^{(r2)}$		$v_3^{(b2)}$		Grid 3
		PE_2		PE_4		PE_6		PE_8		

FIG. 4. *Distribution of the first three grid vectors for the alternating scheme,* $n = 8$. *Grid* 1 = {*Values* 1, 2, 3}, *Grid* 2 = {*Values* 2, 3}, *Grid* 3 = {*Value* 3}.

illustrates the data distribution for $n = 8$. As usual, the points of the initial grid are distributed by horizontal line, with the black lines $v_1^{(b)}$ in the even-numbered processors. For the second grid, the points of $v_1^{(b)}$ are reordered by vertical line and distributed among the $n/2$ even-numbered processors with *two* neighboring lines, each of size $n/2$, in each processor. For example, the horizontal line $v_1^{(b4)}$ originally stored in PE_8 is redistributed among even processors 2 through 8. For the third grid, the black lines $v_2^{(b)}$ of the second grid are simply relabeled as the alternating red and black lines of v_3. The third grid contains $n/2$ points in each direction, and it is distributed by (vertical) line among $n/2$ processors. Distribution of the next two grids on these processors is based on the strategy just described, with the roles of "horizontal" and "vertical" interchanged.

Moving (or permuting) between grids consists of moving the points comprising a vector of black lines on the $(j-1)$st grid to the locations where they coincide with the jth grid (on which they have both colors), and the inverse of this operation. In Fig. 4, $j = 2$ captures the flavor for even j and $j = 3$ handles odd j. For even j, it is convenient to view $v_{j-1}^{(b)}$ as a square *matrix* of order $n/2^{j/2}$ each of whose entries is a pair of neighboring red and black points. For $j = 2$ in the example, $v_{j-1}^{(b)} = v_1^{(b)}$ is a matrix of order four whose entries are the contiguous pairs of points numbered 2 and 3, and it is initially stored by row in the $n/2$ processors 2, 4, 6, and 8. Relocating $v_1^{(b)}$ to use it as v_2 consists of transposing the matrix within these processors. In general for even j, $v_{j-1}^{(b)}$ is stored in the $k = n/2^{j/2}$ processors indexed by a multiple of $2^{j/2}$, and the permutation redistributes the pairs of neighboring red and black points among these processors. For odd j, *no* data movement is required to position v_j. In the example, $v_2^{(b)}$ is already situated in the processors where it is used as v_3.

The costs of data movement for the jth step of the left reduction are as follows. For odd j the permutation is free, and for even j its cost is determined by Lemma 3 with $k = n/2^{j/2}$ and $s = 2$. In addition, the results of the lemma (including number of startups) must be multiplied by two to account for the distance of two between grid neighbors. Consequently, the cost is $2n/2^{j/2}$ with concurrent communication and $2n/2^{j/2} \log_2 (n/2^{j/2})$ without concurrent communication, with $2 \log_2 (n/2^{j/2})$ startups in both cases. Within the jth grid, (16) requires a unidirectional shift of $H_j^{-1} v_j^{(r)}$ for even j and a bidirectional shift for odd j. (See Fig. 4.) The cost of these operations is $2n/27^{\lfloor j/2 \rfloor}$ with two startups.

From this discussion, we have the following result.

THEOREM 4. *On an n-processor hypercube, the communication costs for the m-stage approximate Schur complement preconditioners using* ILU *or* MILU *for the final factorization are bounded by*

$$\begin{cases} 4nm + O(n/2^{m/2}) & \textit{for the one-way scheme,} \\ 12n + O(n/2^{m/2}) & \textit{for the alternating scheme with concurrent communication,} \\ 4n \log_2 n - O(n \log_2 n/2^{m/2}) + O(mn/2^{m/2}) & \textit{for the alternating scheme with communication on one port at a time.} \end{cases}$$

The number of startups is

$$\begin{cases} 4m + 4 + O(n/2^{m/2}) & \textit{for the one-way scheme,} \\ 2m \log_2 n + O(n/2^{m/2}) & \textit{for the alternating scheme.} \end{cases}$$

Proof. We require the sum of the costs of the left reduction $w \leftarrow \mathcal{V}_m v$, right reduction $w \leftarrow \mathcal{W}_m v$, and sparse solves, where the two reductions have the same costs.

For the one-way scheme, the permutation steps of the reductions are free, and the shifts within grids during the $m-1$ reduction steps each cost $2n$ with two startups. From the results of § 5.1, the sparse solves require $4n+O(n/2^{m/2})$ communication and $4m+O(n/2^{m/2})$ startups (where an extra factor of two comes from the paths of length two in the hypercube). The total cost is $4n(m-1)+4n+O(n/2^{m/2})$ communication and $4m+4+O(n/2^{m/2})$ startups.

For the alternating scheme, the $m/2$ transpose operations required for the permutations during the left reduction cost

$$\sum_{i=1}^{m/2} \frac{2n}{2^i} = 2n(1-1/2^{m/2})$$

with concurrent communication, and

$$\sum_{i=1}^{m/2} \frac{2n}{2^i} \log_2\left(\frac{n}{2^i}\right) = 2n \log_2 n - 4n + \frac{n(4+m-2\log_2 n)}{2^{m/2}} \leqq 2n \log_2 n - 4n(1-1/2^{m/2})$$

without concurrent communication. Both cases require $m \log_2 n - (m^2/2+m)$ startups. The shifts of neighboring lines required during the reduction steps (e.g., for (16)) are bounded in cost by twice the cost for even j,

$$2\sum_{i=1}^{m/2} \frac{2n}{2^i} = 4n(1-1/2^{m/2}),$$

with $2(m-1)$ startups. Thus, the left reduction requires less than $6n(1-(1/2^{m/2}))$ data movement with concurrent communication, and $2n \log_2 n$ otherwise. The number of startups is bounded by $m \log_2 n$. From § 5.1, the sparse solves require $O(n/2^{m/2})$ communication and $O(n/2^{m/2})$ startups. The result follows by taking the sum of twice the cost of the left reduction and the sparse solves. □

To avoid additional special cases, this analysis does not take advantage of the two paths cited in Lemma 2. If at least two ports can be used simultaneously (as when all ports are available), then all communicated data blocks can be divided in half by using the extra path. The high-order terms in Theorem 4 then become $2nm$, $6n$, and $2n \log_2 n$, respectively.

The analysis shows that if concurrent communication is available on the hypercube, then the alternating scheme is less expensive to implement, and the number of reductions does not play a significant role in costs. The one-way scheme will be competitive if a small number m of reductions are used. If concurrent communication is not available, then for large n the extra factor of $\log_2 n$ may be a drawback for the alternating scheme. The $O(n/2^{m/2})$ terms come from the sparse solves, and they would not appear at all for the line Jacobi factorization. For the ILU and MILU factorizations, these terms can be made small by increasing the number of reductions. At most $\log_2 n$ reductions are possible for the one-way scheme, but then the ILU factorization reduces to line Jacobi and the term disappears. The alternating scheme can use $2 \log_2 n$ reductions, resulting in a minimum of $2(\log_2 n)^2$ startups. The constants for the "O" notation depend on the relative sizes of arithmetic, bandwidth and startup costs. From the analysis of § 5.1, the constants will be of order one if these costs are similar in scale.

The analysis of this section depends on the assumption that the block rows of each approximate Schur complement $A_j = F_j - \mathrm{sp}(E_j D_j^{-1} C_j)$ are stored by nearest neighbors in a manner analogous to the storage of the original matrix. This distribution can be made during the preprocessing, with data movements and costs essentially the same as those required by the preconditioning. For both schemes, there is movement to neighboring grid lines between the application of D_j^{-1} and E_j, and (if necessary) a permutation before computing A_j. For the one-way scheme, A_j is stored in processors

numbered by a multiple of 2^j (as in (14) for $A_0 \equiv A$). To compute A_j, the sparse approximations to $D_j^{-1}C_j$ are computed in parallel by processors numbered by an odd multiple of 2^{j-1}; the results are then shifted to processors $\{2^j\}$, where the computation is completed with E_j and F_j. The permutations are free, and the communication cost is $O(mn)$. For the alternating scheme, a permutation must be performed before computing A_j for even j, and the total cost is $O(n)$ with concurrent communication and $O(n \log_2 n)$ with communication on one port.

For the one-way scheme, approximately $9nj$ words of storage are needed to store the parts of the approximate Schur complements situated in processors numbered by a multiple of 2^j, so that a maximum of $9nm \leqq 9n \log_2 n$ locations are needed. Note that data is distributed disproportionately, with processors indexed by higher powers of two containing more data. It is also possible to modify this distribution pattern by occasionally rotating data to neighboring processors, at a slight increase in the communication cost per iteration. For the alternating scheme, the size of the blocks decreases at every other stage as the number of reductions goes up, and the maximum storage needed by any processor is $27n - 27n/2^{m/2}$. In both cases, the sparse factorization can overwrite A_m.

We conclude this section with some observations on architectures. First, the approximate Schur complement techniques can also be implemented efficiently on less exotic architectures than the hypercube; it is only necessary that the communication be efficient for permutations and neighboring lines of subgrids. For example, in [16] it is shown that on an n-processor shuffle-shift machine, i.e., a linear array equipped with a shuffle connection [28], [42], the communication costs are $O(mn)$ for the one-way scheme and $O(n \log_2 n)$ for the alternating scheme.

Second, although we have focused on the case of n processors, with one exception all of the analysis generalizes in a straightforward way to fewer than n processors. In particular, if there are $k \ll n$ processors, then an analysis of arithmetic identical to that of § 5.2 shows that the high order $5n^2$ cost of the reductions is reduced to approximately $5n^2/k$, so that the reductions are nearly fully parallelizable. The communication on k processors for the one-way scheme is identical to that described in § 5.3; at each stage of a reduction data must be moved between processors containing neighboring lines, so the cost is $O(mn)$. Moreover, as long as the grid size $n/2^j$ is larger than the number of processors, a linear array of processors is sufficient and a hypercube is not needed. The analysis of [39] shows that the sparse solves are efficient on a small number of processors. The exceptional case is the communication strategy for the alternating scheme, since the grid transpose will be very expensive when $k < n$. Consider as an example the second reduction operator, derived from the matrix of (6). Performing a grid transpose before applying V_2 entails sending $s = n^2/(2k^2)$ data to every other processor. However, prior to the transpose, each of the $n/2$ blocks (of order $n/2$) comprising G_2, H_2, and E_2 is divided equally among the k processors, and for $k < n$, V_2 can be applied without explicitly transposing the grid data. The bidiagonal solves G_2^{-1} and H_2^{-1} must be pipelined, and the cost is $O(n^2/k)$ arithmetic and $O(n) + O(k)$ startups and communication. (For the jth reduction, n is replaced by $n^{j/2}$ here.) Thus, for $k < n$, the alternating reductions should be done in line during the early stages, after which the strategy based on grid transposes should be used.

Finally, more that n processors could be used if the uniprocessors on which the bidiagonal and tridiagonal operations are performed are themselves replaced by clusters of processors. Bidiagonal and tridiagonal solves can be performed in logarithmic time on many types of architectures [24]. However, because the required overall processor configuration is rather complex, we omit further discussion of such strategies.

6. Observations and conclusions. The Schur complement preconditioners appear to be competitive with the best point-oriented preconditioners for solving the nine-point discrete problems considered here, and they offer great potential as effective parallel preconditioners. In particular, the techniques are superior to the ILU factorization and competitive with the MILU factorization (which is extremely effective on the problems considered) in terms of operation counts. It remains to be seen how the methods compare on more complicated linear systems. The strength of the Schur complement methods for parallel computers is that they work with sets of independent block matrices. Most communication is in terms of large blocks of data, so that the methods should not be too dependent on any overhead associated with message passing. This is in contrast with parallel implementation of point preconditioners (with the natural ordering).

Given the vagaries of communication costs and overhead, it is difficult to specify a "best" choice of Schur complement method. The alternating scheme does seem to have an edge over the one-way scheme. It performed better in most experiments, and it has more parallelism in arithmetic. The alternating scheme has a factor of $\log_2 n$ extra communication cost, although this factor is avoided on hypercubes with powerful communication capacities. On the other hand, the one-way costs increase with the number of reductions. In contrast with the nonsymmetrizable versions of [16], both schemes are fairly insensitive to the number of approximate reductions.

Although the techniques discussed here are highly specific to two-dimensional problems, they could be generalized to three-dimensions with either plane or line red-black orderings [13]. For example, for the seven-point operator on a three-dimensional $n \times n \times n$ grid, the plane-ordering results in a coefficient matrix like that of (3), where D_1 and F_1 are block-diagonal with each block containing five nonzeros per row. Application of D_1^{-1} for the matrix-vector product by the Schur complement then entails the simultaneous solution of n linear systems with coefficient matrix corresponding to the five-point operator. We suspect that a more promising approach is to use a line red-black ordering, where the permuted matrix has the form (3) in which D_1 and F_1 are each block-diagonal matrices consisting of approximately $n^2/2$ tridiagonal blocks of order n. C_1 and E_1 each contain four nonzero bands. (This matrix reduces to the two-dimensional point red-black matrix if the number of mesh points in the direction of the lines is one.) Consequently, application of D_1^{-1} consists of n^2 independent tridiagonal solves. The key to the success of such a method will be to compute a good sparse approximation A_1 to the Schur complement S_1, which in this case is a block nine-diagonal matrix of block order $\lfloor n^2/2 \rfloor$ whose block bands consist of dense $n \times n$ matrices.

Last, we note that there are several alternatives to the methods presented here, including polynomial preconditioners [22], vector/parallel variants of point methods [44], multicolor methods [1], [32], [33], [41], and parallel variants [2], [3], [31] of the block factorizations of [4], [7]. We briefly comment on the latter two methods. Two advantages of multicolor methods are that they can easily make use of $O(n^2)$ processors, and they do not require sophisticated processor interconnection schemes. Their numerical performance does not appear to be as good, however. Experiments in [33], [41] show that iteration counts for incomplete factorization preconditioners based on multicolor orderings are at best competitive with *unmodified* (ILU) factorizations based on the natural ordering, and sometimes they are much higher; this suggests that they will be slower than the Schur complement methods on an n-processor hypercube. The block methods require the same type of structure as the Schur complement methods, and (like the Schur methods) their natural parallelism is limited to $O(n)$. Numerical

experiments in [31] show parallel variants of INV (which has no row-sum criterion) to perform efficiently on a four-processor CRAY X-MP, with iteration counts comparable to those of the serial version of INV. We do not know if the modified variants presented in [2], [3] have better performance.

Acknowledgment. I thank Clyde Kruskal for many helpful discussions and Stan Eisenstat for some useful suggestions. In addition, the referees made many useful comments on an earlier version of this paper. One referee in particular is responsible for correcting some errors in the complexity analysis, and for leading me to consider the symmetrizable versions of the preconditioners.

REFERENCES

[1] L. M. ADAMS, *Iterative algorithms for large sparse linear systems on parallel computers*, Ph.D. thesis, Univ. of Virginia, 1982. Available as NASA Contractor Report 166027.

[2] O. AXELSSON, *A survey of vectorizable preconditioning methods for large scale finite element matrix problems*, Report CNA-190, Center for Numerical Analysis, Univ. of Texas at Austin, 1984.

[3] ———, *Incomplete block matrix factorization preconditioning methods. The ultimate answer*? J. Comput. Appl. Math., 12 & 13 (1985), pp. 3-18.

[4] O. AXELSSON, S. BRINKKEMPER, AND V. P. IL'IN, *On some versions of incomplete block-matrix factorization interative methods*, Linear Algebra Appl., 58 (1984), pp. 3-15.

[5] 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.

[6] T. F. CHAN AND Y. SAAD, *Multigrid algorithms on the hypercube multiprocessor*, IEEE Trans. Comput., 35 (1986), pp. 969-977.

[7] P. CONCUS, G. H. GOLUB, AND G. MEURANT, *Block preconditioning for the conjugate gradient method*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 220-252.

[8] 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 *Proc. Symposium on Sparse Matrix Computations*, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1975, pp. 309-332.

[9] S. DEMKO, W. F. MOSS, AND P. W. SMITH, *Decay rates for inverses of band matrices*, Math. Comp., 43 (1984), pp. 491-499.

[10] 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.

[11] M. DRYJA, *A capacitance matrix method for Dirichlet problem on polygon region*, Numer. Math., 44 (1982), pp 51-64.

[12] S. C. EISENSTAT, *Some thoughts on preconditionings for conjugate-gradient-like methods*, Talk presented at Gatlinburg IX, Waterloo, Ontario, 1984.

[13] S. C. EISENSTAT, H. C. ELMAN, AND M. H. SCHULTZ, *Block-preconditioned conjugate-gradient-like methods for numerical reservoir simulation*, Proc. of the Eighth Symposium on Reservoir Simulation, Society of Petroleum Engineers of AIME, 1985, pp. 397-405.

[14] ———, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal. 20 (1983), pp. 345-357.

[15] H. C. ELMAN, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, Ph.D. thesis, Yale Univ. Dept. of Computer Science, 1982.

[16] ———, *Approximate Schur complement preconditioners for serial and parallel computers*, Computer Science Tech. Report 1704, Univ. of Maryland, 1986.

[17] K. FAN, *Note on M-matrices*, Quart. J. Math. Oxford Series (2), 11 (1960), pp. 43-49.

[18] G. E. FORSYTHE AND W. R. WASOW, *Finite Difference Methods for Partial Differential Equations*, John Wiley, New York, 1960.

[19] I. GUSTAFSSON, *A class of first order factorizations*, BIT 18 (1978), pp. 142-156.

[20] L. A. HAGEMAN AND D. M. YOUNG, *Applied Iterative Methods*, Academic Press, New York, 1981.

[21] L. J. HAYES, *Comparative analysis of iterative techniques for solving Laplace's equation on the unit square on a parallel processor*, Master's thesis, Univ. of Texas, Austin, TX, 1974.

[22] O. G. JOHNSON, C. A. MICHELLI, AND G. PAUL, *Polynomial preconditioners for conjugate gradient calculations*, SIAM J. Numer. Anal., 20 (1983), pp. 362-376.

[23] S. L. JOHNSSON, *Communication efficient basic linear algebra computations on hypercube architectures*, J. Parallel Distributed Comput., 4 (1987), pp. 133-172.

[24] ———, *Solving tridiagonal systems on ensemble architectures*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 354–392.

[25] S. L. JOHNSSON AND C.-T. HO, *Spanning graphs for optimum broadcasting and personalized communication in hypercubes*, Report YALEU/DCS/TR-500, Yale Univ. Dept. of Computer Science, 1986.

[26] D. KERSHAW, *Solution of single tridiagonal linear systems and vectorization of the ICCG algorithm on the Cray-1*, in Parallel Computing, G. Rodrigue, ed., Academic Press, New York, 1982, pp. 85–99.

[27] 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., 8 (1987), pp. s166–s202.

[28] D. KLEITMAN, F. T. LEIGHTON, M. LEPLEY, AND G. L. MILLER, *New layouts for the shuffle exchange graph*, Proc. ACM 13th Annual Symposium on Theory of Computing, 1981, pp. 278–292.

[29] 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*, Math. Comp., 31 (1977), pp. 148–162.

[30] J. A. MEYERINK, *Iterative methods for the solution of linear equations based on incomplete block factorization of the matrix*, Proc. Seventh Symposium on Reservoir Simulation, Society of Petroleum Engineers of AIME, 1983, pp. 149–159.

[31] G. MEURANT, *Multitasking the conjugate gradient on the Cray X-MP/48*, Report NA-85-33, Numerical Analysis Project, Stanford Univ., 1985.

[32] D. P. O'LEARY, *Ordering schemes for solving sparse systems of linear equations*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 620–632.

[33] E. L. POOLE AND J. M. ORTEGA, *Multicolor ICCG methods for vector computers*, Report RM-86-06, Dept. of Applied Mathematics, Univ. of Virginia, 1986.

[34] J. K. REID, *The use of conjugate gradients for systems of linear equations possessing "property A,"* SIAM J. Numer. Anal., 9 (1972), pp. 325–332.

[35] E. M. REINGOLD, J. NIEVERGELT, AND N. DEO, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[36] G. RODRIGUE AND D. WOLITZER, *Preconditioning by incomplete block cyclic reduction*, Math. Comp., 42 (1984), pp. 549–565.

[37] Y. SAAD, A. SAMEH, AND P. SAYLOR *Solving elliptic difference equations on a linear array of processors*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 1049–1063.

[38] Y. SAAD AND M. H. SCHULTZ, *Conjugate gradient-like algorithms for solving non-symmetric linear systems*, Math. Comp., 44 (1985), pp. 417–424.

[39] ———, *Parallel implementations of preconditioned conjugate gradient methods*, Report YALEU/DCS/TR-425, Yale Univ. Dept. of Computer Science, 1985.

[40] ———, *Topological properties of hypercubes*, Report YALEU/DCS/TR-389, Yale Univ. Dept. of Computer Science, 1985.

[41] R. SCHREIBER AND W. TANG, *Vectorizing the conjugate gradient methods*, Proc. Symp. CYBER 205 Applications, Fort Collins, CO, 1982

[42] J. T. SCHWARTZ, *Ultracomputers*, ACM TOPLAS, 2 (1980), pp. 484–521.

[43] J. VAN ROSENDALE, *Minimizing inner product data dependencies in conjugate gradient iteration*, Proc. 1983 International Conference Parallel Processing, pp. 44–46.

[44] H. A. VAN DER VORST, *A vectorizable variant of some* ICCG *methods*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 350–356.

[45] R. S. VARGA, *Matrix Interative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 607-630, July 1989
© 1989 Society for Industrial and Applied Mathematics

SPANNING BALANCED TREES IN BOOLEAN CUBES*

CHING-TIEN HO† AND S. LENNART JOHNSSON†‡

Abstract. A *Spanning Balanced n-tree* (SBnT) in a Boolean n-cube is a spanning tree in which the root has fanout n, and all the subtrees of the root have $O(2^n/n)$ nodes. The number of tree edges in each dimension of the n-cube is of order $O(2^n/n)$. The spanning balanced n-tree allows for scheduling disciplines that realize lower bound (within a factor of two) one-to-all personalized communication, all-to-all broadcasting, and all-to-all personalized communication on a Boolean n-cube [C.-T. Ho and S. L. Johnsson, Proc. 1986 International Conference on Parallel Processing, pp. 640-648, IEEE Computer Society, 1986; Tech. Report YALEU/DCS/RR-483, May 1986], [S. L. Johnsson and C.-T. Ho, Tech. Report YALEU/DCS/RR-610, Dept. of Computer Science, Yale Univ., New Haven, CT, November 1987]. The improvement in data transfer time over the familiar binomial tree routing is a factor of $n/2$ for concurrent communication on all ports and one-to-all personalized communication and all-to-all broadcasting. For all-to-all personalized communication on all ports concurrently, the improvement is of order $O(\sqrt{n})$. Distributed routing algorithms defining the spanning balanced n-tree are given. The balanced n-tree is not unique, and a few definitions of n-trees that are effectively edge-disjoint are provided. Some implementation issues are also discussed.

Binary numbers obtained from each other through rotation form necklaces that are full if the period is equal to the length of the number; otherwise, they are degenerate. As an intermediary result, it is shown that the ratio between the number of degenerate necklaces and the total number of necklaces with l bits equal to one is at most $4/(4+n)$ for $1 \leqq l < n$.

Key words. Boolean cubes, balanced trees, spanning trees, personalized communication, routing, shuffles, necklaces, periodicity

AMS(MOS) subject classifications. 68P05, 68Q25

1. Introduction.

High performance computer architectures require a high processing and communication bandwidth, which can be achieved in standard technologies by using a large number of processing elements, and interconnection networks that allow many independent communications concurrently. One such interconnection network is the Boolean cube. The number of dimensions for currently available systems range from 5 to 12. For highly concurrent computations, also known as data parallel computations because the degree of parallelism is determined by the size of the data set rather than control constructs, the number of arithmetic/logic operations per communication is often low, and the efficient use of the communication facilities is important for total performance. Some frequently occurring operations are broadcasting of data from one node to a subset of other nodes, or all other nodes, or the reverse operation: reduction. Common reduction operators are sum, min, max, and logical operators. Other global communications are the sending of a unique piece of data from one node to all other nodes, or the reverse operation. We refer to this type of communication as *personalized communication* [1], [5]. Many linear algebra algorithms require broadcasting [3]. Certain matrix transpose algorithms and other permutations [4] effectively use *one-to-all* or *all-to-all personalized communication* [1].

For global communication a spanning tree, or a composition of many spanning trees, is needed. In this paper we describe and analyze a particular spanning tree, a

* Received by the editors January 20, 1987; accepted for publication (in revised form) December 21, 1987. This work was supported in part by the Office of Naval Research under contracts N00014-84-0043 and N00014-86-K-0564.

† Department of Computer Science, Yale University, New Haven, Connecticut 06520.

‡ This author also has affiliations at the Department of Electrical Engineering, Yale University, New Haven, Connecticut 06520, and Thinking Machine Corporation, 245 First Street, Cambridge, Massachusetts 02142.

Spanning Balanced n-tree (SBnT). It makes possible scheduling disciplines that realize lower bound communication within a factor of two [1]. A balanced n-tree is a tree with fanout n at the root, and approximately the same number of nodes in each subtree of the root. A commonly used spanning tree for global communication in Boolean cubes is a binomial tree. Such a tree has half of the number of nodes in one subtree, a quarter in another, etc. Another important property that makes lower bound algorithms for all-to-all broadcasting and all-to-all personalized communication possible is that for each level of the spanning balanced n-tree, the edges are evenly distributed among the n dimensions of an n-cube, while for the binomial tree, half of the edges are in dimension $n-1$, a quarter in dimension $n-2$, etc. A binomial tree is easily constructed by complementing leading or trailing zeros. The construction algorithm is a distributed algorithm. Each node only needs to know the address of the source node and its own address (and perform an exclusive-or operation on the two before computing the address of children nodes through bit-complementation).

The construction of a balanced n-tree is somewhat more complex but has a resemblance with the construction of the binomial tree. One key observation is that if one subtree of the root is known, then if all the addresses of the nodes in that subtree are rotated by the same amount, another subtree is obtained. That subtree has the same topology as the original subtree. Since n distinct rotations are possible on the node addresses in a Boolean n-cube, the balanced tree can be generated by this strategy, provided that all addresses have a period of n. Nodes with a period less than n will appear in several subtrees of the root. For the outlined construction to work, approximately $2^n/n$ node addresses must have a period of n. Address rotation defines a *shuffle* operation, and the addresses so obtained form a *necklace*. A necklace with n nodes is *full*. Other necklaces are *degenerate*. It follows from a result by Leighton [6] that there exist $O(2^n/n)$ node addresses that cannot be obtained from each other through rotation. We prove a stronger result and show that the ratio between the number of degenerate necklaces and the total number of necklaces for l bits equal to one is at most $4/(4+n)$ for $1 \leqq l < n$. For the construction of the balanced tree, at most one node from every necklace is selected for the generating subtree, and an interconnection scheme needs to be found that makes use of the Boolean cube topology. We describe one such construction that, viewed in the appropriate way, implies complementing some of the leading zeros (but not all) of the node address in order to obtain the address(es) of the children node(s). The definition can be used as a distributed routing algorithm.

In this paper we define the balanced n-tree more precisely, analyze some of its topological properties, and give distributed algorithms for constructing balanced n-trees. We also present a *spanning balanced graph*, SBG, that minimizes the maximum edge load for *one-to-all personalized communication*, *all-to-all broadcasting*, and *all-to-all personalized communication*. The SBG is derived from the balanced n-tree. In personalized communication the root sends a unique piece of information to every node. We also present a few alternative definitions of the SBnT to demonstrate that the SBnT is not unique, and that the technique used to define it can easily be modified to generate other spanning balanced n-trees. These trees may have the same distribution of nodes among subtrees, but the subtrees have different topologies and use different sets of edges in the cube. Fault tolerance for SBnT communication is equivalent to finding an SBnT that does not include the faulty edges. There exist different SBnTs that share only the edges emanating from the root, i.e., that are edge-disjoint below level 1 with the root at level 0. Similarly, the different definitions of the SBnT have different sets of leaf nodes, giving some flexibility for reducing the consequences of node failures.

Note that all the spanning balanced n-trees defined here use all edges from the root. For personalized communication the communication complexity cannot be reduced by using several SBnTs concurrently. The root is the bottleneck, as shown later.

In § 2 the notations and definitions used throughout this paper are introduced. Section 3 contains a definition of the SBnT and an analysis of its properties. Section 4 gives some alternative definitions of the SBnT and compares the characteristics of these alternative SBnTs with those of the SBnT in § 3. In § 5 we prove and give some complexity estimates for personalized communication based on the SBnT. Implementation issues are discussed in § 6 and followed by a summary in § 7.

2. Preliminaries. A Boolean n-cube is an n-dimensional cube with two nodes in each dimension. The total number of nodes is $N = 2^n$, and the total number of bidirectional links is $n(N/2)$. Nodes in the Boolean cube can be given n bit addresses such that adjacent nodes differ in precisely one bit. The distance between nodes $i = (i_{n-1} i_{n-2} \cdots i_0)$, and $i' = (i'_{n-1} i'_{n-2} \cdots i'_0)$ is defined as the *Hamming* distance between the nodes, i.e., *distance* $(i, i') = Hamming(i, i') = \sum_{m=0}^{n-1} (i_m \oplus i'_m)$. The number of 1-bits in the binary representation of i is denoted $\|i\| = \sum_{m=0}^{n-1} i_m = Hamming(i, 0)$. The number of nodes at distance l from a node is $\binom{n}{l}$.

For the definition of the spanning balanced n-tree we make use of *rotations* and *translations*. The *right-rotation of a node* i is $R(i) = (i_0 i_{n-1} i_{n-2} \cdots i_1)$, and $R^m = R^{m-1} \circ R$ is a right rotation of m steps. The inverse operation $R^{-1} = L$ is a left rotation, i.e., $R^{-1}(i) = L(i) = (i_{n-2} i_{n-3} \cdots i_1 i_0 i_{n-1})$. The *right-rotation of a graph* $G = (\mathcal{V}, \mathcal{E})$ is $R(G) = (R(\mathcal{V}), R(\mathcal{E}))$, where $R(\mathcal{V}) = \{R(i) | \forall i \in \mathcal{V}\}$ and $R(\mathcal{E}) = \{(R(i), R(j)) | \forall (i, j) \in \mathcal{E}\}$. The *translation of a node* i by s is $T(i, s) = i \oplus s = c$, where c is the *relative address* of i with respect to s. The *translation of a graph* G by s is $T(G, s) = (T(\mathcal{V}, s), T(\mathcal{E}, s))$ with $T(\mathcal{V}, s) = \{T(i, s) | \forall i \in \mathcal{V}\}$ and $T(\mathcal{E}, s) = \{(T(i, s), T(j, s)) | \forall (i, j) \in \mathcal{E}\}$. The *bit-reversal* of a node i is $B(i) = (i_0 i_1 \cdots i_{n-1})$. Adjacency is preserved under rotation, translation, and bit-reversal [5]. Translation preserves the order of dimensions, and rotation preserves the relative order of dimensions, cyclically. The bit-reversal operation is its own inverse, and the following relationship between bit-reversal and rotation holds: $RBR = B$, $LBL = B$, $R = BLB$, and $L = BRB$.

A *necklace* [6] is a set of nodes with addresses that can be obtained from each other through rotation (shuffle operation). For example, (001001), (010010), and (100100) are in the same necklace. The numbers (110000), (011000), (001100), (000110), (000011), and (100001) are also in the same necklace (but not the same as the preceding ones). The *period* of a binary number i, P_i is the least $u > 0$ such that $i = R^u(i)$. For example, the period of (011011) is 3. A binary number is *cyclic* if its period is less than its length, and it is *noncyclic* otherwise. Note that complementation of a binary number preserves the period. A *cyclic node* i is a node with *cyclic relative address* $c = i \oplus s$ with respect to a given node s. We define cyclic nodes with respect to a given node, since we are only interested in this property for trees with a defined root, and not for the node address itself. We also refer to the root node as the *source* node. Throughout the paper, we use c to represent the *relative* address of node i. Noncyclic nodes belong to *full necklaces* and cyclic nodes to *degenerate necklaces* [6]. The number of nodes in the necklace to which c belongs is P_c. If there were no cyclic nodes, then all necklaces are full, and a balanced n-tree can be generated through rotation of a *generating subtree* of the root. We label the subtrees of the root from 0 to $n-1$, with the generating subtree being subtree 0. The generating subtree is assembled with one node from every necklace. These nodes are called *distinguished nodes*. The spanning

balanced n-trees we define differ in the selection of distinguished nodes. The first SBnT we define uses the node with minimum value of the nodes in a necklace as the distinguished node. This definition of distinguished node is similar, but not identical, to the one used by Leighton [6]. He defines the distinguished node as the node having the longest block of leading zeros, which is also minimum, if there is a unique longest block of zeros. If n is a prime number, then there are only two degenerate necklaces, namely those formed by the nodes with relative addresses $(00\cdots0)$ and $(11\cdots1)$. The construction and analysis of spanning balanced trees for arbitrary n are complicated by the treatment of cyclic nodes.

The addresses of cyclic nodes are made up of repeating blocks. We use the notation $a\,|\,b$ to denote that a divides b. For cyclic nodes there exists at least one rotation of fewer than n steps of the address that yields the original value. Let J_c be the set of distinct rotations that minimize the value of the rotated address, i.e., $J_c = \{j_1, j_2, \cdots, j_m\}$, where $0 \leqq j_1 < j_2 < \cdots < j_m < n$, $R^u(c) = R^v(c)$, $u, v \in J_c$, and $R^u(c) < R^w(c)$, $u \in J_c$, $w \in \{0, 1, \cdots, n-1\} - J_c$. Clearly $|J_c| = n/P_c$. Graphs constructed through rotation of a generating subtree SBG are balanced, but are not trees, since there exist $|J_c|$ paths from the source node s to node i $(c = i \oplus s)$. For a balanced n-tree we select a particular path of the J_c paths from the source node to each cyclic node. For the graph with J_c paths from source to destination, the load on the edges from the source can be made even by dividing the data set for a node into $|J_c|$ pieces, one for each path to node c. The *index* of a node c defines the subtree(s) of the root to which it belongs; $index(c) = j_1$ for a single path to every node, and $index_m(c) = J_c$ for multiple paths to cyclic nodes. The distinguished nodes are defined by the set $\{c \,|\, index(c) = 0, \forall i \in \mathcal{V} - \{s\}\}$.

In a binomial tree defined by complementing leading zeros, all nodes in a given subtree of the root have the same number of trailing zeros. The lowest-order bit that is one can be viewed as the index of the subtree. For the first definition of a generating subtree we also consider the block of leading zeros for defining connections to *children* nodes. However, complementing bits in this block may yield a node i' with relative address c' such that $index(c') \neq index(c)$ for the SBnT, or $index(c') \notin index_m(c)$ for the SBG graph. For instance, $index(00\cdots01) = 0$, but $index(10\cdots01) = n-1$. We define the SBnT and SBG graphs directly from the node addresses without explicit rotation of the generating subtree. It is convenient to introduce the following definitions. Let k be defined by $c_k = 1$, $c_j = 0$, $j \in \{(k+1) \bmod n, (k+2) \bmod n, \cdots, (index(c) - 1) \bmod n\} = \mathcal{M}(c)$, and $k = -1$ if $c = 0$. The kth bit is the last 1-bit to the left of bit $index(c)$, cyclically. The set $\mathcal{M}(c)$ is the maximum set of consecutive indices of 0-bit positions immediately to the right of bit $index(c)$, cyclically. For nonroot nodes, bit $index(c)$ is always a 1-bit. $|\mathcal{M}(c)| = \alpha_c$ is the number of leading zeros of $R^{index(c)}(c)$. For example, for $c = (010110)$, $index(c) = 1$, $k = 4$, $\mathcal{M}(c) = \{5, 0\}$, and $\alpha_c = 2$.

3. A spanning balanced n-tree. Our definition of the balanced n-tree can serve as a distributed routing algorithm. For the complexity estimates of various communication operations, it is of interest to characterize the distribution of nodes among the subtrees of the root as well as the fanout of the nodes. We give a lower bound $N/(n+2)$ on the number of nodes in a subtree of the root with the fewest number of nodes, which is low by at most a term of approximately $2N/n^2$. We also derive an upper bound of $2N/(n+2)$ on the number of nodes in a subtree. We present a table for the actual number of nodes in the maximum and minimum subtrees generated by the SBnT algorithm for up to 20-dimensional cubes, and show that the relative difference approaches 0 as the number of nodes grow, Table 1.

TABLE 1
A comparison of subtree sizes of spanning binomial trees and spanning balanced n-trees.

n	A	B	SBT(max)	SBnT(max)	SBnT(min)	$(N-1)/n$	factor
2	2	2	2	2	1	1.50	1.33
3	2	2	4	3	2	2.33	1.29
4	4	3	8	5	3	3.75	1.33
5	2	2	16	7	6	6.20	1.13
6	10	5	32	13	9	10.50	1.24
7	2	2	64	19	18	18.14	1.05
8	16	6	128	35	30	31.88	1.10
9	8	4	256	59	56	56.78	1.04
10	34	9	512	107	99	102.30	1.05
11	2	2	1,024	187	186	186.09	1.00
12	76	17	2,048	351	335	341.25	1.03
13	2	2	4,096	631	630	630.08	1.00
14	130	21	8,192	1,181	1,161	1,170.21	1.01
15	38	10	16,384	2,191	2,182	2,184.47	1.00
16	256	36	32,768	4,115	4,080	4,095.94	1.00
17	2	2	65,536	7,711	7,710	7,710.06	1.00
18	568	70	131,072	14,601	14,532	14,563.50	1.00
19	2	2	262,144	27,595	27,594	27,594.05	1.00
20	1,036	111	524,288	52,487	52,377	52,428.75	1.00

We define the balanced n-tree by two alternative functions: the $children_{\mathrm{SBnT}}(i, s)$ and $parent_{\mathrm{SBnT}}(i, s)$ functions.

$$children_{\mathrm{SBnT}}(i, s) = \begin{cases} \{(i_{n-1}i_{n-2}\cdots\bar{i}_m\cdots i_0)\}, & \forall m \in \mathcal{M}(c), \quad \text{if } c = 0; \\ \{q_m = (i_{n-1}i_{n-2}\cdots\bar{i}_m\cdots i_0)\}, & \\ \quad \forall m \in \mathcal{M}(c) \text{ and } index(q_m \oplus s) = index(c), & \text{if } c \neq 0. \end{cases}$$

$$parent_{\mathrm{SBnT}}(i, s) = \begin{cases} \phi, & \text{if } c = 0; \\ (i_{n-1}i_{n-2}\cdots\bar{i}_k\cdots i_0), & \text{otherwise.} \end{cases}$$

The $index(parent_{\mathrm{SBnT}}(i, s))$ is the same as $index(c)$, since for any node with relative address c, k is the highest-order bit of $R^{index(c)}(c)$ that is 1. Complementing this bit cannot change the index. It is also readily seen that the $parent_{\mathrm{SBnT}}$ and $children_{\mathrm{SBnT}}$ functions are consistent, i.e., for any node i it is a parent of a node j if and only if j is a child of i.

LEMMA 1. *The $parent_{\mathrm{SBnT}}$ ($children_{\mathrm{SBnT}}$) function defines a spanning tree rooted at node s.*

Proof. The parent node of a node at distance d from node s is at distance $d-1$ from node s, and each node only has one parent node. Traversing the edges defined by successive applications of the $parent_{\mathrm{SBnT}}$ function of any node generates a path to node s for any node. Hence, the graph is a spanning tree rooted at node s. □

Figures 1, 2, and 3 show spanning trees generated by the algorithm above for the root located at node 0 in 3-, 4-, and 5-cubes. Figure 4 shows subtree 0 of an SBnT in a 6-cube. Nodes in square boxes are cyclic nodes.

LEMMA 2. *The number of nodes at level l from the source is $\binom{n}{l}$.*

Proof. From the $parent_{\mathrm{SBnT}}$ function it follows that node i with $\|c\| = l$ is at level l. Furthermore, there exist $\binom{n}{l}$ distinct permutations of l 1-bits out of a string of n bits. □

The SBnT is a *greedy* tree [5] in the sense that the distance for any node in the SBnT to the root is minimal. Personalized communication using a greedy spanning

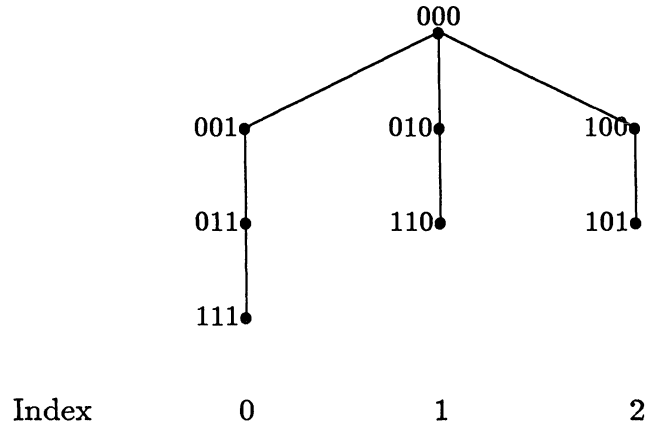


FIG. 1. *A spanning balanced 3-tree in a 3-cube.*

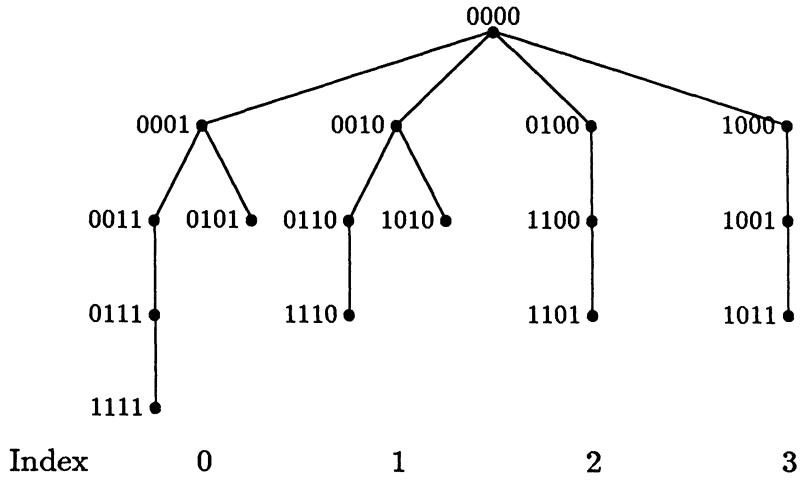


FIG. 2. *A spanning balanced 4-tree in a 4-cube.*

tree guarantees that the minimum communication bandwidth is used. All data is sent via a shortest path.

COROLLARY 1. *The height of one subtree is n, and the height of all other subtrees is* $n-1$.

Proof. There is only one node at distance n from the root, and there are n nodes at distance $n-1$ from the root, each of which has a different index. □

The fanout of nodes is of particular importance for estimating the complexity of scheduling algorithms for personalized communication. If the fanout is nonincreasing as a function of the distance from the root, then the analysis is considerably simplified.

LEMMA 3. *The maximum fanout of a node at level l is* $\lceil (n-l)/2 \rceil$, *for* $1 \leqq l \leqq n$.

Proof. Let the relative address $c = 2^l - 1$. Then $\|c\| = l$ and complementing bits $\{l, l+1, \cdots, l+\lceil (n-l)/2 \rceil - 1\}$ does not change the index, but complementing the higher-order bits does. Hence, the maximum fanout is at least $\lceil (n-l)/2 \rceil$. But, for any other c such that $\|c\| = l$, the maximum size of any block of consecutive zeros cyclically, is also $n-l$; hence $|\mathcal{M}(c)| \leqq n-l$, and the argument can be applied to the leading block of $R^{index(c)}(c)$, and the proof is complete. □

For one-to-all personalized communication in Boolean cubes that allow concurrent communication on all *n-ports* of a node, one effective scheduling discipline is a

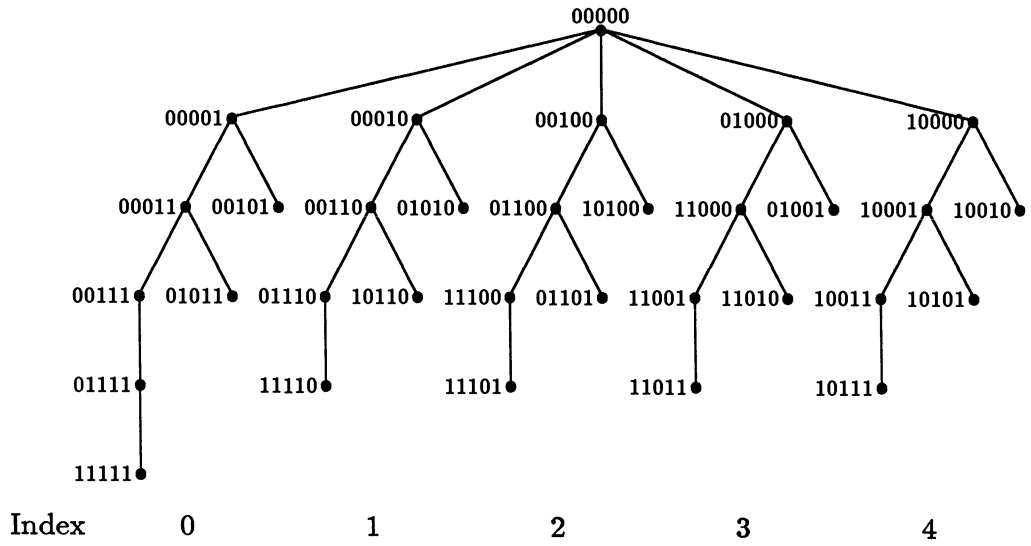


FIG. 3. *A spanning balanced 5-tree in a 5-cube.*

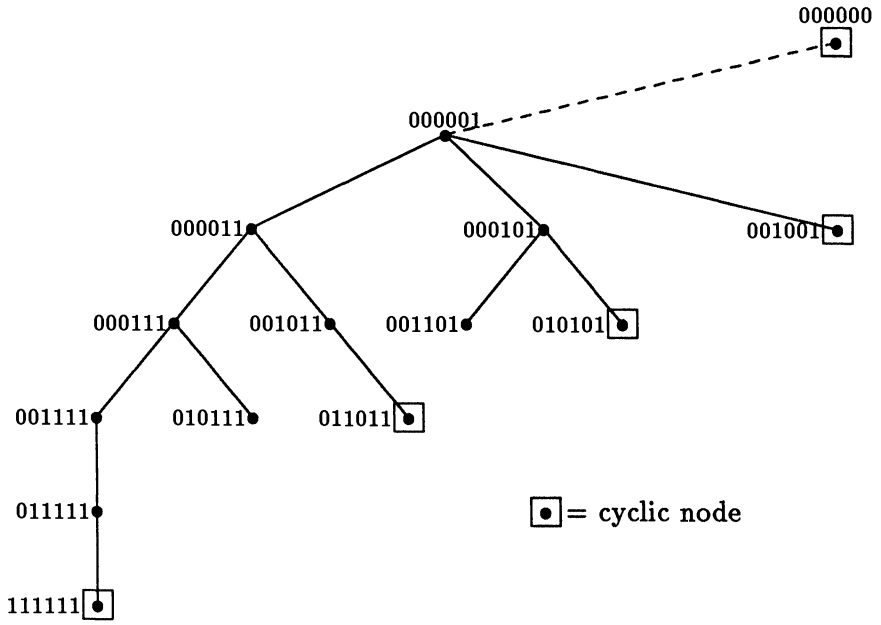


FIG. 4. *Subtree 0 of a spanning balanced 6-tree in a 6-cube.*

reverse-breadth-first order [5], i.e., scheduling data for all nodes at a given distance during the same routing cycle, and in order of decreasing distance from the source. The complexity analysis of this scheduling discipline is simplified, if the communication bandwidth required by nodes forwarding data from the root to the final destination is guaranteed to be at most the same as that required by the root. If the following property holds, then it suffices to consider the root alone.

LEMMA 4. *Let $\phi(c, l)$ be the number of nodes at distance l from node c in the subtree rooted at node c. Then, $\phi(c, l) \geqq \phi(c', l)$, where node c' is a child of node c.*

Proof. Let c' be any nonroot node and node c its parent node. We prove the lemma first for subtree 0 by showing that for any node at distance l below node c' there is a unique corresponding node at distance l below node c. Recall that $\alpha_{c'}$ is the number of leading 0's of $R^{index(c')}(c') = c'$. Clearly, the number of leading 0's of c must be $\alpha_{c'}+\beta+1$, where $\beta \geqq 0$ is the number of consecutive 0's in c' between the two leftmost 1-bits. Any node at distance l below c', say c'_l, has an address that can be derived by complementing l out of the $\alpha_{c'}$ leading 0's of c', and with the index unchanged. There exists a corresponding node c_l at distance l below node c. The address of node c_l can be constructed by leaving $\beta+1$ leading 0's of the address of c unchanged and making the following $\alpha_{c'}$ bits equal to the first $\alpha_{c'}$ bits of node c'_l. This process preserves the index for any given node c'_l. The same argument applies to any other subtree j by considering $R^j(c)$ and $R^j(c')$. □

LEMMA 5. *Excluding node* $c = (11 \cdots 1)$, *all the subtrees of the* SBnT *are isomorphic if* n *is a prime number.*

Proof. For n a prime number there are no cyclic nodes, except the nodes with relative addresses $(00 \cdots 0)$ and $(11 \cdots 1)$. Since different subtrees are obtained through rotations of the addresses, the proof is complete. □

COROLLARY 2. *The subtrees of the root of the* SBnT *are isomorphic, if cyclic nodes are excluded.*

For communication from every node to every other node, *all-to-all communication*, a number of tree edges are mapped to the same cube edge using 2^n distinctly translated balanced n-trees. Since translation preserves the dimension of an edge (but not necessarily the direction), the number of tree edges mapped to any cube edge in a given dimension, say d, is equal to the number of SBnT edges in that dimension. For the complexity analysis of scheduling algorithms, it is also necessary to know at what distance from the source node these edges are [1].

COROLLARY 3. *If* n *is prime, the number of edges in dimension* d *between levels* $l-1$ *and* l *is equal to* $\binom{n}{l}/n$ *for* $l = \{1, 2, \cdots, n-1\}$. *For* $l = n$ *there is only one edge, and it is in dimension* $n-1$. *The total number of edges in dimension* d *is equal to* $(N-2)/n+1$ *for dimension* $n-1$ *and* $(N-2)/n$ *for the other dimensions.*

DEFINITION 1. A *treetop* of a tree T is a tree that is a connected subgraph of T containing the root of T.

LEMMA 6. *Subtree* i *of the root of an* SBnT *is isomorphic to a treetop of subtree* j *of the root of the* SBnT, *if* $i > j$.

Proof. Subtree i is derived from subtree j by pruning away cyclic nodes that have periods p with $j < p \leqq i$ and corresponding incident edges and by performing a left cyclic shift of $i-j$ bits for all nodes in the pruned subtree. □

COROLLARY 4. *Subtree* i *contains all the nodes with periods* $p > i$.

COROLLARY 5. *There are* $n/2$ *subtrees without cyclic nodes if* n *is even, and at least* $2n/3$ *subtrees without cyclic nodes if* n *is odd.*

LEMMA 7. *Subtree* 0 *of an* SBnT *is isomorphic to a treetop of a Spanning Binomial Tree of an* $(n-1)$*-cube. Subtrees* 1 *through* $n-1$ *of an* SBnT *are isomorphic to a treetop of a Spanning Binomial Tree of an* $(n-2)$*-cube.*

Proof. Consider the relative addresses of nodes in subtree 0 of an SBnT of an n-cube. They all have the form $(i_{n-1}i_{n-2} \cdots i_1 1)$, where $i_m = 0$ or 1. From the definition of the children function of an SBnT, if node j is a child of node i, then node j can be derived by complementing one of the leading 0-bits of node i. Recall that the children function of the *spanning binomial tree* (SBT) is defined by complementing *any* one of the leading 0-bits. Hence, subtree 0 of the SBnT is a treetop of an SBT of an $(n-1)$-cube. For the nodes in subtree 1 of the SBnT, they all have a relative address of the form

$(i_{n-1}i_{n-2}\cdots i_2 10)$. Again, the addresses of the children can be derived by complementing one of the leading 0-bits. So, subtree 1 is a treetop of a SBT of an $(n-2)$-cube. By Lemma 6, it follows that subtree 2 to subtree $n-1$ are treetops of a SBT of an $(n-2)$-cube. □

LEMMA 8. *Any cyclic node is a leaf node of the* SBnT.

Proof. From the definition of the *index*(c) and *children*$_{\text{SBnT}}(i, s)$ functions for a node i in a tree rooted at node s, the connections to the children nodes are defined by complementing the subset of the leading zeros of $R^{index(c)}(c)$ that preserves *index*(c). But, if c is periodic, then the index is changed because the leading repetitive pattern of c has a larger value than the following patterns. □

The imbalance between the subtrees of the root is caused by the cyclic nodes. We will now study the distribution of cyclic nodes in some detail. First we give a bound on the total number of cyclic nodes in a subtree.

THEOREM 1. *The number of nodes in a subtree is of order* $O(N/n)$.

Proof. With A cyclic nodes there are at least $(N-A)/n$ nodes in a subtree. Denote the number of degenerate necklaces by B. Since the length of each necklace is at least 2, except for the necklaces formed by nodes $(00\cdots 0)$ and $(11\cdots 1)$, $B \leqq ((A-2)/2)+2$. It follows that the maximum number of nodes in a subtree is $((N-A)/n)+B-1 \leqq (2N+(n-2)A)/2n$. To derive bounds on A, we use the complex-plane diagram used by Hoey and Leiserson [2] in studying the shuffle-exchange network. Leighton [6] shows that $B = O(\sqrt{N})$.

Full necklaces are mapped to circles. Degenerate necklaces are mapped to the origin. In the context of the shuffle-exchange graph, each node that is mapped to the origin of the complex plane is adjacent (via an exchange edge) to a node at position $(1, 0)$ or $(-1, 0)$. Hence, for every full necklace of n nodes there are at most two cyclic nodes. It follows that an upper bound on A is $2N/(n+2)$, and the number of nodes in a subtree is at least $N/(n+2)$ and at most $2N/(n+2)$. The relative difference in the number of nodes in the maximum and minimum subtrees is $B/O(N/n) = O(\sqrt{N})/O(N/n)$, which approaches 0 for $N \to \infty$. □

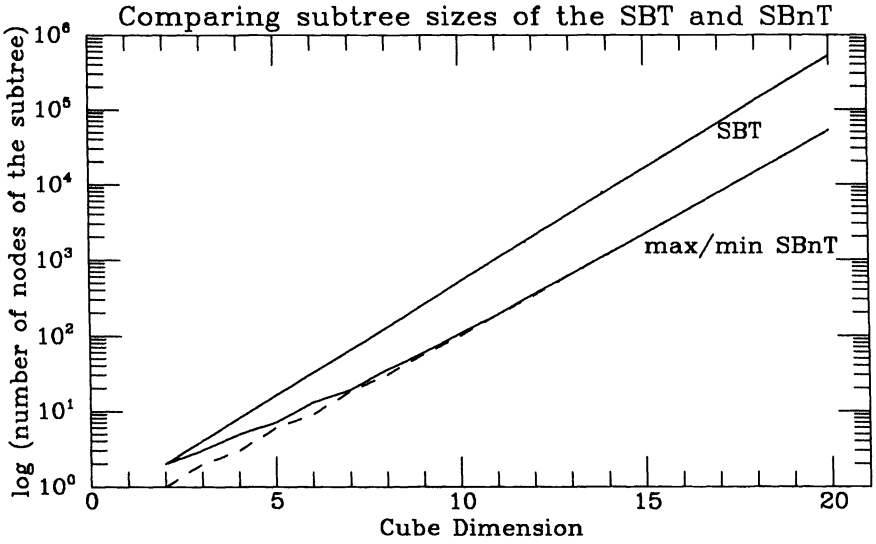
Table 1 gives the sizes of the minimum (SBnT(min)) and maximum (SBnT(max)) subtrees generated according to the definition of the SBnT for up to 20-dimensional cubes. The relative difference approaches 0 rapidly. For comparison we have included the number of nodes in the largest subtree of the corresponding spanning binomial tree. The last column contains the ratio of SBnT(max) to $(N-1)/n$. Figure 5 contains the same information as the table. The curves for the maximum and minimum SBnT become indistinguishable as the cube dimension increases.

In Theorem 1 we showed that the total number of cyclic nodes is at most $2N/(n+2)$. We now show that the ratio of the number of cyclic nodes at level l to the total number of nodes at level l is at most $2/n$. Then we show that for any level of any subtree of the root, the ratio of the number of cyclic nodes to the total number of nodes at the same level of the same subtree is at most $4/(n+4)$, with the exception of the last level of subtree 0. We do that by defining a function described in Lemma 12 and by showing that this function maps each cyclic node to a disjoint set of noncyclic nodes at the same level of the same subtree (except the root node, s, and the node at the last level of subtree 0, $\bar{s}$). Some properties of the period of a cyclic number are needed.

Note that

(1) if $a \mid c$, and $b \mid c$ then lcm $(a, b) \mid c$, where a, b, c are integers,

(2) if c is an *n-bit* cyclic number with period p, then $n/p \mid \|c\|$.



From top to bottom: max subtree size of SBT, max subtree size of SBnT, min subtree size of SBnT.

FIG. 5. *Comparing subtree sizes of spanning binomial trees and* SBnT.

LEMMA 9. *Let c_1 and c_2 be two distinct n-bit cyclic numbers with periods p_1 and p_2, respectively, and $\|c_1\| = \|c_2\|$. Then* $\gcd(p_1, p_2) > 1$.

Proof. Let $\|c_1\| = q$. Assume $\gcd(p_1, p_2) = 1$. Then $p_1 | n$, $p_2 | n$ and $\gcd(p_1, p_2) = 1$ imply $n = \gamma p_1 p_2$ for some positive integer γ. By property 2, we have $(n/p_1) | q$, $(n/p_2) | q$. But $n = \gamma p_1 p_2 \Rightarrow \gamma p_2 | q$ and $\gamma p_1 | q$, which imply $\gamma p_1 p_2 | q$, i.e., $n | q$, by property 1. But $0 < q < n$ since $c_1 \neq c_2$, and we have a contradiction. □

LEMMA 10. *Let c_1 and c_2 be two distinct n-bit cyclic numbers with periods p_1 and p_2, respectively; $p_1 | p_2$ and $\|c_1\| = \|c_2\|$. Then Hamming* $(c_1, c_2) \geqq 2n/p_2$.

Proof. We derive a lower bound for the Hamming distance between c_1 and c_2 by finding the minimum number of bits of c_1 that have to be complemented to yield c_2, for all possible c_1 and c_2. c_2 consists of n/p_2 blocks of length p_2 each. In order to change the period from p_1 to p_2 (or change from c_1 to c_2 if $p_1 = p_2$), at least one bit in each block of p_2 bits of c_1 should be complemented. So, at least n/p_2 bits of c_1 should be complemented. However, either all bits are changed from 1 to 0 or vice versa to maintain periodicity. Hence, the number of 1-bits either decreases or increases by n/p_2, and thus n/p_2 bits should be changed in the opposite direction to satisfy $\|c_1\| = \|c_2\|$. It follows that the Hamming distance between c_1 and c_2 is at least $2n/p_2$. □

LEMMA 11. *Let c_1 and c_2 be two distinct n-bit cyclic numbers with periods p_1 and p_2, respectively, and $\|c_1\| = \|c_2\|$. Then Hamming* $(c_1, c_2) \geqq (2n/p_1p_2)(p_1 + p_2 - 2\gcd(p_1, p_2))$.

Proof. Let $g = \gcd(p_1, p_2)$ and c be an *n-bit* cyclic number with period g and $\|c\| = \|c_1\|$. By Lemma 10, the Hamming distance between c and c_1 is at least $2n/p_1$. Similarly, the Hamming distance between c and c_2 is at least $2n/p_2$. To obtain c_1 from c, we change 1-bits to 0-bits (and 0-bits to 1-bits) for every p_1 bits of c to produce c_1, and we change 1-bits to 0-bits (and 0-bits to 1-bits) for every p_2 bits of c to produce c_2. The number of common bit positions of c that have been changed to generate c_1 and c_2 is $2n/\text{lcm}(p_1, p_2)$, if we changed $2n/p_1$ and $2n/p_2$ bits of c to convert it to c_1 and c_2, respectively. In general, g/p_2 of the bits we changed to generate c_1 and g/p_1 of the bits we changed to generate c_2 correspond to the common bit positions. So, the

Hamming distance between c_1 and c_2 is at least

$$\frac{2n}{p_1}+\frac{2n}{p_2}-\frac{4n}{\text{lcm}\,(p_1,p_2)},$$

i.e., $(2n/p_1p_2)(p_1+p_2-2g)$. □

COROLLARY 6. *Let c_1 and c_2 be two distinct n-bit cyclic numbers with periods p_1 and p_2, respectively; $\|c_1\|=\|c_2\|$ and $\gcd(p_1,p_2)\neq p_1$ and $\neq p_2$. Then Hamming $(c_1,c_2)\geqq 6$.*

Proof. By Lemma 11, *Hamming*$(c_1,c_2)\geqq(2n/p_1p_2)(p_1+p_2-2\gcd(p_1,p_2))$, i.e., *Hamming*$(c_1,c_2)\geqq$

$$\frac{2n}{\text{lcm}\,(p_1,p_2)}\left(\frac{p_1+p_2}{\gcd(p_1,p_2)}-2\right).$$

The maximum value of $\text{lcm}\,(p_1,p_2)$ is n, and the minimum value of

$$\frac{p_1+p_2}{\gcd(p_1,p_2)}$$

is 5 (for $\gcd(p_1,p_2)\neq p_1$ and $\neq p_2$). So, *Hamming*$(c_1,c_2)\geqq 6$ follows. □

COROLLARY 7. *Let c_1 and c_2 be two distinct n-bit cyclic numbers with periods p_1 and p_2, respectively; $\|c_1\|=\|c_2\|$. Then Hamming*$(c_1,c_2)\geqq 4$.

Proof. If $\gcd(p_1,p_2)=p_1$ or p_2, then by Lemma 10, *Hamming*$(c_1,c_2)\geqq 2n/\max(p_1,p_2)$. Since $\max(p_1,p_2)\leqq n/2$, *Hamming*$(c_1,c_2)\geqq 4$.

If $\gcd(p_1,p_2)\neq p_1$ and $\neq p_2$, then, by Corollary 6, *Hamming*$(c_1,c_2)\geqq 6$. □

COROLLARY 8. *Any node has at most one cyclic node as a child.*

Proof. It follows from Corollary 7. □

The following theorem gives a bound on the ratio of the number of cyclic nodes at each level of the whole SBnT.

THEOREM 2. *In an* SBnT, *the ratio of the number of cyclic nodes at level l, $0<l<n$, to the number of nodes at the same level is at most $2/n$.*

Proof. To prove the theorem, we show that for each cyclic number i such that $\|i\|=l$, we can find a set NC_i of noncyclic numbers such that $|NC_i|=(n/2)-1$ and for $j\in NC_i$, $\|j\|=l$. Moreover, the sets for different cyclic numbers are disjoint. A binary number consists of a *string* of bits. Let f be a function that maps a string s of length q to a set of strings of the same length, S_f. We define f to be the function that exchanges any 0-bit with the rightmost 1-bit, or any 1-bit with the rightmost 0-bit. The number of strings in the set S_f is 0, if s contains all 0-bits, or all 1-bits, and $q-1$ otherwise. That $|S_f|=q-1$, if $\|s\|\neq 0$ follows from the fact that each 1-bit and 0-bit determine a unique string, except that the rightmost 1-bit and the rightmost 0-bit determine the same string. For each cyclic string s of length n and period p, we first find the largest p' satisfying $p|p'$, $p'|n$, and $p'<n$. Note that $p'\leqq n/2$. We now want to generate $n-p'-1$ noncyclic strings from the given string s. The first p' bits of these strings are the same as the first p' bits of the string s. The last $n-p'$ bits of the strings are derived by applying the function f to the last $n-p'$ bits of the string s. Each generated string is noncyclic because the Hamming distance between string s and each generated string is 2, and any two cyclic strings of the same length and containing the same number of 1-bits have a Hamming distance of at least 4 (Corollary 7). Since $p'\leqq n/2$, the generated number of noncyclic strings is at least $(n/2)-1$. We now show that the sets S_f generated by two distinct cyclic strings are disjoint. Let c_1 and c_2 be

two distinct cyclic strings with periods p_1 and p_2, respectively, and $\|c_1\| = \|c_2\|$. Consider the following three cases:

- $p_1 = p_2$: Clearly, the two generated sets are disjoint because $c_1 \neq c_2$.
- $\gcd(p_1, p_2) \neq p_1$ and $\neq p_2$: Since the Hamming distance between the generated strings and the given string is 2, and by Corollary 6 the Hamming distance between c_1 and c_2 is at least 6, the two sets are disjoint.
- $\gcd(p_1, p_2) = p_1$ or p_2: Let $\gcd(p_1, p_2) = p_1$ without loss of generality, i.e., $p_1 \mid p_2$. Since the first p_2 bits of the generated strings of c_1 and c_2 are distinct, the two generated sets are disjoint. □

Let c be a cyclic node at level l, $1 \leq l < n$, of subtree 0, and with period p. Let $f(c)$ be the set of nodes obtained by complementing the leftmost 1-bit of c and complementing one of the $\lceil (n-l)/2 \rceil$ closest 0-bits to the right of it. For example, for $c = (0010100101)$, $f(c) = \{(0001100101), (0000110101), (0000101101)\}$. Let $\mathscr{C}_l = \{c_1, c_2, \cdots, c_{\delta_l}\}$ be the set of all cyclic nodes at level l of subtree 0.

LEMMA 12. *Every node in the set* $\bigcup_{c_i \in \mathscr{C}_l} f(c_i)$ *is a noncyclic node at level l of subtree* 0. *Moreover, if* $c_i \neq c_j$, $c_i, c_j \in \mathscr{C}_l$, *then* $f(c_i) \cap f(c_j) = \phi$.

Proof. By Corollary 7, the mapping function $c \to f(c)$ maps a cyclic node to a set of noncyclic nodes. Since the leftmost 1-bit of c and some 0-bit to the right of it are exchanged, the index is preserved, i.e., the node obtained through the exchange operation is in the same subtree as c, and at the same level, since the number of 1-bits is preserved. Let $c_1 = (r_1 r_2 \cdots r_{n/p_1})$ and $c_2 = (s_1 s_2 \cdots s_{n/p_2})$ be two arbitrary nodes in $\mathscr{C}_l$ with periods p_1 and p_2, respectively, and $r_i(s_i)$ be the bit string of length $p_1(p_2)$. Without loss of generality we assume $p_1 \leqq p_2$ and consider the following cases:

- $p_1 < p_2$ and $\gcd(p_1, p_2) \neq p_1$: From Lemma 6, $Hamming(c_1, c_2) \geqq 6$. Since the Hamming distance between c_1 and any node in $f(c_1)$ is 2, and the Hamming distance between c_2 and any node in $f(c_2)$ is 2, $Hamming(c_1', c_2') \geqq 2$, $\forall c_1' \in f(c_1)$, $c_2' \in f(c_2)$, i.e., $f(c_1) \cap f(c_2) = \phi$.
- $p_1 = p_2 = p$: Pick any node $c_1' \in f(c_1)$ and any node $c_2' \in f(c_2)$. Let $x_1(x_2)$ be the number of bits in the string $r_i(s_i)$, which is to the right of the leftmost 1-bit including this 1-bit. From the definition of f, the last x_1 bits of c_1' are the same as the last x_1 bits of c_1. Similarly, the last x_2 bits of c_2' are the same as the last x_2 bits of c_2. If $x_1 = x_2$, then the string formed by the last x_1 bits of c_1 and the string formed by the last x_1 bits of c_2 are distinct since $c_1 \neq c_2$. Hence, the last x_1 bits of c_1' and c_2' are distinct, i.e., $c_1' \neq c_2'$ and $f(c_1) \cap f(c_2) = \phi$, since c_1' and c_2' are arbitrarily chosen.

  If $x_1 < x_2$, then the last x_1 bits of c_1' have nl/p 1-bits and the last x_2 bits of c_2' also have nl/p 1-bits. The leading bit of the last x_2 bits of c_2' is a 1-bit by definition. Hence, the last x_1 bits of c_2' contain at most $(nl/p) - 1$ 1-bits. So, $c_1' \neq c_2'$ or $f(c_1) \cap f(c_2) = \phi$. The proof for $x_1 > x_2$ is similar.
- $p_1 < p_2$ and $\gcd(p_1, p_2) = p_1$: Let $p_2 = \gamma p_1$, $\gamma > 1$. Partition each bit string s_i of c_2, each of length p_2, into γ substrings of length p_1 each. Denote these substrings $s_i^1, s_i^2, \cdots, s_i^\gamma$. So,

$$c_2 = (s_1^1 s_1^2 \cdots s_1^\gamma s_2^1 s_2^2 \cdots s_2^\gamma \cdots s_{n/p_2}^1 s_{n/p_2}^2 \cdots s_{n/p_2}^\gamma).$$

For convenience, we also relabel

$$c_1 = (r_1^1 r_1^2 \cdots r_1^\gamma r_2^1 r_2^2 \cdots r_2^\gamma \cdots r_{n/p_2}^1 r_{n/p_2}^2 \cdots r_{n/p_2}^\gamma).$$

Note that $r_{i_1}^{j_1} = r_{i_2}^{j_2}$, $\forall 1 \leqq i_1, i_2 \leqq n/p_2$, $1 \leqq j_1, j_2 \leqq \gamma$; also $s_{i_1}^j = s_{i_2}^j$, $\forall 1 \leqq i_1, i_2 \leqq n/p_2$, $1 \leqq j \leqq \gamma$. Let the leftmost 1-bit of c_2 be in s_1^j. If $j > 1$, then the last $(\gamma - j + 1)p_1$ bits of

c_2' contain nl/p_2 1-bits, while the last $(\gamma-j+1)p_1$ bits of c_1' contain $(\gamma-j+1)nl/\gamma p_2$ 1-bits, which is less than nl/p_2. So, $c_1' \neq c_2'$. If $j=1$, then consider the following two cases:

- $r_1^y \neq s_1^y$ for some $y, 1<y\leqq\gamma$: By definition, the last $(\gamma-1)p_1$ bits of c_1' are identical to the string $(r_1^2 r_1^3 \cdots r_1^\gamma)$, and the last $(\gamma-1)p_1$ bits of c_2' are identical to the string $(s_1^2 s_1^3 \cdots s_1^\gamma)$. So, $c_1' \neq c_2'$.
- $r_1^1 \neq s_1^1$: Let $x_1(x_2)$ be the number of bits of $r_1^1(s_1^1)$ to the right of the leftmost 1-bit of $r_1^1(s_1^1)$ including this 1-bit. Consider the last $x_1(x_2)$ bits of $r_{n/p_2}^1(s_{n/p_2}^1)$. The proof of $c_1' \neq c_2'$ follows that of the case $p_1=p_2$. □

THEOREM 3. *For an* SBnT, *the ratio of the number of cyclic nodes at any level* l, $1\leqq l<n$, *of any subtree to the number of nodes at the same level of the same subtree is at most*

$$\frac{1}{1+\max\left(\left\lceil\frac{n-l}{2}\right\rceil,\left\lceil\frac{l}{2}\right\rceil\right)}\leqq\frac{4}{4+n}.$$

Proof. By Lemma 12, the function $c\to f(c)$ maps any cyclic node $c\in\mathscr{C}_l$ at level l of subtree 0 to a set of $\lceil(n-l)/2\rceil$ noncyclic nodes at the same level of the same subtree. Moreover, the sets $f(c)$ for different cyclic nodes are disjoint. Since complementing all bits of a binary number preserves the period, the number of degenerate necklaces with l 1-bits for the nodes in the necklace is the same as the number of degenerate necklaces with $n-l$ 1-bits for the nodes in the necklace. So, the ratio of the number of cyclic nodes to the total number of nodes at level l of subtree 0 is the same as the ratio at level $n-l$ of subtree 0. Since subtree 0 has the maximum number of cyclic nodes at each level, and noncyclic nodes at each level are evenly distributed among the n subtrees, the ratio for subtree 0 is an upper bound for all subtrees. □

COROLLARY 9. *With the exception of the last level, the number of cyclic nodes at any level of any subtree of an* SBnT *is at most the same as the number of noncyclic nodes at the same level of the same subtree of the* SBnT.

Proof. By Theorem 3, $4/(4+n)\leqq 1/2$ for $n\geqq 4$. But since for $n=2$ and $n=3$, n is a prime number and there are no cyclic nodes, except nodes $c=(00\cdots00)$ and $(11\cdots11)$ the proof is complete. □

Theorem 3 gives a bound on the ratio of the number of cyclic nodes at each level in each subtree. Figure 6 shows the ratio of the actual number of cyclic nodes to the total number of nodes for each level of up to 16-dimensional cubes for subtree 0. The bound given by the theorem is pessimistic, and we conjecture that the ratio is $2/n$ (which is true for up to 16-dimensional cubes). In Fig. 6 the letters encode cube dimensions greater than 9, and digits cube dimensions less than 10. For the number of cube dimensions being a prime number, there are no cyclic nodes, except at levels 0 and n. Note that the ratio neither decreases monotonically with the cube size for a given level, nor decreases monotonically with increasing level for a given cube size.

COROLLARY 10. *If* n *is even, then the ratio of the number of cyclic nodes at level* 2 *(or* $l-2$*) of subtree* 0 *to the number of nodes at the same level of the same subtree is* $2/n$.

Proof. There is only one cyclic node and $(n/2)-1$ noncyclic nodes at level 2 $(l-2)$. □

If the conjecture is true, then for all even n the maximum ratio occurs at levels 2 and $n-2$ (excluding levels 0 and n).

The definition of the SBnT given above has a unique path from the source to every other node. For personalized communication the number of data elements that need to be transferred to subtree 0 exceeds the number of elements that need to be

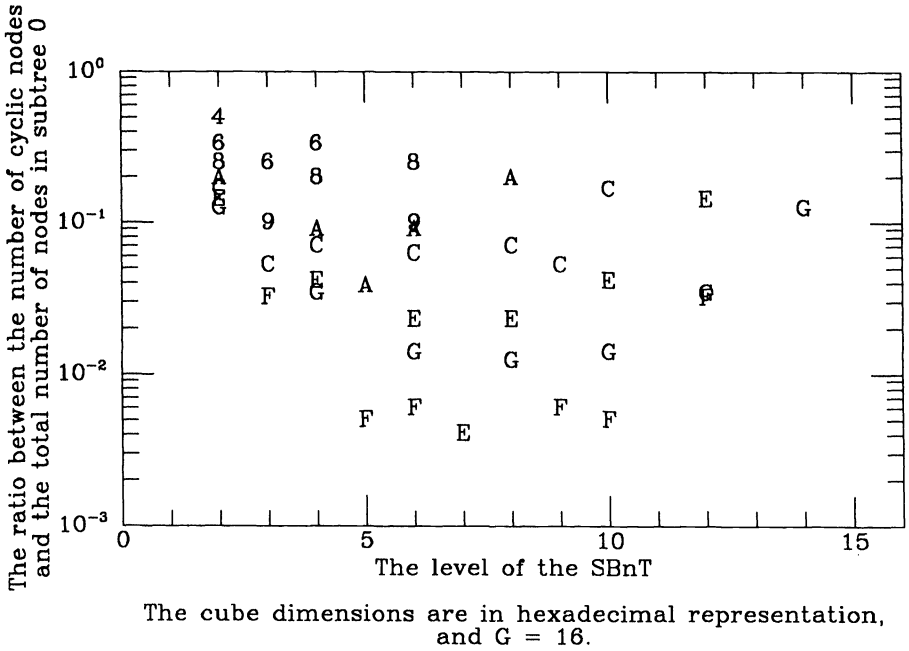


FIG. 6. *The ratio of the number of cyclic nodes to the total number of nodes at level l, $1 \leqq l < n$, of subtree 0.*

transferred to subtree i, $n/2 \leqq i \leqq n-1$, by the number of degenerate necklaces times the data set transferred to each node (assuming the same amount of data to each node). By allowing multiple parent nodes for every cyclic node c, and by splitting the data set for each cyclic node of period P_c into n/P_c parts, the load becomes the same for each subtree of the root. The bandwidth requirement for each link from the root is $(N-1)M/n$, where M is the size of the data set for each node.

To carry out the load-balancing operation, the definition of the set $\mathcal{M}(c)$ is modified by using the set $index_m(c)$ for the definition instead of $index(c)$. Let k_q be such that

$$c_{k_q} = 1,\ c_r = 0,\ r \in \{(k_q+1) \bmod n, (k_q+2) \bmod n, \cdots, (j_q-1) \bmod n\}$$

and

$$k = -1 \text{ if } c = 0.$$

Then,

$$\mathcal{M}_m(c) = \bigcup_{j_q \in J_c} \{(k_q+1) \bmod n, (k_q+2) \bmod n, \cdots, (j_q-2) \bmod n, (j_q-1) \bmod n\}.$$

The $children_{\mathrm{SBnT}}$ and $parent_{\mathrm{SBnT}}$ functions are modified as follows:

$$children_{\mathrm{SBG}}(i, s) = \begin{cases} \{(i_{n-1}i_{n-2}\cdots \bar{i}_m \cdots i_0)\},\ \forall m \in \mathcal{M}_m(c), & \text{if } c = 0; \\ \{q_m = (i_{n-1}i_{n-2}\cdots \bar{i}_m \cdots i_0)\}, \\ \quad \forall m \in \mathcal{M}_m(c) \text{ and } index_m(q_m \oplus s) = index_m(c), & \text{if } c \neq 0. \end{cases}$$

$$parent_{\mathrm{SBG}}(i, s) = \begin{cases} \phi, & \text{if } c = 0; \\ (i_{n-1}i_{n-2}\cdots \bar{i}_{k_q} \cdots i_0),\ q = \{1, 2, \cdots, m\}, & \text{otherwise.} \end{cases}$$

For example, node (011011011) appears in subtrees 0, 3, and 6 with parents (001011011), (011011001), and (011001011), respectively. The modified SBnT, SBG, is a spanning graph [5], which can be viewed as composed of n rotated SBnTs with a weight of $1/n$ each. The parent of a cyclic node c in the jth SBnT is derived by choosing the dimension from the set $index_m(c)$ that is of the lowest order greater than j, cyclically.

4. Other choices of spanning balanced n-trees. For the definition of the balanced n-tree and the balanced graph SBG, we defined the generating subtree based on distinguished nodes selected as the node of a necklace with minimum value. Another obvious choice is to select the node with maximum value. Generating subtrees defined on distinguished nodes obtained from these two definitions are not rotations of each other. For instance, the nodes with relative addresses (0001), (0011), (0111), and (1111) are all minimal in their respective necklaces and belong to the generating subtree for the SBnT we defined previously. Nodes (1000), (1100), (1110), and (1111) are all maximal in their respective necklaces and belong to the generating subtree for an SBnT based on selecting maximum values in necklaces. The number of right rotations to match a node from one selection criterion with a node from the other selection criterion is node dependent. We will now define and compare spanning balanced trees based on maximum values, and minimum and maximum bit-reversed values. Different definitions of the distinguished nodes result in balanced trees, or graphs, that use different edges of the cube, with the exception that they all use all the edges directed away from the root.

We refer to a balanced n-tree defined through maximizing left rotations by $\text{SBnT}^{\max L}$, and the previous tree as $\text{SBnT}^{\min R}$. The set of indices that maximizes the left rotations is denoted $J_c^{\max L} = \{j_1, j_2, \cdots, j_m\}$ and it is defined by $L^u(c) = L^v(c)$, if $u, v \in J_c^{\max L}$, and $L^u(c) > L^w(c)$, if $u \in J_c^{\max L}$ and $w \in \{0, 1, \cdots, n-1\} - J_c^{\max L}$. Moreover $0 \leqq j_1 < j_2 < \cdots < j_m < n$ and $index^{\max L}(c) = j_1$ and $index_m^{\max L}(c) = J_c^{\max L}$. The formal definitions of the $parent_{\text{SBnT}^{\max L}}$ and $children_{\text{SBnT}^{\max L}}$ functions are based on a set of dimensions $\mathcal{M}^{\max L}(c)$ similar to the set $\mathcal{M}(c)$ for right rotations. Let k be such that $c_k = 1$ and $c_j = 0$, $j \in \{(k-1) \bmod n, (k-2) \bmod n, \cdots, (n - index^{\max L}(c)) \bmod n\} = \mathcal{M}^{\max L}(c)$ and $k = n$ if $c = 0$. The $\text{SBnT}^{\max L}$ is defined by

$$children_{\text{SBnT}^{\max L}}(i, s) = \begin{cases} \{(i_{n-1} i_{n-2} \cdots \bar{i}_m \cdots i_0)\}, & \forall m \in \mathcal{M}^{\max L}(c), \quad \text{if } c = 0; \\ \{q_m = (i_{n-1} i_{n-2} \cdots \bar{i}_m \cdots i_0)\}, & \forall m \in \mathcal{M}^{\max L}(c) \\ \quad \text{and } index^{\max L}(q_m \oplus s) = index^{\max L}(c), & \text{if } c \neq 0. \end{cases}$$

$$parent_{\text{SBnT}^{\max L}}(i, s) = \begin{cases} \phi, & \text{if } c = 0; \\ (i_{n-1} i_{n-2} \cdots \bar{i}_k \cdots i_0), & \text{otherwise.} \end{cases}$$

Figure 7 shows the $\text{SBnT}^{\max L}$ in a 5-cube. A balanced $\text{SBG}^{\max L}$ graph can be defined similarly to the balanced $\text{SBG}^{\min R}$ graph. Note that the choice of *minimizing* the *right* rotation in the $\text{SBnT}^{\min R}$ is made such that the root of subtree j is $(00 \cdots 01_j 0 \cdots 0)$. If the number of left rotations that identifies a node with a distinguished node were used instead of the number of right rotations for the assignment of a node to a subtree, then subtrees $1, 2, \cdots, n-1$ in such an $\text{SBnT}^{\min L}$ are relabelings of subtrees $n-1, n-2, \cdots, 1$ of the $\text{SBnT}^{\min R}$, respectively. A $\text{SBnT}^{\min L}$ and an $\text{SBnT}^{\min R}$ are identical and differ only in the labeling of the subtrees. A similar argument applies to the $\text{SBnT}^{\max L}$ and the $\text{SBnT}^{\max R}$.

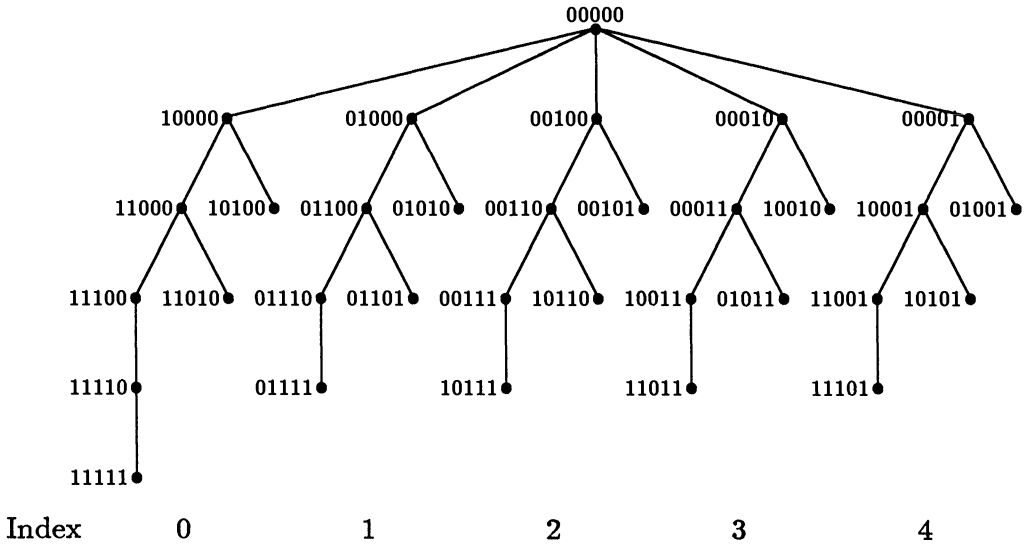


FIG. 7. *A* $\text{SBnT}^{\max L}$ *in a 5-cube.*

LEMMA 13. *The number of nodes of each subtree of the tree* $\text{SBnT}^{\max L}$ *is equal to the number of nodes in the same subtree of the tree* $\text{SBnT}^{\min R}$.

Proof. For a noncyclic node, all the n nodes in the same necklace belong to n different subtrees. For a cyclic node with period p, all the p nodes in the same necklace belong to subtree 0 to $p-1$. □

THEOREM 4. *The* $\text{SBnT}^{\max L}$ *and* $\text{SBnT}^{\min R}$ *are not generally topologically equivalent.*

Proof. We prove the theorem by a direct evaluation for a 6-cube. In a 6-cube, the maximum fanout of any node at level 2 of an $\text{SBnT}^{\min R}$ is 2 (Fig. 4) and the maximum fanout of any node at level 2 of an $\text{SBnT}^{\max L}$ is 3 (Fig. 9). □

Two definitions of balanced n-trees dual to the $\text{SBnT}^{\min R}$ and the $\text{SBnT}^{\max L}$, and that use mostly a different set of edges, can be derived from the bit-reversed representation of the addresses. The dual of the $\text{SBnT}^{\min R}$ is denoted $\text{SBnT}^{\min BL}$, and the dual of the $\text{SBnT}^{\max L}$ is denoted $\text{SBnT}^{\max BR}$. The $\text{SBnT}^{\min BL}$ is defined as follows:

Let $J_c^{\min BL} = \{j_1, j_2, \cdots, j_m\}$, where $0 \leqq j_1 < j_2 < \cdots < j_m < n$, $L^u(c) = L^v(c)$, $u, v \in J_c^{\min BL}$, and $B \circ L^u(c) < B \circ L^w(c)$, $u \in J_c^{\min BL}$, $w \in \{0, 1, \cdots, n-1\} - J_c^{\min BL}$. Then $index^{\min BL}(c) = j_1$ and k is defined by $c_k = 1$ and $c_j = 0$, $j \in \{(k-1) \bmod n, (k-2) \bmod n, \cdots, (n - index^{\min BL}(c)) \bmod n\} = \mathcal{M}^{\min BL}(c)$ and $k = n$ if $c = 0$.

$$children_{\text{SBnT}^{\min BL}}(i, s) = \begin{cases} \{(i_{n-1} i_{n-2} \cdots \bar{i}_m \cdots i_0)\}, & \forall m \in \mathcal{M}^{\min BL}(c), \quad \text{if } c = 0; \\ \{q_m = (i_{n-1} i_{n-2} \cdots \bar{i}_m \cdots i_0)\}, & \forall m \in \mathcal{M}^{\min BL}(c) \\ \quad \text{and } index^{\min BL}(q_m \oplus s) = index^{\min BL}(c), & \text{if } c \neq 0. \end{cases}$$

$$parent_{\text{SBnT}^{\min BL}}(i, s) = \begin{cases} \phi, & \text{if } c = 0; \\ (i_{n-1} i_{n-2} \cdots \bar{i}_k \cdots i_0), & \text{otherwise.} \end{cases}$$

The $\text{SBnT}^{\max BR}$ is defined as follows:

Let $J_c^{\max BR} = \{j_1, j_2, \cdots, j_m\}$, where $0 \leqq j_1 < j_2 < \cdots < j_m < n$, $R^u(c) = R^v(c)$, $u, v \in J_c^{\max BR}$, and $B \circ R^u(c) > B \circ R^w(c)$, $u \in J_c^{\max BR}$, $w \in \{0, 1, \cdots, n-1\} - J_c^{\max BR}$. Then

$index^{\max BR}(c) = j_1$ and k is defined by $c_k = 1$ and $c_j = 0$, $j \in \{(k+1) \bmod n, (k+2) \bmod n, \cdots, (index^{\max BR}(c) - 1) \bmod n\} = \mathcal{M}^{\max BR}(c)$ and $k = -1$ if $c = 0$.

$$children_{\mathrm{SBnT}^{\max BR}}(i, s) = \begin{cases} \{(i_{n-1} i_{n-2} \cdots \bar{i}_m \cdots i_0)\}, & \forall m \in \mathcal{M}^{\max BR}(c), \quad \text{if } c = 0! \\ \{q_m = (i_{n-1} i_{n-2} \cdots \bar{i}_m \cdots i_0)\}, & \forall m \in \mathcal{M}^{\max BR}(c) \\ \quad \text{and } index^{\max BR}(q_m \oplus s) = index^{\max BR}(c), & \text{if } c \neq 0. \end{cases}$$

$$parent_{\mathrm{SBnT}^{\max BR}}(i, s) = \begin{cases} \phi, & \text{if } c = 0; \\ (i_{n-1} i_{n-2} \cdots \bar{i}_k \cdots i_0), & \text{otherwise.} \end{cases}$$

$index^{\min R}(c)$ is the number of right rotations yielding the longest block of leading 0-bits, and $index^{\max L}(c)$ the number of left rotations yielding the longest block of leading 1-bits. Similarly, $index^{\min BL}(c)$ is the number of left rotations yielding the longest block of trailing 0-bits, and $index^{\max BR}(c)$ the number of right rotations yielding the longest block of trailing 1-bits. For example, $index^{\min R}$, $index^{\max L}$, $index^{\min BL}$, and $index^{\max BR}$ of node (1110100010) are 5, 0, 8, and 7, respectively.

Note that the rotation and bit-reversal operations do not commute, i.e., $R \circ B \neq B \circ R$ and $L \circ B \neq B \circ L$.

LEMMA 14. *The bit-reversed value of a bit string with a minimum value among all its rotations is not necessarily the maximum value among the bit-reversals of its rotations, i.e.,* $B \min_x R^x(i) \neq \max_x BR^x(i)$ *for some* i.

Proof. (001101) is the minimum value among all its rotations. But, (101100) is not the maximum bit-reversed value among all rotations of (001101). □

Note that for $n \leqq 5$, the minimum value among all rotations of an address also yields the maximum bit-reversed value of its rotations. This means that the $\mathrm{SBnT}^{\max BR}$($\mathrm{SBnT}^{\max L}$) and the $\mathrm{SBnT}^{\min R}$ are topologically equivalent for up to five-dimensional cubes.

THEOREM 5. *The* $\mathrm{SBnT}^{\min R}$ *and* $\mathrm{SBnT}^{\min BL}$ *are topologically equivalent for all n, and so are the* $\mathrm{SBnT}^{\max L}$ *and* $\mathrm{SBnT}^{\max BR}$.

Proof. Consider any node i in subtree 0 of the $\mathrm{SBnT}^{\min R}$ and the node $B(i)$ in $\mathrm{SBnT}^{\min BL}$. This mapping is one-to-one and onto. We first show that every node of subtree 0 of the $\mathrm{SBnT}^{\min R}$ has a corresponding node in subtree 0 of the $\mathrm{SBnT}^{\min BL}$. To show this property we need to show that $BL^x B(i)$ is minimized for $x = 0$ since i is in subtree 0 of the $\mathrm{SBnT}^{\min R}$. But $BL^x B(i) = BL^{x-1} LB(i) = BL^{x-1} BR(i)$, and it follows that $BL^x B(i) = R^x(i)$, which is minimized for $x = 0$. Correspondingly, every node of subtree 0 of $\mathrm{SBnT}^{\min BL}$ has a counterpart in subtree 0 of the $\mathrm{SBnT}^{\min R}$. The same argument applies for any other subtree, and it follows that the number of nodes in every subtree of the $\mathrm{SBnT}^{\min R}$ is the same as the number of nodes in the same subtree of the $\mathrm{SBnT}^{\min BL}$. To complete the proof, we notice that the bit-reversal operation preserves adjacency.

For the $\mathrm{SBnT}^{\max L}$ and $\mathrm{SBnT}^{\max BR}$ case, we instead use the property that $BR^x B(i) = L^x(i)$. □

Figures 8, 9, and 10 show subtree 0 of an $\mathrm{SBnT}^{\min BL}$, $\mathrm{SBnT}^{\max L}$, and $\mathrm{SBnT}^{\max BR}$ in a 6-cube. The nodes in square boxes are cyclic.

For *n-port one-to-all personalized communication*, the $\mathrm{SBnT}^{\min R}$ routing has an advantage over the $\mathrm{SBnT}^{\max L}$ routing in that the maximum fanout is for most levels lower than for the $\mathrm{SBnT}^{\max L}$ routing. The fanout decreases monotonically for the $\mathrm{SBnT}^{\min R}$ by Lemma 4, but this is only true for the $\mathrm{SBnT}^{\max L}$ for levels $l \geqq 2$. Any spanning tree satisfying Lemma 4 guarantees that the complexity of personalized communication with concurrent communication on all ports is determined by the root. The maximum

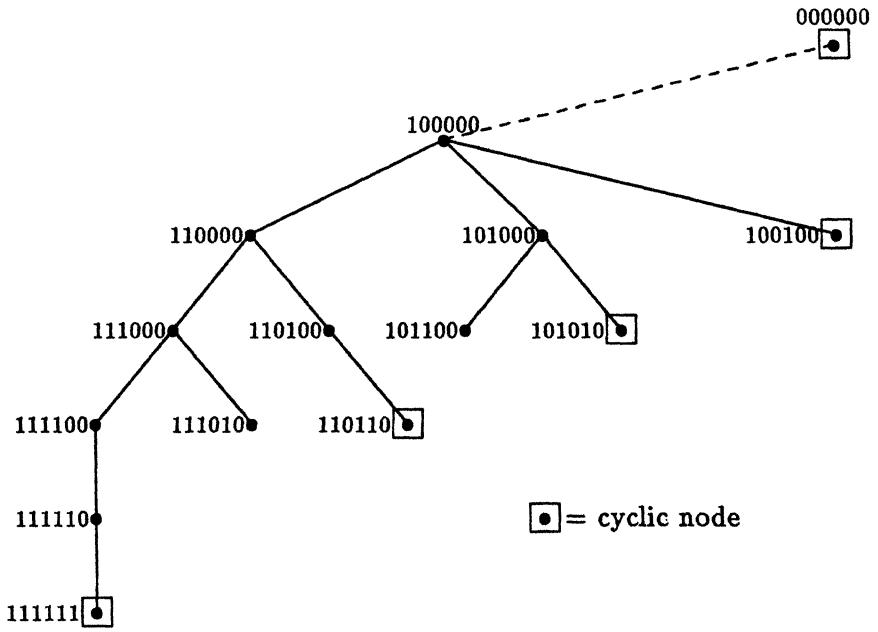


FIG. 8. *Subtree* 0 *of an* $\text{SBnT}^{\min BL}$ *in a* 6*-cube.*

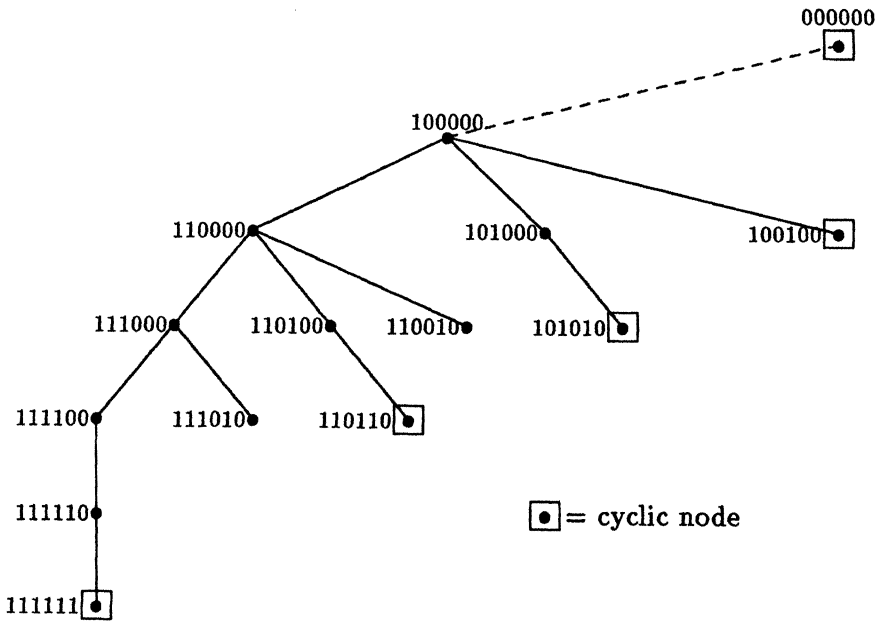


FIG. 9. *Subtree* 0 *of an* $\text{SBnT}^{\max L}$ *in a* 6*-cube.*

fanout of nodes at level l of the $\text{SBnT}^{\max L}$ is

$$\begin{cases} \left\lceil \dfrac{n-l}{2} \right\rceil, & \text{if } l=1; \\ n-l-1, & \text{if } 2 \leqq l \leqq n-2; \\ 1, & \text{if } l=n-1. \end{cases}$$

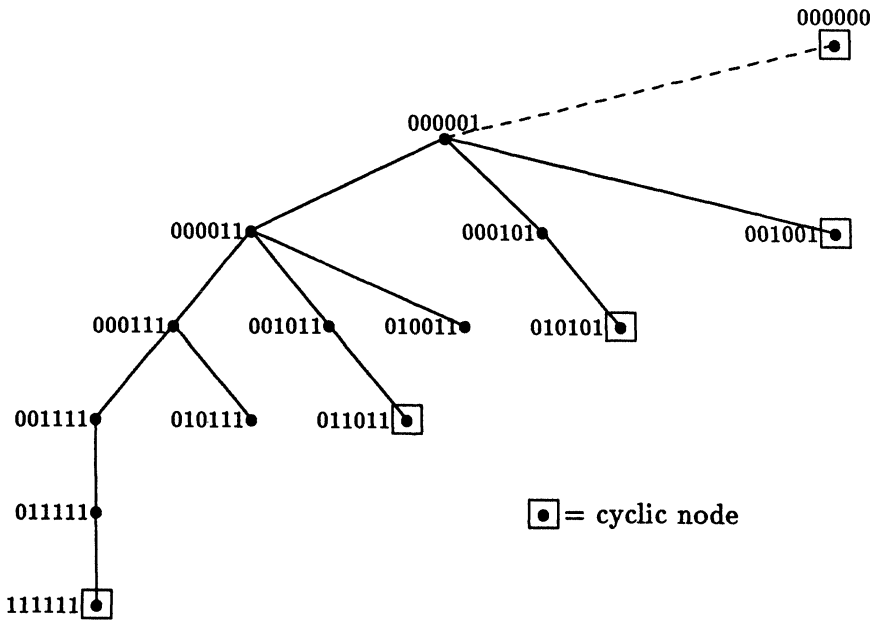


FIG. 10. *Subtree* 0 *of an* $\text{SBnT}^{\max BR}$ *in a* 6*-cube.*

For the $\text{SBnT}^{\min R}$ the fanout at level l is $\lceil (n-l)/2 \rceil$, $1 \leqq i < n$, by Lemma 3. The preference of the $\text{SBnT}^{\min R}$ over the $\text{SBnT}^{\min BL}$ is due to the simpler computation of the index.

LEMMA 15. *For any node below level* 1 *of an* SBnT, *the* $parent_{\text{SBnT}^{\min R}}$ *and* $parent_{\text{SBnT}^{\min BL}}$ *functions define two distinct nodes, if the relative address has a unique longest consecutive block of zeros, cyclically.*

Proof. By definition, $index^{\min R}$ is the dimension of the 1-bit immediately to the left of the longest block of zeros. The parent address can be derived by complementing the 1-bit, which is immediately to the right of the longest block of zeros. Similarly, $index^{\min BL}$ is the dimension of the 1-bit immediately to the right of the longest block of zeros. The parent address can be derived by complementing the 1-bit, which is immediately to the left of the longest block of zeros. □

The $parent_{\text{SBnT}^{\min R}}$ and the $parent_{\text{SBnT}^{\min BL}}$ functions are distinct. It can be shown that the $\text{SBnT}^{\min R}$ and the $\text{SBnT}^{\min BL}$ are edge-disjoint below level 1 for up to four-dimensional cubes. For 5- and 6-cubes, there are 5 and 6 common edges. For 7- and 8-cubes, there are 14 and 16 common edges. The incoming edges of nodes (01011), (010111), and (0010011) are examples of shared edges. Modifications to the $parent_{\text{SBnT}}$ function, such as permutation of the dimensions, can be made to insure that the modified SBnT, and, for instance, $\text{SBnT}^{\min R}$, are edge-disjoint below level 1. The existence of SBnTs that are edge-disjoint below level 1 is important for fault-tolerance.

5. Personalized communication based on an SBnT graph. As an example of the use of the SBnT, we give some complexity results for personalized communication in a Boolean n-cube. We first consider the case of *one-to-all personalized communication* with the communication restricted to *one-port* at a time. With this restriction we assume that the entire data set for the subtree rooted at the sending node is communicated in one routing cycle. For each node we employ a *scheduling discipline* for which data is sent to subtrees in order of increasing dimension of the connecting edge. For the root

there is a minor difference in the data volume to the different subtrees for SBnT routing, but no difference for SBG routing.

With *one-port* communication the root requires a time of $M(N-1)t_c+n\tau$ for packet-switched communication with an unbounded buffer size, a data set M communicated to each node, a transfer time of t_c for each element of the data set, and a start-up time τ for each routing cycle. After the last routing cycle of the root, the last subtree to receive the data has to distribute it to its nodes. We will now prove that a node c in subtree $index(c)$ receives its data during routing cycle $index(c)+n-1-\alpha_c$, and that the data transfer time for each subtree is bounded from above by $O(MN \log n/n)t_c)$. To prove these results we need the following lemmas.

LEMMA 16. *Let k be the dimension of the edge through which a nonroot node c connects to its parent in an* SBnT. *Let the number of children of c be β. Then, the β children of c are connected through edges in dimensions $(k+j) \bmod n$, $1 \leqq j \leqq \beta$.*

Proof. Consider nodes in subtree 0 first. From the definitions of the $parent_{\mathrm{SBnT}}$ and $children_{\mathrm{SBnT}}$ functions, the kth bit is the leftmost 1-bit of c, and the dimension of the edges connecting c to its children can be derived by complementing each of the leading 0-bits for which the index is preserved. Notice that if complementing the $(k+j)$th bit of c $(k+j<n-1)$ changes the index, then so does complementing each of the bits $\{k+j+1, k+j+2, \cdots, n-1\}$. So, the dimensions of the edges connecting to the children nodes of c form a contiguous set of dimensions (modulo n). For nodes in other subtrees, we consider $R^{index(c)}(c)$ instead, and the proof is similar. □

COROLLARY 11. *The longest path in a subtree of the root is the path corresponding to the path* $(00\cdots01), (00\cdots11), \cdots, (01\cdots11), (11\cdots11)$ *in subtree* 0.

LEMMA 17. *With a scheduling of all data for one subtree during a single routing cycle and subtrees in order of increasing dimension of the edge through which they are connected, a node c receives the data for the subtree rooted at it during routing cycle $index(c)+n-1-\alpha_c$ with the first cycle being numbered* 0. *The total number of routing cycles, $2n-2$, is the minimum possible of the* SBnT *routing for one-port communication.*

Proof. We prove the lemma by labeling the edges of the SBnT. A labeling is valid if for each node the labels of edges connecting to its children are distinct and greater than the label of the edge to the parent node. The smallest label is greater than or equal to 0. Define a labeling scheme for subtree 0 first. Edges are labeled according to their corresponding dimensions. Hence, the edge connecting to the parent of a node c with α_c leading 0's is labeled $n-1-\alpha_c$. This labeling is valid by Lemma 16. For the other subtrees, we simply add $index(c)$ to the corresponding labels of subtree 0.

The maximum label is $n-1$ for subtree 0 by Lemma 16 and Corollary 11 and $index(c)+n-2$ for subtree $index(c)$. So, the maximum label of the SBnT is $2n-3$. Interpreting the labels on the edges as routing cycles, the proof is complete by noticing that the total number of cycles, $2n-2$, is the minimum possible since the root has n children and each subtree of the root is of height at least $n-2$. □

Note that the length of a routing cycle is determined by the data volume that needs to be transmitted over a single edge. The data volume depends on the number of nodes in the subtree connected through that edge. Different edges in a given dimension connect to subtrees of different sizes, unlike in a spanning binomial tree for which all tree edges in a given dimension connect to subtrees of the same size. To estimate the data transfer time we need to find the edge in each dimension that transfers the maximum number of elements, since each dimension is routed only once within a subtree.

LEMMA 18. *Each of the edges forming the longest path transfers the maximum number of elements of any edge in that dimension.*

Proof. Consider subtree 0 first. Let T_j be the subtree rooted at the child of node c connected through an edge in dimension $(k+j) \bmod n$, $1 \leqq j \leqq \beta$, and S_j be the number of nodes in subtree T_j. Then the proof is complete by proving that $S_1 \geqq S_2 \geqq \cdots \geqq S_\beta$. Every node in subtree T_{j+1} has addresses with bit $k+j$ equal to zero, bit $k+j+1$ equal to one, and the k least-significant bits equal to the k least-significant bits of c. By moving bit $k+j$ to the most significant bit, we map every node in subtree T_{j+1} to a unique node. Since the mapping preserves the index, the nodes in subtree T_{j+1} are mapped to subtree T_j. So, $S_{j+1} \leqq S_j$ and the proof for subtree 0 is completed by induction. For nodes in other subtrees of the root, we apply the arguments to the binary number $R^{index(c)}(c)$ instead of c. □

For the estimate of the element transfer time we need the following lemma. For proof of this lemma, see [6].

LEMMA 19. *The number of n-bit binary strings for which the longest block of consecutive* 0*-bits* (1*-bits*) *has length* $\log n - \log \ln n - 1$, *or length greater than* $2 \log n$, *is at most* $O(N/n)$.

THEOREM 6. *The communication complexity of an* SBnT *based one-port one-to-all personalized communication with a scheduling of all data for a subtree during a single routing cycle, and subtrees in order of increasing dimension of the edge through which they are connected is bounded from above by* $N(1+O(\log n/n))Mt_c + (2n-2)\tau$.

Proof. The root needs n routing cycles and a time of $(N-1)Mt_c + n\tau$ to send the data to its n children. The communication time required after the final routing cycle of the root is at most the same as the time required for communication in subtree 0. By Lemmas 11 and 18 the communication time for subtree 0 is dominated by the path $(00\cdots01), (00\cdots11), \cdots, (01\cdots11), (11\cdots11)$. The data transfer time is

$$= Mt_c \times \sum_{i=2}^{n} (\text{the number of nodes with at least } i \text{ trailing 1's and } index = 0)$$

$$= Mt_c \times \sum_{i=2}^{n} (i-1) \times (\text{the number of nodes with } i \text{ trailing 1's and } index = 0)$$

$$\leqq Mt_c \times \sum_{i=2}^{n} (i-1) \times (\text{the number of nodes with the length of the longest consecutive 1-bits being } i \text{ and } index = 0)$$

$$\approx Mt_c \times \sum_{i=2}^{n} (i-1) \times \frac{1}{n} \times (\text{the number of nodes with the length of the longest consecutive 1-bits being } i)$$

$$\leqq Mt_c \times \frac{1}{n}\left(n \times O\left(\frac{N}{2}\right) + 2 \log n \times N\right)$$

$$= Mt_c \times O\left(\frac{N \log n}{n}\right).$$

Note that nodes with i trailing 1's and $index = 0$ may have a length of the longest substring of consecutive 1's greater than i, such as node (0000111101). So, the third equation is an upper bound of the second equation. The order of the fourth equation is the same as that of the third by Theorem 3. The fifth equation follows from Lemma 19. The first term in parentheses is a bound for the nodes below level $2 \log n$, and the second term is a bound for nodes above level $2 \log n$. The proof is completed by noticing that the last subtree needs only $n-2$ start-ups after the last routing cycle of the root. □

A lower bound for *one-port one-to-all personalized communication* is $\max(M(N-1)t_c, n\tau)$. Routing according to a spanning binomial tree and the above scheduling discipline is optimal within a factor of two. For *n-port* communication the lower bound is $\max(M(N-1)/n)t_c, n\tau)$. The SBnT routing with a *reverse breadth-first* scheduling [5] is optimal within a factor of two. Routing according to a spanning binomial tree has a complexity that is $O(n)$ worse than the lower bound.

The lower bound for *one-port all-to-all personalized communication* is $\max((nNM/2)t_c, n\tau)$ [5]. A spanning binomial tree routing with an appropriate scheduling discipline is optimal within a factor of two. The minimum number of start-ups for the SBnT routing is approximately twice that of the binomial tree routing, but the data transfer time is approximately the same. For *n-port* communication the SBnT routing again is a factor of two within the lower bound $\max((NM/2)t_c, n\tau)$, but the binomial tree routing is at least $O(\sqrt{n})$ worse than the lower bound [5].

The complexities for some communication algorithms based on the SBnT routing are summarized in Table 2.

6. Implementation issues. We have implemented the $\text{SBnT}^{\min R}$ routing algorithm for personalized communication on an Intel iPSC Boolean cube configured multiprocessor with 64 nodes. In the SBnT routing, the root determines which node belongs to which subtree. If n is a prime number the subtrees are isomorphic (excluding node $(\bar{s}_{n-1}\bar{s}_{n-2}\cdots\bar{s}_0)$) and the root needs to keep only one table of length $\approx N/n$ with each entry of size n bits. The order of the entries corresponds to the transmission order for each port. The table entries point to the messages transmitted over port 0. The pointers for the other ports are obtained by (left) cyclic shifts of the table entries. A one-step cyclic rotation is used for port 1, two steps for port 2, etc. For n not a prime number there are also other cyclic nodes. The period P_c for each table entry needs to be found, and the message divided into P_c pieces for the SBG routing.

TABLE 2
Complexities for some communication algorithms based on the SBnT *routing.*

Comm.	One-to-all person. comm.	All-to-all broadcast	All-to-all person. comm.
one-port	$\leqq N(1+O(\log n/n))Mt_c+(2n-2)\tau$	$(N-1)Mt_c+(2n-1)\tau$	$(nNM/2)t_c+(2n-1)\tau$
n-port	$((N-1)M/n)t_c+n\tau$	$((N-1)M/n)t_c+n\tau$	$(NM/2)t_c+n\tau$

Internal nodes can either route according to the destination address, if it is included, or use tables. If the destination is included, then a node first checks if it is the destination. Otherwise, the output port is determined by finding *index*((*myaddress*)$\oplus$(*source*)) and then finding the first bit that is equal to 1 in ((*myaddress*)$\oplus$(*destination*)) to the left (cyclically) of the bit corresponding to the index. If tables are used instead of a destination field, then for *postorder* scheduling [5] it suffices that each internal node keeps a count for each port. Since the number of ports used in each subtree is at most $n-3$ and the number of nodes in the entire subtree is approximately N/n, a bound on the table size in each node is n^2 bits. A *reversed-breadth-first* scheduling [5] can be implemented by internal nodes keeping a table of how many nodes there are at a given level in each of its subtrees. The table has at most n^2 entries. An upper bound for the number of nodes in a subtree at any level is $N/n^{3/2}$, and the total table size in a node is at most n^3 bits. Hence, without a more sophisticated encoding the *postorder* scheduling discipline requires less table space. It is used for the measurements presented in Fig. 11.

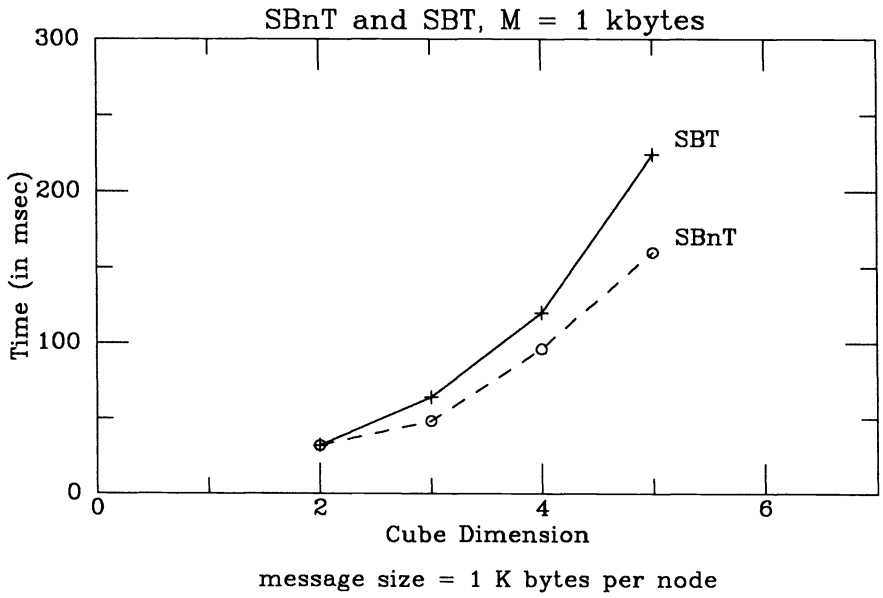


FIG. 11. *One-to-all personalized communication for the SBnT and SBT routings.*

With communication restricted to one port at a time, the expected time of *one-to-all personalized communication* for the SBT routing and SBnT routing is the same for $B = M$. For *one-to-all personalized communication* based on the SBT routing we schedule port communications in a *binary-reflected* Gray code order to take advantage of the partial overlap in communication on different ports that is possible on the Intel iPSC. The observed advantage of the SBnT over the SBT routing is due to the fact that the SBnT can take better advantage of the overlap. In the SBT case, even though messages were communicated over different ports in a binary-reflected Gray code order, the nodes adjacent to the root may not be finished with retransmitting the last packet received when a new packet arrives. In the SBnT, a subtree receives a packet once every n cycles.

7. Summary. The Spanning Balanced n-Tree (SBnT) allows for scheduling disciplines that realize minimum time (within a factor of two) one-to-all personalized communication, all-to-all broadcasting, and all-to-all personalized communication on a Boolean n-cube with *n-port* communication [5]. The number of nodes in each of the n subtrees is $O(N/n)$. The SBnT can be modified to a perfectly balanced graph by allowing multiple parents for cyclic nodes, i.e., splitting the data sets for such nodes. A few different definitions of *Spanning Balanced n-trees* are given, and shown to be essentially edge-disjoint. They are of particular interest with respect to fault-tolerant communications. Single edge failure, with the exception of the edges from the root, and several forms of multiple edge failures can be routed around, given that the failure is known and the proper SBnT is chosen.

We also show that for $0 < l < n$ the ratio of the number of degenerate necklaces to the total number of necklaces with precisely l bits equal to one is at most

$$\frac{1}{1+\max\left(\left\lceil\frac{n-l}{2}\right\rceil,\left\lceil\frac{l}{2}\right\rceil\right)} \leqq \frac{4}{4+n}.$$

REFERENCES

[1] C.-T. HO AND S. L. JOHNSSON, *Distributed routing algorithms for broadcasting and personalized communication in hypercubes*, in Proc. 1986 International Conference on Parallel Processing, pp. 640–648, IEEE Computer Society, 1986; Tech. Report YALEU/DCS/RR-483, Dept. of Computer Science, Yale Univ., New Haven, CT, May 1986.

[2] D. HOEY AND C. E. LEISERSON, *A layout for the shuffle-exchange network*, in Proc. 1980 IEEE International Conference on Parallel Processing, pp. 329–336.

[3] S. L. JOHNSSON, *Communication efficient basic linear algebra computations on hypercube architectures*, J. Parallel Distributed Comput., 4 (1987), pp. 133–172; Tech. Report YALEU/DCS/RR-361, Dept. of Computer Science, Yale Univ., New Haven, CT, January 1985.

[4] S. L. JOHNSSON AND C.-T. HO, *Matrix transposition on Boolean n-cube configured ensemble architectures*, SIAM J. Matrix. Anal. Appl., 9 (1988), pp. 419–454; Tech. Report YALE/DCS/RR-572, Dept. of Computer Science, Yale Univ., New Haven, CT, September 1987. (Revised edition of YALEU/DCS/RR-494, November 1986.).

[5] ———, *Spanning graphs for optimum broadcasting and personalized communication in hypercubes*, Tech. Report YALEU/DCS/RR-500, Dept. of Computer Science, Yale Univ., New Haven, CT, November 1986; revised November 1987, YALEU/DCS/RR-610; IEEE Trans. Computers, to appear.

[6] F. T. LEIGHTON, *Complexity Issues in* VLSI: *Optimal Layouts for the Shuffle-Exchange Graph and Other Networks*, MIT Press, Cambridge, MA, 1983.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 631-653, July 1989
© 1989 Society for Industrial and Applied Mathematics
002

SEQUENTIAL QUADRATIC PROGRAMMING METHODS BASED ON APPROXIMATING A PROJECTED HESSIAN MATRIX*

CHAYA BLEICH GURWITZ† AND MICHAEL L. OVERTON‡

Abstract. The nonlinear programming problem, namely, minimizing a nonlinear function subject to a set of nonlinear equality and inequality constraints, is considered. Sequential quadratic programming (SQP) methods are particularly effective for solving problems of this nature. It is assumed that first derivatives of the objective and constraint functions are available, but that second derivatives may be too expensive to compute. Instead, the methods typically update a suitable matrix that approximates second derivative information at each iteration. The authors are interested in developing SQP methods that maintain an approximation to second derivative information projected onto the tangent space of the constraints. The main motivation for this work is that only the projected matrix enters into the optimality conditions for the nonlinear problem. Updating projected second derivative information reduces the dimension of the matrix to be recurred; the necessity for introducing an augmenting term that can lead to ill-conditioned matrices is avoided; and one can make use of standard quasi-Newton updates that maintain hereditary positive definiteness. Four possible formulations of the quadratic programming subproblem are discussed, and numerical results indicating that these methods may be useful in practice are presented.

Key words. constrained optimization, nonlinear programming, projected Hessian, sequential quadratic programming, quasi-Newton

AMS(MOS) subject classifications. 65, 90

1. Introductory material.

1.1 Introduction. We consider the problem of minimizing a nonlinear function subject to a set of nonlinear constraints:

$$\text{(NP)} \qquad \begin{aligned} &\min_{x \in \mathbf{R}^n} f(x) \\ &\text{s.t.} \quad c_i(x) \geqq 0, \quad i = 1, \cdots, m_i, \\ &\qquad\; c_j(x) = 0, \quad j = m_i + 1, \cdots, m \end{aligned}$$

where the objective and constraint functions are twice continuously differentiable. We are concerned with developing algorithms to find a local minimum of (NP) by using second derivative information in the tangent space to the constraints.

Sequential quadratic programming methods are particularly effective for solving problems of this nature. Algorithms of this type compute the minimizer of (NP) by solving a sequence of quadratic programming subproblems. It is assumed that first derivatives of f and $\{c_i\}$ are available, but that second derivatives may be too expensive to compute. Instead, the methods typically update a suitable $n \times n$ matrix that approximates second derivative information at every iteration.

We are interested in developing sequential quadratic programming methods that maintain an approximation to second derivative information projected onto the tangent space of the constraints. The main motivation for our work is that only the projected

* Received by the editors January 15, 1987; accepted for publication (in revised form) October 10, 1988.

† Department of Computer and Information Science, Brooklyn College, City University of New York, Brooklyn, New York 11210. The work of this author was conducted at New York University, New York, New York 10012 with the support of the U.S. Office of Naval Research Graduate Fellowship Program.

‡ Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. The work of this author was supported in part by National Science Foundation grant DCR-85-02014.

matrix enters into the optimality conditions for the nonlinear problem. One advantage of our methods is that updating projected second derivative information reduces the dimension of the matrix to be recurred. Moreover, we avoid the necessity of introducing an augmenting term that can lead to ill-conditioned matrices. Since the reduced matrix of second derivatives is known to be at least positive semidefinite at the solution, we are able to make use of standard quasi-Newton updates.

Four possible formulations of the quadratic programming subproblem are presented. The difficulties arising from incomplete knowledge of the full second derivative matrix are discussed. Global convergence of some of these methods for a restricted class of problems can be shown. We present numerical results that indicate our algorithms may be useful in practice.

1.2 Notation. The methods that we will be studying will produce a sequence of points in $\mathbf{R}^n$, $x^1, x^2, \cdots$, usually converging to a point x^*, which is a local minimum of (NP). For convenience, we will assume that $x^k \neq x^*$ for all k. Quantities evaluated at iteration i will be denoted by a superscript of i. Rates of convergence refer to the Q-rate of convergence as defined in Ortega and Rheinboldt (1970). The identity matrix of order n will be denoted by I_n and e_i will be used to refer to its ith column.

The following notation will be used in reference to problem (NP): Let $g(x) \equiv \nabla f(x)$, the gradient of the objective function, $c(x) \equiv [c_1(x), \cdots, c_m(x)]^T$, $A(x) \equiv [\nabla c_1(x), \cdots, \nabla c_m(x)]^T$, the matrix whose rows consist of the transposed constraint gradients, and $G(x) \equiv \nabla^2 f(x)$, $G_i(x) \equiv \nabla^2 c_i(x)$, the Hessians of the objective and constraint functions, respectively. We will assume that the functions f and c_i are twice differentiable and that the Hessians of the objective and constraint functions are Lipschitz continuous matrix functions of x.

Let A be partitioned into $\begin{bmatrix} A_I \\ A_E \end{bmatrix}$, where A_I contains the transposed gradients of the inequality constraints and A_E denotes the transposed gradients of the equality constraints. Similarly, let c_I and c_E refer to the values of the inequality and equality constraints, respectively.

A point x is *feasible* for problem (NP) if the constraints are satisfied, i.e., $c_I \geqq 0$, $c_E = 0$. At a solution to (NP), some subset of the inequality constraints, as well as all of the equality constraints, are zero. These constraints, denoted by $\hat{c}(x^*)$ or $\hat{c}^*$, are called the *active constraints* at the solution. Any algorithm for solving (NP) is concerned with identifying the active set at the solution. Since the other inequality constraints have a positive value, they remain satisfied within some neighborhood of x^*. The problem then essentially reduces to one of minimizing the objective function with respect to the active constraints. The index set $J(x^k)$, known as the *working set*, is an approximation to the active set at the solution that is used by the algorithm at the point x^k. Any constraint that is in the working set is called *active*, although it may not actually be included in the active set at the solution. The constraints in the working set are referred to by $\hat{c}(x^k)$ or $\hat{c}^k$.

Let $\hat{A}(x)$ denote the matrix whose rows consist of the transposed gradients of the active constraints and assume that $\hat{A}(x)$ has full row rank. If we partition A_I into $\begin{bmatrix} \tilde{A}_I \\ \hat{A}_I \end{bmatrix}$, where $\tilde{A}_I$ refers to the inactive inequality constraint gradients and $\hat{A}_I$ denotes the gradients of the active inequality constraints, then $\hat{A} = \begin{bmatrix} \hat{A}_I \\ A_E \end{bmatrix}$. A similar partitioning of c_I allows us to express $\hat{c}$ as $(\hat{c}_I^T c_E^T)^T$.

Let $Z(x)$ be a matrix whose columns span the null space of $\hat{A}(x)$, and let $Y(x)$ be a matrix whose columns form a basis for the range space of $\hat{A}(x)^T$. Such matrices can be obtained, for example, by forming the LQ factorization that produces

$$\hat{A}(x)Q(x) = [L(x) \quad 0] \tag{1.1}$$

with Q orthogonal and L lower triangular. Let r be the number of active constraints and let $t = n - r$. The first r columns of $Q(x)$ can be used to define the columns of $Y(x)$, while the columns of $Z(x)$ can be formed from the last t columns of $Q(x)$.

The vector g_L and the matrix W will be used to denote the gradient and Hessian of the *Lagrangian function*, which will be introduced in the next section. The approximation to the Hessian of the Lagrangian function at x^k will be denoted by H^k. We will use B^k to refer to the projected Hessian approximation.

1.3. Optimality conditions. The following optimality conditions are well known and may be found, for example, in Fletcher (1981). Assume that $\hat{A}(x^*)$ has full rank. (A less restrictive assumption uses constraint qualifications; see Fletcher (1981).)

The first-order necessary conditions for a point x^* to be a local minimum of NP are the existence of a vector of Lagrange multipliers, λ^*, such that

$$g(x^*) = \hat{A}(x^*)^T \lambda^*, \tag{1.2a}$$

$$\lambda_i^* \geqq 0 \quad \text{for the active inequality constraints,} \tag{1.2b}$$

$$c_I(x^*) \geqq 0, \qquad c_E(x^*) = 0. \tag{1.2c}$$

A pair (x^*, λ^*) satisfying (1.2) is called a *Kuhn-Tucker pair* of (NP). Condition (1.2a) is equivalent to requiring x^* to be a stationary point of the Lagrangian function

$$L(x, \lambda) = f(x) - \sum \lambda_i \hat{c}_i(x) \tag{1.3}$$

when $\lambda = \lambda^*$.

Second-order optimality conditions are expressed in terms of the Hessian with respect to x of the Lagrangian function,

$$W(x, \lambda) = \nabla^2 f(x) - \sum \lambda_i \nabla^2 \hat{c}_i(x).$$

A necessary condition for optimality is that the projected Hessian, $Z(x^*)^T W(x^*, \lambda^*) Z(x^*)$, is positive semidefinite. Strict positive definiteness of the projected Hessian, together with strict inequality in (1.2b), and the satisfaction of the first-order conditions, is a second-order sufficient condition. In addition to the assumption that $\hat{A}$ has full rank (stated earlier), we shall assume throughout that the second-order sufficient condition holds at x^*.

2. Description of the algorithms. Algorithms based on *sequential quadratic programming* (SQP) have been found particularly effective for solving problem (NP). Methods of this type approximate the minimizer of (NP) by solving a sequence of quadratic programming (QP) subproblems. The solution of the QP is used to provide a search direction p^k. The algorithm then determines a steplength α^k, such that $x^k + \alpha^k p^k$ is a sufficiently better approximation to x^*, in a sense to be defined later.

The QP subproblem is defined by a quadratic objective function and a set of linear constraints. The quadratic objective function is

$$g^T p + \tfrac{1}{2} p^T H p.$$

Here g is the gradient of f at the current point, and H is some approximation to the Hessian of the Lagrangian function. The QP subproblem constraints are linearizations of some, or all, of the nonlinear constraints of NP. There are several possible choices of constraints to include in the QP, which we shall discuss shortly.

Since the Lagrangian function is defined in terms of both x and λ, these methods must incorporate some means of approximating the Lagrange multipliers. At the

solution, the Lagrange multipliers are defined by (1.2a), where $\hat{c}(x^*) = 0$. This suggests that a first-order estimate λ_L^k can be found at x^k by solving the least squares problem

$$\min_{\lambda_L^k} \|\hat{A}^{k^T}\lambda_L^k - g^k\|_2^2. \tag{2.1}$$

A second-order multiplier estimate η^k is given by the multipliers defined by the solution p of the QP subproblem

$$\hat{A}^{k^T}\eta^k = g^k + Hp \tag{2.2}$$

where $\hat{A}$ is defined by the working set at the solution to the QP. These multiplier estimates are referred to as the *QP multipliers.*

Different formulations of the quadratic programming subproblem arise depending on how the QP constraints are chosen. At one extreme the subproblem is posed as an IQP—an *inequality constrained quadratic program*—and all of the equality and inequality constraints of (NP) are included in the subproblem. At the other extreme the constraints of the subproblem are only those in the current working set that are believed to hold with equality. In that case, the subproblem includes only equality constraints. Such subproblems are known as EQPs—*equality constrained quadratic program.* Although the terms EQP and IQP refer to the formulation of the QP subproblem, we also use these terms in a looser sense to mean SQP methods that solve successive QP subproblems of either type.

We are interested in developing algorithms that maintain an approximation to the projected Hessian Z^TWZ rather than to the full second derivative matrix. The main motivation for this approach is that only the projected matrix enters into the optimality conditions for (NP). The idea of recurring an approximation to the projected Hessian rather than to the full second-derivative matrix was suggested by Murray and Wright (1978). Some particular algorithms and their convergence analysis were given for the case of nonlinear equality constraints by Coleman and Conn (1984) and Nocedal and Overton (1985).

We note that the equality constrained case is much simpler than the general case, since the active set, which determines the columns of Z, is known. In the inequality constrained case, as the approximation to the active set is revised, the dimension of Z^TWZ is altered. Our proposed methods are designed to extend the projected Hessian updating algorithms of Nocedal and Overton to the general inequality constrained problem. In the following, we present four different possible formulations of the QP subproblem. The different formulations arise from different choices of the QP subproblem constraints and from considering various ways of retrieving second derivative information for the n-dimensional space from the projected Hessian matrix.

2.1. The EQP subproblem. The EQP subproblem includes only equality constraints, namely those in the current working set. These constraints include all of the equality constraints of the original nonlinear problem, as well as the currently active inequality constraints.

The EQP subproblem for (NP) is therefore

$$\begin{aligned} &\min_{p \in \mathbf{R}^n} \tfrac{1}{2}p^THp + p^Tg, \\ &\text{s.t.} \quad \hat{A}_I^T p = -\hat{c}_I, \\ &\qquad\quad A_E^T p = -c_E. \end{aligned} \tag{2.3}$$

The advantage of posing the subproblem as an EQP is that the solution can be easily computed by solving two systems of linear equations. The solution p^* can be written as

$$p^* = Yp_y^* + Zp_z^*$$

where Y and Z are defined as in (1.1). The vectors p_y^* and p_z^* can be found by solving

$$Lp_y^* = -\hat{c}, \tag{2.4a}$$

$$Z^T H Z p_z^* = -Z^T g - Z^T H Y p_y^* \tag{2.4b}$$

where L is given by (1.1). The solutions are unique when $\hat{A}$ has full row rank and $Z^T HZ$ is positive definite. This method of computing p^* is discussed by Murray and Wright (1982). The solution of the EQP subproblem can also be viewed as the result of applying one step of Newton's method to the nonlinear system of equations

$$\begin{bmatrix} g(x) - \hat{A}(x)^T \lambda \\ \hat{c}(x) \end{bmatrix} = 0 \tag{2.5}$$

(see Nocedal and Overton (1985)).

In our case, since we do not recur the full matrix H, we do not have any value for the $Z^T HY$ term used in (2.4b). However, near the solution we expect the constraints to be satisfied, which would cause p_y to be near zero. Therefore, following Nocedal and Overton (1985), we solve the EQP by replacing $Z^T HZ$ by B, which is an approximation to the projected Hessian, and simply ignoring the $Z^T HY$ term. The solution of our EQP is thus given by

$$Lp_y^* = -\hat{c}, \tag{2.6a}$$

$$Bp_z^* = -Z^T g, \tag{2.6b}$$

$$p^* = Yp_y^* + Zp_z^*. \tag{2.6c}$$

EQP methods are attractive because the formulation of the subproblem parallels Newton's method applied to an unconstrained minimization problem. Moreover, when the correct working set is identified, $Z(x^*)^T W(x^*, \lambda^*) Z(x^*)$ must be at least positive semidefinite. In that case one can use quasi-Newton updating methods to update B. However, the difficulties of determining the correct working set should not be minimized. Algorithms that update the working set at each iteration can suffer from repeated insertion and deletion of the same constraint, known as *zigzagging* (see Wolfe (1972)).

2.2. The IQP subproblem. The IQP formulation of the subproblem for (NP) is given by

$$\begin{aligned} &\min_{p \in \mathbf{R}^n} \tfrac{1}{2} p^T H p + p^T g, \\ &\text{s.t.} \quad \tilde{A}_I^T p \geqq -\tilde{c}_I, \\ &\qquad \hat{A}_I^T p \geqq -\hat{c}_I, \\ &\qquad A_E^T p = -c_E. \end{aligned} \tag{2.7}$$

Note that all of the original inequality constraints are included in the IQP subproblem, even if they are not in the current working set.

In general, the computation of the solution to an IQP is itself a finite iterative process. Essentially, the IQP generates a sequence of working sets and solves the EQP associated with each working set. Therefore, SQP methods which use IQP subproblems involve *major* and *minor* iterations. The *major* iterations refer to the "outer" iterations, and the *minor* iterations are those within the quadratic programming subproblem.

IQP-based methods generally assume that an initial feasible point is available for the subproblem. If necessary, an initial feasible point can be found by a phase-1 procedure. At the current point, some subset of the constraints are active. The method then generates a search direction by solving the EQP defined by minimizing the QP objective function while requiring that the constraints in the working set remain satisfied with equality. However, a full step along this search direction may cause one of the currently inactive constraints to become violated. If such is the case, a step is taken to the closest such constraint, the newly active constraint is added to the working set, and the process is repeated.

Eventually, a point is reached that is the minimum on the subspace defined by the current working set. To determine whether this point is optimal for the original QP (2.7), the QP Lagrange multiplier estimates are checked. If some multiplier is negative, that is an indication that the QP objective function can be further decreased by deleting the corresponding constraint from the working set.

This process of revising the working set and minimizing along the subspace defined by the working set is continued until a point is reached at which the projected gradient is zero, and the Lagrange multipliers are all nonnegative. Thus, each minor iteration involves the solution of an EQP where the defining constraints change from one iteration to the next. Sometimes a constraint is added to, sometimes deleted from, the working set.

Methods based on solving IQP subproblems are not dependent on a strategy for updating the working set during the major iterations. Such algorithms allow for fast recovery from a bad initial approximation to the active set. IQP methods typically involve approximating the full matrix W, since the constraints defining Z^TWZ change within the subproblem. However, there is no a priori reason to assume that W is positive definite. Moreover, when W is indefinite, the solution may not be bounded, and the direction produced by the QP may not be a descent direction. Therefore, many methods that approximate the full second derivative matrix of the Lagrangian function add a term of the form $\rho\hat{A}^T\hat{A}$ to the Hessian matrix so as to force W to be positive definite. This augmenting term can lead to very ill-conditioned matrices, although in practice this does not appear to impede the performance of such methods (Gill et al. (1984)).

A discussion of the relative merits of EQP and IQP methods can be found in Murray and Wright (1982) and Gill et al. (1985c). In the neighborhood of a solution, one would expect similar performance from both the IQP and EQP methods. Close to x^* it is expected that the working set would correspond to the correct active set. Therefore, the solution to an IQP subproblem would be found by taking one step along the subspace defined by the current working set, which is equivalent to the solution of the EQP defined by those constraints.

We now consider how to define the Hessian approximation matrix H, given the fact that second derivative information is available only on the tangent space to the constraints. The full matrix H is needed by the subproblem both to compute the gradient of the QP objective function during the solution of the QP and to compute the QP multiplier estimates (1.5).

The simplest solution approximates H by "unprojecting" the projected Hessian, i.e., by defining

$$H = Q\begin{bmatrix} B & 0 \\ 0 & D \end{bmatrix}Q^T \tag{2.8}$$

where B is an approximation to the projected Hessian, Z^TWZ, Q is defined by (1.1),

and D is a positive diagonal matrix, such as the identity matrix, of the appropriate dimension. The matrix D is used to fill out the matrix in order to ensure that the "unprojected" matrix remains positive definite. If some knowledge of the scaling of the problem is known, that information can be used in forming D. The motivation here is to be able to use the smaller matrix B in place of the n-dimensional matrix H. We have

$$H = [Z \quad Y]\begin{bmatrix} B & 0 \\ 0 & D \end{bmatrix}\begin{bmatrix} Z^T \\ Y^T \end{bmatrix} = ZBZ^T + YDY^T$$

and

$$Z^THZ = Z^T(ZBZ^T + YDY^T) = B$$

so that the projection of H onto the reduced space is just B.

This method is attractive because convergence results known for IQP methods can be applied directly. In addition, as is the case in any IQP-based method, the determination of the working set is done automatically by the subproblem. However, since the matrix is not accurate on the space spanned by the constraint gradients, the search direction produced by the IQP may not be meaningful when far from the solution.

There are two basic problems associated with this formulation of the subproblem. First, the revision of the working set within the IQP subproblem is predicated on multiplier estimates that may not be reliable. Second, when a constraint is deleted from the current working set, the dimension of Z and hence of B is increased by one. When the full Hessian is being recurred, an estimation of the new column and row of Z^THZ can be obtained by expanding the matrix Z and multiplying the Hessian onto the new column of Z. However, when only reduced Hessian information is available, it is not clear how to expand the projected Hessian. In general, we simply set the new column to some multiple of the corresponding column of the identity matrix.

Therefore, we consider two additional approaches in which the subproblems posed are midway between EQP and IQP methods. In the following sections we describe these formulations of the subproblem.

2.3. The PQP subproblem. In the first method, we partially solve the IQP by allowing the subproblem to add constraints, but terminate the iterative process as soon as a constraint is to be dropped. We call this method a PQP—*partial* QP—since we are *partially* solving the IQP. By allowing the PQP to partially solve the full IQP, we allow many constraints to be picked up in the course of one major iteration. However, since the PQP cannot delete any constraints, we avoid the problems of expanding the projected Hessian approximation and of relying upon multiplier estimates of dubious accuracy. In this manner, we hope to retain some of the advantages of the IQP method, while reducing the possible ill effects of solving the full IQP when the full Hessian is not being approximated.

The PQP is implemented by following the steps of the IQP until a point is reached where the IQP would delete a constraint. In solving the IQP subproblem, a sequence of minor iterations is performed. At each iteration an EQP subproblem is solved. At the end of each minor iteration, if the current projected gradient is zero, that is an indication that the QP objective function cannot be further decreased while remaining on all of the constraints in the current working set. At this stage, in solving the IQP, a constraint could be dropped from the working set and the process would be repeated with the diminished working set. It is at this point that the PQP method differs from the IQP. The PQP would simply stop after having reached a point where the projected

gradient is zero. Since the PQP is never allowed to delete a constraint from within the QP subproblem, this method entails updating the active set approximation in the course of the outer iteration.

2.4. The MQP subproblem. In the second method we allow only those constraints that have been added during the current QP to be deleted. Thus if a constraint is to be deleted, second derivative information on the appropriate subspace can be recovered from the projected Hessian at the start of the QP. This formulation of the subproblem will be called an MQP—*modified* QP. We restrict the QP by not allowing it to drop any constraints that were active at the start of the QP. The QP may add constraints, and may even delete constraints that have been added within the QP, but may not move off the constraints that were active previously. In this manner, we hope to minimize the effects of dropping constraints based on inaccurate multiplier estimates and of expanding the projected Hessian approximation with inaccurate second derivative information.

Constraining the QP to remain on the original active set is accomplished by treating all the initially active constraints as equalities. Since there is no restriction on the sign of the multiplier associated with an equality constraint, these constraints will not be deleted.

The MQP subproblem is therefore

$$\begin{aligned} &\min_{p\in R^n} \frac{1}{2} p^T Q \begin{bmatrix} B & 0 \\ 0 & D \end{bmatrix} Q^T p + p^T g, \\ &\text{s.t.} \quad \hat{A}^T p = -\hat{c}, \\ &\qquad \tilde{A}_I^T p \geqq -\tilde{c}_I. \end{aligned} \tag{2.9}$$

We note that it is indeed possible for the QP to add a constraint and subsequently delete it. This scenario occurs when there is another constraint between the original constraint and the solution. In that case, the second constraint may override the first. (See Fig. 2.1.)

In the neighborhood of a solution, we would expect the correct active set to be identified. In that case, an IQP method would find the solution after solving one EQP subproblem. Since the IQP method would not involve deleting any constraints, the PQP and MQP methods would both behave in exactly the same fashion as the IQP. Thus, near the solution all four formulations usually reduce to an EQP method.

3. Implementation issues. In order to concentrate on the issues involved in restricting the second derivative approximations to the reduced space without concerning ourselves with other implementation details, we have developed our algorithms by modifying NPSOL, an SQP software package developed at the Systems Optimization Laboratory at Stanford University (see Gill et al. (1983a), (1983b)). We have endeavored to deviate from the original routines as little as possible. Since our algorithms correspond to NPSOL, aside from the differences induced by the different approximations to the Hessian, we are able to compare the effect of the reduced approximation upon the performance of SQP algorithms.

Detailed descriptions of the NPSOL algorithm and of our methods can be found in Gurwitz (1986). In the following sections we will discuss some important aspects of the implementation of these methods.

3.1. Multiplier estimates. Lagrange multiplier estimates are used to update the Hessian approximation, to determine whether constraints should be deleted from the active set, and as parameters of the augmented Lagrangian penalty function.

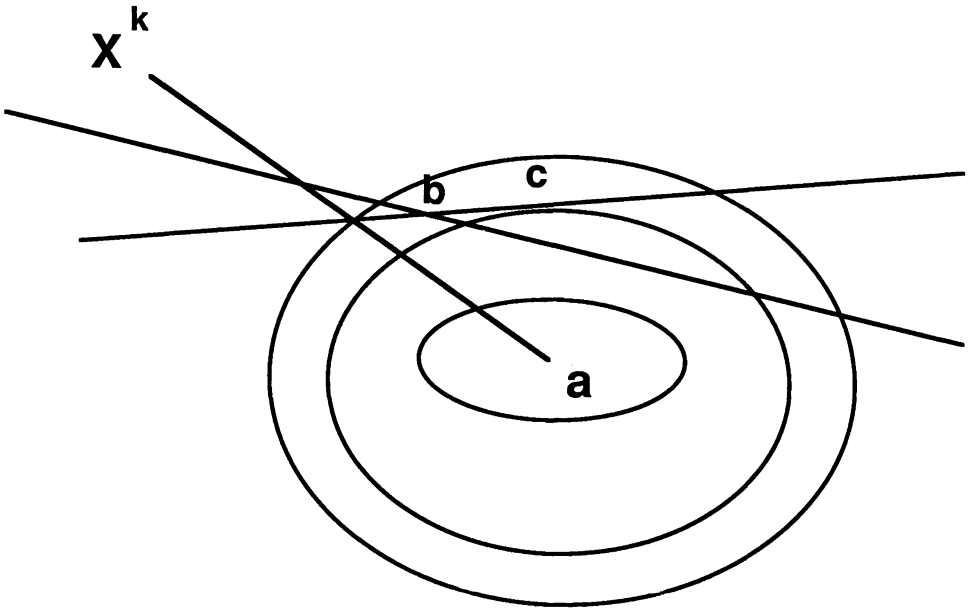


FIG. 2.1. *The difference between the directions produced by the* MQP *and* PQP *methods. The contours of* f_{QP} *are shown, along with two linear constraints. The unconstrained solution would be at a. The solution to the* PQP *is found at the intersection of the two constraints, b. For the* MQP *method, the algorithm can move off the first constraint and find a better solution at c.*

Since the definition of the objective function of the subproblem depends on the second derivative matrix, convergence of SQP methods depends critically on convergence of the multiplier estimates.

A first-order estimate λ_L^k at x^k can be found by solving the least squares problem (2.1). This system is compatible at any point where the gradient is a linear combination of the rows of $\hat{A}^k$. Therefore, even when the residual is zero, λ_L^k is only an estimate of the multipliers defined by the current active set, unless $\hat{c}(x^k) = 0$. A second-order multiplier estimate η^k is given by the multipliers defined by the solution p of the QP subproblem, as in (2.2).

A second-order multiplier estimate at x^k is a prediction of the first-order multipliers at the new point. It should be emphasized that η^k depends critically on p^k and for a poor choice of p^k can be completely inaccurate. Moreover, when the projected Hessian is recurred, second derivative information is not available for the entire space, and thus the second-order multiplier estimate may not be reliable. Wright (1976) shows that using the least squares multiplier estimates at the new point, x_{k+1}, for updating the Hessian approximation cannot impede the fast rate of convergence of the underlying method. In fact, Goodman (1985) shows that such an approach is indeed a Newton method (in the x variables) and is quadratically convergent. Therefore, we use first-order multipliers at x^{k+1}, not second-order estimates at x^k, to update the Hessian approximation.

3.2. Merit functions. An essential element of any algorithm for solving optimization problems is a technique for forcing convergence from a poor starting point. This is generally accomplished by viewing the step to the new point as a search direction and choosing a steplength along that path in a manner such that the new point is a "sufficiently better" approximation to the solution.

Let m denote some merit function. In the unconstrained case, the "acceptable" reduction in f can be enforced by requiring the steplength α^k to satisfy the *Goldstein–Armijo conditions*

$$f(x^k+\alpha^k p^k) \leqq m(x^k)+\alpha^k \gamma \nabla f(x^k)^T p^k, \qquad \gamma \in (0,1), \tag{3.1a}$$

$$\nabla f(x^k+\alpha^k p^k)^T p^k \geqq \delta \nabla f(x^k)^T p^k, \qquad \delta \in (\gamma,1). \tag{3.1b}$$

When nonlinear constraints are present, it is not straightforward to produce a sequence of points that remain feasible. The line search routine must therefore balance the reduction of the objective function against the satisfaction of the constraints. Various *merit functions* have been proposed for measuring the quality of a steplength.

A nondifferentiable, but well-conditioned, merit function is the *exact penalty function* (used by Han (1977))

$$P_E(x,\rho) \equiv f(x)+\rho \|c_v(x)\|_1$$

where c_v denotes the constraint violations. There exists a finite value ρ^* such that for any $\rho > \rho^*$, x^* will be a local minimum of P_E. However, the correct value depends on values computed at x^*, and thus is not known a priori. The algorithm, therefore, may suffer from bad choices of the penalty parameter. If ρ is chosen too small, P_E may be unbounded below, or the region in which x^* is a minimum may be very small. For large values of ρ the penalty function becomes ill conditioned.

The optimality conditions require x^* to be a minimum of the Lagrangian function on the tangent space to the active constraints at $\lambda = \lambda^*$. It is possible, therefore, to augment the Lagrangian function by adding a term that alters the function in the range space of $\hat{A}(x^*)^T$ so as to cause x^* to be a local minimum. Such an *augmented Lagrangian penalty function* (used by Fletcher (1981), Schittkowski (1981)) is given by

$$P_A(x,\lambda,\rho) \equiv f(x)-\lambda^T c_v(x)+\frac{\rho}{2} c_v(x)^T c_v(x).$$

As is the case for the exact penalty function, there exists some threshold value $\bar{\rho}$ such that x^* will be a minimum of P_A for all $\rho > \bar{\rho}$. In addition, such merit functions have the advantage of being continuously differentiable. It should be emphasized that x^* is a minimizer of the augmented Lagrangian only when $\lambda = \lambda^*$. Methods incorporating an augmented Lagrangian merit function are thus particularly sensitive to poor multiplier estimates.

The exact penalty merit function has the advantage of not being overly sensitive to multiplier estimates. Global convergence can generally be proved for methods that reduce an exact penalty function at each iteration. However, insisting on a decrease in an exact penalty function can result in steplengths of unity being rejected, even in a very small neighborhood of the solution. This phenomenon, known as the "Maratos effect," prevents some algorithms from converging superlinearly (Mayne (1979), Chamberlain et al. (1982)). Chamberlain et al. (1982) and Fletcher (1982), (1985) discuss techniques for avoiding the Maratos effect.

Global convergence of methods using an augmented Lagrangian penalty function involves proving convergence of the multipliers. Therefore, Schittkowski (1982) presents a globally convergent modification of the augmented Lagrangian merit function in which the line search is performed over x and λ, rather than over x alone. The merit function used in NPSOL (Gill et al. (1985c)), based on that of Schittkowski, is a smooth augmented Lagrangian function of the form

$$P_N(x,\lambda,s,\rho) = f(x)-\sum_{i=1}^{m} \lambda_i(c_i-s_i)+\frac{1}{2}\sum_{i=1}^{m} \rho_i(c_i-s_i)^2$$

where s are *slack variables.* The line search evaluates $P_N(x^k + \alpha p^k, \lambda^k + \alpha \delta\lambda^k, s, \rho)$ to produce

$$\begin{pmatrix} x^{k+1} \\ \lambda^{k+1} \\ s^{k+1} \end{pmatrix} = \begin{pmatrix} x^k \\ \lambda^k \\ s^k \end{pmatrix} + \alpha \begin{pmatrix} p^k \\ \delta\lambda^k \\ \delta s^k \end{pmatrix}.$$

The quantity $\delta\lambda^k$ is defined by $\delta\lambda = \eta^k - \lambda^k$, where η^k are the QP multipliers. Hence, when a step of length one is taken, the QP multiplier estimates are accepted. In a neighborhood of the solution, a steplength of unity will, in fact, produce an acceptable point. Such a merit function thus enforces global convergence, while maintaining fast local convergence. It should be emphasized, however, that this line search depends critically on the QP multipliers. It is expected that whenever a unit step is taken, the QP multipliers provide a good estimate of the Lagrange multipliers at the new point.

Since our algorithms were developed by modifying the NPSOL package, and we have tried to deviate from the original code as little as possible, we prefer to be able to use the NPSOL merit function P_N. As mentioned above, use of this merit function avoids the "Maratos effect" and enforces global convergence for the original NPSOL algorithm. We would have liked to show that the global convergence associated with the use of the merit function P_N can be extended to our modifications of the original method. However, as noted above, this merit function depends critically on the QP multipliers. Since our non-IQP methods do not solve a full IQP subproblem, the QP multipliers associated with a solution of one of our subproblems are not reliable estimates of the Lagrange multipliers of the original problem. In fact, since the non-IQP subproblems do not allow for deleting constraints that were in the working set at the start of the QP (the PQP and EQP methods cannot delete any constraints at all, while the MQP can delete only those constraints that have been added within the current QP), the QP multipliers may have the wrong sign. In that case, obviously it is not logical to use a merit function that would set the new multiplier estimate, λ^{k+1} to $\lambda^k + \alpha^k(\eta^k - \lambda^k)$. For our IQP method, the QP multipliers will all be of the correct sign. However, since the multipliers are defined by the full Hessian matrix (see (2.2)), the QP multipliers associated with our IQP subproblem, in which the full Hessian is approximated by "unprojecting" the projected Hessian approximating matrix as in (2.8), may be poor estimates of the Lagrange multipliers of the original problem.

We have considered various ways of adapting the NPSOL merit function for use with our modified methods. One possibility is to use the least squares multiplier estimates at $x^k + p^k$ in place of η^k. These multiplier estimates may be better approximations to the Lagrange multipliers at x^{k+1} (see § 3.1). However, there is no restriction on the signs of the least squares multipliers and therefore they, too, may have the wrong sign. Moreover, far from the solution, it may be necessary to take a small step along p^k, and the least squares estimates at the full step $x^k + p^k$ may be totally unreliable.

As a means of avoiding the use of a multiplier estimate with the wrong sign, we consider simply setting $\delta\lambda^k$ to zero whenever η^k has the wrong sign. Although this keeps us from revising the multiplier estimates in the wrong direction, if we consistently set $\delta\lambda^k$ to zero, we will not be modifying the multiplier estimates.

Near the solution we expect that the working set will be an accurate prediction of the active set J^*. In that case, all of our methods find the solution to the subproblem in one step, the QP multiplier estimates are all of the correct sign, and the NPSOL merit function generally performs very well. In fact, for many problems, the working set becomes a good prediction of the true active set J^* even far from the solution, and the NPSOL merit function can be used.

In implementing our algorithms, we allow the user the option of choosing the merit function. The first option is to use the NPSOL merit function and set $\delta\lambda_j^k$ to zero whenever η_j^k has the wrong sign; the second option is to use an exact penalty function, and the third option is to use the objective function itself as the merit function. Examples of cases for which the objective function itself can be used as the merit function are discussed in § 4.

3.3. Quasi-Newton updates. Given B^k, a positive definite approximation to $Z^{k^T}W^kZ^k$, we wish to apply a standard updating method such as DFP or BFGS to obtain B^{k+1}. Empirical results and the analysis of Powell (1986) and Byrd, Nocedal, and Yuan (1987) indicate that the BFGS update is preferable. For the unconstrained case, the BFGS update is defined by

$$B^{k+1} = B^k + \frac{y^k y^{k^T}}{y^{k^T}s^k} - \frac{B^k s^k s^{k^T} B^k}{s^{k^T} B^k s^k} \tag{3.2}$$

where

$$s^k = x^{k+1} - x^k, \qquad y^k = g^{k+1} - g^k.$$

When the updates are to be performed in the projected space, alternative definitions must be used for s^k and y^k. Nocedal and Overton (1985), in discussing the equality-constrained problem, suggest the following choices:

(a) $s^k = Z^{k+1^T}(x^{k+1} - x^k)$, or

(b) $s^k = Z^{k^T}(x^{k+1} - x^k)$

(c) $y^k = Z^{k+1^T}(g^{k+1} - (g^k - \hat{A}^{k^T}\lambda_L^{k+1}))$, or

(d) $y^k = Z^{k+1^T}(g^{k+1} - (g^k - \hat{A}^{k^T}\lambda_L^k))$, or

(e) $y^k = Z^{k^T}(g^{k+1} - \hat{A}^{k+1^T}\lambda_L^{k+1} - g^k)$, or

(f) $y^k = Z^{k+1^T}g^{k+1} - Z^{k^T}g^k$.

The definitions for y^k are derived from considering the change in the gradient of the Lagrangian function (1.3). For example, (c) is equivalent to

$$y^k = Z^{k+1^T}(\nabla_x L(x^{k+1}, \lambda_L^{k+1}) - \nabla_x L(x^k, \lambda_L^{k+1})).$$

For the inequality constrained problem, the number of active constraints may change from one iteration to the next. Thus Z^k and Z^{k+1} and hence B^k and B^{k+1} may not be of the same dimension. In order to extend these definitions of s^k and y^k to the more general problem, we replace Z^k in (a)-(f) by Z_F^k, which is defined as a basis for the null space of $\hat{A}_F^k$, i.e., the matrix whose rows consist of the gradients of the constraints that are active at x^{k+1}, but evaluated at x^k. If more constraints are active at x^{k+1} than at x^k, the transformations applied to update the LQ factorization are applied to the Cholesky factor of B^k as well. When fewer constraints are active at x^{k+1}, the approximation to the projected Hessian matrix must be expanded by adding a row and a column. When second derivative information is available for the full space, the new column can be computed by projecting this information into the new space. For our purposes, we simply use a positive multiple of the identity matrix in place of the full Hessian.

Unlike unconstrained optimization, there is no guarantee that $s^{k^T}y^k > 0$. Therefore, Womersley (1984), (1985), Womersley and Fletcher (1986) use a modification of the BFGS method proposed by Powell (1978a) that guarantees hereditary positive definiteness. Powell's modification consists of replacing y^k in (3.2) by the vector

$$r^k = \theta y^k + (1-\theta)B^k s^k,$$

where θ is defined by

$$\theta = \begin{cases} 1, & s^{k^T} y^k \geqq 0.2 s^{k^T} B^k s^k, \\ \dfrac{0.8 s^{k^T} B^k s^k}{s^{k^T} B^k s^k - s^{k^T} y^k}, & s^{k^T} y^k < 0.2 s^{k^T} B^k s^k. \end{cases}$$

This modification to the update prevents the determinant of B^{k+1} from being less than 0.2 times the determinant of B^k. (The constant 0.2 was chosen as a reasonable size, but not because of any theoretical properties.)

However, using the above definitions of s^k and y^k, it is possible to construct examples where $\|s^k\|$ is small while $\|y^k\|$ is not. The Powell update does not avoid this problem, since it may be the case that $s^{k^T} y^k \gg s^{k^T} B^k s^k$. In this event, the updates may be undefined or may cause the Hessian approximation to blow up. Coleman and Conn (1984) circumvent this problem by introducing an intermediate point, $\hat{x}^k \equiv x^k + Z^k p_z$, and defining

$$s^k = Z^{k^T}(\hat{x}^k - x^k), \qquad y^k = Z^{k^T}(\nabla_x L(\hat{x}^k, \lambda_L^k) - \nabla_x L(x^k, \lambda_L^k)).$$

This procedure ensures that $\|s^k\|$ cannot be too small compared to $\|y^k\|$, but requires an additional set of gradient evaluations at each iteration.

Nocedal and Overton suggest an alternative strategy. If $\|s^k\|$ is too small, they simply do not perform the update. Let

$$v^k = Y^{k+1^T}(x^{k+1} - x^k) \quad \text{or} \quad v^k = Y^{k^T}(x^{k+1} - x^k)$$

corresponding to whether (a) or (b) is used to define s^k. The update is done if and only if $\|v^k\| < (\omega/(k+1)^{1+\nu})\|s^k\|$, where ω and ν are appropriately chosen constants. Under certain conditions this update rule will not impede local two-step superlinear convergence.

In practice, we have found that it is generally preferable not to skip updates. The update rule of Nocedal and Overton may be too restrictive and may cause the algorithm to skip updates even in a situation in which updating the projected Hessian would not cause any difficulty. Ideally, we would like to use some update rule that would guarantee that the eigenvalues of the projected Hessian approximating matrix lie within given bounds. Calamai and Moré (1987) have recently developed a quasi-Newton method for solving systems of nonlinear equations, based on the Broyden update, that preserves bounds on the elements of the Jacobian matrix. However, it is not clear how to extend this method to preserve positive definiteness nor how to apply it in the projected space.

Instead, we consider another simple updating rule based on an idea of Sargent (1986). We use the Powell modification of the BFGS update if this preserves some bounds on estimates of the eigenvalues of the projected Hessian matrix. Since we recur the Cholesky factor of the projected Hessian matrix, R^k, we use the values $r_{\max}^2$, $r_{\min}^2$, and $(r_{\max}/r_{\min})^2$ as estimates of the maximum and minimum eigenvalues and the condition number of the projected Hessian, respectively, where $r_{\max} = \max_i \|R_{i,i}^k\|$ and $r_{\min} = \min_i \|R_{i,i}^k\|$. (Of course, in the worst case, these estimates may not be very good.) If the estimates of the eigenvalues of the new projected Hessian approximation do not lie within the set bounds, we do not update. This update rule is less restrictive than the Nocedal and Overton rule and yet will not allow the approximating matrix to blow up.

Nocedal and Overton discuss an example for which $\|s^k\| \ll \|y^k\|$ and performing the update would cause the norm of the reduced Hessian approximation to become very large, while use of their updating rule would avoid this difficulty. We note that our updating rule would provide the same results. Another example presented by Nocedal and Overton demonstrates the sensitivity of their update rule to the proper

choice of ω. An incorrect choice of ω can allow the update to be performed and cause the projected Hessian approximation to become nearly singular. The advantage of our updating rule is that any such update that would result in a near-singular matrix would almost always be skipped.

3.4. Dropping criteria. Our EQP and PQP methods do not allow any constraints to be deleted from the working set during the course of solving the QP subproblem. Similarly, the MQP method cannot delete a constraint that had been active before the start of the QP. Therefore, these methods are dependent on a strategy for updating the working set during the major iterations.

Many criteria for deciding when to drop a constraint from the working set have been studied. On one extreme, exact minimization on the current subspace can be required before a constraint is dropped. On the other extreme, some criteria call for deleting a constraint whenever its associated Lagrange multiplier estimate is negative. In general, requiring subspace minimization tends to cause the algorithm to take a greater number of iterations, since minimizing on the wrong subspace does not necessarily produce much progress toward the solution. However, an algorithm that may drop constraints at every step may suffer from zigzagging. An example due to Wolfe (1972) shows that such a method may converge to a nonoptimal point.

When there is more than one constraint with a multiplier estimate of the wrong sign, it is usual to delete the constraint corresponding to the largest multiplier estimate in absolute value with an incorrect sign (see Fletcher (1981)). However, we have found that where the constraints are nonlinear, this may not be the best choice. A nonlinear constraint may be included in the working set and yet may not be exactly satisfied. In fact, the linearized constraint may be exactly satisfied while the current iterate is actually infeasible with respect to the original nonlinear constraint.

Another related issue is how much to modify the working set at each iteration. Allowing only one constraint to be deleted at each iteration may cause the method to take many steps until the working set is correctly determined; dropping many constraints too early may cause the method to take extra steps in order to pick them up again. In practice, we have found that constraints can be deleted from the working set until the gradient projected onto the new space is no longer small.

The user may control how far to minimize on the current subspace before allowing a delete operation by setting the tolerance level. If desired, the user may allow a default tolerance level to be used, which is computed by the program and is dependent on the scaling of the problem. If the projected gradient is less than the tolerance level, the method may delete one or more constraints until the gradient projected onto the larger space is above the tolerance level. The constraints to be deleted are chosen to correspond to the largest least squares multiplier estimates in absolute value with an incorrect sign, chosen from among those constraints that are satisfied.

Because of the difficulties involved in choosing an appropriate merit function (see § 3.2), it may be the case that in order to find an acceptable point for the merit function along the search direction p^k, the steplength must be extremely small. If this occurs, unless the working set is modified, the new iterate, x^{k+1}, will be very close to x^k, and p^{k+1} will be close to p^k, and the method will keep taking extremely small steps until it moves far enough away from x^k. Therefore, if we find that a very small step is taken, we immediately try to modify the working set by deleting a constraint, regardless of the size of the projected gradient.

3.5. Theoretical results. Our methods can be shown to be globally convergent for a restricted class of problems. Our IQP method combined with a suitable merit function

is globally convergent for problems in which the constraint functions are concave. This method is also globally convergent for minimax problems, which will be discussed in the next section. An active set strategy, presented by Byrd and Shultz (1982), can be used to guarantee nonzigzagging and global convergence for our EQP, PQP, and MQP methods on linearly constrained problems. Details of these convergence results can be found in Gurwitz (1986).

4. Test problems and results. The results were obtained by running the NPSOL package Version 2.0 and a Fortran implementation of our methods on a VAX-11/780 at the Courant Mathematics and Computing Laboratory. A detailed description of both NPSOL and our methods can be found in Gurwitz (1986). Double precision arithmetic was used throughout.

As discussed in § 3.2, there is no merit function with which our methods will perform uniformly well. Two problems are presented in Gurwitz (1986) that demonstrate how either the exact penalty function or the augmented Lagrangian merit function used in NPSOL can degrade the performance of our methods, while using the other merit function on the same problem would not cause any difficulty.

Incorrect choice of merit function can cause our methods to fail or to reject the full step $\alpha = 1$, even near the solution. Therefore, we are interested in studying the performance of our methods without being concerned with the possible adverse effects of a bad choice of merit function. There is a class of problems for which a "natural" merit function is available. Let $f(x): \boldsymbol{R}^n \to \boldsymbol{R}^m$ be a twice continuously differentiable function. Consider the problems

$$(l_1P) \qquad \min_{x \in \boldsymbol{R}^n} \zeta(x) \equiv \|f(x)\|_1$$

and

$$(l_\infty P) \qquad \min_{x \in \boldsymbol{R}^n} \gamma(x) \equiv \|f(x)\|_\infty .$$

These problems arise from data-fitting applications, where one attempts to fit the m observations with a nonlinear function of n variables. The problem consists of selecting x so that the fit is optimal in some sense. By minimizing the l_1 norm, x is chosen so that the average deviation of the model from the data is minimized, while the infinity norm is used to minimize the maximum deviation.

Problem $(l_\infty P)$, also known as the *minimax* problem, is equivalent to the following nonlinear programming problem in $n+1$ variables:

$$(P_\infty) \qquad \begin{aligned} &\min_{(x \in \boldsymbol{R}^n, \delta \in \boldsymbol{R}^1)} \delta, \\ &\text{s.t.} \quad \delta - f_i(x) \geqq 0, \\ &\qquad \delta + f_i(x) \geqq 0, \qquad i = 1, \cdots, m. \end{aligned}$$

Similarly, by introducing m extra variables, (l_1P) can be transformed into

$$(P_1) \qquad \begin{aligned} &\min_{(x \in \boldsymbol{R}^n, u \in \boldsymbol{R}^m)} \sum_{i=1}^{m} u_i, \\ &\text{s.t.} \quad u_i - f_i(x) \geqq 0, \\ &\qquad u_i + f_i(x) \geqq 0, \qquad i = 1, \cdots, m. \end{aligned}$$

Any algorithm for solving nonlinearly constrained optimization problems can be used to compute the solution to these formulations of (l_1P) and $(l_\infty P)$. These problems have the additional property that feasibility can always be maintained. Specifically, any value of x, δ and u can be chosen by

$$\delta = \|f(x)\|_\infty, \qquad u_i = |f_i(x)|, \quad i = 1, \cdots, m.$$

The constraint functions, therefore, can be ignored when measuring progress toward the solution, and the objective functions themselves can be used directly as the merit function.

Following the standard definition of the QP subproblem, the IQP formulation of the subproblem for (P_∞) would be given by

$$\begin{aligned} &\min_{p \in \mathbf{R}^{n+1}} \tfrac{1}{2} p^T H p + e_{n+1}^T p, \\ &\text{s.t.} \quad A^T p \geqq -c \end{aligned} \tag{4.1}$$

where H is an $(n+1)\times(n+1)$ matrix that approximates the Hessian of the nonlinear problem and A and c are defined appropriately. We note that since the second derivative of δ is zero, the Hessian approximation in the QP should be of the form

$$H = \begin{bmatrix} \bar{H} & 0 \\ 0 & 0 \end{bmatrix}.$$

Although H is not positive definite, Z^THZ will normally be positive definite at the solution. This approach is used by Han (1978), (1981). Since our IQP method corresponds to that of Han, we are able to apply Han's results to prove the global convergence of our IQP method for the minimax problem (see Gurwitz (1986)).

Note that although our implementations are a reasonable way to solve minimum l_∞ problems, they give a very inefficient way of solving minimum l_1 problems because of the extra variables introduced. See Overton (1982) for a discussion of efficient implementations of minimum l_1 methods.

The test problems listed below can be found in Gill and Murray (1976). Many of these problems are also discussed by Watson (1979) and Moré, Garbow, and Hilstrom (1981). These problems were originally used for testing minimum least squares methods, but they are of interest for testing other data-fitting methods as well. Note, however, that the minimizer and minimum values reported in the literature for the least squares case usually differ from the optimal values for the l_1 and l_∞ norms.

For each of the test problems, we list below the number of variables, $\bar{n}$, and the number of functions, $\bar{m}$. Further information can be found in Gurwitz (1986). Our methods found a different solution from that found by NPSOL for both the minimum l_∞ and the minimum l_1 norms on Problem Chebyquad. Note that for minimum l_1 problems, the equivalent nonlinear programming problem has $\bar{n}+\bar{m}$ variables and $2\bar{m}$ constraints. Similarly, the nonlinear programming problem for the minimum l_∞ problem has $\bar{n}+1$ variables and $2\bar{m}$ constraints.

Problem 4.1. Woods. $\bar{n}=4$, $\bar{m}=7$.
Problem 4.2. Singular. $\bar{n}=4$, $\bar{m}=4$.
Problem 4.3. Rosenbrock. $\bar{n}=2$, $\bar{m}=2$.
Problem 4.4. Chebyquad. $\bar{n}=8$, $\bar{m}=8$.
Problem 4.5. Helix. $\bar{n}=3$, $\bar{m}=3$.
Problem 4.6. Large Residual (Davidon 2). $\bar{n}=4$, $\bar{m}=20$.
Problem 4.7. Cragg and Levy. $\bar{n}=4$, $\bar{m}=5$.
Problem 4.8. Beale. $\bar{n}=2$, $\bar{m}=3$.

Problem 4.9. Branin. $\bar{n} = 2$, $\bar{m} = 2$.
Problem 4.10. Freudenstein and Roth. $\bar{n} = 2$, $\bar{m} = 2$.
Problem 4.11. Bard. $\bar{n} = 3$, $\bar{m} = 15$.
Problem 4.12. Kowalik and Osborne. $\bar{n} = 4$, $\bar{m} = 11$.
Problem 4.13. Osborne 1. $\bar{n} = 5$, $\bar{m} = 33$.
Problem 4.14. Engvall 1. $\bar{n} = 5$, $\bar{m} = 5$.
Problem 4.15. Zangwill 2. $\bar{n} = 3$, $\bar{m} = 3$.
Problem 4.16. Madsen. $\bar{n} = 2$, $\bar{m} = 3$.

Tables 4.1 and 4.2 compare the performance of our methods, using the objective function itself as the merit function, with NPSOL, which uses an augmented Lagrangian merit function. The three values indicated for each method represent the number of major iterations, the number of minor iterations, and the number of function calls (i.e., the number of evaluations of f, g, c, and A). The entry $*$ in the table indicates failure. The tolerance level for deleting constraints was set to 10^{-3}. In fact, all of the problems were also run using the default tolerance level (see § 3.4) and produced nearly identical results. Definitions (a) and (f) of § 3.3 were used for s^k and y^k in the quasi-Newton update. The matrix D, used in forming the approximation to the full Hessian matrix from the projected matrix (see (2.8)), was set to the identity matrix.

TABLE 4.1
Minimum l_∞ problems.

Problem	nz	NPSOL	IQP	MQP	PQP	EQP
Woods	0	32, 88, 83	5, 13, 6	7, 13, 7	7, 11, 7	9, 10, 9
Singular	1	31, 51, 33	31, 15, 34	12, 37, 13	14, 37, 49	14, 29, 48
Rosenbrock	1	24, 29, 38	7, 12, 10	7, 11, 11	7, 8, 11	7, 7, 11
Chebyquad	5	28, 57, 33	6, 14, 8	6, 14, 8	6, 10, 8	7, 10, 15
Helix	2	14, 20, 22	9, 22, 13	9, 21, 13	9, 14, 13	9, 12, 13
Large Res.	2	16, 71, 17	42, 311, 152	*	*	*
C & L	4	41, 89, 45	21, 41, 58	18, 31, 53	17, 24, 53	16, 17, 50
Beale	0	15, 47, 17	11, 29, 17	*	*	*
Branin	1	1, 3, 7	2, 5, 4	2, 3, 3	2, 2, 3	2, 2, 3
F & R	1	11, 20, 13	11, 24, 17	*	*	*
Bard	1	7, 19, 8	6, 28, 7	8, 13, 9	8, 9, 9	9, 8, 10
K & O	0	*	8, 16, 10	8, 12, 12	7, 10, 11	9, 7, 16
Osborne 1	0	7, 33, 9	20, 60, 39	16, 23, 21	16, 19, 21	19, 18, 24
Engvall 1	4	*	*	14, 46, 24	15, 43, 91	*
Zangwill 2	2	2, 5, 6	2, 5, 3	2, 5, 3	2, 4, 3	4, 4, 3
Madsen	1	6, 9, 7	7, 12, 9	7, 12, 9	6, 8, 7	8, 9, 10

The number of function calls is determined by the number of trial points the merit function must consider before the new iterate is accepted. At best, the merit function always accepts the unit step, and the number of function calls is one more than the number of major iterations. However, for the EQP, and sometimes the PQP and MQP methods, the algorithm may take a "zero" step, i.e., add some constraints to the working set without making a move. In such a case, the number of major iterations may be larger than the number of function calls.

As evidenced by the results shown, our methods generally converge somewhat faster than NPSOL on this class of problems. We note that for these problems we use a different merit function than the one used in NPSOL. However, for problems on which the number of function calls is about the same as the number of major iterations,

TABLE 4.2
Minimum l_1 problems.

Problem	nz	NPSOL	IQP	MQP	PQP	EQP
Woods	1	6, 20, 14	3, 16, 4	5, 16, 3	5, 16, 3	13, 14, 4
Singular	0	11, 19, 12	11, 12, 12	11, 10, 12	11, 10, 12	16, 9, 12
Rosenbrock	0	4, 8, 12	4, 9, 5	4, 6, 5	4, 6, 5	6, 5, 5
Chebyquad	1	12, 30, 16	9, 29, 13	8, 25, 9	8, 25, 9	22, 23, 25
Helix	0	8, 14, 11	7, 10, 8	7, 10, 8	7, 10, 8	9, 7, 8
Large Res	4	*	37, 464, 48	*	*	*
C & L	1	46, 77, 53	19, 53, 20	19, 23, 20	19, 23, 20	5, 22, 22
Beale	0	11, 33, 13	10, 21, 11	7, 12, 8	7, 12, 8	8, 6, 8
Branin	0	7, 11, 8	4, 5, 5	4, 5, 5	4, 5, 5	7, 5, 5
F & R	2	9, 14, 10	7, 12, 8	9, 13, 10	9, 13, 10	10, 12, 10
Bard	0	12, 92, 13	8, 47, 9	15, 35, 14	16, 32, 17	32, 33, 20
K & O	0	10, 34, 14	12, 52, 21	9, 23, 12	11, 23, 15	34, 33, 30
Osborne 1	0	45, 182, 128	*	8, 93, 13	*	*
Engvall 1	*	*	*	*	*	*
Zangwill 2	0	1, 7, 4	2, 7, 3	2, 7, 3	2, 7, 3	6, 7, 3
Madsen	0	10, 7, 11	11, 7, 12	11, 7, 12	11, 7, 12	14, 7, 12

the merit function did accept steplengths of one, and the different results can be attributed to the differences in the second derivative information.

The column of the table headed *nz* indicates the number of columns of Z at the solution, which is the dimension of the projected Hessian matrix. Many of these problems are characterized by having a large number of constraints active at the solution, and hence the reduced space is of much lower dimension. We note that although a zero value of *nz* indicates that the projected Hessian is null at the solution, it generally is the case that there is some matrix of small dimension recurred during intermediate stages of the computation, until the working set is correctly identified.

As discussed in § 2.2, methods that update an approximation to the full second derivative matrix often introduce an augmenting term in order to force the approximating matrix to remain positive definite. These problems demonstrate the ill-conditioning of the approximating matrices that can result from adding this augmenting term. The estimated condition number of the NPSOL Hessian approximation for the Woods minimax problem was on the order of 10^{10}, for the Helix minimax problem, 10^9 and for the Cragg and Levy minimax problem, 10^8. However, for none of the minimax problems did the estimated condition number of the projected Hessian approximation recurred by our methods grow larger than 10^3.

We note that for some problems, our methods indicate a very large number of function calls, although the number of major and minor iterations may be small. In most cases, this phenomenon occurred on problems for which some Lagrange multipliers at the solution were near zero. In such cases, the methods may have essentially converged earlier. At the last step, the merit function may consider many trial points in an attempt to find the next iterate, causing the number of function calls to increase.

Table 4.3 compares the results of our methods and NPSOL on a group of general problems. As above, a tolerance of 10^{-3} is used for determining when a constraint may be deleted. The NPSOL merit function was used first. For the problems on which our methods failed because of using the NPSOL merit function, two sets of results are listed. The first corresponds to using the NPSOL merit function, and the second refers to the results obtained from using the exact penalty merit function with a suitable

TABLE 4.3
General problems.

Problem	nz	NPSOL	IQP	MQP	PQP	EQP
HS64-1	2	34, 56, 51	17, 38, 33	*	*	*
HS64-2[1]	2	—	39, 88, 63	41, 45, 227	41, 45, 227	47, 49, 238
HS78	2	8, 9, 10	6, 7, 8	7, 8, 9	7, 8, 9	7, 8, 9
HS84	0	5, 9, 6	2, 4, 3	2, 4, 3	2, 4, 3	5, 4, 6
HS104	7	7, 11, 8	6, 6, 7	5, 6, 6	5, 6, 6	6, 8, 11
HS111	7	56, 57, 59	56, 57, 60	56, 57 60	56, 57, 60	56, 57, 60
HS118	0	14, 33, 30	9, 29, 11	9, 25, 11	9, 23, 11	17, 17, 17
Manne-1	7	12, 37, 13	9, 31, 12	*	*	*
Manne-2[1]	7	—	9, 31, 12	40, 64, 41	40, 64, 41	58, 67, 110

[1] Using the exact penalty merit function.

choice of ρ (for HS 64, $\rho = 3000$; for Manne, $\rho = 10$). An entry of * indicates failure; a dashed line indicates that the corresponding combination of method and parameter does not apply.

The first seven problems can be found in Hock and Schittkowski (1981). The starting points and solutions are given in Gurwitz (1986). For each problem we list below the number of variables, *n*, the number of bound constraints, *mb*, the number of linear constraints, *ml*, and the number of nonlinear constraints, *mn*. The solutions found by both our program and NPSOL differ from those given in Hock and Schittkowski for Problems 4.20 and 4.21. The solution listed in Hock and Schittkowski for Problem 4.21 is not accurate. Our solution agrees with the one given in Nocedal and Overton (1985).

Problem 4.17. HS64. $n = 3$, $mb = 3$, $ml = 0$, $mn = 1$.

Problem 4.18. HS78. $n = 5$, $mb = 0$, $ml = 0$, $mn = 3$.

Problem 4.19. HS84. $n = 5$, $mb = 5$, $ml = 0$, $mn = 6$.

Problem 4.20. HS104. $n = 8$, $mb = 8$, $ml = 0$, $mn = 6$.

Problem 4.21. HS111. $n = 10$, $mb = 10$, $ml = 0$, $mn = 3$.

Problem 4.22. HS118. $n = 15$, $mb = 15$, $ml = 17$, $mn = 0$.

Problem 4.23. Manne. This is a smaller version of a problem presented in Murtagh and Saunders (1982). $n = 30$, $mb = 30$, $ml = 10$, $mn = 10$.

The results demonstrate that the NPSOL merit function can generally be used for all our methods when the IQP method performs only one minor iteration per major iteration. In that case, our other methods perform very similarly to the IQP method, and the QP multiplier estimates should have the correct signs. On the other hand, for problems on which the IQP method takes many minor iterations during the course of one major iteration, the other methods, which are forced to terminate the subproblem prematurely, suffer from inaccurate multiplier estimates. In most cases, the exact penalty merit function can be used, although, as mentioned above, it is extremely sensitive to the proper choice of ρ. For HS64, setting $\rho = 3000$ provided satisfactory results; however, for $\rho = 2280$, the algorithms failed.

The problems for which *nz* is relatively large, HS104, HS111, and Manne, were chosen in order to compare our methods with NPSOL on problems for which the dimension of the projected Hessian matrix is not significantly smaller than the dimension of the full Hessian. Problems HS78 and HS111 are equality constrained problems. Therefore, the active set is immediately identified and one would expect the projected methods to perform well, even when nz is large. Problem Manne is a problem

for which constraints must be added to the working set but are later deleted from it. Our non-IQP methods suffer from this need to often revise the working set. Problem HS118 is a linearly constrained problem for which nz is zero at the solution but is 14 at the first iteration. This indicates that many constraints must be added to the working set. Since the EQP method can add only one constraint per iteration, it is not surprising that it takes almost twice as many major iterations as our other methods. The other methods, however, take many more minor iterations.

Problem 4.24. Penalty Function I. This problem can be found in Moré, Garbow, and Hillstrom (1981). $n = 50$, $mb = 50$, $ml = 0$, $mn = 0$.

This problem is an example where an EQP method works best (see Table 4.4). On this problem the IQP method picks up all 50 bound constraints at the first iteration and then must subsequently delete them. Since the EQP method cannot pick up many constraints at once, it does not go through this process of unnecessarily modifying the working set. Nevertheless, the IQP method does recover from its initially bad start. The MQP and PQP methods, which also add many constraints during the first iteration, can recover faster than the IQP method, provided that a very loose dropping tolerance is used. If any reasonably small tolerance level is used, these methods suffer from not being able to delete constraints until much effort has been spent in finding the minimum on the wrong subspace. We note that MQP and PQP methods produce identical iterates for this problem. This is due to the fact that at the first iteration the IQP adds many constraints to the working set but does not delete any of them until subsequent iterations. Therefore, at the end of the first subproblem all of the non-EQP methods are at the same point. For subsequent QP subproblems, both the MQP and the PQP methods start with the same initial working sets. Since constraints are no longer being added within the subproblems, the MQP method cannot drop any constraints within the QP, and thus is the same as the PQP method for this problem.

5. Conclusions. The results indicate that our IQP method is the most robust of our methods. In many cases the superiority of the IQP method is due to the ability to use the NPSOL merit function rather than the exact penalty merit function. However, even for the minimax and minimum l_1 problems, for which a merit function is available for all the methods, the IQP method performs the best. Yet, for problems on which none of the methods have any difficulty, the IQP method may be slightly worse due to the fact that it often takes more minor iterations.

The MQP and PQP methods almost always perform identically. These methods can differ only in a case where the IQP would pick up many constraints and then drop some. Usually, if there is a difference between the two methods, it is evidenced at the first iteration. During subsequent iterations, the IQP may modify the working set by adding new constraints, or deleting old ones, but it rarely adds many constraints and

TABLE 4.4
Penalty function I.

	delete tolerance	
Method	10^6	10^{-1}
NPSOL	9, 131, 27	9, 131, 27
IQP	16, 164, 44	16, 164, 44
MQP	9, 59, 27	>200
PQP	9, 59, 27	>200
EQP	9, 10, 27	29, 30, 93

then drops some within one QP once the computation has passed the first few iterations. Therefore, at that stage the MQP and PQP methods are identical.

Since the MQP and PQP methods can fail when used with the NPSOL merit function, a possible improvement would be to use a merit function that is more suitable to these methods. It would appear that such a merit function might be defined as an adaptive method that does not rely on multiplier estimates when far from the solution but that allows for fast local convergence by considering the multipliers once the working set has settled down.

The EQP method is preferable in cases in which the IQP picks up many constraints but then must delete them all from the working set. Since the EQP can pick up only one constraint at a time, it does not have the chance to add many constraints that are incorrectly determined to be active. However, for a problem in which many constraints must be made active, the EQP will take many more steps (and thus will make more function calls) in order to add each constraint separately to the working set.

In general, our results indicate that the projected Hessian updating methods are most successful when the dimension of the projected Hessian is small, and the active constraints are correctly identified in the early stages of the computation. These methods perform well even when the dimension of the projected space is large, provided that the active set is easily identified. Performance of these methods is worst on problems for which the working set changes many times, since the method does not get a chance to build up second derivative information on the projected subspace.

The most important question is whether projected Hessian updating methods have a useful role to play in future choices of the best methods for solving nonlinear programming problems. We feel that our results for the IQP algorithm are encouraging in that regard. Although the NPSOL results are very good, our IQP algorithm, using the NPSOL merit function, can give somewhat better results, as demonstrated by Table 4.3. Note that essentially the only difference between the two methods is that we recur the projected Hessian rather than the full second derivative matrix. The numerical results indicate that projected Hessian updating methods may be useful in practice.

Acknowledgments. We would particularly like to thank Philip E. Gill, Walter Murray, and Margaret H. Wright of Stanford University for many long and helpful conversations and for providing us with both their NPSOL code and test problems to use in our work.

REFERENCES

C. G. BROYDEN, J. E. DENNIS, JR., AND J. J. MORÉ (1973), *On the local and superlinear convergence of quasi-Newton methods*, J. Inst. Math. Appls., 12, pp. 223–245.

R. H. BYRD, J. NOCEDAL, AND Y. YUAN (1987), *Global convergence of a class of quasi-Newton methods on convex problems*, SIAM J. Numer. Anal., 24, pp. 1171–1190.

R. H. BYRD AND G. A. SHULTZ (1982), *A practical class of globally convergent active set strategies for linearly constrained optimization*, Tech. Report CU-CS-238-82, Computer Science Department, University of Colorado, Boulder, CO.

P. H. CALAMAI AND J. J. MORÉ (1987), *Quasi-Newton methods with bounds*, SIAM J. Numer. Anal., 24, pp. 1434–1441.

R. M. CHAMBERLAIN, C. LEMARECHAL, H. C. PEDERSEN, AND M. J. D. POWELL (1982), *The watchdog technique for forcing convergence in algorithms for constrained optimization*, Math. Programming Stud., 16, pp. 1–17.

T. F. COLEMAN AND A. R. CONN (1982a), *Nonlinear programming via an exact penalty function: asymptotic analysis*, Math. Programming, 24, pp. 123–136.

T. F. COLEMAN AND A. R. CONN (1982b), *Nonlinear programming via an exact penalty function: global analysis*, Math. Programming, 24, pp. 137-161.

———, (1984), *On the local convergence of a quasi-Newton method for the nonlinear programming problem*, SIAM J. Numer. Anal., 21, pp. 755-769.

J. E. DENNIS, JR. AND J. J. MORÉ (1974), *A characterization of superlinear convergence and its application to quasi-Newton methods*, Math. Comp., 28, pp. 549-560.

———, (1977), *Quasi-Newton methods, motivation and theory*, SIAM Rev., 19, pp. 46-89.

R. FLETCHER (1980), *Practical Methods of Optimization, Vol.* 1, *Unconstrained Optimization*, John Wiley, New York, Toronto.

———, (1981), *Practical Methods of Optimization, Vol.* 2, *Constrained Optimization*, John Wiley, New York, Toronto.

———, (1982), *Second order corrections for non-differentiable optimization*, in Numerical Analysis, G. A. Watson, ed., Dundee, 1981, Lecture Notes in Mathematics 912, Springer-Verlag, Berlin, New York, pp. 85-114.

———, (1985), *An l_1 penalty method*, in Numerical Optimization 1984, P. T. Boggs, R. H. Byrd, and R. B. Schnabel, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 26-40.

P. E. GILL AND W. MURRAY (1976), *Algorithms for the solution of the non-linear least squares problem*, Report NAC 71, National Physics Laboratory, Middlesex, England.

———, (1979), *The computation of Lagrange-multiplier estimates for constrained minimization*, Math. Programming, 17, pp. 32-60.

P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND M. H. WRIGHT (1983a), *User's guide for* SOL/QPSOL: *A Fortran package for quadratic programming*, Tech. Report SOL 83-7, Department of Operations Research, Stanford University, Stanford, CA.

———, (1983b) *User's guide for* SOL/NPSOL: *a Fortran package for nonlinear programming*, Tech. Report SOL 83-12, Department of Operations Research, Stanford University, Stanford, CA.

———, (1983c), *Procedures for optimization problems with a mixture of bounds and general linear constraints*, Tech. Report SOL 82-6, Department of Operations Research, Stanford University, Stanford, CA.

———, (1983d), *On the representation of a basis for the null space*, Tech. Report SOL 83-19, Department of Operations Research, Stanford University, Stanford, CA.

———, private communication, 1984.

———, (1985a), *Software and its relationship to methods*, in Numerical Optimization 1984, P. T. Boggs, R. H. Byrd, and R. B. Schnabel, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 139-159.

———, (1985b), *Some issues in implementing a sequential quadratic programming algorithm*, ACM Signum Newsletter, 20, no. 2, pp. 13-19.

———, (1985c), *Model building and practical aspects of nonlinear programming*, in Computational Mathematical Programming, K. Schittkowski, ed., NATO ASI Series F: Computer and Systems Sciences 15, Springer-Verlag, Berlin, New York, pp. 209-247.

P. E. GILL, W. MURRAY, AND M. H. WRIGHT (1981), *Practical Optimization*, Academic Press, London, New York.

J. GOODMAN (1985), *Newton's method for constrained optimization*, Math. Programming, 33, pp. 162-171.

C. B. GURWITZ (1986), *Sequential quadratic programming methods based on approximating a projected Hessian matrix*, Ph.D. thesis, New York University, New York, NY; Tech. Report No. 219, Department of Computer Science, New York University, New York, NY.

S.-P. HAN (1976), *Superlinearly convergent variable metric algorithms for general nonlinear programming problems*, Math. Programming, 11, pp. 263-282.

——— (1977), *A globally convergent method for nonlinear programming*, J. Optim. Theory Appl., 22, pp. 297-310.

——— (1978), *Superlinear convergence of a minimax method*, Tech. Report TR78-336, Computer Science Department, Cornell University, Ithaca, NY.

———, (1981), *Variable metric methods for minimizing a class of nondifferentiable functions*, Math. Programming, 20, pp. 1-13.

W. HOCK AND K. SCHITTKOWSKI (1981), *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems 187, Springer-Verlag, Berlin, New York.

D. Q. MAYNE (1979), *On the use of exact penalty functions to determine step length in optimization algorithms*, in Numerical Analysis, G. A. Watson, ed., Dundee, 1979, pp. 98-109; Lecture Notes in Mathematics 773, Springer-Verlag, Berlin, New York.

J. J. MORÉ, B. S. GARBOW, AND K. H. HILSTROM (1981), *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7, pp. 17-41.

H. MUKAI AND E. POLAK (1978), *On the use of approximations in algorithms for optimization problems with equality and inequality constraints*, SIAM J. Numer. Anal., 15, pp. 674–693.

W. MURRAY AND M. H. WRIGHT (1978), *Projected Lagrangian methods based on the trajectories of penalty and barrier functions*, Systems Optimization Laboratory Report 78-23, Stanford University, Stanford, CA.

———, (1982), *Computation of the search direction in constrained optimization algorithms*, Math. Programming Stud., 16, pp. 62–83.

B. A. MURTAGH AND M. A. SAUNDERS (1982), *A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints*, Math. Programming Stud., 16, pp. 84–117.

J. NOCEDAL AND M. L. OVERTON (1985), *Projected Hessian updating algorithms for nonlinearly constrained optimization*, SIAM J. Numer. Anal., 22, pp. 821–850.

J. M. ORTEGA AND W. C. RHEINBOLDT (1970), *Iterative Solutions of Nonlinear Equations in Several Variables*, Academic Press, New York, London.

M. L. OVERTON (1982), *Algorithms for nonlinear l_1 and l_∞ fitting*, in Nonlinear Optimization 1981, M. J. D. Powell, ed., Academic Press, London, New York, pp. 91–101.

M. J. D. POWELL (1978a), *Algorithms for nonlinear constraints that use Lagrangian functions*, Math. Programming, 14, pp. 224–248.

———, (1978b), *The convergence of variable metric methods for nonlinearly constrained optimization calculations*, in Nonlinear Programming 3, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, eds., Academic Press, New York, London, pp. 27–63.

———, (1986), *How bad are the* BFGS *and* DFP *methods when the objective function is quadratic*?, Math. Programming, 34, pp. 34–47.

R. W. H. SARGENT, personal communication, 1986.

K. SCHITTKOWSKI (1981), *The nonlinear programming method of Wilson, Han, and Powell with an augmented Lagrangian type line search function, parts* 1 *and* 2, Numer. Math., 38, pp. 83–127.

———, (1982), *On the convergence of a sequential quadratic programming method with an augmented Lagrangian line search functions*, Tech. Report SOL 82-4, Department of Operations Research, Stanford University, Stanford, CA.

G. A. WATSON (1979), *The minimax solution of an overdetermined system of nonlinear equations*, J. Inst. Math. Appl., 23, pp. 167–180.

P. WOLFE (1972), *On the convergence of gradient methods under constraints*, IBM J. Res. Develop., 16, pp. 407–411.

R. S. WOMERSLEY (1984), *Minimizing nonsmooth composite functions*, Report CMA R12-84, Center for Mathematical Analysis, Australian National University, Canberra, Australia.

——— (1985), *Local properties of algorithms for minimizing nonsmooth composite functions*, Math. Programming 32, pp. 69–89

R. S. WOMERSLEY AND R. FLETCHER (1986), *An algorithm for composite nonsmooth optimization problems*, J. Optim. Theory Appl., 48, pp. 493–523.

M. H. WRIGHT (1976), *Numerical methods for nonlinearly constrained optimization*, Ph.D. thesis, Stanford University, Stanford, CA.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 654-670, July 1989
© 1989 Society for Industrial and Applied Mathematics
003

NONSTANDARD SCALING MATRICES FOR TRUST REGION GAUSS-NEWTON METHODS*

HUBERT SCHWETLICK† AND VOLKER TILLER†

Abstract. For solving large nonlinear least-squares problems via trust region Gauss-Newton methods, nonstandard scaling matrices are proposed for scaling the norm of the step. The scaling matrices are rectangular, of full rank, and contain a block of the Jacobian matrix of the residual function.

Three types of such matrices are investigated. The corresponding trust region methods are shown to have qualitatively the same convergence properties as the standard method. Nonstandard scaling matrices are especially intended for solving large and structured problems such as orthogonal distance regression or surface fitting. Initial computational experience suggests that for such problems the proposed scaling implies sometimes a modest increase in the number of iterations but reduces the overall computational costs.

Key words. nonlinear least squares, trust region Gauss-Newton method, Levenberg-Marquardt algorithm, nonstandard scaling, orthogonal distance regression

AMS(MOS) subject classifications. primary, 65H10; secondary, 65D10, 65H05

1. Introduction.

The trust region Gauss-Newton method is in common use for solving nonlinear least-squares problems

$$\text{minimize } \tfrac{1}{2}\|f(x)\|^2, \quad f: R^n \to R^m, \quad m > n. \tag{1}$$

The norm is always the Euclidean vector norm, or the spectral norm for matrices. The solution proceeds iteratively

$$x^{k+1} = x^k - s^k, \qquad k = 0, 1, 2, \cdots$$

where the correction s^k is an approximate solution of the constrained linear least-squares problem

$$\text{minimize } \{\|J(x^k)s - f(x^k)\|: \|D_k s\| \leqq \Delta_k\}. \tag{2}$$

Here $J(x^k)$ denotes the Jacobian matrix of $f(x)$ at the current iterate x^k, D_k is a nonsingular (n, n)-matrix scaling the norm of the step s, and $\Delta_k > 0$ is the trust region radius.

Let

$$s_{GN} = J(x^k)_D^+ f(x^k), \qquad J(x^k)_D^+ = D_k^{-1}[J(x^k)D_k^{-1}]^+$$

be the (generalized) Gauss-Newton step, i.e., that solution of the unconstrained linear least-squares problem

$$\text{minimize } \|J(x^k)s - f(x^k)\| \tag{3}$$

that has minimal norm $\|D_k s\|$. It is known that the Gauss-Newton step $s = s_{GN}$ solves (2) if $\|D_k s_{GN}\| \leqq \Delta_k$. Otherwise, problem (2) has the unique solution $s = s(\lambda)$ where $s(\lambda)$ is defined as the solution of the unconstrained regularized linear least-squares problem

$$\text{minimize } \|J(x^k)s - f(x^k)\|^2 + \lambda \|D_k s\|^2, \tag{4}$$

and $\lambda > 0$ is determined from the scalar equation

$$\Phi(\lambda) = \|D_k s(\lambda)\| - \Delta_k = 0. \tag{5}$$

* Received by the editors January 21, 1987; accepted for publication August 22, 1988.

† Department of Mathematics, Martin Luther University, Halle-Wittenberg, Halle, 4020, German Democratic Republic.

An appropriate algorithm for solving (4), (5) was described by Moré [8], and now it is part of most trust region codes. Since $\Phi(\lambda)$ is nonlinear in λ, (5) also has to be solved iteratively so that the solution $s(\lambda)$ of (4) has to be evaluated repeatedly for different values of λ. As reported by Moré [8] and confirmed by Gay [4], on the average, 1.8 evaluations of $s(\lambda)$ per step are required in case of medium-size problems. When $J(x^k)$ is unstructured and a QR factorization of it is known, then for scaling matrices commonly used the number of operations necessary to compute $s(\lambda)$ for a given λ is of order n^3. Thus, when n is not small compared to m, the costs for computing $s(\lambda)$ for a given λ are of the same order as the costs for the QR factorization of $J(x^k)$, i.e., the costs for solving (3). Moreover, for large problems the number of inner iterations for solving (5) seems to increase (see Moré and Sorensen [11]). Therefore, when both m and n are large, solving (2) requires a multiple of costs necessary for solving the unrestricted problem (3). As a consequence for such problems, the standard trust region approach cannot compete with the damped Gauss-Newton method.

In this paper we propose a nonstandard choice of the scaling matrices D_k that allows us to solve (2) with costs not essentially higher than those for solving (3), provided that most of the columns of $J(x^k)$ are linearly independent, i.e.,

$$\text{rank}\,(J(x^k)) \geqq t \quad \text{and} \quad l := n - t \ll n. \tag{6}$$

The iteration with respect to λ is realized by solving only l-dimensional linear subproblems with costs of order l^3 for each λ. Let us remark that in (6) only a lower bound t for the rank of $J(x^k)$ and not the rank itself is needed.

There are classes of real problems for which the basic assumption (6) is automatically satisfied. As an example we mention the parameter estimation problem in explicit models $y = r(z, \Theta)$ with errors in the variables, sometimes also referred to as orthogonal distance regression or total approximation problems (cf. Schwetlick and Tiller [15]; Boggs, Byrd, and Schnabel [1]; and Watson [16]). In this case the vector x of variables has the form

$$x = (z^1, \cdots, z^M, \Theta),$$

and its dimension is $n = d(x) = M \cdot d(z) + d(\Theta)$ where $d(v)$ denotes the dimension of a vector v. Here z^j is the vector of state variables at the jth state ($j = 1, \cdots, M$) of the process, and all these states z^j have to be estimated simultaneously with the parameter vector Θ. This problem can be formulated as nonlinear least-squares problem (1) with an appropriate f of dimension $m = d(f) = M \cdot d(z) + M \cdot d(y)$. Since the number M of measurements may be large, the problem is of high dimension, in general, and n is not small compared to m. More precisely, there holds

$$n \leqq m \leqq (1 + d(y)/d(z))n. \tag{7}$$

However, the Jacobian of f has the structure

$$J = \left[\begin{array}{c|c} I & 0 \\ \hline A_1 & A_2 \end{array}\right] = \left[\begin{array}{c} \diagdown \\ \hline \square \quad \square \\ \ddots \quad \vdots \\ \square \quad \square \end{array}\right]. \tag{8}$$

The matrix I is the identity matrix of dimension $M \cdot d(z)$, A_1 is a block diagonal matrix with the Jacobian matrices $r_z(z^j, \Theta)$ of r with respect to z as diagonal blocks,

and A_2 consists of the Jacobian matrices $r_\Theta(z^j, \Theta)$ with respect to Θ, the blocks computed at the M different states. For more details, see § 4 of this paper.

A possible rank deficiency of J is obviously restricted to the block A_2 because the left part of J always has full rank, i.e., $\operatorname{rank}(J) \geqq M \cdot d(z)$. Thus (6) is satisfied with $t = M \cdot d(z)$ and $l = n - t = d(\Theta)$, where $l \ll n$ for sufficiently large M.

The following § 2 is addressed to the choice of scaling matrices D_k. If t linearly independent columns of $J(x^k)$ are known, then we can construct three versions of scaling matrices D_k having the desired property to reduce the repeated solution of (4) to the solution of only $(n-t)$-dimensional subproblems. In order to apply the general convergence theory for trust region methods given in Moré [9], the matrices D_k have to satisfy certain conditions. This problem is discussed in § 3. These conditions are satisfied in the structured problem mentioned above. However, they can be fulfilled also in case of an unstructured Jacobian matrix if, at the kth step, $t = t_k$ is adaptively chosen by using appropriate factorization techniques. In § 4 we apply our concepts for choosing D_k to structured problems such as orthogonal distance regression and surface fitting, both allowing a fixed, a priori choice of t. It turns out that the third version of scaling matrices D_k yields the same step s which was proposed by Schwetlick and Tiller [15] in a special algorithm for solving errors-in-variables problems. However, in that paper the step was controlled as in Marquardt [7] and Schwetlick [13], [14], i.e., by directly manipulating the parameter λ rather than by changing the trust region radius Δ. The last § 5 contains some numerical examples.

2. The choice of scaling matrices and the computation of the step. As mentioned above, the step computation in ellipsoidal trust region algorithms for nonlinear least-squares problems involves solving

$$\text{minimize } \{\|Js - f\|^2 + \lambda \|Ds\|^2\} \tag{9}$$

for different $\lambda > 0$. Here and in the following we omit the argument and the index k, i.e., J stands for $J(x^k)$, D for D_k, and so on. The unconstrained linear least-squares problem (9) is equivalent to

$$\text{minimize } \left\| \begin{bmatrix} J \\ \lambda^{1/2} D \end{bmatrix} s - \begin{bmatrix} f \\ 0 \end{bmatrix} \right\|^2. \tag{9'}$$

For the repeated solution of (9) via orthogonal transformations Moré [8] described the following two-stage method.

Stage 1. *Reduction of J to upper triangular form.* Perform the orthogonal transformation

$$Q_1^T J P = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, \qquad Q_1^T f = \begin{bmatrix} \tilde{g}_1 \\ \tilde{g}_2 \end{bmatrix}$$

where Q_1 is (m, m)-orthogonal, R_1 is (n, n)-upper-triangular, P is a permutation that commutes the columns of J, and $\tilde{g}_1$ has n components. Then, with

$$\tilde{s} = P^T s, \qquad \tilde{D} = P^T D P,$$

(9) is equivalent to

$$\text{minimize } \left\| \begin{bmatrix} R_1 \\ \lambda^{1/2} \tilde{D} \end{bmatrix} \tilde{s} - \begin{bmatrix} \tilde{g}_1 \\ 0 \end{bmatrix} \right\|^2. \tag{10}$$

Stage 2. *Computation of* $s(\lambda)$ *for given* $\lambda > 0$. Perform the orthogonal transformation

$$Q^T \begin{bmatrix} R_1 \\ \lambda^{1/2}\tilde{D} \end{bmatrix} = \begin{bmatrix} R(\lambda) \\ 0 \end{bmatrix}, \qquad Q^T \begin{bmatrix} \tilde{g}_1 \\ 0 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$$

where Q is $(2n, 2n)$-orthogonal, R is (n, n)-upper-triangular, and g_1 has n components. Solve the triangular system

$$R(\lambda)\tilde{s} = g_1$$

for $\tilde{s}$, and set $s = P\tilde{s}$.

Stage 1 does not depend on λ and is usually realized by Householder or Fast Givens transformations which require $\sim(mn^2 - n^3/3)$ flops where one flop means one multiplication plus one addition. Stage 2 realized by Fast Givens transformations has the operation count $\sim n^3/3$.

In general, only a few iterations with respect to λ are necessary, since the Newton-type method used for solving (5) converges quickly and, moreover, only an approximate solution of this equation is needed. On the other hand, the step $s = s^k$ may be rejected by the trust region algorithm if the new residual $\|f(x^k - s^k)\|$ is not sufficiently smaller than $\|f(x^k)\|$; see Algorithm 2 in § 3. In this case $\Delta = \Delta_k$ is reduced, and the computation of the corresponding step s is restarted by passing through the second stage only.

For many practical problems the relation $m \gg n$ is valid. Then the leading term in the costs for computing the step s becomes mn^2, caused by Stage 1, and even the repeated realization of Stage 2 does not essentially affect the total costs. If, however, the relation $n \ll m$ is not valid, then the situation is completely different.

As an example we consider the errors-in-variables problem mentioned in the Introduction. There both m and n are large if the number M of measurements is large (see (7)). In such cases the computational costs for the second stage are of the same order as the costs for Stage 1, so that the iteration with respect to λ becomes much more expensive.

The special scaling matrices proposed in the following are intended to overcome these difficulties. The construction of the scaling matrices D_k rests essentially on the assumption that in a large Jacobian matrix J only a few columns, if any, depend linearly on the others. Let J_1 be the matrix of those t columns of J that are known to be sufficiently linearly independent. The remaining columns, which may depend on those of J_1, are collected in the matrix J_2. Without loss of generality we assume that the first t columns of J form the regular part J_1, and the last $l = n - t$ columns form J_2, i.e.,

$$\begin{gathered} J = [\underset{t}{J_1} | \underset{l}{J_2}] = m, \\ t + l = n, \quad l < n, \quad \operatorname{rank}(J_1) = t. \end{gathered} \tag{11}$$

Now let us return to problem (9). Obviously the matrix D that scales the norm in (9) regularizes the matrix J in (9′) in the sense of Tihonov. Moreover, for large λ, i.e., for small Δ, the information contained in J becomes negligible in the coefficient matrix of the corresponding normal equations $(J^TJ + \lambda D^TD)s = J^Tf$. But the matrix $J = [J_1 | J_2]$ need not be regularized in its first part J_1, and the information contained in J_1 should not become negligible in $J^TJ + \lambda D^TD$. Therefore we take D as a rectangular matrix having J_1 as the block in its first t columns. Obviously the trust region approach does not require D to be square. Any full rank matrix D can be used for defining the

ellipsoidal norm $\|v\|_D = (v^T D^T D v)^{1/2} = \|Dv\|$. Motivated by these ideas, in the following we describe three possibilities for the choice of D.

Version 1. Set

$$D = D^{(1)} = \left[\begin{array}{c|c} J_1 & 0 \\ \hline 0 & C \end{array}\right]$$

where C is a regular (l, l)-matrix. If

$$s = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}$$

is partitioned analogously to $J = [J_1 | J_2]$, then problem (9′) reads

$$\text{minimize} \left\| \left[\begin{array}{c|c} J_1 & J_2 \\ \hline \lambda^{1/2} J_1 & 0 \\ \hline 0 & \lambda^{1/2} C \end{array}\right] \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} - \begin{bmatrix} f \\ 0 \\ 0 \end{bmatrix} \right\|. \tag{12}$$

To solve this problem via orthogonal transformations, first a QR factorization of J is computed as in Stage 1 of the general case:

$$Q_1^T [J_1 | J_2] = \left[\begin{array}{c|c} K_{11} & K_{22} \\ \hline 0 & K_{22} \\ \hline 0 & 0 \end{array}\right], \qquad Q_1^T f = \begin{bmatrix} \tilde{g}_1 \\ \tilde{g}_2 \\ \tilde{g}_3 \end{bmatrix}. \tag{13}$$

The block K_{11} is (t, t)-upper-triangular, K_{22} is (l, l)-upper triangular, and $\tilde{g}_1$ and $\tilde{g}_2$ have t and l components, respectively. K_{11} is always regular, K_{22} may be rank-deficient. Multiplying the residual of (12) from the left by the orthogonal matrix

$$\begin{bmatrix} Q_1^T & & \\ & Q_1^T & \\ & & I_l \end{bmatrix} \tag{13′}$$

leads to the equivalent problem

$$\text{minimize} \left\| \left[\begin{array}{c|c} K_{11} & K_{12} \\ \hline 0 & K_{22} \\ \hline \lambda^{1/2} K_{11} & 0 \\ \hline 0 & \lambda^{1/2} C \end{array}\right] \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} - \begin{bmatrix} \tilde{g}_1 \\ \tilde{g}_2 \\ 0 \\ 0 \end{bmatrix} \right\|. \tag{14}$$

The squared norm of the residual in (14) is

$$\rho^2 = \|K_{11} s_1 + K_{12} s_2 - \tilde{g}_1\|^2 + \|K_{22} s_2 - \tilde{g}_2\|^2 + \lambda \|K_{11} s_1\|^2 + \lambda \|C s_2\|^2.$$

By considering the identity

$$\|u + v\|^2 + \lambda \|u\|^2 = \|(1+\lambda)^{1/2} u + (1+\lambda)^{-1/2} v\|^2 + \lambda (1+\lambda)^{-1} \|v\|^2 \tag{15}$$

and introducing $u = K_{11} s_1$, $v = K_{12} s_2 - \tilde{g}_1$, ρ^2 can be expressed as

$$\begin{aligned} \rho^2 = {} & \|(1+\lambda)^{1/2} K_{11} s_1 + (1+\lambda)^{-1/2} (K_{12} s_2 - \tilde{g}_1)\|^2 \\ & + \lambda (1+\lambda)^{-1} \|K_{12} s_2 - \tilde{g}_1\|^2 + \|K_{22} s_2 - \tilde{g}_2\|^2 + \lambda \|C s_2\|^2. \end{aligned}$$

Because K_{11} is regular, s_1 can always be chosen in dependence on s_2 so that the first term vanishes. Thus, problem (14) can be solved by computing the solution of the regularized linear least-squares problem

$$\text{minimize} \left\| \begin{bmatrix} \left(\frac{\lambda}{1+\lambda}\right)^{1/2} K_{12} \\ K_{22} \\ \lambda^{1/2} C \end{bmatrix} s_2 - \begin{bmatrix} \left(\frac{\lambda}{1+\lambda}\right)^{1/2} \tilde{g}_1 \\ \tilde{g}_2 \\ 0 \end{bmatrix} \right\| \tag{16}$$

followed by the solution of the linear system

$$(1+\lambda) K_{11} s_1 = \tilde{g}_1 - K_{12} s_2. \tag{17}$$

Remember that K_{11} is already upper-triangular so that solving (17) is a simple back-substitution. Moreover, we will see later that this backsubstitution is to be done only once after having obtained a final λ accepted as an approximate solution of (5).

The least-squares problem (16) is again solved via orthogonal factorization. Since K_{12} is a long matrix, it is useful to shorten this block in advance by the orthogonal transformation

$$Q_2^T K_{12} = \begin{bmatrix} R_{12} \\ 0 \end{bmatrix}, \qquad Q_2^T \tilde{g}_1 = \begin{bmatrix} \tilde{g}_{11} \\ \tilde{g}_{12} \end{bmatrix} \tag{18}$$

where Q_2 is (t, t)-orthogonal and R_{12} is (l, l)-upper-triangular. Then (16) is equivalent to

$$\text{minimize} \left\| \begin{bmatrix} \left(\frac{\lambda}{1+\lambda}\right)^{1/2} R_{12} \\ K_{22} \\ \lambda^{1/2} C \end{bmatrix} s_2 - \begin{bmatrix} \left(\frac{\lambda}{1+\lambda}\right)^{1/2} \tilde{g}_{11} \\ \tilde{g}_2 \\ 0 \end{bmatrix} \right\| \tag{19}$$

with a $(3l, l)$-matrix instead of the $(2l+t, l)$-matrix in (16). Finally, the transformation

$$Q_3^T \begin{bmatrix} \left(\frac{\lambda}{1+\lambda}\right)^{1/2} R_{12} \\ K_{22} \\ \lambda^{1/2} C \end{bmatrix} = \begin{bmatrix} R(\lambda) \\ 0 \\ 0 \end{bmatrix}, \qquad Q_3^T \begin{bmatrix} \left(\frac{\lambda}{1+\lambda}\right)^{1/2} \tilde{g}_{11} \\ \tilde{g}_2 \\ 0 \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} \tag{20}$$

with an orthogonal $(3l, 3l)$-matrix Q_3 and an upper-triangular (l, l)-matrix $R(\lambda)$ is performed, and $s_2 = s_2(\lambda)$ is computed as the solution of

$$R(\lambda) s_2 = g_1. \tag{21}$$

In the Newton-type algorithm for solving $\Phi(\lambda) = \|Ds(\lambda)\| - \Delta = 0$ described in Moré [8], at each step the current estimate λ is improved using the numbers $\Phi(\lambda)$ and $\Phi'(\lambda)$. In the following we express these values in terms of the quantities already introduced above. By definition of D there holds

$$\|Ds(\lambda)\|^2 = \|J_1 s_1(\lambda)\|^2 + \|C s_2(\lambda)\|^2.$$

From (13), (17), and (28) we obtain

$$\begin{aligned} \|J_1 s_1\|^2 = \|K_{11} s_1\|^2 &= \|\tilde{g}_1 - K_{12} s_2\|^2/(1+\lambda)^2 \\ &= (\|\tilde{g}_{11} - R_{12} s_2\|^2 + \|\tilde{g}_{12}\|^2)/(1+\lambda)^2; \end{aligned}$$

hence,

$$\|Ds(\lambda)\|^2 = (\|\tilde{g}_{11} - R_{12} s_2\|^2 + \|\tilde{g}_{12}\|^2)/(1+\lambda^2) + \|C s_2\|^2. \tag{22}$$

Indeed, $\|Ds(\lambda)\|$ and, therefore, $\Phi(\lambda)$, can be expressed without explicitly using s_1. The derivative $\Phi'(\lambda)$ is given by

$$\Phi'(\lambda) = -(D^T Ds(\lambda))^T (J^T J + \lambda D^T D)^{-1} D^T Ds(\lambda) / \|Ds(\lambda)\|. \tag{23}$$

Next we examine the term $(J^T J + \lambda D^T D)^{-1}$. From the definition of D, (12), (13), (18), and (20) it follows that

$$\begin{aligned} J^T J + \lambda D^T D &= \left[\begin{array}{c|c} (1+\lambda) K_{11}^T K_{11} & K_{11}^T K_{12} \\ \hline K_{12}^T K_{11} & K_{12}^T K_{12} + K_{22}^T K_{22} + \lambda C^T C \end{array}\right] \\ &= \bar{R}(\lambda)^T \bar{R}(\lambda) \end{aligned} \tag{24}$$

with

$$\bar{R}(\lambda) = \left[\begin{array}{c|c} (1+\lambda)^{1/2} K_{11} & (1+\lambda)^{-1/2} K_{12} \\ \hline 0 & R(\lambda) \end{array}\right]. \tag{25}$$

Therefore, (23) yields

$$\Phi'(\lambda) = -(D^T Ds)^T (\bar{R}^T \bar{R})^{-1} D^T Ds / \|Ds\| = -\|\bar{R}^{-T} D^T Ds\|^2 / \|Ds\|. \tag{26}$$

By considering

$$\bar{R}(\lambda)^{-T} = \left[\begin{array}{c|c} (1+\lambda)^{-1/2} K_{11}^{-T} & 0 \\ \hline -(1+\lambda)^{-1} R(\lambda)^{-T} K_{12}^T K_{11}^{-T} & R(\lambda)^{-T} \end{array}\right],$$

(12), (13), (17), and (18), the numerator of (26) becomes

$$\begin{aligned} \|\bar{R}^{-T} D^T Ds\|^2 &= (\|\tilde{g}_{11} - R_{12} s_2\|^2 + \|\tilde{g}_{12}\|^2)/(1+\lambda)^3 \\ &\quad + \|R(\lambda)^{-T} (C^T C s_2 - R_{12}^T (\tilde{g}_{11} - R_{12} s_2)/(1+\lambda)^2)\|^2 \end{aligned} \tag{27}$$

so that also $\Phi'(\lambda)$ can be expressed without using s_1 explicitly. We summarize all the considerations above in the following algorithm.

ALGORITHM 1 (*Version* 1). Computation of the trust region step s for given Δ, $0 < \Delta < \|J_D^+ f\|$.

Step 0: Choose a tolerance ε and an initial estimate $\lambda > 0$.

Step 1: Perform the transformations (13) and (18).

Step 2: Perform the transformation (20) and compute s_2 from (21).

Step 3: Compute $\|Ds\|^2$ from (22), set $\Phi(\lambda) = \|Ds\| - \Delta$.

Step 4: *If* $|\Phi(\lambda)| \leqq \varepsilon \Delta$

then accept λ, solve (17) for s_1, set

$$s = \left[\begin{array}{c} s_1 \\ \hline s_2 \end{array}\right].$$

else compute $\|\bar{R}^{-T} D^T Ds\|^2$ from (27), set $\Phi'(\lambda) = -\|\bar{R}^{-T} D^T Ds\|^2 / \|Ds\|$, update λ by using $\Phi(\lambda)$ and $\Phi'(\lambda)$, *goto* Step 2.

If not only J_1 but the whole J is taken as part of D we obtain Version 2.

Version 2. Set

$$D = D^{(2)} = \left[\begin{array}{c|c} J_1 & J_2 \\ \hline 0 & C \end{array}\right]$$

where C is a regular (l, l)-matrix.

Now the corresponding regularized linear least-squares problem has the form

$$\text{minimize} \left\| \left[\begin{array}{c|c} J_1 & J_2 \\ \hline \lambda^{1/2} J_1 & \lambda^{1/2} J_2 \\ \hline 0 & \lambda^{1/2} C \end{array}\right] \left[\begin{array}{c} s_1 \\ \hline s_2 \end{array}\right] - \left[\begin{array}{c} f \\ \hline 0 \\ \hline 0 \end{array}\right] \right\|. \tag{28}$$

After having transformed the residual of (28) by the orthogonal matrix (13′), the squared residual norm ρ^2 of the equivalent problem can be expressed as

$$\rho^2 = \|K_{11}s_1 + K_{12}s_2 - \tilde{g}_1\|^2 + \|K_{22}s_2 - \tilde{g}_2\|^2 + \lambda \|K_{11}s_1 + K_{12}s_2\|^2 + \lambda \|K_{22}s_2\|^2 + \lambda \|Cs_2\|^2.$$

By using the identity (15) we obtain

$$\begin{aligned} &\|K_{11}s_1 + K_{12}s_2 - \tilde{g}_1\|^2 + \lambda \|K_{11}s_1 + K_{12}s_2\|^2 \\ &\quad = \|(1+\lambda)^{1/2} K_{11}s_1 + (1+\lambda)^{1/2} K_{12}s_2 - (1+\lambda)^{-1/2} \tilde{g}_1\|^2 + \frac{\lambda}{1+\lambda} \|\tilde{g}_1\|^2 \end{aligned} \tag{29}$$

and

$$\|K_{22}s_2 - \tilde{g}_2\|^2 + \lambda \|K_{22}s_2\|^2 = \|(1+\lambda)^{1/2} K_{22}s_2 - (1+\lambda)^{-1/2} \tilde{g}_2\|^2 + \frac{\lambda}{1+\lambda} \|\tilde{g}_2\|^2.$$

Therefore, as in Version 1, (28) can be solved in two stages, namely, at first,

$$\text{minimize} \left\| \left[\begin{array}{c} (1+\lambda)^{1/2} K_{22} \\ \hline \lambda^{1/2} C \end{array}\right] s_2 - \left[\begin{array}{c} (1+\lambda)^{-1/2} \tilde{g}_2 \\ \hline 0 \end{array}\right] \right\|$$

and, secondly, solve the triangular system

$$K_{11}s_1 = \frac{1}{1+\lambda} \tilde{g}_1 - K_{12}s_2.$$

Obviously also in this version it is possible to express $\Phi(\lambda)$ and $\Phi'(\lambda)$ without explicit use of s_1 analogously to Version 1.

In the third version, (2) is first transformed into the equivalent problem

$$\text{minimize} \left\| \left[\begin{array}{c|c} K_{11} & K_{12} \\ \hline 0 & K_{22} \end{array}\right] \left[\begin{array}{c} s_1 \\ \hline s_2 \end{array}\right] - \left[\begin{array}{c} \tilde{g}_2 \\ \hline \tilde{g}_2 \end{array}\right] \right\|$$

by using the orthogonal factorization (13). Now the transformed part $(K_{11} | K_{12})$ is used as part of D. This leads to Version 3 of Algorithm 1.

Version 3. Set

$$D = D^{(3)} = \left[\begin{array}{c|c} K_{11} & K_{12} \\ \hline 0 & C \end{array}\right]$$

where C is a regular (l, l)-matrix. The corresponding regularized least-squares problem is

$$\text{minimize} \left\| \left[\begin{array}{c|c} K_{11} & K_{12} \\ \hline 0 & K_{22} \\ \hline \lambda^{1/2} K_{11} & \lambda^{1/2} K_{12} \\ \hline 0 & \lambda^{1/2} C \end{array}\right] \left[\begin{array}{c} s_1 \\ \hline s_2 \end{array}\right] - \left[\begin{array}{c} \tilde{g}_1 \\ \hline \tilde{g}_2 \\ \hline p \\ \hline 0 \end{array}\right] \right\|.$$

With (29) it follows that this problem can again be separated into two problems:

$$\text{minimize} \left\| \begin{bmatrix} K_{22} \\ \lambda^{1/2} C \end{bmatrix} s_2 - \begin{bmatrix} \tilde{g}_2 \\ 0 \end{bmatrix} \right\|$$

and

$$\text{solve } K_{11} s_1 = \frac{1}{1+\lambda} \tilde{g}_1 - K_{12} s_2.$$

Obviously this version differs only slightly from Version 2, and also here the values $\Phi(\lambda)$ and $\Phi'(\lambda)$ can be computed without forming s_1 explicitly.

The algorithms described above are all based on orthogonal transformations. An alternative realization is possible via the normal-equations approach, with the advantage of a lower operation count and the disadvantage of a smaller class of problems that can be handled in this way. Also then Φ and Φ' can be computed with low costs but now terms of order t^2 occur because each λ requires two backsubstitutions with a triangular (t, t)-matrix. For brevity we omit the details that can be readily found when the normal equations are solved by block elimination. If the partition of the Jacobian matrix is not known a priori, it can be found by truncated Cholesky factorization of $J^T J$ with pivoting on the diagonal analogously to the strategy described in the next section.

Let us finish this section with some remarks concerning the choice of C. The simplest choice is, of course, $C_k = I$ but it does not perform well for badly scaled problems. In the literature usually a diagonal $C_k = \operatorname{diag}(c_{k,i})$ is taken with $c_{k,i} = \|J_2(x^k)e^i\|$ if this norm is not zero. The choice

$$\begin{aligned} & c_{k,i} = \max\{c_{k-1,i}, \|J_2(x^k)e^i\|\} \quad \text{for } k > 0 \\ & c_{0,i} > 0, \end{aligned} \tag{30}$$

mentioned by Moré [8] works also in case of vanishing columns of J_2 and has some advantages with respect to convergence properties; see § 3. In the following we assume C_k to be chosen according to (30).

3. Convergence properties. Up to now we were only concerned with the choice of scaling matrices $D = D_k$ and the computation of trust region steps $s = s^k$ as essential parts of a trust region algorithm. In order to get a complete trust region algorithm some other parts have to be added.

ALGORITHM 2. Model trust region algorithm with nonstandard scaling matrices.

Step 0: Choose x^0, Δ_0, $\delta \in (0, 1)$, set $k = 0$.

Step 1: Compute the residual $f = f(x^k)$ and the Jacobian matrix $J = J(x^k)$.

Step 2: Choose the permutation matrix $P = P_k$ and the index $t = t_k \in \{1, \cdots, n\}$ such that $JP = (J_1 | J_2)$, where J_1 consists of t linearly independent columns.

Step 3: Choose $D = D_k$ according to one of the versions given in the previous section.

Step 4: Compute $s = s^k$ as an approximate solution of problem (2) by using Algorithm 1 (or the corresponding algorithm valid for Versions 2 or 3), set $s = Ps$.

Step 5: *If*

$$\frac{\|f(x^k)\|^2 - \|f(x^k - s^k)\|^2}{\|f(x^k)\|^2 - \|f(x^k) - J(x^k)s^k\|^2} > \delta \tag{31}$$

then $x^{k+1} = x^k - s^k$

else $x^{k+1} = x^k$.

Step 6: Update Δ_k, set $k = k+1$, *goto* Step 1.

A survey of trust region algorithms and related convergence results has been given by Moré [9]. Algorithm 2 above is a special case of the general trust region method described by Moré. If the function f, the scaling matrices D_k, and the stepsize strategy used in Step 6 satisfy the conditions formulated in the paper of Moré, then the algorithm converges globally in a sense explained there. Since the contribution of our paper is the choice of special nonstandard scaling matrices, the conditions on D_k are of special interest to us. These conditions are

$$\|J_k\bar{D}_k^{-1}\| \leqq b_1 \quad \text{and} \quad \|\bar{D}_k^{-1}\| \leqq b_2 \tag{32}$$

with constants $b_1, b_2 > 0$ where the scaling matrices $\bar{D}_k$ are assumed to be square and regular. Because our scaling matrices D_k are rectangular of type (p, n), $p \geqq n$, and of full rank, we express them as

$$D_k = U_k\bar{D}_k \tag{33}$$

where $\bar{D}_k$ is a regular (n, n)-matrix, and U_k is a (p, n)-matrix with $U_k^T U_k = I$. Then there holds

$$\|D_k s\| = \|\bar{D}_k s\| \quad \text{for all } s,$$

i.e., $\bar{D}_k$ yields the same scaling as D_k. In the following we construct the factorization (33) by using the information known from computing the step s and investigate when the conditions (32) are satisfied.

Version 1. From (13) we get

$$J_1 = Q_1 \begin{bmatrix} K_{11} \\ 0 \end{bmatrix}.$$

If Q_1 is partitioned according to $Q_1 = (Q_{11} | Q_{12})$, where Q_{11} has t columns, then we get $J_1 = Q_{11}K_{11}$ and, hence,

$$D = \left[\begin{array}{c|c} Q_{11}K_{11} & 0 \\ \hline 0 & C \end{array}\right]$$

so that (33) holds with

$$U = \left[\begin{array}{c|c} Q_{11} & 0 \\ \hline 0 & I \end{array}\right] \quad \text{and} \quad \bar{D} = \left[\begin{array}{c|c} K_{11} & 0 \\ \hline 0 & C \end{array}\right].$$

The terms that have to be bounded are

$$\|\bar{D}_k^{-1}\| = \left\| \left[\begin{array}{c|c} K_{11}^{-1} & 0 \\ \hline 0 & C^{-1} \end{array}\right] \right\|$$

and

$$\|J_k\bar{D}_k^{-1}\| = \|Q_1^T J_k\bar{D}_k^{-1}\| = \left\| \left[\begin{array}{c|c} I & K_{12}C^{-1} \\ \hline 0 & K_{22}C^{-1} \end{array}\right] \right\|.$$

Let us examine the second equation. Due to the definition of C (see (30)), all columns of the matrix occurring in the last term have Euclidean norm less than one. Thus, the Frobenius norm of that matrix is bounded by $n^{1/2}$, so $\|J_k\bar{D}_k^{-1}\| \leqq n^{1/2}$.

From the first equation we get

$$\|\bar{D}_k^{-1}\| = \max\{\|K_{11}^{-1}\|, \|C^{-1}\|\}.$$

Since $\|C_k^{-1}\|$ is bounded by construction (see again (30)), we have only to guarantee that K_{11}^{-1} is uniformly bounded. For doing this we have to consider two cases.

In some problems a fixed permutation matrix P and number t are known a priori such that $JP=(J_1|J_2)$, and $\|J_1^+\|=\|K_{11}^{-1}\|$ is uniformly bounded; see the following section for examples.

Other problems, however, have no natural partition leading to a block J_1 that has the desired property. Then $P=P_k$ and $t=t_k$ can be determined adaptively by using a QR factorization of J with interchange of columns such that

$$JP=Q\begin{bmatrix}K\\0\end{bmatrix} \quad \text{with } K=\begin{bmatrix}k_{11} & \cdots & k_{1n}\\ & \ddots & \vdots\\ 0 & & k_{nn}\end{bmatrix}$$

where Q is an orthogonal (m, m)-matrix and the relation

$$|k_{11}|\geqq|k_{22}|\geqq\cdots\geqq|k_{nn}| \tag{34}$$

holds. Now we choose

$$t=t_k=\max\{j\colon |k_{jj}|\geqq b_3|k_{11}|\} \tag{35}$$

with a given tolerance $b_3>0$. This choice of t implies

$$\|K_{11}^{-1}\|\leqq 2^{t-1}/(b_3|k_{11}|); \tag{36}$$

see Lawson and Hanson [6, formula 6.32], for example. The right-hand side of (36) is bounded if $|k_{11}|$ is bounded from below. Due to the strategy of permutation we have $|k_{11}|=\max\{\|Je^j\|\colon j=1,\cdots,n\}$ such that $|k_{11}|$ is bounded from below if and only if

$$\|J_k\|\geqq b_4 \tag{37}$$

with some $b_4>0$, i.e., if not all elements of the Jacobian matrix are simultaneously small.

Version 2. Here the matrix Q_1 from (13) is partitioned according to $Q_1=(\bar{Q}_{11}|\bar{Q}_{12})$, where $\bar{Q}_{11}$ has n columns. Further, let Q_2 be an orthogonal $(2l, 2l)$-matrix that realizes the transformation

$$Q_2^T\begin{bmatrix}K_{22}\\C\end{bmatrix}=\begin{bmatrix}\tilde{R}\\0\end{bmatrix} \tag{38}$$

with an upper-triangular (l, l)-matrix $\tilde{R}$. If Q_2 is partitioned into blocks with l columns $Q_2=(Q_{21}|Q_{22})$, then (33) is valid with

$$U=\left[\begin{array}{c|c}\bar{Q}_{11} & 0\\ \hline 0 & I_l\end{array}\right]\left[\begin{array}{c|c}I_t & 0\\ \hline 0 & Q_{21}\end{array}\right] \quad \text{and} \quad \bar{D}=\left[\begin{array}{c|c}K_{11} & K_{12}\\ \hline 0 & \tilde{R}\end{array}\right].$$

This leads to

$$\|\bar{D}_k^{-1}\|=\left\|\left[\begin{array}{c|c}K_{11}^{-1} & -K_{11}^{-1}K_{12}\tilde{R}^{-1}\\ \hline 0 & \tilde{R}^{-1}\end{array}\right]\right\|$$

and

$$\|J_k\bar{D}_k^{-1}\|=\|Q_1^TJ_k\bar{D}_k^{-1}\|=\left\|\left[\begin{array}{c|c}I & 0\\ \hline 0 & K_{22}\tilde{R}^{-1}\end{array}\right]\right\|.$$

Now

$$\|K_{22}\tilde{R}^{-1}\|\leqq\left\|\begin{bmatrix}K_{22}\tilde{R}^{-1}\\ C\tilde{R}^{-1}\end{bmatrix}\right\|=\left\|Q_2^T\begin{bmatrix}K_{22}\\C\end{bmatrix}\tilde{R}^{-1}\right\|=\left\|\begin{bmatrix}I\\0\end{bmatrix}\right\|=1$$

holds so that the first condition (32) is always satisfied. To bound $\bar{D}_k^{-1}$ we consider (38) and conclude

$$\|\tilde{R}x\|=\left\|\begin{bmatrix}\tilde{R}\\0\end{bmatrix}x\right\|=\left\|\begin{bmatrix}K_{22}\\C\end{bmatrix}x\right\|\geqq\|Cx\|$$

which implies $\|\tilde{R}^{-1}\| \leqq \|C^{-1}\|$. However, due to the choice of C according to (30), C^{-1} and hence $\tilde{R}^{-1}$ are uniformly bounded. Since K_{11} is constructed as in Version 1 of Algorithm 1, its inverse is bounded if (37) holds. It remains to show that the term $K_{12}\tilde{R}^{-1}$ is bounded. For doing this we consider $z^j = \tilde{R}e^j$. Then we get

$$\|z^j\| = \|\tilde{R}e^j\| = \left\| \begin{bmatrix} K_{22} \\ C \end{bmatrix} e^j \right\| \geqq \|Ce^j\| = c_j$$

$$\geqq \|K_{12}e^j\| = \|K_{12}\tilde{R}^{-1}z^j\|$$

for $j = 1, \cdots, l$. Thus, the columns of $K_{12}\tilde{R}^{-1}$ are bounded by 1, which yields $\|K_{12}\tilde{R}^{-1}\| \leqq l^{1/2}$.

Version 3. Here we have a square scaling matrix

$$D = \bar{D} = \left[\begin{array}{c|c} K_{11} & K_{12} \\ \hline 0 & C \end{array}\right],$$

i.e., we have the same situation as in Algorithm 1, Version 2 with $\tilde{R}$ replaced by C. Thus, we have to show that $K_{12}C^{-1}$ and $K_{22}C^{-1}$ are bounded which, however, follows immediately from the definition of C.

Summarizing the results obtained above we get the following assertion.

Assertion. (i) Let Algorithm 2 be realized with a partition $JP = (J_1 J_2)$ such that

$$\|J_1^+\| = \|K_{11}^{-1}\| \leqq b_5 \tag{39}$$

uniformly with $b_5 > 0$. Then the conditions (32) are satisfied.

(ii) The assumption (39) is fulfilled if J_1 and K_{11} are chosen adaptively according to (35), (36) provided that (37) holds.

We see that all the versions described deliver admissible scaling matrices under weak conditions on f, at least from a theoretical point of view.

In concluding this section we discuss the asymptotic behavior of the step s for $\Delta \to 0$, i.e., for $\lambda \to \infty$. From the normal equations belonging to (12) we get for Algorithm 1, Version 1 the explicit representation

$$\begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \frac{1}{1+\lambda} \left(\begin{array}{c|c} J_1^T J_1 & 1/(1+\lambda) J_1^T J_2 \\ \hline 1/(1+\lambda) J_2^T J_1 & 1/(1+\lambda) J_2^T J_2 + \lambda/(1+\lambda) C^T C \end{array} \right)^{-1} \begin{pmatrix} \nabla_1 g \\ \nabla_2 g \end{pmatrix}$$

where

$$\begin{pmatrix} \nabla_1 g \\ \nabla_2 g \end{pmatrix} = \nabla g = \begin{pmatrix} J_1^T f \\ J_2^T f \end{pmatrix}$$

is the gradient of the objective function. For increasing λ the step is damped and the vanishing off-diagonal blocks of the matrix lead to a decoupling of the system. The direction of s_1 turns to the Gauss–Newton direction $(J_1^T J_1)^{-1} \nabla_1 g$ and the direction of s_2 is the scaled gradient direction $(C^T C)^{-1} \nabla_2 g$.

In Version 3 there holds

$$\begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \frac{1}{1+\lambda} \left[\begin{array}{c|c} K_{11}^T K_{11} & K_{11}^T K_{12} \\ \hline K_{12}^T K_{11} & K_{12}^T K_{12} + 1/(1+\lambda) K_{22}^T K_{22} + \lambda/(1+\lambda) C^T C \end{array} \right]^{-1} \begin{bmatrix} \nabla_1 g \\ \nabla_2 g \end{bmatrix}.$$

For increasing λ the system does not decouple and the change of direction comes mainly from the term $(1/(1+\lambda)) K_{22}^T K_{22}$ in the southeast block of the matrix since $\lambda/(1+\lambda)$ is nearly equal to one for a wide range of λ. The asymptotic behavior of Algorithm 1, Version 2 differs only slightly from that of Version 3.

4. Practical least-squares problems with a priori partition of the Jacobian matrix.

4.1. Parameter estimation in explicit models with errors in the variables. Let a process be described by the explicit model equations

$$y = r(z, \Theta)$$

where $z \in R^{d(z)}$ and $y \in R^{d(y)}$ are the state variables. The vector $\Theta \in R^{d(\Theta)}$ of parameters is to be estimated by fitting the measured data $\tilde{z}^i$, $\tilde{y}^i$, $i = 1, \cdots, M$. In classic regression analysis, the $\tilde{z}^i$ are supposed to be free of errors, and the estimate of Θ is determined as the solution of the nonlinear least-squares problem

$$\text{minimize} \frac{1}{2} \sum_{i=1}^{M} \| r(\tilde{z}^i, \Theta) - \tilde{y}^i \|^2. \tag{40}$$

Here only the errors of $\tilde{y}^i$ are taken into consideration, whereas the errors of the $\tilde{z}^i$ are neglected. Sometimes the measurement errors in z^i are of the same magnitude as errors in $\tilde{y}^i$. Then we speak of models with errors in the variables. In order to take into consideration also the errors in $\tilde{z}^i$, the usual criterion (40) should be replaced by the more general criterion

$$\text{minimize} \frac{1}{2} \sum_{i=1}^{M} \{ \| r(z^i, \Theta) - \tilde{y}^i \|^2 + \| z^i - \tilde{z}^i \|^2 \}. \tag{41}$$

For simplicity of notation, in (41) all the deviations are equally weighted. However, we can also allow arbitrary weighting matrices.

Problem (41) is a large-scale problem with $M \cdot d(z) + d(\Theta)$ variables $z^1, \cdots, z^M$ and Θ where z^i are the estimated states and M is the possibly large number of measurements. With the notation

$$f(Z, \Theta) = \begin{bmatrix} z^1 - \tilde{z}^1 \\ \vdots \\ z^M - \tilde{z}^M \\ r(z^1, \Theta) - \tilde{y}^1 \\ \vdots \\ r(z^M, \Theta) - \tilde{y}^M \end{bmatrix}, \qquad Z = \begin{bmatrix} z^1 \\ z^2 \\ \vdots \\ z^M \end{bmatrix}$$

problem (41) reads

$$\text{minimize} \tfrac{1}{2} \| f(Z, \Theta) \|^2.$$

The Jacobian matrix of $f(Z, \Theta)$ has the form described in the introduction; see (8) and the remarks there. For $P = I$ and $t = M \cdot d(z)$ the condition $\| J_1^+ \| = \| K_{11}^{-1} \| \leqq 1$ is satisfied instead of (36). In Schwetlick and Tiller [15] an efficient algorithm for solving problem (41) was proposed which uses the same steps $s(\lambda)$ as in Version 3 of this paper. However, the steps are controlled by λ instead of Δ.

4.2. Surface fitting. Let a surface in R^3 be given in parametric form

$$y = r(u, v, \Theta)$$

where $y \in R^3$ is a point of the surface, u and v are the scalar parameters belonging to the point y, and Θ is a vector of parameters specifying the surface. Then the surface is fitted to the data $\tilde{y}^i$, $i = 1, \cdots, M$, by solving the nonlinear least-squares problem

$$\text{minimize} \sum_{i=1}^{M} \| r(u_i, v_i, \Theta) - \tilde{y}^i \|^2. \tag{42}$$

With

$$f(u_1, v_1, \cdots, u_M, v_M, \Theta) = \begin{pmatrix} r(u_1, v_1, \Theta) - \tilde{y}^1 \\ \vdots \\ r(u_M, v_M, \Theta) - \tilde{y}^M \end{pmatrix},$$

(42) reads

$$\text{minimize } \tfrac{1}{2}\|f\|^2.$$

The Jacobian matrix of f has the structure

$$J = (J_1 J_2) = \begin{bmatrix} \square & & & \square \\ & \square & & \square \\ & & \ddots & \vdots \\ & & \square & \square \end{bmatrix}$$

where the blocks in J_1 are the $(3, 2)$-Jacobian matrices of r with respect to u and v, and the blocks in J_2 are the $(3, N(\Theta))$-Jacobian matrices of r with respect to Θ; all the blocks computed at (u_i, v_i), $i = 1, \cdots, M$.

Now it is a natural assumption for this problem that the surface is regular in the following sense: For any (u, v) the $(3, 2)$-Jacobian matrix

$$G = \left[\frac{\partial r}{\partial u}, \frac{\partial r}{\partial v}\right]$$

has full rank. If this is even true uniformly in the sense of $\|G^+\| \leqq b_5$, then (39) is satisfied and our nonstandard scaling matrices can be used. However, in this problem no good initial estimates (u_i^0, v_i^0) to the surface parameters are available, in general.

5. Some numerical examples.

5.1. Orthogonal distance regression. In a first set of examples, nonstandard scaling matrices have been applied to orthogonal distance regression problems described in § 4.1. The set consists of four one-dimensional model equations

$$y = r(z, \Theta) \quad \text{with } d(y) = d(z) = 1, \quad d(\Theta) = 3.$$

Simulated data $(\tilde{z}_i, \tilde{y}_i)$ have been generated from exact data

$$z_i^* = -3 + (i-1)/5, \quad y_i^* = r(z_i, \Theta), \quad i = 1, \cdots, 30$$

belonging to the true parameter $\Theta^* = (1, 1, 1)^T$ according to

$$\tilde{z}_i = z_i^*(1 + \delta_i), \qquad \tilde{y}_i = y_i^*(1 + \varepsilon_i)$$

where ε_i, δ_i are random errors uniformly distributed on $[-0.1, 0.1]$. The problems have been solved by the trust region method of § 2 with nonstandard scaling matrices of Version 1 and 3 (denoted by NS1 and NS3, respectively) and, for comparison, by the trust region method with standard diagonal scaling (abbreviated as STR) as well as by a damped Gauss–Newton method (GN). The step control has been done as proposed by Moré [8]. Moreover, each problem is treated three times with different initial values

$\Theta^0 = \alpha\Theta^*$ for the parameters. The objective of choosing α was to get convergence via regularized steps in the first part of iteration. The initial values for z_i are always the data, i.e., $z_i^0 = \tilde{z}_i$. For each model equation, the Tables 1-4 contain in the columns the number of iterations needed by the corresponding method for the different values of α. In case of divergence a hyphen is written.

It turns out that the iteration count of the nonstandard methods is about the same as for the standard trust region method. So the new modifications seem to be appropriate for reducing the overall costs in solving orthogonal distance regression problems.

5.2. Adaptive choice of scaling matrices for general least-squares problems. In a second set we have investigated general, nonstructured least-squares problems. For a given, fixed number t, $t \in \{0, \cdots, n\}$, the partition (11) has been defined by using a QR decomposition of J with column interchange along the lines of LINPACK routines (cf. Dongarra et al. [3]). The value $t = 0$ corresponds to standard diagonal scaling, whereas $t = n$ gives the damped Gauss-Newton method with stepsize control via $\|Ds\|$. Therefore, the parameter t allows a discrete variation between the extremal methods,

TABLE 1
$y = \Theta_1 \sin(\Theta_2 z) \cdot \exp(\Theta_3 z)$.

α	STR	NS1	NS3	GN
0.1	6	6	6	—
2	8	16	16	9
3	11	8	8	13

TABLE 2
$y = \sin(\Theta_1 z)/(\Theta_2 z^2 + \Theta_3)$.

α	STR	NS1	NS3	GN
0.1	4	4	4	4
2	6	6	6	7
3	21	21	21	16

TABLE 3
$y = \Theta_1 \sin(\Theta_2 z + \Theta_3)$.

α	STR	NS1	NS3	GN
0.1	7	8	8	—
0.2	9	8	8	18
2	10	10	10	9

TABLE 4
$y = \Theta_1 \exp(\Theta_2 z) + \Theta_3$.

α	STR	NS1	NS3	GN
0.05	8	7	7	34
0.1	10	8	8	24
0.5	6	6	6	6

namely, standard trust region and Gauss–Newton. We would expect that for t nearby n, i.e., with somewhat more computational effort per step than required by the Gauss–Newton method, the new methods should take fewer iterations than damped Gauss–Newton.

The presented problems are a subset of the investigated problems and they will illustrate what happens when general least-squares problems are solved. All the problems are well known from literature.

In Tables 5–9, the number of iterations is given for varying values of t and for the versions NS1 and NS3 (cf. § 5.1).

Unfortunately, the expected reduction in the number of iterations sometimes does not take place. It seems that the matrices generated by the first t columns of JP are not always so suitable for measuring the trust region step.

TABLE 5
Variably dimensioned function (see Moré et al. [10]). $n = 30$.

t	0	5	10	15	20	25	30
NS1	13	14	14	15	15	17	29
NS3	13	31	39	29	30	29	29

TABLE 6
Example II in Wright and Holt [17]. $n = 12$.

t	0	1	2	3	6	9	12
NS1	21	22	26	21	26	26	26
NS3	21	30	28	44	26	26	26

TABLE 7
Generalized Rosenbrock function (cf. Gill and Murray [5]). $n = 10$.

t	0	2	4	6	8	10
NS1	23	12	12	27	38	13
NS3	23	22	21	21	29	13

TABLE 8
Generalized singular function (cf. Conn, Gould, and Toint [2]). $n = 40$.

t	0	8	16	24	32	40
NS1	18	21	21	21	21	26
NS3	18	26	26	26	26	26

TABLE 9
Exponential functions with 11 parameters (see Osborne [12]).

t	0	1	3	5	7	9	11
NS1	10	10	9	8	16	8	11
NS3	10	11	11	11	12	9	11

Summarizing our restricted numerical experience we would recommend the use of nonstandard scaling matrices for large problems with an a priori known structure, where they can essentially reduce the computational costs. For general nonstructured problems the method so far cannot be recommended. In this case further investigations seem to be necessary.

REFERENCES

[1] P. T. BOGGS, R. H. BYRD, AND R. B. SCHNABEL, *A stable and efficient algorithm for nonlinear orthogonal distance regression*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 1052–1078.

[2] A. R. CONN, N. I. M. GOULD, AND PH. L. TOINT, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Report NR 86/3, Namur, 1986.

[3] 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.

[4] D. M. GAY, *Computing optimal locally constrained steps*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 186–197.

[5] P. E. GILL AND W. MURRAY, *Conjugate gradient methods for large scale nonlinear optimization*, Report TR SOL 79-15, 1979.

[6] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[7] D. W. MARQUARDT, *An algorithm for least squares estimation of nonlinear parameters*, SIAM J. Appl. Math., 11 (1963), pp. 431–441.

[8] J. J. MORÉ, *The Levenberg–Marquardt algorithm: Implementation and theory*, Lecture Notes in Mathematics 630, G. A. Watson, ed., Springer-Verlag, New York, 1978, pp. 105–116.

[9] ———, *Recent developments in algorithms and software for trust region methods*, Mathematical Programming—The State of the Art, A. Bachem, M. Grötschel, and B. Korte, eds., Springer-Verlag, Berlin, 1983, pp. 258–287.

[10] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM, *Testing unconstrained optimization software*, ACM Trans. Math. Software, Vol. 6, No. 1, 1981, pp. 17–41.

[11] J. J. MORÉ AND D. C. SORENSEN, *Computing a trust region step*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 553–572.

[12] M. R. OSBORNE, *Some aspects of non-linear least squares calculations*, in Numerical Methods for Nonlinear Optimization, F. Lootsma, ed., Academic Press, New York, 1972, pp. 171–189.

[13] H. SCHWETLICK, *Zur Konvergenz regularisierter Gauss–Newton Verfahren*, Ž. Vyčisl. Mat. i Mat. Fiz., 13 (1973), pp. 1371–1382.

[14] ———, *Numerische Lösung nichtlinearer Gleichungen*, VEB Deutscher Verlag der Wissenschaften, Berlin, 1979.

[15] H. SCHWETLICK AND V. TILLER, *Numerical methods for estimating parameters in nonlinear models with errors in the variables*, Technometrics, 27 (1985), pp. 17–24.

[16] G. A. WATSON, *The solution of generalized least squares problems*, International Series of Numerical Mathematics, Vol. 75, Birkhäuser-Verlag, Basel, 1985.

[17] S. J. WRIGHT AND J. N. HOLT, *An inexact Levenberg–Marquardt method for large sparse nonlinear least squares*, J. Austral. Math. Soc. Ser. B26, (1986), pp. 387–403.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 671-685, July 1989
© 1989 Society for Industrial and Applied Mathematics
004

NUMERICAL DETERMINATION OF AN EMANATING BRANCH OF HOPF BIFURCATION POINTS IN A TWO-PARAMETER PROBLEM*

DIRK ROOSE† AND BART DE DIER‡

Abstract. In a two-parameter problem a branch of Hopf bifurcation points can bifurcate from a branch of simple turning points of the steady state problem, at a point for which the Fréchet derivative has a double eigenvalue zero with a one-dimensional nullspace. It is indicated how the origin of a branch of Hopf points can be detected during the continuation of a branch of simple turning points.

Further, an augmented system of equations is presented, for which this "origin for Hopf bifurcation" is an isolated solution. If the steady state problem is described by a system of algebraic equations, Newton's method for the solution of the augmented system can be implemented very efficiently. The authors also discuss switching to the branch of Hopf points.

Results are given for the one-dimensional "Brusselator" model, a system of four partial differential equations.

Key words. bifurcation, nonlinear dynamics, continuation, differential equations

AMS(MOS) subject classifications. 65J15, 65P05

1. Introduction.

We consider nonlinear evolution equations of the following form:

$$\frac{du}{dt} = F(u, \lambda, \gamma) \qquad F: D \subset X \times \mathbb{R} \times \mathbb{R} \to X \tag{1}$$

where X is a real Hilbert space, and λ and γ denote real parameters. Although the theoretical results in this paper are valid in the general setting of Hilbert spaces, we will assume $X = \mathbb{R}^n$, so that (1) represents a system of ordinary differential equations or a discretization of a system of partial differential equations. For the analysis of a system of the form (1), we are often interested in a (qualitative) description of the solution set as a function of the parameters.

For the one-parameter problem

$$\frac{du}{dt} = G(u, \lambda) = F(u, \lambda, \gamma^f) \qquad (\gamma^f \text{ fixed}) \tag{2}$$

branches of steady state solutions, satisfying $G(u, \lambda) = 0$, can be determined numerically by a continuation procedure (see, e.g., [18], [22], [23]). These branches can contain "critical points" such as turning points, regular bifurcation points, and Hopf bifurcation points. At these points the multiplicity of the solution set and the stability of the solutions can change. Hopf bifurcation occurs when a branch of time-periodic solutions bifurcates from the branch of steady-state solutions.

In order to provide more insight into the solution set of the two-parameter problem (1), it can be useful to determine paths of turning points, Hopf bifurcation points, and (in some cases) regular bifurcation points of $F(u, \lambda, \gamma) = 0$. This can be done by a continuation procedure, applied to a suitable determining system, starting from a known "critical point" (see, e.g., [15], [29], [30]).

* Received by the editors May 4, 1987; accepted for publication (in revised form) August 4, 1988.

† Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3030 Leuven, Belgium.

‡ Department of Chemical Engineering, Katholieke Universiteit Leuven, Decroylaan 2, B-3030 Leuven, Belgium.

A branch of Hopf bifurcation points can bifurcate from a branch of turning points and in this paper we present a method for the computation of the origin of such a branch of Hopf points. Starting from these "origins for Hopf bifurcation" we can easily compute branches of Hopf points. The knowledge of these origins also allows the investigation of "global Hopf bifurcation."

In the next section we recall some theoretical results on Hopf bifurcation in a one-parameter problem. In § 3 we give the conditions for an emanating branch of Hopf points in a two-parameter problem. In § 4 we introduce a numerical method for the calculation of an origin of such a branch and in § 5 we indicate how this method can be implemented efficiently. In § 6 we discuss how we can proceed to determine the emanating branch of Hopf points. Finally, in § 7 we illustrate our method on a model problem, i.e., the one-dimensional "Brusselator" model.

2. Hopf bifurcation in one-parameter problems. Sufficient conditions for the occurrence of Hopf bifurcation in a one-parameter problem of the form (2) are [11], [19]:

(H1) There exists a smooth branch of steady state solutions $u^{(s)}(\lambda)$ with $u^{(s)}(\lambda^h) = u^h$.

(H2) G is sufficiently smooth in a neighbourhood of (u^h, λ^h).

(H3) The Jacobian matrix $G_u(u^{(s)}(\lambda), \lambda)$ with $\lambda \in (\lambda^h - \varepsilon, \lambda^h + \varepsilon)$ has a pair of simple complex conjugate eigenvalues $\mu(\lambda)$, $\overline{\mu(\lambda)}$ such that $\mu(\lambda) = \alpha(\lambda) \pm i\omega(\lambda)$ with $\omega(\lambda^h) = \omega^h > 0$, $\alpha(\lambda^h) = 0$ and $\alpha'(\lambda^h) \neq 0$.

(H4) $G_u(u^h, \lambda^h)$ has no eigenvalues of the form $ki\omega^h$, $k \in \{0, 2, 3, 4, \cdots\}$.

The steady state solution (u^h, λ^h) is called a Hopf bifurcation point.

Several methods are proposed in the literature for the numerical determination of a Hopf bifurcation point. A classification and references can be found in [24]. All these iterative methods require a sufficiently good estimate of the Hopf point as a starting point to insure convergence. The performance and the reliability of some methods with respect to the starting point is investigated in [12] and [25].

However, the detection of a Hopf bifurcation point during a continuation process along a branch of steady state solutions is a much more difficult problem than the detection of a turning point or regular bifurcation point. At the latter points the Jacobian matrix $G_u(u, \lambda)$ is singular, which can easily be detected during continuation [15], [18], [22]. At a Hopf point (u^h, λ^h) the matrix $[G_u(u^h, \lambda^h) - i\omega^h]$ is singular but ω^h is unknown a priori.

Jepson [14] discusses four techniques for detecting a Hopf point during continuation:

(a) The first method is based on the calculation of all eigenvalues of $G_u(u, \lambda)$ at each continuation point.

(b) At each continuation point, the Routh-Hurwitz theorem can be used to determine the number of eigenvalues of $G_u(u, \lambda)$ with positive real part.

(c) Starting from a perturbation of the steady state solution at each continuation point, the solution of the time-dependent problem can be computed to detect a change in the stability of the steady state solutions along the branch.

(d) The last technique, the "multiply shifted power method," can be useful if the eigenvalues of $G_u(u, \lambda)$ lie in a few known clusters.

In contrast with the last two methods, the first two techniques are generally applicable. However they are very expensive compared to the cost of the continuation, especially when the Jacobian matrix has large dimension and is sparse or banded.

3. The origin of a branch of Hopf points in a two-parameter problem. Since Hopf bifurcation points are structurally stable in a one-parameter problem, branches of Hopf points can exist in a two-parameter problem of the form (1). Such a branch of Hopf points $(u^h, \lambda^h, \gamma^h)$ can be computed by a continuation procedure applied to a determining system for Hopf points, starting from a known Hopf point (see, e.g., [15], [25]). However, as indicated in the previous section, the calculation of an initial Hopf bifurcation point can be very difficult or expensive.

Guckenheimer and Holmes [10, pp. 364–374] analyse the unfolding of a degenerate vector field with normal form

$$\frac{dx}{dt} = y, \quad \frac{dy}{dt} = x^2 + xy \quad (x, y \in \mathbb{R}). \tag{3}$$

The analysis shows that in a two-parameter problem of the form (1), a branch of Hopf bifurcation points emanates from a steady state solution $(u^o, \lambda^o, \gamma^o)$, for which the Jacobian matrix $F_u(u^o, \lambda^o, \gamma^o)$ has a double eigenvalue zero and a one-dimensional nullspace. Also a branch of simple quadratic turning points passes through $(u^o, \lambda^o, \gamma^o)$. Thus at $(u^o, \lambda^o, \gamma^o)$ a branch of Hopf points bifurcates from the branch of quadratic turning points and the branching is one sided. (The unfolding of this singularity is also studied by other authors; see [7] and [10] for references.)

We will call such a steady state solution $(u^o, \lambda^o, \gamma^o)$ an "*origin for Hopf bifurcation*" or, following Fiedler and Kunkel [7], [8], a "*B-point.*"

Thus at an origin for Hopf bifurcation the Riesz index of the eigenvalue zero of $F_u(u^o, \lambda^o, \gamma^o)$ is two [13], and the Jordan canonical form of this Jacobian matrix contains a Jordan block of order two. It follows that the right and left nullvectors of $F_u(u^o, \lambda^o, \gamma^o)$, denoted by ϕ^o and ψ^o, are orthogonal [13], i.e.,

$$F_u(u^o, \lambda^o, \gamma^o)\phi^o = 0, \qquad \|\phi^o\| = 1, \tag{4a}$$

$$F_u^T(u^o, \lambda^o, \gamma^o)\psi^o = 0, \qquad \|\psi^o\| = 1, \tag{4b}$$

$$\psi^{oT}\phi^o = 0. \tag{5}$$

Then the Jacobian matrix $F_u(u^o, \lambda^o, \gamma^o)$ also has a generalized eigenvector ζ^o satisfying

$$F_u(u^o, \lambda^o, \gamma^o)\zeta^o = \phi^o, \qquad \phi^{oT}\zeta^o = 0. \tag{6}$$

Note also that other normalization conditions for ζ^o can be used [13] and that the range R of the Jacobian matrix can be expressed as

$$R(F_u(u^o, \lambda^o, \gamma^o)) = \{y \in \mathbb{R}^n \mid \psi^{oT}y = 0\}. \tag{7}$$

Assuming that the origin for Hopf bifurcation is a simple quadratic turning point with respect to λ, the following conditions must also be satisfied (see, e.g., [20], [29], [30]):

$$F_\lambda(u^o, \lambda^o, \gamma^o) \notin R(F_u(u^o, \lambda^o, \gamma^o)) \Leftrightarrow \psi^{oT}F_\lambda(u^o, \lambda^o, \gamma^o) \neq 0, \tag{8}$$

$$\psi^{oT}F_{uu}(u^o, \lambda^o, \gamma^o)\phi^o\phi^o \neq 0. \tag{9}$$

We now derive some formulas we will use in the next section. In a two-parameter problem $F(u, \lambda, \gamma) = 0$ branches of simple quadratic turning points with respect to λ can exist, which can be parametrized by γ in the neighbourhood of each quadratic turning point [29]. Along the branch of turning points passing through the origin for Hopf bifurcation $(u^o, \lambda^o, \gamma^o)$ and parametrized by γ, the following identities hold:

$$F(u(\gamma), \lambda(\gamma), \gamma) = 0, \tag{10a}$$

$$F_u(u(\gamma), \lambda(\gamma), \gamma)\phi(\gamma) = 0, \tag{10b}$$

$$\phi(\gamma)^T\phi(\gamma) = 1, \tag{10c}$$

$$\psi(\gamma)^T F_u(u(\gamma), \lambda(\gamma), \gamma) = 0, \tag{10d}$$

$$\psi(\gamma)^T\psi(\gamma) = 1. \tag{10e}$$

Here $\phi(\gamma)$ and $\psi(\gamma)$ denote, respectively, the right and left nullvectors of $F_u(u(\gamma), \lambda(\gamma), \gamma)$. Differentiation of these identities along the branch of turning points and evaluation at $(u^o, \lambda^o, \gamma^o)$ gives

$$F_u^o u'(\gamma^o) + F_\lambda^o \lambda'(\gamma^o) + F_\gamma^o = 0, \tag{11a}$$

$$F_{uu}^o \phi^o u'(\gamma^o) + F_{u\lambda}^o \phi^o \lambda'(\gamma^o) + F_{u\gamma}^o \phi^o + F_u^o \phi'(\gamma^o) = 0, \tag{11b}$$

$$\phi^{oT}\phi'(\gamma^o) = 0, \tag{11c}$$

$$\psi^{oT}F_{uu}^o u'(\gamma^o) + \psi^{oT}F_{u\lambda}^o \lambda'(\gamma^o) + \psi^{oT}F_{u\gamma}^o + \psi'(\phi^0)^T F_u^o = 0, \tag{11d}$$

$$\psi^{oT}\psi'(\gamma^o) = 0. \tag{11e}$$

Here the superscript ^{o} denotes that the derivatives are evaluated at $(u^o, \lambda^o, \gamma^o)$. The matrix

$$A = \begin{bmatrix} F_u^o & F_\lambda^o & 0 \\ F_{uu}^o \phi^o & F_{u\lambda}^o \phi^o & F_u^0 \\ 0 & 0 & 2\phi^{oT} \end{bmatrix} \tag{12}$$

is nonsingular [20], [29] and from (11a)-(11c) it follows that

$$\begin{bmatrix} u'(\gamma^0) \\ \lambda'(\gamma^0) \\ \phi'(\gamma^o) \end{bmatrix} = -A^{-1} \begin{bmatrix} F_\gamma^o \\ F_{u\lambda}^o \phi^o \\ 0 \end{bmatrix}. \tag{13}$$

4. The calculation of an origin of Hopf bifurcation. Based on the theoretical results of the previous section, the following procedure can be used to compute the origin of a branch of Hopf points in a two-parameter problem:

—Determine branches of quadratic turning points by continuation applied to a suitable determining system [15], [30].

—During the continuation process, calculate at each quadratic turning point $(u^{t_i}, \lambda^{t_i}, \gamma^{t_i})$

$$c^{t_i} = \cos(\psi^{t_i}, \phi^{t_i}) \tag{14}$$

where ϕ^{t_i} and ψ^{t_i} are the right and left nullvectors of $F_u^{t_i}$, which can be computed by inverse iteration. These nullvectors must be scaled such that ${\phi^{t_{i-1}}}^T\phi^{t_i} > 0$ and ${\psi^{t_{i-1}}}^T\psi^{t_i} > 0$. This ensures that ϕ^{t_i} and ψ^{t_i} lie in (nearly) the same direction as $\phi^{t_{i-1}}$ and $\psi^{t_{i-1}}$, respectively; i.e., the nullvectors ϕ^t and ψ^t change continuously during the continuation process.

—If $c^{t_{i-1}}c^{t_i} < 0$, an origin for Hopf bifurcation must lie between these two consecutive continuation points (see (5)) and can be computed by solving the determining system given below.

Consider the augmented system

$$C(y) = 0, \qquad C : \mathbb{R}^{3n+2} \to \mathbb{R}^{3n+2} \tag{15a}$$

with $y=(u, \lambda, \phi, \zeta, \gamma)$ and

$$C(u, \lambda, \phi, \zeta, \gamma)=\begin{cases} F(u, \lambda, \gamma), \\ F_u(u, \lambda, \gamma)\phi, \\ \phi^T\phi-1, \\ F_u(u, \lambda, \gamma)\zeta-\phi, \\ \zeta^T\phi. \end{cases} \tag{15b}$$

The system $C(y)=0$ is a regular determining system for an origin for Hopf bifurcation $(u^o, \lambda^o, \gamma^o)$ since $(u^o, \lambda^o, \phi^o, \zeta^o, \gamma^o)$ is a solution of (15) and since the following theorem holds.

THEOREM. *Let $(u^o, \lambda^o, \gamma^o)$ be an origin for Hopf bifurcation lying on a branch of simple quadratic turning points $(u(\gamma), \lambda(\gamma), \gamma)$ with respect to λ of $F(u, \lambda, \gamma)=0$. Then the Jacobian matrix $C_y(y^o)$ of (15) evaluated at $y^o=(u^o, \lambda^o, \phi^o, \zeta^o, \gamma^o)$ is nonsingular if*

$$\left.\frac{d(\psi(\gamma)^T\phi(\gamma))}{d\gamma}\right|_{\gamma=\gamma^o}\neq 0 \tag{16}$$

where $\phi(\gamma)$ and $\psi(\gamma)$ denote, respectively, the right and left nullvectors of $F_u(u(\gamma), \lambda(\gamma), \gamma)$.

Proof.

$$C_y(y^o)=\begin{bmatrix} A & B \\ C^T & D \end{bmatrix}=\left[\begin{array}{ccc|cc} F_u^o & F_\lambda^o & 0 & 0 & F_\gamma^o \\ F_{uu}^o\phi^o & F_{u\lambda}^o\phi^o & F_u^o & 0 & F_{u\gamma}^o\phi^o \\ 0 & 0 & 2\phi^{oT} & 0 & 0 \\ \hline F_{uu}^o\zeta^o & F_{u\lambda}^o\zeta^o & -I & F_u^o & F_{u\gamma}^o\zeta^o \\ 0 & 0 & \zeta^{oT} & \phi^{oT} & 0 \end{array}\right]. \tag{17}$$

Based on a well-known lemma of Keller [18], $C_y(y^o)$ is nonsingular if and only if $D-C^TA^{-1}B$ is nonsingular. From (13) it follows that

$$A^{-1}B=\begin{bmatrix} 0 & -u'(\gamma^o) \\ 0 & -\lambda'(\gamma^o) \\ 0 & -\phi'(\gamma^o) \end{bmatrix} \tag{18}$$

where $u'(\gamma^o)$, $\lambda'(\gamma^o)$, and $\phi'(\gamma^o)$ are the derivatives of $u(\gamma)$, $\lambda(\gamma)$, and $\phi(\gamma)$ along the branch of quadratic turning points, evaluated at the origin of Hopf bifurcation. Thus

$$D-C^TA^{-1}B=\begin{bmatrix} F_u^o & F_{u\gamma}^o\zeta^o+F_{uu}^o\zeta^o u'(\gamma^o)+F_{u\lambda}^o\zeta^o\lambda'(\gamma^o)-\phi'(\gamma^o) \\ \phi^{oT} & \zeta^{oT}\phi'(\gamma^o) \end{bmatrix}. \tag{19}$$

Since $\dim(N(F_u^o))=1$, the same lemma of Keller states that this matrix is nonsingular if and only if

$$\dim R(F_{u\gamma}^o\zeta^o+F_{uu}^o\zeta^o u'(\gamma^o)+F_{u\lambda}\zeta^o\lambda'(\gamma^o)-\phi'(\gamma^o))=1, \tag{20a}$$

$$\dim R(\phi^{oT})=1, \tag{20b}$$

$$N(F_u^o)\cap N(\phi^{oT})=0, \tag{20c}$$

$$R(F_u^o)\cap R(F_{u\gamma}^o\zeta^o+F_{uu}^o\zeta^o u'(\gamma^o)+F_{u\lambda}^o\zeta^o\lambda'(\gamma^o)-\phi'(\gamma^o))=0. \tag{20d}$$

Clearly, conditions (20b), (20c) are satisfied. We now show that the conditions (20a) and (20d) are satisfied if (16) holds. Indeed, in this case

$$\left.\frac{d(\psi(\gamma)^T\phi(\gamma))}{d\gamma}\right|_{\gamma=\gamma^o}=\psi'(\gamma^o)^T\phi^o+\psi^{oT}\phi'(\gamma^o)\neq 0 \tag{21}$$

or

$$\psi'(\gamma^o)^T F_u^o \zeta + \psi^{oT} \phi'(\gamma^o) \neq 0. \tag{22}$$

Using (11d), this can be written as

$$\psi^{oT}(F_{u\gamma}^o \zeta^o + F_{uu}^o \zeta^o u'(\gamma^o) + F_{u\lambda}^o \zeta^o \lambda'(\gamma^o) - \phi'(\gamma^o)) \neq 0. \tag{23}$$

Thus (20a) must hold and also (20d), since this last condition is equivalent to (23). □

Note that condition (16) only means that the inner product $\psi(\gamma)^T \phi(\gamma)$ must be nonzero along the branch of quadratic turning points for $\gamma \in [\gamma^o - \varepsilon, \gamma^o + \varepsilon] \backslash \gamma^o$.

5. Implementation of Newton's method. We now describe some efficient procedures to implement Newton's method applied to the augmented system (15):

$$C_y(y^{(k)}) \Delta y = -C(y^{(k)}), \tag{24a}$$

$$y^{(k+1)} = y^{(k)} + \Delta y, \qquad k = 1, 2, \cdots \tag{24b}$$

where $y^{(k)} = (u^{(k)}, \lambda^{(k)}, \phi^{(k)}, \zeta^{(k)}, \gamma^{(k)})^T$ and $\Delta y = (\Delta u, \Delta\lambda, \Delta\phi, \Delta\zeta, \Delta\gamma)^T$.

We can take advantage of the structure of the Jacobian matrix $C_y(y^{(k)})$ to reduce the computational work for the calculation of the Newton correction Δy.

We first show how Δy can be computed by solving only systems with the matrix $F_u^{(k)}$ (Procedure A). However, this procedure will be unstable, since $F_u^{(k)}$ becomes singular at an origin for Hopf bifurcation. We then discuss two stable variants of the algorithm, which require additional computational work and programming effort (Procedures B and C). Further we indicate how sufficiently good starting values $\phi^{(1)}$ and $\zeta^{(1)}$ can be obtained for the Newton iteration (24).

PROCEDURE A. Equation (24a) can be written as follows:

$$F_u^{(k)} \Delta u + \Delta\lambda F_\lambda^{(k)} + \Delta\gamma F_\gamma^{(k)} = -F^{(k)}, \tag{25a}$$

$$F_{uu}^{(k)} \phi^{(k)} \Delta u + \Delta\lambda F_{u\lambda}^{(k)} \phi^{(k)} + \Delta\gamma F_{u\gamma}^{(k)} \phi^{(k)} + F_u^{(k)} \Delta\phi = -F_u^{(k)} \phi^{(k)}, \tag{25b}$$

$$F_{uu}^{(k)} \zeta^{(k)} \Delta u + \Delta\lambda F_{u\lambda}^{(k)} \zeta^{(k)} + \Delta\gamma F_{u\gamma}^{(k)} \zeta^{(k)} - \Delta\phi + F_u^{(k)} \Delta\zeta = -F_u^{(k)} \zeta^{(k)} + \phi^{(k)}, \tag{25c}$$

$$2\phi^{(k)T} \Delta\phi = -\phi^{(k)T} \phi^{(k)} + 1, \tag{25d}$$

$$\zeta^{(k)T} \Delta\phi + \phi^{(k)T} \Delta\zeta = -\phi^{(k)T} \zeta^{(k)} \tag{25e}$$

where the superscript $^{(k)}$ denotes that F and the derivatives are evaluated at $(u^{(k)}, \lambda^{(k)}, \gamma^{(k)})$. Equation (25a) can be solved in terms of $\Delta\lambda$ and $\Delta\gamma$, i.e.,

$$\Delta u = -p_1 - \Delta\lambda p_2 - \Delta\gamma p_3 \tag{26}$$

by solving the systems

$$A p_1 = F^{(k)}; \quad A p_2 = F_\lambda^{(k)}; \quad A p_3 = F_\gamma^{(k)} \tag{27}$$

with

$$A = F_u^{(k)} \in \mathbb{R}^{n \times n}. \tag{28}$$

Using (26), $\Delta\phi$ can be computed in terms of $\Delta\lambda$ and $\Delta\gamma$ from (25b):

$$\Delta\phi = -q_1 - \Delta\lambda q_2 - \Delta\gamma q_3 \tag{29}$$

by solving

$$\begin{aligned} A q_1 &= F_u^{(k)} \phi^{(k)} - B_1 p_1, \\ A q_2 &= F_{u\lambda}^{(k)} \phi^{(k)} - B_1 p_2, \\ A q_3 &= F_{u\gamma}^{(k)} \phi^{(k)} - B_1 p_3 \end{aligned} \tag{30}$$

with $B_1 = F_{uu}^{(k)}\phi^{(k)} \in \mathbb{R}^{n\times n}$. Similarly, $\Delta\zeta$ can be computed as

$$\Delta\zeta = -z_1 - \Delta\lambda z_2 - \Delta\gamma z_3 \tag{31}$$

by solving

$$\begin{aligned} A z_1 &= F_u^{(k)}\zeta^{(k)} - \phi^{(k)} - B_2 p_1 + q_1, \\ A z_2 &= F_{u\lambda}^{(k)}\zeta^{(k)} - B_2 p_2 + q_2, \\ A z_3 &= F_{u\gamma}^{(k)}\zeta^{(k)} - B_2 p_3 + q_3 \end{aligned} \tag{32}$$

with $B_2 = F_{uu}^{(k)}\zeta^{(k)} \in \mathbb{R}^{n\times n}$. Substituting (29) and (31) into (25d), (25e) gives rise to a 2×2 system for $\Delta\lambda$ and $\Delta\gamma$:

$$\begin{aligned} &2\phi^{(k)T}q_2\Delta\lambda + 2\phi^{(k)T}q_3\Delta\gamma = (\phi^{(k)T}\phi^{(k)} - 1) - 2\phi^{(k)T}q_1, \\ &(\zeta^{(k)T}q_2 + \phi^{(k)T}z_2)\Delta\lambda + (\zeta^{(k)T}q_3 + \phi^{(k)T}z_3)\Delta\gamma = \phi^{(k)T}\zeta^{(k)} - \zeta^{(k)T}q_1 - \phi^{(k)T}z_1. \end{aligned} \tag{33}$$

Having solved this system, Δu, $\Delta\phi$, and $\Delta\zeta$ can be computed from (26), (29), and (31). Using this technique, the solution of system (24a) of dimension $3n+2$ mainly consists of the following:

(a) Evaluation of the vectors $F^{(k)}$, $F_\lambda^{(k)}$, $F_\gamma^{(k)}$, $F_{u\lambda}^{(k)}\phi^{(k)}$, $F_{u\gamma}^{(k)}\phi^{(k)}$, $F_{u\lambda}^{(k)}\zeta^{(k)}$, and $F_{u\gamma}^{(k)}\zeta^{(k)}$;

(b) Evaluation of the matrices $F_u^{(k)}$, $F_{uu}^{(k)}\phi^{(k)}$, and $F_{uu}^{(k)}\zeta^{(k)}$;

(c) Computation of six matrix-vector products ($B_1 p_i$ and $B_2 p_i$, $i = 1, 2, 3$);

(d) One LU-decomposition of the $n\times n$-matrix $A = F_u^{(k)}$;

(e) Nine backsubstitutions.

This procedure is very efficient but can be unstable, since $A = F_u^{(k)}$ becomes singular at $(u^o, \lambda^o, \gamma^o)$. However this procedure can be used if we only wish to compute an origin for Hopf bifurcation with an accuracy of a few digits, which is sufficient to switch to the bifurcating branch of Hopf points (see § 7). In general the matrix A will only be very ill conditioned if the current approximation is already very close to the origin for Hopf bifurcation. Note also that the ill-conditioning of A will only cause a large relative error on the *correction* Δy of the Newton iteration; even if only a few digits of Δy are correct, this will still improve the approximation y. But clearly, using this procedure, the accuracy that can be obtained is severely limited and y^o cannot be computed to full machine precision.

PROCEDURE B. To improve the numerical stability of the procedure given above, a deflated decomposition of the solutions of the systems with matrix A can be incorporated. In [2] Chan describes deflation techniques that can be used to compute an accurate representation for the solution of a system with a nearly singular matrix A. We will work with the deflation based on the singular value decomposition and we use the same notation as in [2].

Let A_S be a singular matrix close to A with a left nullvector ψ_{SV} and a right nullvector ϕ_{SV} ($\|\psi_{SV}\|_2 = \|\phi_{SV}\|_2 = 1$). Let $P_u = 1 - uu^T$, with $\|u\|_2 = 1$, denote the orthogonal projector.

A deflated decomposition of the solution of $Az = p$ is given by

$$z = z_D + \frac{c_p}{\delta}\phi_{SV} \tag{34}$$

where z_D is the unique solution of

$$A_S z_D = P_{\psi_{SV}} p, \qquad P_{\phi_{SV}} z_D = z_D \tag{35}$$

and $c_p = \psi_{SV}^T p$. The scalar δ tends to zero if A tends to being singular. We refer to [2] for the equations that determine ψ_{SV} and ϕ_{SV} (which define implicitly A_S), z_D and δ.

The calculation of the nullvectors ψ_{SV} and ϕ_{SV} requires the solution of two systems with matrix A. Afterwards, the computation of the deflated decomposition of the solution of $Az = p$ mainly consists of computing $A^{-1}(P_{\psi_{SV}} p)$, i.e., one backsolve with the projected right-hand side.

We now describe how Procedure A can be made numerically stable using the deflated decomposition of the solutions of all systems with matrix A.

First note that only the representation (34) must be used in the algorithm. If we carry out the division by δ and the addition in (34), the second term will dominate z_D. Then the accuracy in z_D, which is important for the final solution of the system (25), will be lost. A similar situation occurs if we use deflation in the block-elimination algorithm for the solution of bordered systems with a matrix of the form

$$M = \begin{bmatrix} A & b \\ c^T & d \end{bmatrix},$$

which requires the solution of two systems with matrix A [18]. The deflated block-elimination algorithm given in [2] uses only the representation (34).

Instead of computing p_i, q_i, z_i $(i = 1, 2, 3)$, we compute the deflated solutions p_{iD}, q_{iD}, z_{iD}. Then we can replace (26) by

$$\Delta u = -p_{1D} - \Delta\lambda p_{2D} - \Delta\gamma p_{3D} - \alpha_1 \phi_{SV} \tag{36}$$

where α_1 is given in terms of $\Delta\lambda$ and $\Delta\gamma$ by

$$\alpha_1 = \frac{1}{\delta}[\psi_{SV}^T(A^{-1}p_1) + \Delta\lambda\psi_{SV}^T(A^{-1}p_2) + \Delta\gamma\psi_{SV}^T(A^{-1}p_3)]. \tag{37}$$

Here $A^{-1}p_i$ $(i = 1, 2, 3)$ simply denote the right-hand sides of (27). Using (36), $\Delta\phi$ can be written in terms of $\Delta\lambda$, $\Delta\gamma$, and α_1:

$$\Delta\phi = -q_1 - \Delta\lambda q_2 - \Delta\gamma q_3 - \alpha_1 q_4 \tag{38}$$

where q_1, q_2, and q_3 are determined by (30), where p_i must be replaced by p_{iD}, and q_4 satisfies

$$Aq_4 = -B_1\phi_{SV}. \tag{39}$$

Since we only compute deflated solutions q_{iD} $(i = 1, \cdots, 4)$, (38) must be replaced by

$$\Delta\phi = -q_{1D} - \Delta\lambda q_{2D} - \Delta\gamma q_{3D} - \alpha_1 q_{4D} - \alpha_2\phi_{SV} \tag{40}$$

where α_2 is given by

$$\alpha_2 = \frac{1}{\delta}[\psi_{SV}^T(A^{-1}q_1) + \Delta\lambda\psi_{SV}^T(A^{-1}q_2) + \Delta\gamma\psi_{SV}^T(A^{-1}q_3) + \alpha_1\psi_{SV}^T(A^{-1}q_4)]. \tag{41}$$

In an analogous way, we obtain

$$\Delta\zeta = -z_{1D} - \Delta\lambda z_{2D} - \Delta\gamma z_{3D} - \alpha_1 z_{4D} - \alpha_2 z_{5D} - \alpha_3\phi_{SV} \tag{42}$$

where z_1, z_2, and z_3 are determined by (32), where p_i and q_i must be replaced by p_{iD} and q_{iD}; z_4 and z_5 satisfy

$$Az_4 = -B_2\phi_{SV} + q_{4D}, \qquad Az_5 = -\phi_{SV}. \tag{43}$$

Finally, α_3 is given by

$$\alpha_3 = \frac{1}{\delta}[\psi_{SV}^T(A^{-1}z_1) + \Delta\lambda\psi_{SV}^T(A^{-1}z_2) + \Delta\gamma\psi_{SV}^T(A^{-1}z_3) + \alpha_1\psi_{SV}^T(A^{-1}z_4) + \alpha_2\psi_{SV}^T(A^{-1}z_5)]. \tag{44}$$

Substituting (40) and (42) into (25d, e), together with the expressions (37), (41), (44) gives rise to a 5×5 system for $\Delta\lambda$, $\Delta\gamma$, α_1, α_2, α_3. Having solved this system, Δu, $\Delta\phi$, and $\Delta\zeta$ can be obtained from (36), (40), and (42).

Compared with Procedure A, this procedure requires additional projections of the right-hand sides and the computed solution vectors, eight matrix-vector products and 14 backsubstitutions (including 2 backsubstitutions for the computation of ψ_{SV} and ϕ_{SV}).

PROCEDURE C. For the implementation of Newton's method applied to a determining system for simple quadratic turning points, Moore and Spence [20] describe a procedure in which only systems have to be solved with a matrix $\tilde{A}$, obtained by replacing a column of $A = F_u^{(k)}$ by $F_\lambda^{(k)}$ (say the ipth column). At a turning point, and hence also at an origin for Hopf bifurcation, $\tilde{A}$ is nonsingular (see (8)).

In [5] we describe a procedure for the implementation of Newton's method applied to (15), which is similar to the algorithm given in [20]. In that case, the normalization of ϕ in (15) is replaced by $e_{ip}^T\phi = 1$ $(1 \leq ip \leq n)$. Hence (25d) must be replaced by $e_{ip}^T\Delta\phi = 0$.

This procedure requires the solution of nine systems with matrix $\tilde{A}$. Of course, the structure of A will be destroyed in forming $\tilde{A}$ and it is possible that solving a system with $\tilde{A}$ is much more expensive than solving a system with A.

In case the Jacobian matrix A is a bandmatrix, we can choose ip as large as possible. If $ip = n$, no fill-in will occur during the LU-factorization of $\tilde{A}$ and the systems can be solved using an "adapted bandsolver." Another possibility in this case is using the block-elimination algorithm [18] to compute the solutions of the systems with matrix $\tilde{A}$. Then Procedure C requires the solution of 10 systems with matrix $\hat{A}$, the leading $(n-1)\times(n-1)$-matrix of $\tilde{A}$. In [5] we show that the determinant of $\hat{A}$ can be used to detect higher singularities during the continuation of a branch of origins for Hopf bifurcation in a three-parameter space.

Starting values for ϕ^o and ζ^o. Assume that sufficiently good starting values $u^{(1)}$, $\lambda^{(1)}$, $\gamma^{(1)}$ for u^o, λ^o, γ^o are available, e.g., by continuation of a branch of quadratic turning points as outlined above. Note that $u^{(1)}$, $\lambda^{(1)}$, $\gamma^{(1)}$ must not be exactly a quadratic turning point to avoid the matrix $F_u^{(1)} = F_u(u^{(1)}, \lambda^{(1)}, \gamma^{(1)})$ being singular. Then starting values $\phi^{(1)}$ and $\zeta^{(1)}$ for ϕ^o and ζ^o can be obtained by the following procedure:

(45a) $\quad \nu^{(0)} = [1 \quad 1 \quad \cdots \quad 1]^T$ (or any arbitrary vector $\in\mathbb{R}^n$),

(45b) $\quad$ solve $F_u^{(1)}\nu^{(1)} = \nu^{(0)}$,

(45c) $\quad \phi^{(1)} = \nu^{(1)}/\|\nu^{(1)}\|_2$,

(45d) $\quad$ solve $F_u^{(1)}\nu^{(2)} = \phi^{(1)}$,

(45e) $\quad \zeta^{(1)} = \nu^{(2)} - (\nu^{(2)T}\phi^{(1)})\phi^{(1)}$.

If $(u^{(1)}, \lambda^{(1)}, \gamma^{(1)})$ is close to a quadratic turning point, one inverse iteration step applied to $F_u^{(1)}$ (steps (45a)–(45c)) will be sufficient to obtain a good approximation of the eigenvector, corresponding to the smallest eigenvalue of $F_u^{(1)}$. This eigenvector will be close to ϕ^o. Performing more than one inverse iteration step can be disadvantageous if $F_u^{(1)}$ is very close to $F_u^{(0)}$, which contains a Jordan block of order two [9].

The vector $\nu^{(2)}$ obtained in step (45d) will have a large component in the direction of $\phi^{(1)}$ (step (45d) can be considered as a second inverse iteration step). Taking the component of $\nu^{(2)}$ orthogonal on $\phi^{(1)}$, a vector $\zeta^{(1)}$ is obtained which will be a sufficiently good approximation for ζ^o. In our numerical experiments the residual $F_u^{(1)}\zeta^{(1)} - \phi^{(1)}$ was in general not larger than the residual $F_u^{(1)}\phi^{(1)}$.

6. Switching to the branch of Hopf points. Assume that an origin for Hopf bifurcation $(u^o, \lambda^o, \gamma^o)$ is determined accurately by solving the determining system (15). Then also the eigenvector ϕ^o and the generalized eigenvector ζ^o are known.

The emanating branch of Hopf bifurcation points can then be computed by a continuation procedure applied to a suitable determining system for Hopf points, starting from a Hopf point $(u^h, \lambda^h, \gamma^h)$ close to the origin for Hopf bifurcation. In [24], [25] we proposed the following determining system

$$\begin{aligned} &F(u, \lambda, \gamma) = 0, \qquad ([F_u(u, \lambda, \gamma)]^2 + \omega^2 I)p = 0, \\ &p^T p = 1, \qquad q^T p = 0. \end{aligned} \tag{46}$$

Assume that $F_u(u^h, \lambda^h, \gamma^h)$ has a pair of purely imaginary eigenvalues $\mu = \pm i\omega^h$, with corresponding eigenvectors $\phi^h = \phi_r^h \pm \phi_i^h$. If $p^h \in \text{span}\{\phi_r^h, \phi_i^h\}$ and p^h is orthogonal on the arbitrary vector q then $(u^h, \lambda^h, \gamma^h, p^h, \omega^h)$ is a regular solution of (46). Indeed, the matrix $[F_u(u^h, \lambda^h, \gamma^h)]^2$ has a double real eigenvalue $-(\omega^h)^2$ with a two-dimensional eigenspace spanned by ϕ_r^h and ϕ_i^h.

At the origin for Hopf bifurcation the Jacobian matrix $[F_u(u^h, \lambda^h, \gamma^h)]^2$ has a double eigenvalue zero with a two-dimensional eigenspace spanned by ϕ^o and ζ^o. Thus if the Hopf point $(u^h, \lambda^h, \gamma^h)$ is close to $(u^o, \lambda^o, \gamma^o)$, then span $\{\phi_r^h, \phi_i^h\}$ must be close to span $\{\phi^o, \zeta^o\}$.

It follows that a Hopf point $(u^h, \lambda^h, \gamma^h)$ close to $(u^o, \lambda^o, \gamma^o)$ can be computed by solving (46) with a fixed small value of $\omega = \omega_{\text{fixed}}$. Note that the derivative of ω along the branch of Hopf points becomes infinite at the origin of the branch [25]. Good starting values for u, λ, γ, and p are given by a slight perturbation of u^o, λ^o, and γ^o and a vector belonging to span $\{\phi^o, \zeta^o\}$. In practice we can choose $q = \phi^o$ and $p = \zeta^o$ or $q = \zeta^o$ and $p = \phi^o$. The perturbation of u^o, λ^o, and γ^o is necessary if we use the efficient implementation of Newton's method for solving (46) as described in [24], since in this case systems have to be solved with the matrix F_u.

In fact, it can be proved that the use of $(u^o, \lambda^o, \gamma^o, \phi^o, \omega_{\text{fixed}})$ as starting values for (46) with $q = \zeta^o$ corresponds to a predictor step along the tangent to the bifurcating branch of Hopf points at $(u^o, \lambda^o, \gamma^o)$.

The only remaining problem is the choice of ω_{fixed}. If ω_{fixed} is chosen too large, the Hopf point $(u^h, \lambda^h, \gamma^h)$ will be rather far from $(u^o, \lambda^o, \gamma^o)$ and Newton's method applied to (46) will diverge, due to the poor starting values. However, if ω_{fixed} is chosen too small, problems can arise during the continuation procedure. In that case the first continuation point lies very close to the origin for Hopf bifurcation, i.e., close to a turning point. This can cause numerical difficulties as indicated in [24], [25], since a turning point corresponds to a singular point for (46).

7. Branches of Hopf points for a model problem: the trimolecular Brusselator scheme. Chemical reaction and diffusion in a one-dimensional steady state reactor where auto- and cross-catalytic reactions take place, are described for the trimolecular Brusselator scheme [32], [21] by

$$\begin{aligned} &\frac{\partial A}{\partial t} = \frac{\partial^2 A}{\partial z^2} - \frac{L^2}{D_A} A, \qquad \frac{\partial X}{\partial t} = \frac{\partial^2 X}{\partial z^2} - \frac{L^2}{D_X}((B+1)X - X^2 Y - A), \\ &\frac{\partial Y}{\partial t} = \frac{\partial^2 Y}{\partial z^2} - \frac{L^2}{D_Y}(X^2 Y - BX), \qquad \frac{\partial B}{\partial t} = \frac{\partial^2 B}{\partial z^2} - \frac{L^2}{D_B} BX. \end{aligned} \tag{47a}$$

Here D_A, D_B, D_X, and D_Y are the respective diffusion coefficients of the components, while A, B, X, and Y represent their concentrations; L is the parameter of the scheme

and indicates the length of the one-dimensional reactor; z is the dimensionless space coordinate and varies between 0 and 1. The boundary conditions considered are

$$(47b) \qquad A = A_0 = 2, \quad B = B_0 = 4.6, \quad \frac{dX}{dz} = 0, \quad \frac{dY}{dz} = 0 \quad \text{for } z = 0 \text{ and } 1.$$

The bifurcation behaviour of this system with respect to L, the reactor length, has been studied extensively (see, e.g., [3]).

Using an $O(h^4)$-discretization on an equidistant mesh, system (47) can be approximated by a system of the form (1). We will consider D_A, the diffusion coefficient of component A, as the second parameter; thus L and D_A are the parameters previously denoted by λ and γ. Then the Jacobian matrix $F_u(u, L, D_A)$ has bandwidth $d = 7$. In our calculations we used 42 discretization points ($h = 1/41$).

De Dier et al. [3] computed bifurcation diagrams for the steady state problem with respect to L, for $D_A = 0.5$, 0.02, and ∞, $D_X = 0.0016$, $D_Y = 0.008$, and $D_B = \infty$. A part of the bifurcation diagram for $D_A = 0.5$ and $L < 0.5$ is given in Fig. 1. Several turning points were computed. Starting from these turning points, we performed a two-parameter study by computing branches of turning points. Therefore we used the continuation code PITCON of Rheinboldt and Burkardt [22] to compute solutions of the following determining system for quadratic turning points, introduced by Seydel [28] and Moore and Spence [20]

$$(48) \qquad F(u, L, D_A) = 0, \quad F_u(u, L, D_A)\phi = 0, \quad l(\phi) = 1.$$

In each continuation step we have to solve systems of dimension $2n+2$. Spence and Werner [29] have indicated how this can be implemented efficiently, requiring only the LU-decomposition of a matrix of order n and eight backsubstitutions (four backsubstitutions for the calculation of the tangent to the curve of turning points). In our implementation we exploited the bandstructure of the Jacobian matrix F_u as much as possible. More details about the numerical procedure and a discussion of the results can be found in [4].

In Fig. 2 we show the projection of some of the computed branches of turning points on the (L, D_A)-plane. The branches labeled B_1, B_2, and B_3 contain, respectively, the turning points b_1, b_2, and b_3 of branch b in Fig. 1.

Using the algorithm presented in § 4, we tried to find origins for Hopf bifurcation on these branches of turning points. At the points marked T_i the test function (14)

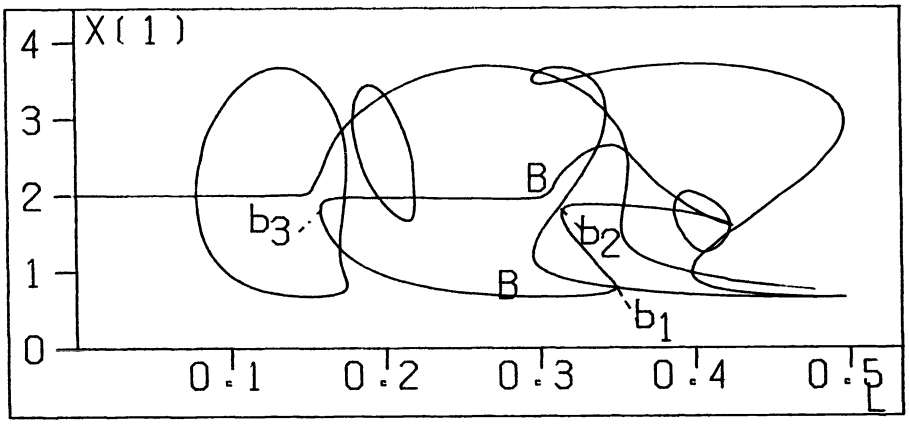


FIG. 1. *Bifurcation diagram for* (47) *with respect to L for* $D_A = 0.5$, $D_X = 0.0016$, $D_Y = 0.008$, *and* $D_B = \infty$.

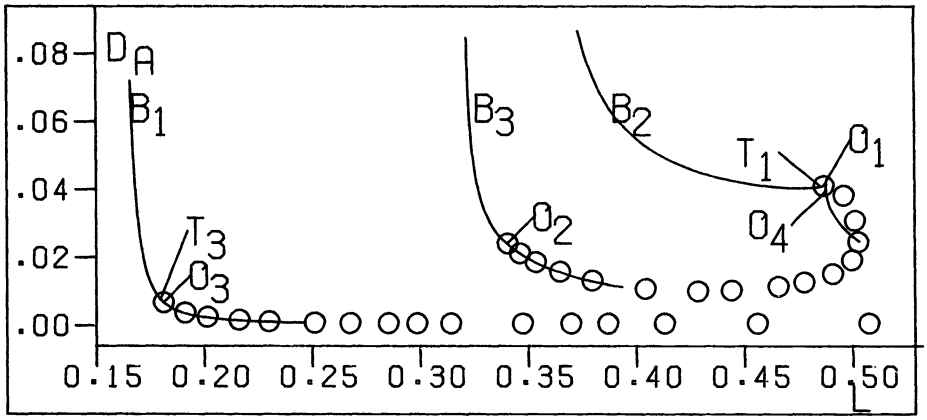


FIG. 2. *Branches of quadratic turning points, origins for Hopf bifurcation, and branches of Hopf points for* (47) *with* $D_X = 0.0016$, $D_Y = 0.008$, *and* $D_B = \infty$. *Projection on the* (L, D_A)*-plane.* ——: *branches of turning points*; ∘∘∘∘: *branches of Hopf points.*

changed sign. Starting from these points, origins for Hopf bifurcation O_i were computed by solving the determining system (15) with a damped Newton iteration, implemented as described in § 5. The results are given in Table 1.

Working with a machine precision of $\approx 10^{-16}$ we were able to compute these origins for Hopf bifurcation with a relative accuracy of $\approx 10^{-6}$. Procedure A discussed in § 5 does not allow more accurate results, due to the ill-conditioning of $F_u^{(k)}$ in the subsequent Newton iterations.

To determine the emanating branch of Hopf points we first computed a solution $(u^h, L^h, D_A^h, \omega_{\text{fixed}}, p^h)$ of the determining system for Hopf points (46) with $\omega_{\text{fixed}} = 0.001$. Starting from this Hopf point, we used PITCON to compute a branch of solutions of the determining system (46). The solution of the $(2n+3)$-dimensional systems to be solved in each continuation step can be implemented efficiently by combining the block-elimination algorithm of Keller [18] with the procedure described in [24] to solve (46) for a fixed value of γ. In this case the computational work mainly consists of the following:

(a) The computation of the matrix product F_u^2;

(b) The computation of 15 matrix-vector products (nine for the calculation of the tangent);

(c) Two LU-decompositions of $n \times n$-matrices;

TABLE 1

Origins for Hopf bifurcation O_i *of* (36) *with* $D_X = 0.0016$, $D_Y = 0.008$, *and* $D_B = \infty$.

T_i: *used as starting points for the computation of* O_i.

	L	D_A	$X(1)$	$Y(1)$
O_1	0.48656	0.04108	0.71105	4.61031
O_2	0.34046	0.02406	0.89000	3.50599
O_3	0.18125	0.00673	0.99478	3.21672
O_4	0.48799	0.03976	0.78142	4.62694
T_1	0.48539	0.04088	0.69910	4.60001
T_3	0.17976	0.00767	1.03609	3.11000
T_4	0.48802	0.04072	0.77707	4.61010

(d) Six backsubstitutions (three for the calculation of the tangent).

Also in this case the bandstructure of the matrices can be exploited.

In Fig. 2 we show the branch of Hopf points that originates from O_1 and ends in O_2 and also a part of the branch of Hopf points starting at O_3. We also computed the branch starting at O_4; this branch has a "vertical asymptote" at $L \approx 0.408$ ($X(1) \approx 1.87$, $Y(1) \approx 2.69$). Note that O_1 and O_4 lie very close to the cusp point C. Projected on the (L, D_A)-plane the branches of Hopf points must bifurcate tangentially from the branches of turning points (see [10, pp. 364–374] and [8, Fig. 1.1]). The projection on the $(D_A, X(1))$-plane of the branch between O_1 and O_2 is given in Fig. 3 and indicates that, close to the origin for Hopf bifurcation, this branch of Hopf points nearly coincides with the branches of turning points.

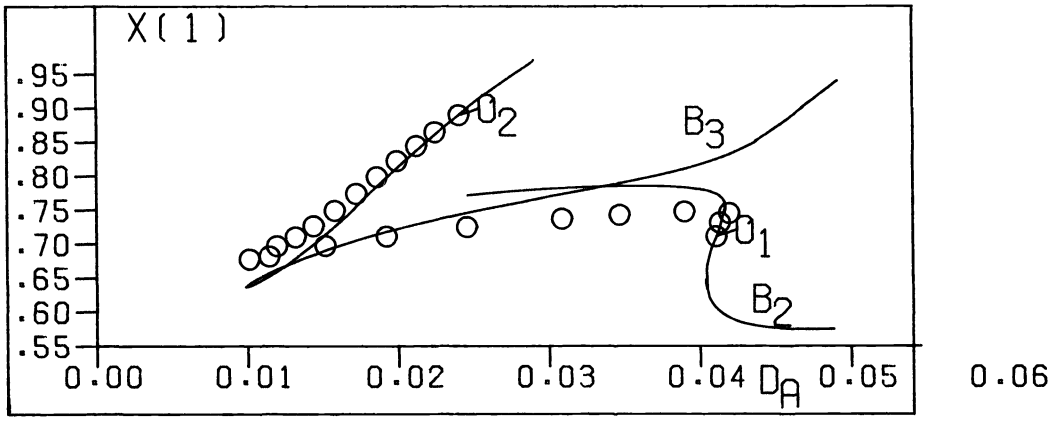


FIG. 3. *Branches of quadratic turning points, origins for Hopf bifurcation, and branches of Hopf points for* (47) *with* $D_X = 0.0016$, $D_Y = 0.008$, *and* $D_B = \infty$. *Projection on the* $(D_A, X(1))$*-plane.* ——: *branches of turning points*; ∘∘∘∘: *branches of Hopf points.*

8. Conclusion. The calculation of origins for Hopf bifurcation by the method introduced in this paper is a powerful technique for the analysis of Hopf bifurcation in a two-parameter problem. First, the determination of these origins is very useful for the computation of branches of Hopf points in a two-parameter problem, especially since it is difficult (or expensive) to detect Hopf bifurcation in a one-parameter problem. However this approach does not allow the determination of closed branches of Hopf bifurcation points. Second, the location of the origins for Hopf bifurcation is important for the analysis of "global Hopf bifurcation" along a one-dimensional closed curve in a two-parameter space, as shown by Keener [17] and Fiedler [7].

We used our method also for the computation of origins for Hopf bifurcation in a model for the behaviour of a tubular reactor [26]. Origins for Hopf bifurcation have been determined also for some other problems, see e.g. [27] (two well-mixed cells with the Brusselator reaction scheme), [1] (Oregonator-model), [8] (Coupled Oregonators), and [16] (Taylor problem in Navier–Stokes equations).

In [16] Jepson, Spence, and Cliffe develop a general theory for the behaviour of the steady state solutions near an origin for Hopf bifurcation (see also [31]). They also present numerical techniques for the computation of origins for Hopf bifurcation and the bifurcating branch, which are related to the method presented in this paper.

If a branch of origins for Hopf bifurcation is followed in a three-parameter space, higher singularities can be detected. This is studied in the papers [5] and [6].

REFERENCES

[1] D. BARKLEY AND J. RINGLAND, *Bifurcations of a torus in a model of the Belousov-Zhabotinskii reaction*, Report, Center for Studies in Statistical Mechanics, University of Texas, Austin, TX, 1986.

[2] T. F. CHAN, *Deflation techniques and block-elimination algorithms for solving bordered singular systems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 121-134.

[3] B. DE DIER, F. WALRAVEN, R. JANSSEN, P. VAN ROMPAY, AND V. HLAVACEK, *Bifurcation and stability analysis of a one-dimensional diffusion autocatalytic reaction system*, Z. Naturforsch., 42a (1987), pp. 992-1002.

[4] B. DE DIER AND D. ROOSE, *Determination of bifurcation points and catastrophes for the "Brusselator" model with two parameters*, in Bifurcation: Analysis, Algorithms, Applications, ISNM 79, T. Küpper, R. Seydel, and H. Troger, eds., Birkhäuser, Basel, 1987, pp. 38-46.

[5] ———, *Computation of branches of codimension two bifurcation points in a three-parameter space*, Report TW99, Department of Computer Science, Katholieke Univ. Leuven, Belgium, 1987.

[6] ———, *Interaction between fold and Hopf curves leads to new bifurcation phenomena*, Report, Department of Chemical Engineering, Katholieke Univ. Leuven, Belgium, 1988; J. Comput. Appl. Math., (1989), to appear.

[7] B. FIEDLER, *Global Hopf bifurcation of two-parameter flows*, Arch. Rational Mech. Anal., 94 (1986), pp. 59-81.

[8] B. FIEDLER AND P. KUNKEL, *A quick multiparameter test for periodic solutions*, in Bifurcation: Analysis, Algorithms, Applications, ISNM 79, T. Küpper, R. Seydel, and H. Troger, eds., Birkhäuser, Basel, 1987, pp. 61-70.

[9] G. GOLUB AND J. WILKINSON, *Ill-conditioned eigensystems and the computation of the Jordan canonical form*, SIAM Rev., 18 (1976), pp. 578-619.

[10] J. GUCKENHEIMER AND P. HOLMES, *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Applied Mathematical Sciences 42, Springer-Verlag, New York, 1983.

[11] B. HASSARD, N. KAZARINOFF, AND Y-H. WAN, *Theory and Applications of Hopf Bifurcation*, Cambridge University Press, Cambridge, 1981.

[12] M. HOLODNIOK AND M. KUBICEK, *New algorithms for the evaluation of complex bifurcation points in ordinary differential equations. A comparative numerical study*, Appl. Math. Comput., 15 (1984), pp. 261-274.

[13] G. IOOSS AND D. D. JOSEPH, *Elementary Stability and Bifurcation Theory*, Springer-Verlag, New York, 1980.

[14] A. D. JEPSON, *Numerical Hopf bifurcation*, Ph.D. thesis, Part II, California Institute of Technology, Pasadena, CA, 1981.

[15] A. D. JEPSON AND H. B. KELLER, *Steady state and periodic solution paths: their bifurcations and computations*, in Numerical Methods for Bifurcation Problems, ISNM 70, T. Küpper, H. D. Mittelmann, and H. Weber, eds., Birkhäuser, Basel, 1984, pp. 219-246.

[16] A. D. JEPSON, A. SPENCE, AND K. A. CLIFFE, *Hopf along a fold*, Harwell Report TP1264, AERE Harwell, U.K., 1988; submitted to IMA J. Numer. Anal.

[17] J. P. KEENER, *Infinite period bifurcation and global bifurcation branches*, SIAM J. Appl. Math., 41 (1981), pp. 127-144.

[18] 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.

[19] J. E. MARSDEN AND M. MCCRACKEN, *The Hopf Bifurcation and Its Applications*, Applied Mathematical Sciences 19, Springer-Verlag, Berlin, 1976.

[20] G. MOORE AND A. SPENCE, *The calculation of turning points of nonlinear equations*, SIAM J. Numer. Anal., 17 (1980), pp. 567-576.

[21] I. PRIGOGINE AND R. LEFEVER, *Symmetry breaking instabilities in dissipative systems* II, J. Chem. Phys., 48 (1968), pp. 1695-1700.

[22] W. C. RHEINBOLDT AND J. V. BURKARDT, *A locally parametrized continuation process*, ACM Trans. of Math. Software, 9 (1983), pp. 215-241.

[23] W. C. RHEINBOLDT, *Numerical Analysis of Parametrized Nonlinear Equations*, John Wiley, New York, 1986.

[24] D. ROOSE AND V. HLAVACEK, *A direct method for the computation of Hopf bifurcation points*, SIAM J. Appl. Math., 45 (1985), pp. 879-894.

[25] D. ROOSE, *An algorithm for the computation of Hopf bifurcation points in comparison with other methods*, J. Comput. Appl. Math., 12 & 13 (1985), pp. 517-529.

[26] ———, *Numerical computation of origins for Hopf bifurcation in a two-parameter problem*, in Bifurcation: Analysis, Algorithms, Applications, ISNM 79, T. Küpper, R. Seydel, and H. Troger, eds., Birkhäuser, Basel, 1987, pp. 268-276.

[27] I. SCHREIBER, M. HOLODNIOK, M. KUBICEK, AND M. MAREK, *Periodic and aperiodic regimes in coupled dissipative chemical oscillators*, J. Statist. Phys., 43 (1986), pp. 489-519.

[28] R. SEYDEL, *Numerical computation of branch points in nonlinear equations*, Numer. Math., 33 (1979), pp. 339-352.

[29] A. SPENCE AND B. WERNER, *Nonsimple turning points and cusps*, IMA J. Numer. Anal., 2 (1982), pp. 413-427.

[30] A. SPENCE AND A. JEPSON, *The numerical calculation of cusps, bifurcation points and isola formation points in two-parameter problems*, in Numerical Methods for Bifurcation Problems, ISNM 70, T. Küpper, H. D. Mittelmann, and H. Weber, eds., Birkhäuser, Basel, 1984, pp. 502-514.

[31] A. SPENCE, A. D. JEPSON, AND K. A. CLIFFE, *A note on the calculation of paths of Hopf bifurcation points*, J. Comput. Appl. Math., (1989), to appear.

[32] A. TURING, *The chemical basis of morphogenesis*, Phil. Trans. Roy. Soc. London Ser. B, 237 (1952), pp. 37-72.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 686-698, July 1989

© 1989 Society for Industrial and Applied Mathematics
005

AN OPTIMAL COMPLEXITY ALGORITHM FOR COMPUTING THE TOPOLOGICAL DEGREE IN TWO DIMENSIONS*

T. BOULT† AND K. SIKORSKI‡

Abstract. An algorithm is presented for computing the topological degree for any function from a class F. The class F consists of functions $f: C \to \mathbb{R}^2$ defined on C, the unit square, which satisfy the Lipschitz condition with constant $K > 0$, and whose infinity norm on the boundary of C is at least $d > 0$. For functions in this class, the algorithm could be useful in solving nonlinear systems of equations and also in other aspects of nonlinear analysis.

A worst-case lower bound, $m^* = 4\lfloor K/4d \rfloor$, is established on the number of function evaluations necessary to compute the topological degree for any function f from the class F. The parallel information used by our algorithm permits the computation of the degree for every f in F with m^* function evaluations. The cost of our algorithm is shown to be almost equal to the complexity of the problem.

The algorithm has been implemented and tested. Performance information for a set of 10 systems of equations is reported.

Key words. complexity, topological degree of a mapping, Lipschitz function

AMS(MOS) subject classification. 65H10

0. Introduction. The problem of computing the topological degree of a function has been studied in many recent papers ([12], [11], [18], [21]-[25]). From the topological degree, one may ascertain whether there exists a zero of a function inside a domain. Namely, Kronecker's theorem (see [16]) states that if the degree is not zero, then there exists at least one zero of a function inside the domain. By computing a sequence of domains with nonzero degrees and decreasing diameters, one can obtain a region with arbitrarily small diameter that contains at least one zero of the function (see [12]). Such methods, called generalized bisection, have been implemented and tested by several authors(see, e.g., [6], [10], [12], [20], [22]). These methods have been touted as appropriate when the function f is not smooth or cannot be evaluated accurately. For such functions they yield approximations to roots, even when all other methods tested have failed (see, e.g., [10]).

Another class of bisection methods, based on interval analysis, has been proposed by Moore and Jones [15], Hansen [8], Hansen and Sengupta [9], and others. These methods are claimed to be robust and appropriate for finding starting points for Newton-type methods. A nice summary of the properties of these methods has been given by Kearfott in [14].

The generalized bisection methods based on the degree computation are related to simplicial continuation methods surveyed, for example, in [1], [5], [7], [26].

There are numerous applications of degree computation in nonlinear analysis. In addition to the existence test for roots, the degree computation is used in methods for finding direction proceedings from bifurcation points in the solution of nonlinear functional differential equations (see, e.g., [17], [19]) and could be used in others, e.g., [13].

* Received by the editors November 3, 1986; accepted for publication (in revised form) June 23, 1988. This work was supported in part by National Science Foundation grants DCR-82-14322, CCR-86-03674, and CCR-88-09022; in part by Amoco Foundation Engineering Faculty grant 0-46403; and in part by an IBM Graduate Fellowship.

† Department of Computer Science, Columbia University, New York, New York 10027.

‡ Department of Computer Science, University of Utah, Salt Lake City, Utah 84112.

Previous algorithms for degree computation have been tested on several examples. Their authors have observed that they may require arbitrarily many function evaluations (see, e.g., [11], [12], [23], [24]).

In this paper we restrict the class of functions to be able to compute the degree for every element in the restricted class using an a priori bounded number of evaluations. We consider the class F of Lipschitz functions, with constant K, defined on the unit square $C \subset \mathbb{R}^2$, $f: C \to \mathbb{R}^2$ such that for every $f \in F$, $\|f(x)\|_\infty \geqq d > 0$ for all $x \in \partial C$, the boundary of C, and also such that $K/4d \geqq 1$. Note that if $K/2d < 1$, then the functions in F do not have zeros and therefore the degree is zero for every f. The case $1 \leqq K/2d < 2$ is open. The information on f, $N_m(f)$, consists of m values of f on ∂C, which may be computed sequentially (adaptively). This form of information is assumed since the topological degree is uniquely determined by the values of f on ∂C (see [16]). The topological degree is computed by means of an algorithm φ that is a mapping depending on the information, $\varphi: N_n(F) \to I$, where I denotes the set of all integers.

In § 1 we present basic definitions and give the formulation of the problem. In § 2 we exhibit an algorithm φ^* using information N_{m^*}, which allows us to compute the degree of any $f \in F$, using $m^* = 4\lfloor K/4d \rfloor$ function evaluations. We then show, in § 3, that m^* is a lower bound on the number of function evaluations necessary to compute the degree for every $f \in F$ using arbitrary information. In § 4 we prove that the algorithm φ^* computes the degree for any $f \in F$. We conclude in § 5, with a calculation of the cost of φ^* and show that φ^* is an almost optimal complexity algorithm.

We remark that information N_{m^*} is *parallel* (*nonadaptive*), i.e., the evaluation points are given a priori. Thus it can be easily implemented on a parallel computer yielding an almost optimal speedup (see [28] for further details).

The reader interested in the actual details of of implementation should consult [2], [3], where an extension of the algorithm φ^* is detailed and tested. The extensions presented therein include the ability to calculate the degree for arbitrary polygonal regions in two dimensions and for the unit cube in N dimensions. We remark that while the complexity bounds in N dimensions are not as tight, [4] presents lower and upper bounds on the complexity of the computation of topological degree for Lipschitz functions in N dimensions.

1. Basic definitions. Let $C \equiv [0, 1] \times [0, 1]$ be the unit square in $\mathbb{R}^2$, $\|\cdot\| \equiv \|\cdot\|_\infty$ the infinity norm in $\mathbb{R}^2$, $\theta \equiv [0, 0] \in \mathbb{R}^2$, and I be the set of all integers. For a given positive d and K define

$$F = \{f: C \to \mathbb{R}^2, f = (f_1, f_2): \|f(x) - f(y)\| \leqq K\|x - y\| \ \forall x, y \in C \tag{1.1}$$
$$\|f(x)\| \geqq d \ \forall x \in \partial C \text{ and } K/4d \geqq 1\}.$$

Our problem is to find the topological degree, deg (f, C, θ) of f relative to C at θ (see [16]), for every f in F. To solve this problem, we use *information* N_m and an algorithm φ using N_m. These are defined as in [27], as follows:

Let $f \in F$ and define

$$N_m(f) = [f(x_1), \cdots, f(x_m)], \tag{1.2}$$

where $x_1 \in \partial C$ is given a priori, $x_j = \underline{x}_j(f(x_1), \cdots, f(x_{j-1}))$ and $\underline{x}_j$ is a transformation, $\underline{x}_j: \mathbb{R}^{2\cdot(j-1)} \to \partial C$, $j = 2 \cdots m$. If $\underline{x}_j$ are constant transformations, then the information is called *parallel* (*nonadaptive*); otherwise, it is called *sequential* (*adaptive*).

Knowing N_m, we approximate deg (f, C, θ) by an algorithm φ, which is an arbitrary mapping

$$\varphi: N_m(F) \to I, \tag{1.3}$$

where I is the set of all integers.

By *minimal cardinality number* m^* we mean the minimal integer m for which there exists information N_m that uniquely determines the degree of every f in F, i.e., $N_m(f') = N_m(f) \Rightarrow \deg(f', C, \theta) = \deg(f, C, \theta)$ for all $f', f \in F$. We define a segment $[x, y]$ to be the closed counterclockwise-oriented portion of ∂C with endpoints x and y, and by (x, y) we mean the interior of $[x, y]$. We define a partition P of ∂C^2 to be either the empty set or a set $\{p_i\}_{i=1}^q$ of counterclockwise-ordered points from ∂C^2 such that

$$\partial C^2 = \sum_{i=1}^{q} [p_i, p_{i+1}] \quad \text{where } p_{q+1} = p_1.$$

DEFINITION 1.1. A nonempty partition P forms a sufficient refinement of a boundary ∂C with respect to the sign of a function f if and only if $(p_i, p_{i+1}) \cap (p_j, p_{j+1}) = \varnothing$ for all $i \neq j$, and on each $[p_i, p_{i+1}]$, there exists a component of f say f_{j_i} that is of constant sign (i.e., $\neq 0$) on $[p_i, p_{i+1}]$, and the remaining component of f is nonzero at p_i and p_{i+1}.

Stenger showed (see [22]) that given a sufficient refinement of the boundary ∂C, then the topological degree can be computed by

$$\deg(f, \partial C, \theta) = \frac{1}{4} \sum_{i=1}^{q} (-1)^{j_i - 1} \cdot \deg(f_{j_i+1}, [p_i, p_{i+1}], 0) \cdot \operatorname{sgn}(f_{j_i}(p_i)), \tag{1.4}$$

where j_i is the index of the component of f that has constant sign on $[p_i, p_{i+1}]$, $\dot{f}_3 = f_1$, $\deg(f_j, [p_i, p_{i+1}], 0) = (\operatorname{sgn}(f_j(p_{i+1})) - \operatorname{sgn}(f_j(p_i)))/2$, and $\operatorname{sgn}(f_j(p_i)) = (1$ if $f_j(p_i) > 0$, -1 if $f_j(p_i) < 0)$.

2. Description of the algorithm. The algorithm calculates the degree using (1.4). Using nonadaptive information $N_{m^*}(f)$ of cardinality $4\lfloor K/4d \rfloor$, the algorithm constructs conceptually a partition P that forms a sufficient refinement of ∂C with respect to the sign of f, for every f in F. The algorithm calculates the sign of both components of f at each point of the partition P (the proof that this can be done using only $N_{m^*}(f)$ is given in § 4) and calculates the degree using (1.4). A flow chart of the major steps of the algorithm appears in Fig. 2.1. Consult [2], [3] for implementation details.

3. A lower bound. In this section we show that the minimal cardinality number m^* is larger than $4\lfloor K/4d \rfloor - 1$. Namely, we prove the following lemma.

LEMMA 3.1. *For any adaptive information operator N_m, of cardinality $m \leqq 4\lfloor K/4d \rfloor - 1$, there exist two functions f^*, f^{**} in F, such that*

$$N_m(f^*) = N_m(f^{**}), \quad \deg(f^*, C, \theta) = 0, \quad \deg(f^{**}, C, \theta) = \pm 1.$$

Proof. Let $M = \lfloor K/(4d) \rfloor$, and define

$$x_j = \begin{cases} (j/M, 0) & \text{for } j = 1, 2, \cdots, M, \\ (1, (j-M)/M) & \text{for } j = M+1, \cdots, 2M, \\ (1-(j-2M)/M, 1) & \text{for } j = 2M+1, \cdots, 3M, \\ (0, 1-(j-3M)/M) & \text{for } j = 3M+1, \cdots, 4M, \end{cases}$$

with the notation $x_{j \pm 4M} = x_j$ for all j. Thus the x_j subdivides ∂C into $4M$ segments $[x_j, x_{j+1}]$ of size $\geqq 4d/K$, i.e., $\|x_j - x_{j+1}\| = 1/M \geqq 4d/K$. For a given point $c \in \mathbb{R}^2$, define the functions $h_i, h_i : \mathbb{R}^2 \to \mathbb{R}$, $i = 1, 2$, by

$$\begin{aligned} h_1(w; c) &= \min(d, -d + K\|w - c\|), \\ h_2(w; c) &= \min(d, \max(-d, -3d + K\|w - c\|)) \quad \forall w \in \mathbb{R}^2. \end{aligned} \tag{3.1}$$

For given points $c_i \in \mathbb{R}^2$, define the functions $g_i : \mathbb{R}^2 \to \mathbb{R}$, $i = 1, 2$, by

$$g_i(w; c_i) = \min(d, \max(-d, -2d + K\|w - c_i\|)) \quad \forall w \in \mathbb{R}^2. \tag{3.2}$$

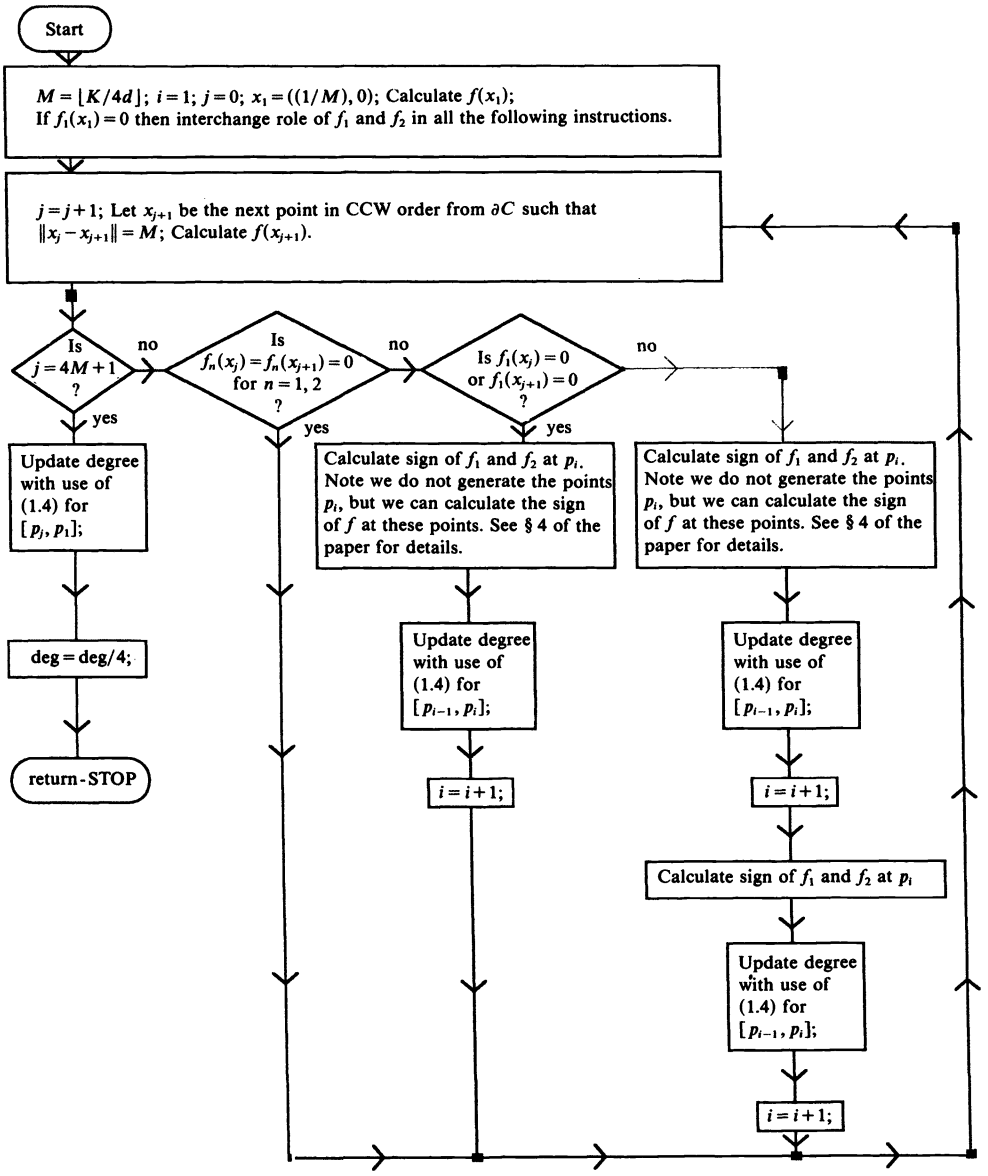


FIG. 2.1 *Flow chart of algorithm to calculate topological degree.*

Figure 3.1 shows $f = (g_1, g_2)$ where $c_1 = (-d/K, -d/K)$, $c_2 = (d/K, d/K)$. Now we define the function $f^*: C \to \mathbb{R}^2$ by

$$f^*(w) = (\min(h_1(w, x_1), h_1(w, x_3), \cdots, h_1(w, x_{4M-3}), h_1(w, x_{4M-1})),$$
$$\min(h_2(w, x_1), h_2(w, x_3), \cdots, h_2(w, x_{4M-3}), h_2(w, x_{4M-1}))).$$

(See Fig. 3.2.) Note that f^* is Lipschitz with constant K, as the minimum of such functions, and $\|f^*(w)\| = d$, for all $w \in \partial C$. Thus f^* belongs to F. Note also that $\deg(f^*, C, \theta) = 0$ since f^* has no zeros at all!

Next we generate the nonadaptive information $M_{m,f^*}(f) = [f(w_1), \cdots, f(w_m)]$ for all $f \in F$, where $w_i = \underline{w}_i(f^*(w_1), \cdots, f^*(w_{i-1}))$. Since the cardinality of this information is $m \leqq 4M - 1$, we know that there exists a segment on ∂C, say $[x_{j-1}, x_j]$, that was not sampled by the information operator N_{m,f^*}, i.e., $w_i \notin [x_{j-1}, x_j]$ for all $i = 1 \cdots m$.

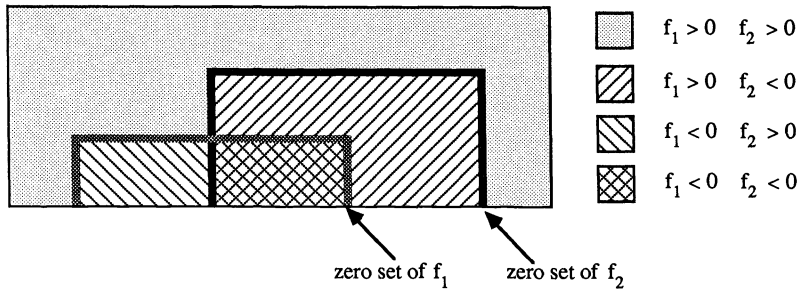


FIG. 3.1

Assume without loss of generality that j is odd, and define $f^{**}: C \to \mathbb{R}^2$ by

$$f^{**}(w) = (\min(h_1(w, x_1), h_1(w, x_3), \cdots, h_1(w, x_{j-2}), g_1(w, x_j + \zeta), h_1(w, x_{j+2}), \cdots, h_1(w, x_{4M-1})),$$
$$\min(h_2(w, x_1), h_2(w, x_3), \cdots, h_2(w, x_{j-2}), g_2(w, x_j + \xi), h_2(w, x_{j+2}), \cdots, h_2(w, x_{4M-1}))),$$

where ζ and ξ depend on j, as shown in Table 3.1. To see that f^{**} is a member of the class F, note that f^{**} is a minimum of Lipschitz functions with constant K, and so it too is Lipschitz with constant K. Obviously, for every $w \in \partial C - [x_{j-1}, x_{j+1}]$ we have $\|f^{**}(w)\| = d$. For every $w \in [x_{j-1}, x_{j+1}]$, the choice of the parameters ζ and ξ assures that $\|f^{**}(w)\| = d$, because either g_1 or g_2 is $= \pm d$ on that interval.

Now note that for every $w \in [x_j, x_{j+1}]$, we have $f^*(w) = f^{**}(w) = (\min(d, -d + K\|z - x_j\|), \max(-d, -3d + K\|z - x_j\|))$ (see Fig. 3.1 for the case with $0 < j < M$). Also note that $f^*(w) = f^{**}(w)$, for all w from ∂C such that $w \notin [x_{j-1}, x_{j+1}]$. This implies that $N_m(f^*) = N_m(f^{**})$. However, by the choice of ζ and ξ, f^{**} has exactly one zero in C (see Fig. 3.1), with the Jacobian at that zero $= \pm K^2$, which implies that $\deg(f^{**}, C, \theta_2) = \pm 1$. This completes the proof of Lemma 3.1.

4. Upper bound. Lemma 3.1 implies that the minimal cardinality number m^* must be greater than $4\lfloor K/4d \rfloor - 1$. In this section (in Lemma 4.1) we prove that m^* is equal to $4\lfloor K/4d \rfloor$, by exhibiting a nonadaptive information N_q, $q = 4\lfloor K/4d \rfloor$, such that it generates a refinement of ∂C with respect to the sign of f, and therefore allows us to calculate the degree for any f in F.

In § 2 of this paper we outlined the idea of an algorithm using N_q to compute the degree. Here (in Lemma 4.2) we exhibit the details of the algorithm and prove that it calculates the degree of any $f \in F$.

Let $f \equiv (f_1, f_2) \in F$; let $M \equiv 4d/K - \varepsilon$, for arbitrarily small $\varepsilon > 0$; and let $q = 4\lfloor 1/M \rfloor$. Observe that $q = 4\lfloor K/4d \rfloor$.

Let $x_0 = (0, 0)$ and x_j, $j = 1, \cdots, q$ be counterclockwise-(CCW-) ordered points along ∂C such that $\|x_j - x_{j+1}\| = M$, $j = 0, \cdots, q$, where $x_{q+1} = x_1$.

Define the information operator $N_q(f) \equiv [f(x_1), \cdots, f(x_q)]$.

TABLE 3.1

Range for j	Value for ζ	Value for ξ
$0 < j \leqq M$	$(-d/K, -d/K)$	$(d/K, d/K)$
$M < j \leqq 2m$	$(d/K, -d/K)$	$(-d/K, d/K)$
$2M < j \leqq 3M$	$(d/K, d/K)$	$(-d/K, -d/K)$
$3M < j \leqq 4M$	$(-d/K, d/K)$	$(d/K, -d/K)$

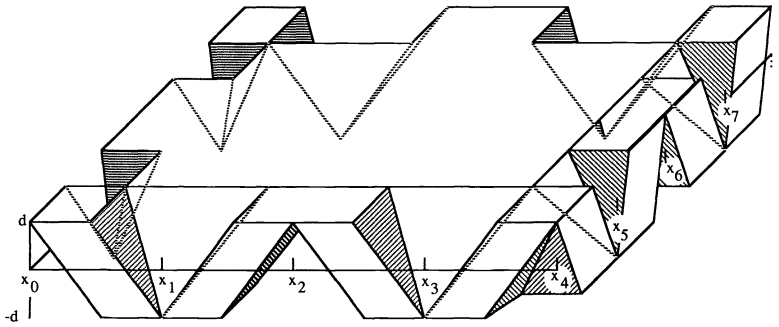


FIG. 3.2. *First component of function f^* for the case $d/K = 1/16$.*

LEMMA 4.1. *For every f in F, the information $N_q(f)$ uniquely determines the degree of f, i.e., for all $\tilde{f} \in F$, $N_q(f) = N_q(\tilde{f}) \Rightarrow \deg(f) = \deg(\tilde{f})$, by generating a partition P of ∂C such that either P is empty and the degree of f is zero, or P forms a sufficient refinement of ∂C with respect to the sign of f.*

Proof. For every function f in F we build a partition $P = \{p_j\}_{j=1}^{Q}$ of ∂C (where $0 \leqq Q < 2 \cdot q$ is the number of points in the partition, and Q depends on f) using only x_i and $f(x_i)$, $i = 1, \cdots, q$, such that P is either empty (and the degree of f is zero) or P forms a sufficient refinement of ∂C with respect to the sign of f.

First we construct the partition P. Assume that $f_1(x_1) \neq 0$; if not, then interchange the roles of f_1 and f_2 in the definition below.

DEFINITION 4.1. Let $i = j = 0$, and $P = \varnothing$. Then while $i \leqq q+1$, iteratively define the partition P as follows.

Let $i = j$ and redefine j as the smallest index $> i$ such that $f_1(x_j) \neq 0$. Then let

$$
P := \begin{cases}
P \quad \text{if } f_2(x_i) = f_2(x_j) = 0 & \\
P \cup \{\beta_L\} \cup \{\beta_R\} \quad \text{if } \exists z \in [x_i, x_j] \text{ such that } f_1(z) = 0, & \\
\quad \text{where } \beta_L, \beta_R \text{ are such that} & \\
\quad\quad [\beta_L, \beta_R] \subset (x_i, x_j),\ \|\beta_L - x_i\| < \|\beta_R - x_i\| & \\
\quad \text{and } \beta_L \in (x_i, x_{i+1}) \text{ is such that } f_1 \neq 0 \text{ on } [x_i, \beta_L], \text{ and} & \quad (4.1) \\
\quad\quad \exists z_1 \in [x_i, x_{i+1}] \text{ such that } f_1(z_1) = 0 \text{ and } \|z_1 - \beta_L\| \leqq M/2, & \\
\quad \text{and } \beta_R \in (x_{j-1}, x_j) \text{ is such that } f_1 \neq 0 \text{ on } [\beta_R, x_j], \text{ and} & \quad (4.2) \\
\quad\quad \exists z_2 \in [x_{j-1}, x_j] \text{ such that } f_1(z_2) = 0 \text{ and } \|z_2 - \beta_R\| \leqq M/2; & \\
P \cup \{\gamma_L\} \cup \{\gamma_R\} \quad \text{otherwise, (i.e., } j = i+1), & \\
\quad \text{where } \gamma_L, \gamma_R \text{ are such that} & \\
\quad\quad [\gamma_L, \gamma_R] \subset (x_i, x_{i+1}),\ \|\gamma_L - x_i\| < \|\gamma_R - x_i\|, \text{ and} & \\
\quad \|\gamma_R - x_i\| = \min(M/2, |f_2(x_i)|/(2K)) \quad \text{if } p = i & \quad (4.3) \\
\quad \|\gamma_L - x_{i+1}\| = \min(M/2, |f_2(x_{i+1})|/(2K)) \quad \text{if } p = i+1, & \quad (4.4)
\end{cases}
$$

where p is given by

$$
p = \begin{cases}
i & \text{if } ((|f_2(x_i)| < d \text{ or } |f_2(x_{i+1})| < d) \text{ and } |f_2(x_{i+1})| \leqq |f_2(x_i)|), \\
i+1 & \text{if } ((|f_2(x_i)| < d \text{ or } |f_2(x_{i+1})| < d) \text{ and } |f_2(x_{i+1})| > |f_2(x_i)|), \\
i & \text{if } ((|f_2(x_i)| \geqq d \text{ and } |f_2(x_{i+1})| \geqq d) \text{ and } |f_1(x_i)| \leqq |f_1(x_{i+1})|), \\
i+1 & \text{otherwise.}
\end{cases}
$$

□

Note that because we do not always add the same number of points to the partition, it is not possible to know in advance how many points will be in the partition. We let Q be the size of the partition, i.e., $P = \cup_{i=1}^{Q} \{p_i\}$, where $Q = 0$ if $P = \varnothing$.

First we show that if $P = \emptyset$, then $\deg(f, C, \theta) = 0$. Notice that the only information which will result in $P = \emptyset$ has $f_2(x_i) = 0$ for all i, $1 \leqq i \leqq q$. Thus information N_q is sufficient to determine if P is empty. Combining the fact that $f_2(x_i) = 0$ for all i with the definition of the class F, we see that there exists no z such that $|f_2(z)| \geqq d$ and $\|z - x_i\|_\infty < d/K$. In addition, we know that if $|f_2(z)| < d$, then $|f_1(x_i)| \geqq d$. Therefore, there exists no z such that $f_1(z) = 0$ and $\|z - x_i\|_\infty < 2d/K$. But recall that $\|x_i - x_{i+1}\|_\infty = M < 4d/k$; thus $f_1 \neq 0$ on the boundary of C^2.

We can now use the Poincaré-Bohl theorem to show that $\deg(f, C, \theta) = 0$ (see [16] for a full description of this theorem). Briefly, this theorem says that to conclude that $\deg(f, C, \theta) = \deg(g, C, \theta)$, for all $f, g \in F$ it is enough to show that the homotopy $h(t, z)$, $h(t, z) = t \cdot f(z) + (1 - t) \cdot g(z)$ is nonzero for every $t \in [0, 1]$ and every $z \in \partial C$.

Recall $f_1 \neq 0$ on the boundary of C^2, and without loss of generality let us assume that $f_1(z) > 0$ for all $z \in \partial C$. Then let $g(z) = (d, f_2(z))$ and $h(t, z) = t \cdot f(z) + (1 - t) \cdot g(z)$. Above, we have shown that $f_1 > 0$ on the boundary of C^2; thus the homotopy $h(t, z)$ obviously is nonzero for every $t \in [0, 1]$ and every $z \in \partial C$. Combining the fact that $\deg(g, C, \theta) = 0$ (since $g(z)$ has no zeros at all) with the Poincaré-Bohl theorem, we see that if $P = \emptyset$, then $\deg(f, C, \theta) = \deg(g, C, \theta) = 0$.

For the remainder of the paper let us assume that P is nonempty. In the above definition of the partition P, we define two types of segments: $\beta = [\beta_L, \beta_R]$ segments where f_1 has a zero; and $\gamma = [\gamma_L, \gamma_R]$ segments where f_1 does not have a zero. The endpoints of the β and γ segments form our partition. For clarity in the following discussion, let us index these segments (each type independently) using the CCW order, i.e., we have $\beta_j = [\beta_{j,L}, \beta_{j,R}]$, $j = 1 \cdots Q_1$, $\gamma_j = [\gamma_{j,L}, \gamma_{j,R}]$, $j = 1 \cdots Q_2$, where $Q_1 + Q_2 = Q/2$. Observe that the above definition results in the following properties:

(4.5) $\forall j$ and $\forall x \in \beta_j$, $\exists z \in \beta_j$ such that $f_1(z) = 0$ and $\|x - z\| \leqq M/2$. (This follows from (4.1)-(4.2) and $\|x_j - x_{j+1}\| = M$.)

(4.6) $\forall i, f_1(x_i) = 0$ if and only if $\exists j$ such that $x_i \in \beta_j$.

(4.7) If $\beta_i \subset [x_j, x_{j+1}]$, then $\forall z \in [x_j, x_{j+1}]$, $f_1(z) = 0 \Rightarrow z \in \operatorname{Int}(\beta_j)$.

(4.8) $\forall z \in \partial C$ such that $f_1(z) = 0$, $\exists j$ such that $z \in \operatorname{Int}(\beta_j)$.

(4.9) $\forall i, j, (\beta_i \cap \gamma_j) = \emptyset$.

(4.10) $\forall i, j, x_i \notin \gamma_j$.

(4.11) $\forall j, f_1 \neq 0$ on γ_j and $f_2 \neq 0$ on γ_j.

(4.12) $\forall i, \quad x_i \notin P$.

(4.13) A segment $[x_j, x_{j+1}]$ contains no points of P iff $f_1(x_j) = f_1(x_{j+1}) = 0$ or $f_2(x_j) = f_2(x_{j+1}) = 0$.

(4.14) A segment $[x_j, x_{j+1}]$ contains one point of P iff $f_1(x_j) = 0$ or $f_1(x_{j+1}) = 0$ but not both.

(4.15) A segment $[x_j, x_{j+1}]$ never contains more than two points of P.

Now we prove a less obvious property.

(4.16) Let z be a point such that $f_1(z)=0$. By definition of the class F, we have $|f_2(z)| \geqq d$. Since it would take a distance greater or equal d/K from z for $|f_1|$ to be $\geqq d$, and at least d/K more for f_2 to reach zero, there exists no point $y \in \partial C$ such that $f_2(y)=0$ and $\|y-z\| \leqq M/2 < 2d/K$.

Next we show that P forms a sufficient refinement of ∂C with respect to the sign of f for arbitrary f in F. Namely, for each segment $[L, R]$ $L, R \in P$ such that $(L, R) \cap P = \varnothing$, we show

$$f_1 \neq 0 \quad \text{on } [L, R] \quad \text{or} \quad f_2 \neq 0 \quad \text{on } [L, R] \tag{4.17}$$

and

$$f_1 \text{ and } f_2 \text{ are nonzero at both } L \text{ and } R. \tag{4.18}$$

That is, by property (4.8) we know that f_1 is of constant sign (and $\neq 0$) on all segments except when there exists i such that $[L, R] = [\beta_{i,L}, \beta_{i,R}]$. By properties (4.5) and (4.16), we have $f_2 \neq 0$ on $[\beta_{i,L}, \beta_{i,R}]$ for all i, which yields (4.17). By definition, we know that for all j, $f_1(\beta_{j,L}) \neq 0$ and $f_1(\beta_{j,R}) \neq 0$, which yields that $f_1 \neq 0$ at all points of P. That $f_2 \neq 0$ at all points of P follows from (4.11) and the above argument. Thus (4.18) holds, which completes the proof of Lemma 4.1. □

LEMMA 4.2. *For every function f in F, the following formulas hold for the sign of f_1 and f_2 at all points of the partition P from Lemma* 4.1, *contained in* $[x_i, x_{i+1}]$, *for all* $i = 1, \cdots, q$:

If $[x_i, x_{i+1}]$ *contains exactly one point T of P, then*

$$\begin{aligned} &\operatorname{sign} f_1(T) = \operatorname{sign} f_1(x_{i+1}) \quad \text{if } f_1(x_i)=0 \text{ and } f_1(x_{i+1}) \neq 0, \\ &\operatorname{sign} f_2(T) = \operatorname{sign} f_2(x_i) \end{aligned} \tag{4.19}$$

and

$$\begin{aligned} &\operatorname{sign} f_1(T) = \operatorname{sign} f_1(x_i) \quad \text{if } f_1(x_i) \neq 0 \text{ and } f_1(x_{i+1}) = 0, \\ &\operatorname{sign} f_2(T) = \operatorname{sign} f_2(x_{i+1}). \end{aligned} \tag{4.20}$$

If $[x_i, x_{i+1}]$ *contains two points L and R of P, then for* $j = 1, 2$

$$\begin{aligned} &\operatorname{sign} f_j(L) = \operatorname{sign} f_j(x_i) \quad \text{if } \operatorname{sign} f_2(x_i) = \operatorname{sign} f_2(x_{i+1}), \\ &\operatorname{sign} f_j(R) = \operatorname{sign} f_j(x_{i+1}) \end{aligned} \tag{4.21}$$

and

$$\begin{aligned} &\operatorname{sign} f_1(L) = \operatorname{sign} f_1(x_i) \quad \text{if } \operatorname{sign} f_2(x_i) \neq \operatorname{sign} f_2(x_{i+1}), \\ &\operatorname{sign} f_1(R) = \operatorname{sign} f_1(x_{i+1}), \\ &\operatorname{sign} f_2(L) = \operatorname{sign} f_2(R) = \operatorname{sign} f_2(x_p), \end{aligned} \tag{4.22}$$

where p is given in Definition 4.1.

COROLLARY. *Observe that* (4.13)-(4.15) *imply that for every f in F, we can calculate the number of points of P in each* $[x_i, x_{i+1}]$. *Thus using* (4.19)-(4.22) *we can calculate the degree of every f in F using* (1.4).

Proof of Lemma 4.2. For $[x_i, x_{i+1}]$ containing exactly one point of P, we have two cases:

If $f_1(x_i)=0$ and $f_1(x_{i+1}) \neq 0$, then for some j, $T = \beta_{j,r}$ and (4.2) implies $\operatorname{sign} f_1(T) = \operatorname{sign} f_1(x_{i+1})$. Moreover, (4.5) and (4.6) directly yield $\operatorname{sign} f_2(T) = \operatorname{sign} f_2(x_i)$.

If $f_1(x_i) \neq 0$ and $f_1(x_{i+1}) = 0$, then for some j, $T = \beta_{j,L}$ and (4.20) follows by similar arguments as (4.19).

Now we consider the case of $[x_i, x_{i+1}]$ containing exactly two points of P. Note that L and R in this case are the endpoints of a β_j or γ_j for some j. Moreover, (4.13) and (4.14) imply that $f_1(x_i) \neq 0 \neq f_1(x_{i+1})$ and ($f_2(x_i) \neq 0$ or $f_2(x_{i+1}) \neq 0$).

For (4.21) observe that if f_1 is not zero in $[x_i, x_{i+1}]$, then for some j, $L = \gamma_{j,L}$ and $R = \gamma_{j,R}$. Formula (4.21) for f_1 in this case is obvious and for f_2 follows directly from (4.4) and (4.11). If f_1 has a zero in $[x_i, x_{i+1}]$, then for some j, $L = \beta_{j,L}$ and $R = \beta_{j,R}$. Thus (4.7) yields the formula for f_1. As in (4.16), it follows that f_2 is of constant sign within $M/2 < 2d/K$ from a zero of f_1, which combined with (4.1), (4.2), and $\|x_i - x_{i+1}\| = M$ yields the formula for f_2.

Now we prove (4.22). If f_1 is not zero in $[x_i, x_{i+1}]$, then (4.22) obviously holds for f_1, and since in this case $L = \gamma_{j,L}$ and $R = \gamma_{j,R}$ for some j, then (4.22) for f_2 follows from (4.3), (4.4) and the definition of the index p. Suppose now that f_1 has some zero in $[x_i, x_{i+1}]$. Then, as in (4.20), for some j, $L = \beta_{j,L}$ and $R = \beta_{i,R}$. Thus (4.7) yields (4.22) for f_1. Since L and R are both within $M/2$ from some zero of f_1, then all zeros of f_2 must be in $(x_i, L) \cup (R, x_{i+1})$. Thus $\operatorname{sign} f_2(L) = \operatorname{sign}(R) = \operatorname{sign} f_2(\alpha_1)$, where α_1 is an arbitrary zero of f_1 from $[x_i, x_{i+1}]$. To show (4.22) for f_2 we prove that $\operatorname{sign} f_2(x_p) = \operatorname{sign} f_2(\alpha_1)$.

Assume first that $|f_2(x_i)| < d$ or $|f_2(x_{i+1})| < d$ and that $p = i$, i.e., $|f_2(x_{i+1})| \leqq |f_2(x_i)|$, and suppose by contrast that $\operatorname{sign} f_2(x_p) \neq \operatorname{sign} f_2(\alpha_1)$, where α_1 is any zero of f_1. Thus there exists α_2, a zero of f_2, in $[x_i, L] \subset [x_i, \alpha_1]$. Then $\|\alpha_2 - x_i\| \geqq |f_2(x_i)|/K$ and (4.16) yields $\|\alpha_2 - \alpha_1\| \geqq 2d/K$. It takes a distance d/K from α_1 for $|f_1|$ to become at least d, say at z, and a distance $d/K - |f_2(x_{i+1})|/K$ from z for $|f_2|$ to become less than d at x_{i+1}. The sum of these distances is obviously equal to $\|x_i - x_{i+1}\|$; thus

$$\begin{aligned}\|x_i - x_{i+1}\| &= \|\alpha_2 - x_i\| + \|\alpha_2 - \alpha_1\| + \|\alpha_1 - z\| + \|z - x_{i+1}\| \\ &\geqq |f_2(x_i)|/K + 2d/K + d/K + d/K - |f_2(x_{i+1})|/K \\ &\geqq 4d/K > \|x_i - x_{i+1}\|,\end{aligned}$$

a contradiction. The proof for $p = i+1$ is essentially the same.

Assume finally that $|f_2(x_1)| \geqq d$ and $|f_2(x_{i+1})| \geqq d$. Then $|f_1(x_i)| < d$ or $|f_1(x_{i+1})| < d$ since it is impossible for both $|f_1|$ and $|f_2|$ to be at least d at both x_i, x_{i+1} and have zeros in $[x_i, x_{i+1}]$.

As before, we show that the $\operatorname{sign} f_2(x_p) = \operatorname{sign} f_2(\alpha_1)$ for some zero α_1 of f_1. Suppose the contrary, and also assume that $p = i$. Then there would exist a zero of f_2, α_2, in $[x_i, \alpha_1]$. Since in this case $|f_1(x_i)| < d$, then

$$\begin{aligned}\|x_i - x_{i+1}\| &= \|x_i - \alpha_2\| + \|\alpha_2 - \alpha_1\| + \|\alpha_1 - x_{i+1}\| \\ &= d/K - |f_1(x_i)|/K + d/K + 2d/K + |f_1(x_{i+1})|/K \\ &\geqq 4d/K > \|x_i - x_{i+1}\|,\end{aligned}$$

a contradiction. The proof for $p = i+1$ is essentially the same. This finally completes the proof of Lemma 4.2. □

5. Complexity of the problem and of our algorithm. In this section we examine the complexity, i.e., minimal cost, of the problem of the calculation of topological degree for functions from our class. We then give the complexity of our algorithm, φ^*, and conclude that it is an almost optimal complexity algorithm.

We assume that each arithmetic operation ($+, -, *, \div, abs(\,)$), each logical operation (and, or, not) or comparison ($>, <, \geqq, \leqq, =, \neq$) costs unity, and that each function evaluation costs c.

Lemma 3.1 implies that any algorithm that solves the problem must use at least $m^* = 4 * \lfloor K/4d \rfloor$ function evaluations. Thus a lower bound on the computational complexity is $m^* \times (c+2) - 1$, since there are $2m^*$ numbers that must go into the calculation and combining them takes at least $2m^* - 1$ operations.

The complexity (cost) of our algorithm, φ^*, is equal to the cost of computing the information and the algebraic complexity of the algorithm. The location of the evaluation points can, for known K and d, be calculated once (i.e., precomputed), in which case the cost of the information would be $m^* \times c$. However, the algorithm as implemented in [2], [3] calculates the location of the next information point based on the location of the current one. This approach results in an information cost of $m^* \times (c+17)$. The algebraic complexity for φ^* is $31 \times m^* + 12$. Thus the computational complexity of φ^* as implemented is $m^* \times (c+48) + 12$. However, with precomputation of the information points and removal of the bells and whistles, we can reduce the algorithm cost down to $m^* \times (c+20) + 5$. Therefore φ^* is an almost optimal complexity algorithm.

6. Numerical results. We report on some of the numerical results of the algorithm implemented in [2], [3]. The algorithm presented therein is an extension of the algorithm described here to the case where the domain of interest is an arbitrary polygon in $\mathbb{R}^2$. The extension is trivial and consists of choosing points on the boundary of the polygon separated by a distance $1/\lfloor K/(4d) \rfloor$ in the infinity norm.

The algorithm was implemented in FORTRAN IV and the timing results are from an HP-9000 (a Motorola-68020-based workstation with floating point coprocessor). In Table 6.1 we present the timing results for a number of test functions. The CPU seconds were obtained by averaging the total CPU time of 10,000 successive calls to the algorithm to reduce the effect of extraneous CPU time that is counted in the system timing figures (e.g., loading the algorithm into memory, page swaps, user timeouts, etc.).

TABLE 6.1
Timing results for proper use of program.

Function No.	Domain	Degree	K	d	No. of Evaluations	CPU seconds $\times 10^{-4}$
1	$[0,1]^2$	1	80.000	.0100	8,000	59.3
1	$[0,1]^2$	1	80.000	.0050	16,000	99.4
2	$[0,1]^2$	0	4.000	.5000	8	0.1
2	$[0,1]^2$	0	4.000	.0500	80	0.5
3	$[-2,2]\times[-.25,.25]$	−1	8.000	.4375	40	0.5
3	$[-2,2]\times[-.25,.25]$	−1	8.000	.0100	1,800	11.0
4	$[0,1]^2$	−1	10.000	.0100	1,000	7.3
4	$[0,1]^2$	−1	10.000	.0010	10,000	64.2
5	$[-1,1]^2$	2	4.000	.8284	8	0.1
5	$[-1,1]^2$	2	4.000	.3300	24	0.3
6	$[-4,-4],[0,4],[4,0]$	2	32.000	.4000	400	2.5
6	$[-4,-4],[0,4],[4,0]$	2	32.000	.0100	15,998	93.5
7	$[-1,1]^2$	−3	12.000	.6667	36	0.2
7	$[-1,1]^2$	−3	12.000	.0067	3,596	22.9
8	$[-4,-4],[0,4],[4,0]$	3	64.000	4.3300	73	0.5
8	$[-4,-4],[0,4],[4,0]$	3	64.000	.3300	968	6.4
9	$[-1,1]^2$	4	36.000	.7000	76	0.6
9	$[-1,1]^2$	4	36.000	.0700	772	6.8
10	$[-4,-4],[0,4],[4,0]$	4	256.000	7.0000	183	1.5
10	$[-4,-4],[0,4],[4,0]$	4	256.000	.7000	1,828	13.6

The test functions have been chosen to verify the algorithm in many different cases. We have included oscillatory functions having several zeros (F_1); polynomials (F_2, F_3); oscillatory functions with exactly one zero (F_4); and the functions F_5, F_6, F_7, which have been used by other authors (e.g., see [12]).

In several other tests, see Table 6.1, we have overestimated the Lipschitz constant K and underestimated d, the minimum of the norm on the boundary D. With such a choice of parameters, the algorithm is *guaranteed* to compute the correct value for the degree, albeit with an above-minimal cost. On the other hand, if one underestimates K and/or overestimates d, the algorithm is only a heuristic, and in general may not compute the correct degree.

As a final series of tests (see Table 6.2), we have underestimated the Lipschitz constant K. (We do not underestimate d since the algorithm checks to see that no information points violate that constraint.) With such a choice of parameters, the algorithm is *not guaranteed* to compute the correct value for the degree; however, one might consider it a heuristic algorithm. As can be seen in the example, even when K is underestimated by a factor of two or more, most of the test results in the correct degree computation.

TABLE 6.2
Timing results for heuristic use of program.

Function No.	Domain	Degree	K	d	No. of Evaluations	CPU seconds $\times 10^{-4}$
1	$[0,1]^2$	1	.800	.0050	160	1.1
2	$[0,1]^2$	0	2.000	.5000	4	0.1
3	$[-2,2]\times[-.25,.25]$	−1	2.000	.4375	8	0.1
4	$[0,1]^2$	−1	3.000	.0100	300	2.1
5	$[-1,1]^2$	−1†	2.000	.8284	4	0.1
6	$[-4,-4],[0,4],[4,0]$	2	8.000	.4000	98	0.8
7	$[-1,1]^2$	−3	6.000	.6667	16	0.3
8	$[-4,-4],[0,4],[4,0]$	2†	16.000	4.3300	18	0.3
8	$[-4,-4],[0,4],[4,0]$	3	32.000	4.3300	36	0.4
9	$[-1,1]^2$	4	16.000	.7000	34	0.5
10	$[-4,-4],[0,4],[4,0]$	4	128.000	7.0000	90	0.8

† Incorrect values computed.

Test function 1: $F_1 = ((\sin(11.5\times\pi\times x)+.01), \cos(11\times\pi\times y))$. This function has 121 zeros within the unit square, the minimum value of its infinity norm on the boundary is .01, and it is Lipschitz with constant $2\times 11.5\times\pi < 80.0$.§ The correct degree is 1.

Test function 2: $F_2 = ((x^2+y^2-.5), (2xy-.5))$. This function has one zero (of multiplicity 2) within the unit square, the minimum value of its infinity norm on the boundary is .5, and it is Lipschitz with constant 4. The correct degree is 0.

Test function 3: $F_3 = ((x^2-4y), (y^2-2x+4y))$ with the domain as the rectangle $-2\leqq x\leqq 2$ and $-.25\leqq y\leqq .25$. This function has one zero within the domain, the minimum value of its infinity norm on the boundary is $\geqq .4375$, and it is Lipschitz with constant 8. The correct degree for this function is −1.

Test function 4: $F_4 = ((\sin(1.5\times\pi\times x)+.01), -\cos(\pi\times y))$. This function has one zero within the unit square, the minimum value of its infinity norm on the boundary

§ Remember, we are using the infinity norm.

is .01, and it is Lipschitz with constant $2 \times 1.5 \times \pi < 10.0$. The true degree of this function in this domain is -1.

Test function 5: $F_5 = ((x^2 - y^2), (2xy))$. The function was tested over the square $[-1, 1] \times [-1, 1]$, on which the function has degree 2. The minimum value of its infinity norm on the boundary is $>.8284$ (i.e., $2.0 * \sqrt{2} - 1$), and it is Lipschitz with constant 4.0.

Test function 6: $F_6 = z^2$ over the triangular region defined by vertices $[-4, -4]$, $[4, -4]$, and $[0, -4]$. This is the region used in [12] and is presented for comparison purposes. On this region the minimum value of its infinity norm on the boundary is $>.4$ (computed numerically using knowledge of the Lipschitz constant), and it is Lipschitz with constant 32.0. The true degree of this function in this domain is obviously 2.

Test function 7: $F_7 = ((x^3 - 3xy^2), (y^3 - 3x^2y))$. The function was tested over the square $[-1, 1] \times [-1, 1]$, on which the function has degree -3. The minimum value of its infinity norm on the boundary is $>.66666$, and it is Lipschitz with constant 12.0.

Test function 8: $F_8 = z^3 = ((x^3 - 3xy^2), (-y^3 + 3x^2y))$ over the triangular region defined by vertices $[-4, -4]$, $[4, -4]$, and $[0, -4]$. This is the region used in [12] and is presented for comparison purposes. In this region the function has degree 3. The minimum value of its infinity norm on the boundary is >4.33333, and it is Lipschitz with constant 64.0.

Test function 9: $F_9 = z^4 = ((x^4 + y^4 - 6x^2y^2), (4xy \cdot (x^2 - y^2)))$. The function was tested over the square $[-1, 1] \times [-1, 1]$, on which the function has degree 4. The minimum value of its infinity norm on the boundary is $>.7$, and it is Lipschitz with constant 36.

Test function 10: $F_{10} = z^4 = ((x^4 + y^4 - 6x^2y^2), (4xy \cdot (x^2 - y^2)))$ over the triangular region defined by vertices $[-4, -4]$, $[4, -4]$, and $[0, -4]$. This is the region used in [12] and is presented for comparison purposes. In this region the function has degree 4. The minimum value of its infinity norm on the boundary is >7.0, and it is Lipschitz with constant 256.0.

REFERENCES

[1] E. L. ALLGOWER AND K. GEORG, *Simplicial and continuation methods for approximating fixed points and solution to systems of equations*, SIAM Rev., 22 (1980), pp. 28–85.

[2] T. E. BOULT, *Information-based complexity in nonlinear equations and computer vision*, Ph.D. thesis, Computer Science Department, Columbia University, NY, 1986.

[3] T. E. BOULT AND K. SIKORSKI, *A Fortran subroutine for computing topological degree of Lipschitz functions*, in progress.

[4] ———, *Complexity of computing topological degree of Lipschitz functions in N dimensions*, J. Complexity, 2 (1986), pp. 44–59.

[5] B. C. EAVES, *A short course in solving equations with PL homotopies*, SIAM-AMS Proc., 9 (1976), pp. 73–143.

[6] A. EIGER, K. SIKORSKI, AND F. STENGER, *A method of bisection for solving n nonlinear equations*, ACM Trans. Math. Software, 10 (1984), pp. 367–377.

[7] C. B. GARCIA AND W. I. ZANGWILL, *Pathways to Solutions, Fixed Points and Equilibria*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[8] E. HANSEN, *Global optimization using interval analysis—the multidimensional case*, Numer. Math., 34 (1980), pp. 247–270.

[9] E. HANSEN AND S. SENGUPTA, *Bounding solutions of systems of equations using interval analysis*, BIT, 21 (1981), pp. 203–211.

[10] C. HARVEY AND F. STENGER, *A two-dimensional analogue to the method of bisection for solving nonlinear equations*, Quart. Appl. Math., 33 (1976), pp. 351–368.

[11] R. B. KEARFOTT, *Computing the degree of maps and a generalized method of bisection*, Ph.D. thesis, University of Utah, Salt Lake City, UT, 1977.

[12] R. B. KEARFOTT, *An efficient degree-computation method for a generalized method of bisection*, Numer. Math., 32 (1979), pp. 109-127.

[13] ———, *On a general technique for finding directions proceeding from bifurcation points*, in Numerical Methods for Bifurcation Problems, T. Küpper, H. D. Mittlemann, and H. Weber, eds., Birkhäuser, Boston, 1984, pp. 210-218.

[14] ———, *Abstract generalized bisection and a cost bound*, Math. Comp., 49 (1987), pp. 187-202.

[15] R. E. MOORE AND S. T. JONES, *Safe starting regions for iterative methods*, SIAM J. Numer. Anal., 14 (1977), pp. 1051-1065.

[16] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[17] H. O. PEITGEN AND M. PRÜFER, *A constructive approach to global bifurcation, the Schauder continuation method and coincidence problems*, in Approximation of Fixed Points and Functional Differential Equations, H. O. Peitgen, ed., Springer Lecture Notes, Springer-Verlag, New York, 1979.

[18] M. PRÜFER AND H. W. SIEGBERG, *On computational aspects of topological degree in* $\mathbb{R}^n$, Preprint No. 252, Sonderforschungsbereich 72, Approximation und Optimierung, Universität Bonn.

[19] M. PRÜFER, *Calculating global bifurcation*, in Continuation Methods, H. Wacher, ed., Academic Press, New York, 1978, pp. 187-214.

[20] K. SIKORSKI, *A three-dimensional analogue to the method of bisection for solving nonlinear equations*, Math. Comp., 33 (1979), pp. 722-738.

[21] ———, *Minimal number of function evaluations for computing topological degree in two dimensions*, Tech. Report, Department of Computer Science, Columbia University, New York, NY, 1984.

[22] F. STENGER, *An algorithm for computing the topological degree of a mapping in* $\mathbb{R}^n$, Numer. Math., 25 (1976), pp. 23-28.

[23] M. STYNES, *An algorithm for numerical calculation of topological degree*, Appl. Anal., 9 (1979), pp. 63-77.

[24] ———, *A simplification of Stenger's topological degree formula*, Numer. Math., 33 (1979), pp. 147-156.

[25] ———, *On the construction of sufficient refinements for computation of topological degree*, Numer. Math., 37 (1981), pp. 453-462.

[26] M. J. TODD, *The computation of fixed points and applications*, Springer Lecture Notes in Economics and Mathematical Systems 124, Springer-Verlag, New York, 1976.

[27] J. F. TRAUB, G. W. WASILKOWSKI, AND H. WOŹNIAKOWSKI, *Information, Uncertainty, and Complexity*, Addison-Wesley, Reading, MA, 1983.

[28] J. F. TRAUB AND H. WOŹNIAKOWSKI, *Information and computation*, Adv. Computers, 23 (1984), pp. 35-92.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 699-717, July 1989
© 1989 Society for Industrial and Applied Mathematics
006

ON THE ROBUSTNESS OF ILU SMOOTHING*

GABRIEL WITTUM†

Abstract. In the present paper, a detailed analysis of a multigrid method with an ILU smoother applied to a singularly perturbed problem is given. Based on the analysis of a simple anisotropic model problem, a variant of the usual incomplete LU factorization is introduced, which is especially suited as robust smoother. For this variant a detailed analysis and a proof of robustness is given. Furthermore, some contradictions between the smoothing rates predicted by local Fourier analysis and the practically observed convergence factors are explained (see [W. Hackbusch, *Multi-grid Methods and Applications*, Springer-Verlag, Berlin, Heidelberg, 1985; R. Kettler, "Analysis and comparison of relaxation schemes in robust multi-grid and preconditioned conjugate gradient methods," in *Multi-grid Methods, Lecture Notes in Math.* 960, Springer-Verlag, Berlin, 1982; C. A. Thole, *Beiträge zur Fourieranalyse von Mehrgitterver fahren*, Diplomarbeit, Universität Bonn, 1983]. The theoretical results are confirmed by numerical tests.

Key words. singular perturbation problems, robust multigrid methods, anisotropic problem, ILU factorizations, robust smoothing, smoothing property, local Fourier analysis, convergence proof

AMS(MOS) subject classifications. 65F10, 65N20, 65N30

1. Introduction.

The treatment of singular perturbations is a severe problem in solving partial differential equations. The discretization of a singularly perturbed operator contains characteristic difficulties which may affect multigrid convergence, as the discretization process is hidden in the hierarchy of grids Ω_l, $l=1,\cdots,1_{\max}$, used in multigrid technique. To handle such problems, "robust" multigrid methods are required. In the present paper, we introduce a robust multigrid method based on a variant of incomplete LU smoothing. This method is proven to be robust for the anisotropic model problem and related problems. The theoretical results are confirmed by numerical tests.

1.1. Preliminaries and notations.

An operator $K=K(\varepsilon)$ depending on a parameter ε is called singularly perturbed, if the limiting operator $K(0)=\lim_{\varepsilon\to 0}K(\varepsilon)$ is of a type other than $K(\varepsilon)$ for $\varepsilon>0$. We are especially interested in operators of the form

$$K(\varepsilon)=\varepsilon K^{\mathrm{I}}+K^{\mathrm{II}} \tag{1.1.1}$$

where $K(\varepsilon)$ is elliptic for $\varepsilon>0$ and K^{II} is nonelliptic or elliptic of lower order.

A simple model for this situation is

$$K(\varepsilon)u=f \quad \text{in } \Omega, \qquad u=g \quad \text{on } \partial\Omega \tag{1.1.2a}$$

with

$$K(\varepsilon)=-\left(\varepsilon\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right) \tag{1.1.2b}$$

in a domain $\Omega\subset\mathbf{R}^2$. For the moment, let $\Omega=(0,1)\times(0,1)$ and (1.1.1) be discretized by

$$K_l(\varepsilon)=h_l^{-2}\begin{bmatrix} & -1 & \\ -\varepsilon & 2(1+\varepsilon) & -\varepsilon \\ & -1 & \end{bmatrix} \tag{1.1.3}$$

* Received by the editors April 25, 1988; accepted for publication (in revised form) August 10, 1988. This work was supported in part by Deutsche Forschungsgemeinschaft.

† SFB 123, Universität Heidelberg, Im Neuenheimer Feld 294, D-6900 Heidelberg, Federal Republic of Germany.

on a rectangular equidistant grid with stepsize h_l, ordering the gridpoints lexicographically. Stencil (1.1.3) applies to gridpoints lying totally inside Ω. For points having a boundary point as neighbor, the corresponding entry of the stencil has to be skipped. For the stencil notation of difference operators, see [8].

This problem is very simple; however, some typical difficulties arising from singular perturbations show up there and can be analyzed thoroughly. In addition, the insight obtained by the analysis of the model problem gives rise to certain generalizations as presented in § 2.2. The restriction to $\Omega = (0, 1) \times (0, 1)$ is only to simplify the notation of the technical parts in § 2. The results, however, apply to a general region (see Theorem 2.6).

Concerning multigrid technique, we refer to Hackbusch's book [7]. The two-grid operator, corresponding to a two-grid method on levels l and l-1 is given by

$$T_{2,l}(\nu, 0) = (I - pK_{l-1}^{-1} rK_l) S_l^{\nu}, \tag{1.1.4}$$

using ν steps of a smoother S_l, a prolongation p, and a restriction r. We restrict our analysis to this two-grid operator, as from there the convergence of the corresponding multigrid method can be concluded easily (see [7, § 7]).

Given two matrices A and B, we denote

$$A > B \qquad (A \geqq B) \tag{1.1.5}$$

as the positive (semi-)definiteness of the matrix A-B.

1.2. Background. There are several approaches treating problems of type (1.1.1) by multigrid methods. On the one hand there are methods based on special coarsening techniques and sophisticated extensions of Gauss–Seidel smoothers (see [3], [18]); on the other hand there are methods based on incomplete LU smoothing. Furthermore, Hackbusch introduced the frequency-decomposition multigrid method in [9]. This method looks very promising towards a general-purpose robust and efficient multigrid method being applicable even to three-dimensional problems. In the two-dimensional case, however, ILU will still be competitive, at least for nonparallel computers. The ILU smoothers were introduced by Wesseling and Kettler (cf. [21]–[23], [13]) on account of their "robustness." For robustness we give the following definition.

DEFINITION 1.1. Denote by $\zeta(\varepsilon)$ the contraction number of a multigrid method for the singularly perturbed problem (1.1.1). We call the method $K_l(\varepsilon)$-robust if and only if

$$\zeta(\varepsilon) \leqq \zeta < 1 \quad \forall \varepsilon \geqq 0 \tag{1.2.1}$$

holds.

Although this term was widely used, there was no proof or rigorous theoretical treatment of robustness up until now. The only analysis existing was given by Hackbusch, establishing robustness for a special line Jacobi smoother applied to problem (1.1.2) (see [7, Lemma 10.1.2]).

Wesseling and Kettler found the robustness of ILU for model problem (1.1.2) to be due to the fact that ILU solves $K_l(0)$ exactly. This fundamental observation led Hackbusch to the criterion that a smoothing iteration suitable for a singularly perturbed problem $K_l(\varepsilon)$ has to be a very fast or even exact solver for the limit case $K_l(0)$ (cf. [7, Crit. 10.1.1]). Actually, this is the main tool for achieving robustness and it is equivalent to assumption (2.1.10a); unfortunately it is not sufficient as results from our analysis. This also becomes visible when doing computations on extremely fine grids using a straightforward ILU approach. There the convergence rate deteriorates for certain values of ε. These difficulties led to a very different esteem of ILU smoothing

among several groups. This is the reason we introduced some variant called ILU_β controlled by a parameter β, similar to the modifications of incomplete LU decompositions as suggested by Gustafsson [5] and Jennings and Malik [12]. In § 2 we prove robustness of ILU_β for problem (1.1.2) and a generalized problem for certain values of β.

Section 3 is concerned with the role of local Fourier analysis to predict convergence rates. Influences of the finite dimension of the discrete problems are usually neglected, thus providing unrealistic results for ILU smoothing applied to problem (1.1.2) (see [19], [13]). These discrepancies are explained and give rise to a slight modification of local mode analysis.

In § 4 we optimize the parameter β, using the estimates deduced in § 2. We check this prediction of the optimal parameter by comparing it with the results from practical computations performed on problem (1.1.2).

We were led to the ILU_β variant by a careful analysis of robustness and the behavior of incomplete LU smoothing, based on the convergence theory by Hackbusch (see [7]) and some extensions by the author (see [24]). Using local Fourier techniques Khalil [14] and Oertel [17] suggested the same modification in lectures during the Fourth Gesellschaft für Angewandte Mathematik und Mechanik Seminar on Robust Multigrid Methods in Kiel, 22–24 January, 1988. We presented the main results of the present paper at the same conference.

2. A robust smoother.

2.1. The model problem. Let K_l from (1.1.3) be split into

$$K_l = M - N \tag{2.1.1}$$

with

$$M = (L+D)D^{-1}(L+D)^T \tag{2.1.2}$$

and

$$D = h^{-2}\,\mathrm{diag}\,\{d_{i,j}\}_{i=1,\cdots,m,j=1,\cdots,m}, \tag{2.1.3a}$$

$d_{i,j}$ corresponding to the gridpoint $x=(jh, ih)$, $h=1/(m+1)$, and

$$L = h^{-2}\begin{bmatrix} & \cdot & \\ -\varepsilon & 0 & \cdot \\ & -1 & \end{bmatrix}. \tag{2.1.3b}$$

The entries of D are defined by

$$d_{i,j} = \begin{cases} 2(1+\varepsilon), & i=j=1, \\ 2(1+\varepsilon)-\varepsilon(\varepsilon-\beta)/d_{i,j-1}, & i=1,\quad j>1, \\ 2(1+\varepsilon)-(1-\beta\varepsilon)/d_{i-1,j}, & i>1,\quad j=1 \\ 2(1+\varepsilon)-\varepsilon(\varepsilon-\beta)/d_{i,j-1}-(1-\varepsilon\beta)/d_{i-1,j}, & i,j>1 \end{cases} \tag{2.1.4}$$

with $\beta \in \mathbf{R}$. We are especially interested in $\beta \in [0, 1]$.

The corresponding iterative method is given by

(2.1.5) Let $x^{(0)}$ be given. Compute $x^{(i+1)}$ from $x^{(i)}$ by $x^{(i+1)} = x^{(i)} - M^{-1}(K_l x^{(i)} - b)$.

We call this iteration ILU_β.

For certain values of β we regain well-known ILU methods. Thus ILU_0 is identical with the usual five-point ILU factorization, as described in [16]. ILU_{-1} is identical with the so-called modified incomplete LU decomposition by Gustafsson [5], introduced to improve the asymptotic condition number of the preconditioned system,

while ILU_1 corresponds to a modification suggested by Jennings and Malik [12] yielding a positive definite rest matrix. These modifications were applied as preconditioners for *cg* algorithms. In multigrid framework, only Hemker [10] investigated some modifications of the standard ILU algorithm.

To prove robustness of a two-grid method with this smoother, we consider the splitting

$$\|T_{2,l}(\nu,0)\| \leqq \|K_l^{-1} - pK_{l-1}^{-1}r\| \, \|K_l S_l^\nu\| \tag{2.1.6}$$

of a suitable norm of the two-grid operator $T_{2,l}(\nu,0)$ from (1.1.4), which is the basic tool of Hackbusch's multigrid convergence theory (cf. [7]). The factors are estimated separately by the "approximation property"

$$\|K_l^{-1} - pK_{l-1}^{-1}r\| \leqq C_A h_l^2, \tag{2.1.7}$$

and the "smoothing property"

$$\|K_l S_l^\nu\| \leqq C_S h_l^{-2} \eta(\nu) \tag{2.1.8a}$$

with

$$\eta(\nu) \to 0 \quad \text{for } \nu \to \infty. \tag{2.1.8b}$$

In [7] Hackbusch uses a more general definition of smoothing property (2.1.8) in order to also incorporate diverging iterations as smoothers. For our purpose, however, definition (2.1.8) is sufficient.

This construction applies to the case of $\varepsilon = \text{const}$. Considering $K_l(\varepsilon)$ with ε as parameter, we realize that now the approximation property behaves like

$$\|K_l^{-1}(\varepsilon) - pK_{l-1}^{-1}(\varepsilon)r\| \leqq C_A \frac{1}{\varepsilon} h_l^2 \tag{2.1.7ε}$$

(cf. [7, p. 202]), where $\|\cdot\|$ now and in the following denotes the Euclidean or the spectral norm. There are, in general, two possibilities to get rid of the factor $1/\varepsilon$. First, the coarse-grid corrrection may be improved, say by semicoarsening and related, very specialized techniques (cf. [18]), or as a general purpose approach by the frequency decomposition multigrid method, introduced by Hackbusch in [9]. Then the factor $1/\varepsilon$ drops out of (2.1.7ε). Second, we may use a smoother satisfying

$$\|K_l(\varepsilon) S_l^\nu\| \leqq C_S h_l^{-2} \varepsilon \eta(\nu) \tag{2.1.8aε}$$

with $\eta(\nu)$ from (2.1.8b). Wesseling and Kettler followed this method when introducing their robust multigrid method (cf. [21], [13]). The following criterion gives sufficient conditions for (2.1.8aε).

CRITERION 2.1. *Let $K_l(\varepsilon)$ be a symmetric and positive definite matrix. Further, let $K_l(\varepsilon)$ be split according to* (2.1.1) *with symmetric and positive definite M. Suppose further*

$$\exists \alpha > 2: \quad K_l(\varepsilon) + \alpha N \geqq 0 \quad \forall \varepsilon > 0, \tag{2.1.9}$$

$$\|N\| \leqq \varepsilon h^{-2} C_N, \tag{2.1.10a}$$

and

$$\|S_l\| \leqq C. \tag{2.1.10b}$$

Then (2.1.8aε) *is valid with* $\eta(\nu) = \eta_0(\nu - 2, 1/(\alpha - 1))$, η_0 *from* (2.1.12b).

Proof. As $M = M^T > 0$, there exists $M^{1/2} = M^{1/2^T} > 0$. Let $X := M^{-1/2} N M^{-1/2}$. Since $K_l(\varepsilon) > 0$, we get $X = I - M^{-1/2} K_l(\varepsilon) M^{-1/2} \leqq I$. On account of (2.1.9), $(1+\vartheta)/\vartheta N + K_l(\varepsilon) \geqq 0$ holds with $\vartheta = 1/(\alpha - 1)$. Using this, we obtain $0 \leqq (1+\vartheta)M - K_l(\varepsilon)$. By virtue of that and by the symmetry of the matrices involved, we

end up with $(I+\vartheta)I - M^{-1/2}K_l(\varepsilon)M^{-1/2} \geqq 0$ and conclude that $X \geqq -\vartheta I$. Thus $-\vartheta I \leqq X \leqq I$ holds. Using (2.1.2), the symmetry of the problem, the following lemma, and $\|S_l\| \leqq C$, we obtain

$$\|K_l(\varepsilon)S_l^\nu\| = \|(M-N)(M^{-1}N)^\nu\| = \|M^{1/2}X(I-X)X^{\nu-2}XM^{1/2}\|$$
$$\leqq \|(I-X)X^{\nu-2}\|\,\|S_l\|\,\|N\| \leqq CC_N\varepsilon h^{-2}\eta_0(\nu-2, 1/(\alpha-1)). \qquad \square$$

Criterion 2.1 is a generalization of the criterion used in [24] to prove the smoothing property for incomplete LU decompositions. The technique is based on Hackbusch's work on the smoothing property (see [7, § 6.2]).

LEMMA 2.2. *Let A be symmetric. Furthermore let*

$$-\vartheta I \leqq A \leqq I. \tag{2.1.11}$$

Then the spectral norm of $A^\nu(I-A)$ is bounded by

$$\|A^\nu(I-A)\| \leqq \eta_0(\nu, \vartheta) \tag{2.1.12a}$$

with

$$\eta_0(\nu, \vartheta) = \max\{\nu^\nu/(\nu+1)^{(\nu+1)}, (1+\vartheta)\vartheta^\nu\}. \tag{2.1.12b}$$

Proof. According to [7, Lemma 1.3.5(iii)], we have

$$\|A^\nu(I-A)\| \leqq \max_{-\vartheta \leqq x \leqq 1} |x^\nu(1-x)| = \max\{\nu^\nu/(\nu+1)^{(\nu+1)}, (1+\vartheta)\vartheta^\nu\}. \qquad \square$$

To apply Criterion 2.1 to ILU_β, we prove the following lemma.

LEMMA 2.3. *Let $K_l(\varepsilon)$ from* (1.1.3) *be split as given in* (2.1.2)–(2.1.4). *Then* (2.1.9) *and* (2.1.10) *hold for $\beta \in (0, 1)$.*

Proof. For (2.1.10a). N from (2.1.1) is given by

$$N = h^{-2}\begin{bmatrix} \gamma_{ij}^r & & \\ & \beta(\gamma_{ij}^r+\gamma_{ij}^l) & \\ & & \gamma_{ij}^l \end{bmatrix} \tag{2.1.13}$$

with

$$\begin{cases} \gamma_{ij}^r = \dfrac{\varepsilon}{d_{i,j-1}} & \text{for } i \geqq 1 \text{ and } j > 1, \\ \gamma_{ij}^l = \dfrac{\varepsilon}{d_{i-1,j}} & \text{for } i > 1 \text{ and } j \geqq 1. \end{cases} \tag{2.1.14}$$

As $d \geqq d_{i,j} \geqq \delta$ (d, δ from (2.1.27)), (2.1.10) holds.

For (2.1.10b) Using (2.1.3), (2.1.26c), and (2.1.27c), assumption (2.1.10b) is readily proven with $C = 1$ by $\|S_l\| \leqq \|M^{-1}\|\,\|N\| \leqq 1$.

For (2.1.9). We split $(K_l(\varepsilon)+\alpha N)$ in the following way. Let $i, j \leqq m$; then

$$h^2(K_l(\varepsilon)+\alpha N) = \begin{bmatrix} \alpha\gamma_{ij}^r & -1 & \\ -\varepsilon & 2(1+\varepsilon)+\alpha\beta(\gamma_{ij}^r+\gamma_{ij}^l) & -\varepsilon \\ & -1 & \alpha\gamma_{ij}^l \end{bmatrix} =: A$$
$$= B_1 + B_2$$

with

$$B_1 = \begin{bmatrix} q\alpha\varepsilon/\delta & -1 & \\ -\varepsilon & 2(1+\varepsilon+(\alpha\beta\varepsilon/\delta)) & -\varepsilon \\ & -1 & q\alpha\varepsilon/\delta \end{bmatrix} \text{ and} \tag{2.1.15a}$$

(2.1.15b)

$$B_2 = \begin{bmatrix} \alpha\varepsilon((1/d_{i,j-1}) - q/\delta) & & \\ & \alpha\beta\varepsilon((1/d_{i,j-1}) + (1/d_{l-j,j}) - 2/\delta) & \\ & & \alpha\varepsilon((1/d_{i-1,j}) - q/\delta) \end{bmatrix}$$

where

$$q = \frac{1-\beta}{2} \tag{2.1.16}$$

and δ from (2.1.27). Denote by $\lambda_{\min}(C)$ the smallest eigenvalue of the symmetric matrix C. Then

$$\lambda_{\min}(A) \leqq \lambda_{\min}(B_1) - \|B_2\|_\infty \tag{2.1.17}$$

holds according to usual perturbation theory. Now we show the estimate

$$\|B_2\|_\infty \leqq 2(1-q)\frac{\alpha\varepsilon}{\delta}. \tag{2.1.18}$$

With $d = \min(d_{i,j-1}, d_{i-1,j})$ and by Lemma 2.4 we have

$$\begin{aligned} \|B_2\|_\infty &= \alpha\varepsilon\left(\left|\frac{1}{d_{i,j-1}} + \frac{1}{d_{i-1,j}} - 2\frac{q}{\delta}\right| + \beta\left|\frac{1}{d_{i,j-1}} + \frac{1}{d_{i-1,j}} - \frac{2}{\delta}\right|\right) \\ &\leqq \frac{\alpha\varepsilon}{\delta}\frac{2(1-\beta)\delta + (3\beta-1)d}{d}. \end{aligned}$$

Since $d \geqq \delta$ (cf. (2.1.27)), $(2(1-\beta)\delta + (3\beta-1)d)/d \leqq 1+\beta = 2(1-q)$ holds. Thus (2.1.18) is proved. So it is sufficient to show that

$$\lambda_{\min}(B_1) - 2(1-q)\frac{\alpha\varepsilon}{\delta} \geqq 0. \tag{2.1.19}$$

We prove (2.1.19) by constructing a diagonal matrix Φ with

$$B_1 - 2(1-q)\frac{\alpha\varepsilon}{\delta} I \geqq (\Lambda+\Phi)\Phi^{-1}(\Lambda+\Phi)^T \tag{2.1.20}$$

and $\Lambda := h^2 L$ with L from (2.1.3b). The left-hand side of (2.1.20) reads

$$B_1 - 2(1-q)\frac{\alpha\varepsilon}{\delta} I = \begin{bmatrix} q\frac{\alpha\varepsilon}{\delta} & -1 & \\ -\varepsilon & 2(1+\varepsilon) + \frac{2\alpha\varepsilon\beta}{\delta} - 2(1-q)\frac{\alpha\varepsilon}{\delta} & -\varepsilon \\ & -1 & q\frac{\alpha\varepsilon}{\delta} \end{bmatrix} \tag{2.1.21}$$

while the right-hand side is

$$(\Lambda+\Phi)\Phi^{-1}(\Lambda+\Phi)^T = \begin{bmatrix} \varepsilon/\varphi & -1 & \\ -\varepsilon & \varphi + (1+\varepsilon^2)/\varphi & -\varepsilon \\ & -1 & \varepsilon/\varphi \end{bmatrix} \tag{2.1.22}$$

with $\Phi = \operatorname{diag}\{\varphi, \cdots, \varphi\}$. Comparing both, we obtain

$$\varphi = \frac{2\delta}{(1-\beta)\alpha} \tag{2.1.23}$$

and

$$2(1+\varepsilon)+\frac{\alpha\varepsilon}{\delta}(\beta-1)\geqq\varphi+\frac{1+\varepsilon^2}{\varphi}. \tag{2.1.24}$$

Thus we chose

$$\alpha=\frac{2\delta}{(1-\beta)(1+\varepsilon)}, \tag{2.1.25}$$

the value of α still satisfying (2.1.24). As $\beta>0$ is a fixed parameter and $\delta=1+\varepsilon+\sqrt{2\varepsilon(1+\beta)}>1+\varepsilon$, $\alpha>2$ holds. □

Some properties of recursion (2.1.4), used in the above proof, are given in Lemma 2.4.

LEMMA 2.4. *Let $d_{i,j}$ be computed by* (2.1.4) *and $\beta\in[0,1]$. Then for $\varepsilon\geqq\beta$,*

$$d_{i,j}\leqq d_{i-1,j},\quad i>1,\quad j\geqq 1, \tag{2.1.26a}$$

$$d_{i,j}\leqq d_{i,j-1},\quad i\geqq 1,\quad j>1, \tag{2.1.26b}$$

hold and $d_{i,i}$ converges:

$$d_{i,i}\searrow\delta_1=1+\varepsilon+\sqrt{2\varepsilon(1+\beta)}; \tag{2.1.26c}$$

whereas, in general,

$$d\geqq d_{i,j}\geqq\delta \tag{2.1.27a}$$

with

$$d=\max\{d_{11},d_{12}\} \tag{2.1.27b}$$

and

$$\delta=\min\{\delta_1,\delta_2=1+\varepsilon+\sqrt{(2+\beta+\varepsilon)\varepsilon}\} \tag{2.1.27c}$$

is satisfied.

Proof. Formulae (2.1.26a, b) are readily proved by induction, using the monotonicity property of M-matrices. Thus these propositions apply to incomplete decompositions of a general M-matrix. By virtue of (2.1.26a, b) $d_{i,i}$ is monotonically decreasing. By the stability theorem of Meijerink and Van der Vorst [16], it is also bounded below by zero. Hence $d_{i,i}$ converges and the limit $\delta=1+\varepsilon+\sqrt{2\varepsilon(1+\beta)}$ can be computed from $\delta=2(1+\varepsilon)-(\varepsilon(\varepsilon-\beta)/\delta)-(1-\varepsilon\beta)/\delta$. Formula (2.1.27) follows from (2.1.26) for $\varepsilon\geqq\beta$ and for $\varepsilon<\beta$ from the inequalities $d_{12}>d_{i2}$ for $i>1$, $d_{i1}<d_{ik}<d_{i2}$, for $k>2$ and $d_{i,1}\geqq d_{i+1,1}$ for $i\geqq 1$, which is still valid. □

Remark 2.5. Lemma 2.3 carries over to a general domain, provided an equidistant rectangular grid is used, treating boundaries by linear interpolation. Then the corresponding stencils are of the form

$$\begin{bmatrix} & -1 & \\ \cdot & g_{1j} & -\varepsilon \\ & -1 & \end{bmatrix}$$

at the left-hand boundary. The other ones are given analogously. g_{ij} satisfies $g_{ij}\geqq 2(1+\varepsilon)$. Thus it is sufficient to replace the splitting of $h^2(K_l(\varepsilon)+\alpha N)=B_1+B_2$ in the proof of Lemma 2.3 by $h^2(K_l(\varepsilon)+\alpha N)=B_1+B_2+G$ with a diagonal matrix $G=\operatorname{diag}\{g_{ij}-2(1+\varepsilon)\}\geqq 0$ and B_1, B_2 from (2.1.15).

Combining Criterion 2.1, Lemma 2.3, and Remark 2.5 and regarding that (2.1.9) and (2.1.10) are satisfied for $\beta \geqq 1$, we obtain the following theorem.

THEOREM 2.6. *Let $K_l(\varepsilon)$ be the usual five-point discretization of the anisotropic problem from* (1.1.2) *on an arbitrary connected domain $\Omega \subset \mathbf{R}^2$. Then the* ILU_β *smoother, defined by* (2.1.1)-(2.1.5) *is $K_l(\varepsilon)$-robust, provided $\beta > 0$. More precisely, the smoothing property*

$$\|K_l(\varepsilon) S_l^\nu\| \leqq \varepsilon h^{-2} C_S \eta_0(\nu - 2, 1/(\alpha - 1)), \qquad \nu \geqq 2, \tag{2.1.28a}$$

with

$$C_S = \frac{2(1+\beta)}{\delta}, \tag{2.1.28b}$$

is satisfied, and

$$\|T_{2,1}(\nu, 0)\| \leqq C \eta_0(\nu - 2, 1/(\alpha - 1)), \qquad \nu \geqq 2, \tag{2.1.29}$$

with

$$\alpha = \begin{cases} \dfrac{2\delta}{(1-\beta)(1+\varepsilon)} & \text{for } 0 < \beta < 1, \\ \infty & \text{for } \beta \geqq 1 \end{cases} \tag{2.1.30}$$

holds independently of the stepsize h_l and the anisotropy constant ε.

Remark 2.7. By symmetry the above results hold as well in the case $\varepsilon \to \infty$. Thus ILU_β is robust without assumptions on the direction of the anisotropy, provided it is aligned with the gridlines.

2.2. The general case. ILU_β can easily be extended to a general problem.

Let

$$Ku = f \tag{2.2.1}$$

be a system of linear equations with a sparse $m \times m$-matrix K. Suppose that the usual incomplete decomposition using some prescribed sparse pattern exists for K (cf. [16], [1]). For example, there exist strictly lower- and upper-triangular matrices L and U and a diagonal matrix $D > 0$, so that

$$K = M - N \tag{2.2.2a}$$

with

$$M = (L + D) D^{-1} (U + D) \tag{2.2.2b}$$

and

$$N = \begin{pmatrix} n_{11} & \cdots & n_{1m} \\ \vdots & & \vdots \\ n_{m1} & \cdots & n_{mm} \end{pmatrix} \tag{2.2.2c}$$

is some rest matrix. Then for $\beta \in \mathbf{R}$ we define

$$D_\beta = \operatorname{diag}\left\{ d_i + \beta \sum_{j=1}^{m} |n_{ij}| \right\} \tag{2.2.3}$$

where d_i denote the elements of D. Replacing D in (2.2.2b) by D_β, we obtain ILU_β for (2.2.1):

$$K = M_\beta - N_\beta \tag{2.2.4a}$$

with

$$M_\beta = (L + D_\beta) D_\beta^{-1} (U + D_\beta). \tag{2.2.4b}$$

As mentioned above, this corresponds to some known modifications for certain values of β. All these modifications are designed to shift the spectrum of the rest-matrix N.

Remark 2.8. Let K be positive definite and the sparsity pattern on which the incomplete decomposition is based include the one of K. Then for $\beta = 0$ the rest matrix is indefinite; for $\beta = -1$, it is negative definite; while for $\beta \geqq 1$ it is positive (semi)-definite. In addition, for $\beta = -1$ M can again be regarded as discretization of some differential operator. This allows MILU ($=\text{ILU}_{-1}$) to improve the asymptotic condition number (cf. [5]). As smoother, however, it is not suited at all, as the eigenvalues of $M^{-1}N$ are unbounded. So we concentrate here on $\beta \geqq 0$.

Remark 2.9. Since $D_\beta \geqq D$ holds elementwise, the stability of ILU_β is ensured for M-matrices. Further, ILU_β is at least as stable as the usual ILU_0.

Proof. The proof follows immediately from [16, Thm. 2.3]. □

Remark 2.10. Let K be an M-matrix and $\beta \geqq 0$. Then ILU_β from (2.2.4) is a regular splitting of K (see [20]).

Proof. By virtue of Remark 2.9 the ILU_β decomposition is stable. According to [16, Thm. 2.3] and its proof, ILU_0 from (2.2.2) is a regular splitting of K. As $D_\beta \geqq D$ elementwise, $(U + D_\beta)^{-1} \geqq 0$, $(L + D_\beta)^{-1} \geqq 0$, $D_\beta > 0$ hold elementwise. So we obtain $M_\beta^{-1} \geqq 0$ and $N_\beta \geqq 0$, elementwise, completing the proof. □

Suppose $K_l(\varepsilon)$ is a discretization of a differential operator, depending on some parameter ε. Let $K_l(\varepsilon)$ be positive definite and symmetric. Further let $(2.1.7\varepsilon)$ be the corresponding approximation property. To compensate for the factor $1/\varepsilon$, we again need smoothing property $(2.1.8\varepsilon)$. Using Criterion 2.1, we obtain the following theorem.

THEOREM 2.11. *Let $K_1(\varepsilon)$ be positive definite and symmetric for all $\varepsilon \geqq 0$. Suppose that* ILU_1 *exists for $K_l(\varepsilon)$. Further, let* (2.1.10a) *hold in the ∞-norm for $N = N_0$.*

Then ILU_β *satisfies* $(2.1.8a\varepsilon)$ *for $\beta \geqq 1$ and $\varepsilon \geqq 0$, provided* (2.1.10b) *holds for* ILU_β.

Proof. First, we have $N_\beta = M_\beta - M + N = N + D_\beta - D + L(D^{-1} - D_\beta^{-1})U = N + D_\beta - D + L(D^{-1} - D_\beta^{-1})L^T \geqq 0$, since $U = L^T$ by symmetry of K_l and $D_\beta \geqq D$. Thus, (2.1.9) holds for N_β. Second, by $D^{-1} - D_\beta^{-1}$,

$$(D_\beta - (D_\beta - D))^{-1} - D_\beta^{-1} = D_\beta^{-1}((I - (D_\beta - D)D_\beta^{-1})^{-1} - I) = D_\beta^{-1} \sum_{\kappa=1}^{\infty} ((D_\beta - D)D_\beta^{-1})^\kappa$$

$$= (D_\beta - D)D_\beta^{-1} \sum_{\kappa=0}^{\infty} ((D_\beta - D)D_\beta^{-1})^\kappa D_\beta^{-1}$$

(since $(D_\beta - D)D_\beta^{-1} < I$), and we obtain

$$\|N_\beta\| \leqq \|N\|_\infty \left(1 + \beta\left(1 + \|L\|^2 \left\| D_\beta^{-1} \left(\sum_{\kappa=0}^{\infty} ((D_\beta - D)D_\beta^{-1})^\kappa D_\beta^{-1} \right\| \right)\right) \leqq C\|N\|_\infty,$$

proving (2.1.10a) for N_β. The proposition then follows from Criterion 2.1. □

Thus ILU_β is $K_l(\varepsilon)$-robust for $\beta \geqq 1$.

The above construction also covers factorizations of higher order such as seven-point and nine-point stencils (see [24]) and incomplete block decompositions (see [1], [13]). There the same modification can be applied and yields a robust method provided the original ILU decomposition satisfies (2.1.10a), which is the crucial assumption.

Now we turn to the practical aspects.

3. The value of local Fourier analysis to predict the practical behavior of a multigrid method. The first convergence proofs for multigrid methods were given by analyzing model problems with Fourier techniques (see [4], [6], [2]). Although not a rigorous proof for more complicated situations, local Fourier analysis has been established as a tool to predict the practical performance of a multigrid method (cf. [7], [3]). In many cases, the results are rather reliable and offer a nice possiblity to check the correctness of a program (cf. [3]). This is not so with ILU. The practical computations for model problem (1.1.3) using a five-point ILU smoother corresponding to ILU_0 from § 2 show the convergence factor κ_{10} to approach zero for $\varepsilon \to 0$, as can be seen from Fig. 1. κ_i is defined by

$$\kappa_i = \sqrt[i]{\frac{\|r_i\|}{\|r_0\|}} \tag{3.1}$$

where r_i denotes the residual after step i.

This does not work with a simple Gauss-Seidel method, as shown in Fig. 1.

The local Fourier analysis, however, yields as prediction for the smoothing rate

$$\rho_B(\varepsilon) = \frac{1}{1+2(\varepsilon+\sqrt{2\varepsilon})}, \tag{3.2}$$

tending to 1, $\varepsilon \to 0$ (cf. [13], [19]). Up to now, this difference was thought to arise from boundary effects and/or rounding errors (cf. [19], [17]). Interested in the reason for this discrepancy and also in the reason for the peak in Fig. 1, we computed the convergence rates for the same multigrid method as in Fig. 1, gradually refining the stepsize of the finest grid. Using $h_f = 1/2^i$, $i = 5, \cdots, 10$, as finest stepsize we obtained the results shown in Fig. 2.

Obviously, the peak depends on ε and h. So we cannot expect a realistic prediction of the convergence rate by usual local Fourier analysis, since $h \to 0$ is incorporated

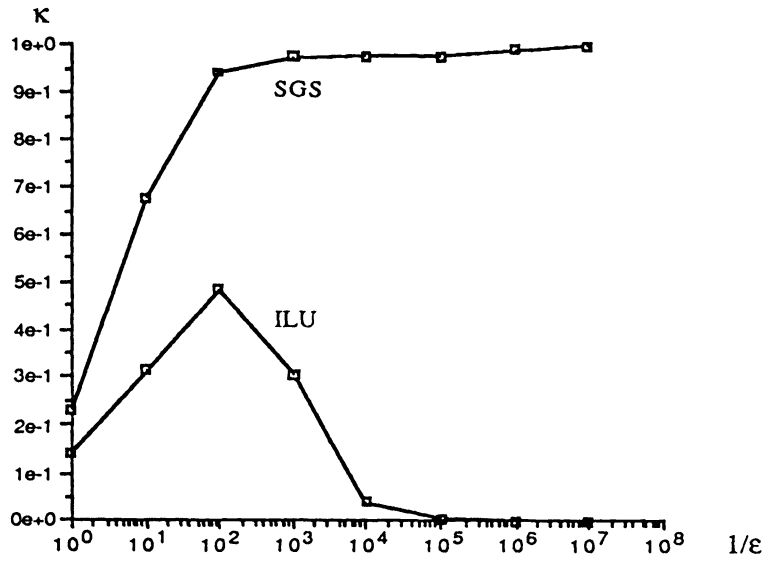


FIG. 1. *Comparison of the convergence rate* κ_{10} *of a standard multigrid method for* (1.1.3) *using a five-point* ILU *smoother* (ILU_0) *and a symmetric Gauss-Seidel smoother* (SGS) (*W-cycle*) *with* $h = 1/64$ *as finest stepsize.*

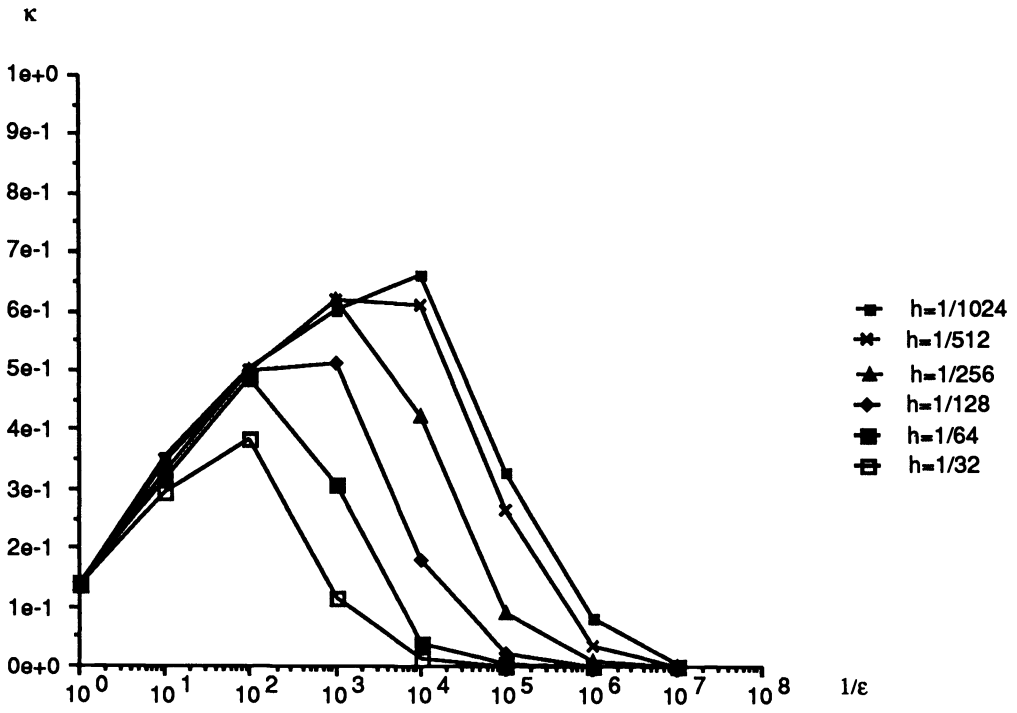


FIG. 2. *Convergence rate for the multigrid method from Fig.* 1 (ILU_0) *with varying finest stepsize.*

there (cf [2]). This limit, however, will never be reached on a real computer, first since computing time and storage are limited and second the condition number of $K_l(\varepsilon)$ is proportional to $1/h^2$, making computations with too small equidistant stepsizes and a finite number of digits senseless. It is mainly used since multigrid methods, involving rather fine grids, quickly approach the limit rates for usual problems (cf. [7]). Model problem (1.1.3), however, consists of a set of problems controlled by the parameter ε, which is ranging in a much larger interval than h. Hence it is reasonable to assume

$$h \geqq h_0 > 0 \tag{3.3}$$

for practical purposes. Usually, $h_0 \approx 10^{-3}$. Looking for a reasonable prediction of the practical behavior of our method, we simply include the finite stepsize h into the local Fourier analysis.

To predict the smoothing rate, we have to compute

$$\rho(\varepsilon, h_l) = \max_{\lambda \in \Lambda_{\text{high}}(M_l^{-1}(\varepsilon)K_l(\varepsilon))} |1 - \lambda(M_l^{-1}(\varepsilon)K_l(\varepsilon)| \tag{3.4}$$

with Λ_{high} being the set of eigenvalues corresponding to the high frequency eigenfunctions (cf. [7], [11], [18]) and $M_l(\varepsilon) = M$ from (2.1.2). Computing these eigenvalues by local Fourier analysis results in

$$\rho(\varepsilon, h_l) = \max_{\mu,\nu \in J} \frac{\varepsilon}{\delta} \frac{|\cos(\mu - \nu)h_l\pi|}{|1 + \varepsilon - \varepsilon \cos \mu h_l\pi - \cos \nu h_l\pi + (\varepsilon/\delta)\cos(\mu - \nu)h_l\pi|} \tag{3.5}$$

with $J = [1, n_l = 1/h_l - 1]^2 \backslash [1, n_{l-1} = 1/h_{l-1} - 1]^2 \cap \mathbf{N}^2$ and $\delta = 1 + \varepsilon + \sqrt{2\varepsilon}$.

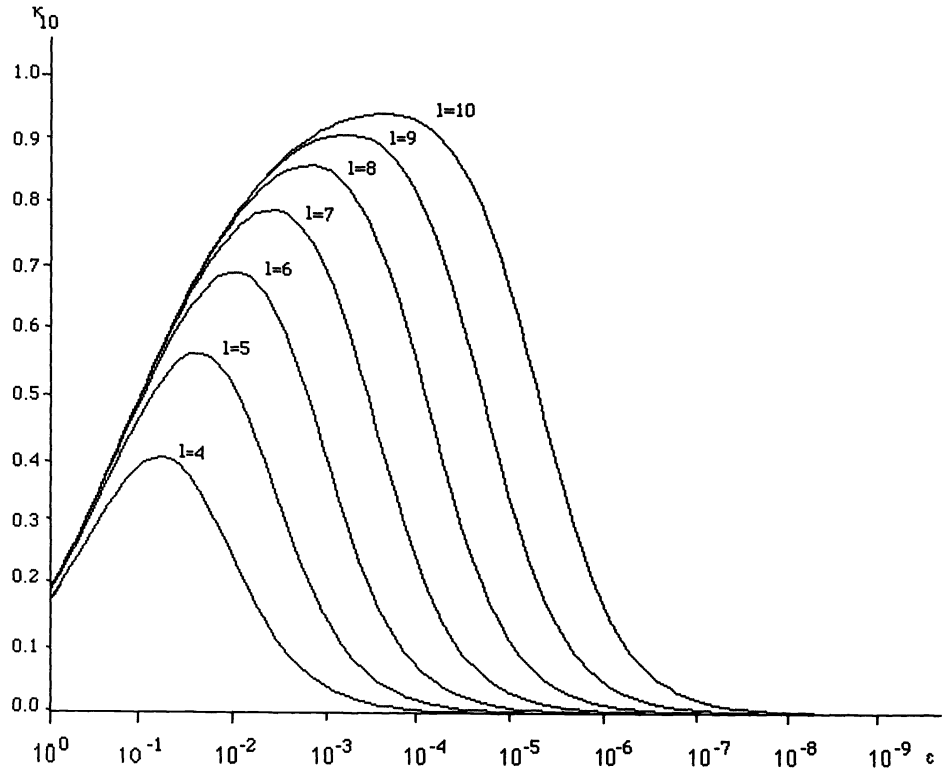


FIG. 3. *Estimate* (3.7) *for* $\rho(\varepsilon, h_1)$ *for* $h_1 = 1/2^1$.

The maximum of the right-hand side of (3.5) is attained at $\mu = (1-h_l)/h_l$, $\nu = 1$. Expanding the cosine terms, we obtain

$$\rho(\varepsilon, h_l) = \frac{\varepsilon}{\delta} \frac{1-2h_l^2\pi^2}{\varepsilon\left(2-\frac{1}{\delta}\right)+\left(1+\varepsilon+\frac{4\varepsilon}{\delta}\right)\frac{h_l^2\pi^2}{2}+O(h_l^4)} + O(h_l^4). \tag{3.6}$$

For small h we neglect terms of order greater than or equal to four and obtain

$$\rho(\varepsilon, h_l) \leqq 2\varepsilon \frac{1-2h_l^2\pi^2}{2\varepsilon(2\delta-1)+((1-\varepsilon)\delta+4\varepsilon)h_l^2\pi^2} \tag{3.7}$$

for small h and ε. Using (3.7) we readily predict the actual behavior of our method, as can be seen by comparing Figs. 2 and 3.

Thus we conclude with the following remark.

Remark 3.1. Suppose (3.3) is true. Then the usual five-point ILU scheme (ILU_0) is robust for $K_l(\varepsilon)$. This confirms rather well the early assertion by Wesseling [21] and Kettler [13] about the robustness of incomplete LU smoothing.

Admitting $h \to 0$, however, we are in the same situation as in § 2, where we had to require $\beta > 0$ in order to obtain robustness.

4. The optimal choice of β.

4.1. Optimizing the smoothing property estimate. For practical purposes we are interested in the dependence of the convergence factor of a multigrid method with

ILU_β smoothing on β. To investigate the influence of β in (2.1.28), we define

$$\chi(\beta)=\frac{2(1+\beta)}{\delta}\,\eta_0\left(\nu-2,\frac{1}{\alpha(\beta)-1}\right). \tag{4.1.1}$$

First we compute

$$\eta_0 := \min_{\beta\in(0,1]}\eta_0\left(\nu-2,\frac{1}{\alpha(\beta)-1}\right)=\min_{\beta\in(0,1]}\max\left\{\frac{(\nu-2)^{\nu-2}}{(\nu-1)^{\nu-1}},\frac{\alpha(\beta)}{(\alpha(\beta)-1)^{\nu-1}}\right\}. \tag{4.1.2}$$

As $(\nu-2)^{(\nu-2)}/(\nu-1)^{(\nu-1)}$ does not depend on β, while $\alpha/(\alpha-1)^{(\nu-1)}$ is monotonously decreasing with $\beta\in[0,1]$, we look for β_2:

$$\frac{\alpha(\beta_2)}{(\alpha(\beta_2)-1)^{\nu-1}}=\frac{(\nu-2)^{\nu-2}}{(\nu-1)^{\nu-1}}. \tag{4.1.3}$$

Let $\xi=\xi(\nu)$ denote a solution of

$$\xi=\frac{(\nu-2)^{\nu-2}}{(\nu-1)^{\nu-1}}(\xi-1)^{\nu-1} \tag{4.1.3'}$$

for fixed $\nu\in\mathbf{N}$. For $3\leqq\nu\leqq11$ the maximum values of ξ are given in Table 1.

Equating $\alpha(\beta_2)=\xi(\nu)$ with $\alpha(\beta)$ from (2.1.30), we obtain for $\beta\leqq1$

$$\frac{2\delta}{(1-\beta_2)(1+\varepsilon)}=\xi. \tag{4.1.4}$$

Using (2.1.27) and performing some elementary calculations, we end up with

$$\beta_2=\frac{\xi(\xi-2)(1+\varepsilon)^2-4\varepsilon-4\sqrt{\varepsilon(\varepsilon+\xi(1+\varepsilon)^2)}}{\xi^2(1+\varepsilon)^2}. \tag{4.1.5}$$

β_2 satisfies $\beta_2\leqq1$, and $\beta_2\geqq0$ holds, provided

$$\xi\geqq2+\frac{\sqrt{8\varepsilon}}{1+\varepsilon}. \tag{4.1.6}$$

For $3\leqq\nu\leqq11$ the values of ε for which (4.1.6) is valid are given in Table 2.

TABLE 1
$\xi(\nu)$ from (4.1.3') for $3\leqq\nu\leqq11$.

ν	3	4	5	6	7	8	9	10	11
$\xi(\nu)$	5.828	4.0	3.379	3.063	2.879	2.740	2.646	2.574	2.518

TABLE 2
ε_0, so that (4.1.6) holds for ε: $0\leqq\varepsilon\leqq\varepsilon_0$.

ν	3	4	5	6	7	8	9	10	11
ε_0	1	1	0.638	0.205	0.121	0.079	0.058	0.045	0.036

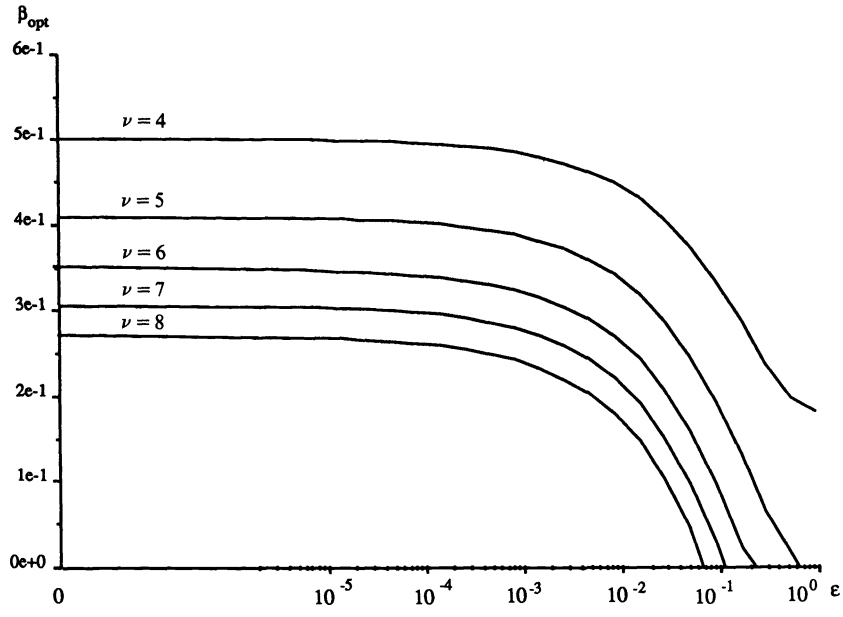


FIG. 4. β_{opt} *from* (4.1.7) *for* $\nu = 4, \ldots, 8$.

Hence

$$\beta_{opt} = \begin{cases} \beta_2 & \text{for } \xi \geqq 2 + \dfrac{\sqrt{8\varepsilon}}{1+\varepsilon}, \\ 0 & \text{else,} \end{cases} \tag{4.1.7}$$

and

$$\eta_0(\nu) = \begin{cases} \dfrac{\alpha(\beta)}{(\alpha(\beta)-1)^{\nu-1}} & \text{for } 0 \leqq \beta < \beta_{opt} \\ \dfrac{(\nu-2)^{\nu-2}}{(\nu-1)^{\nu-1}} & \text{for } \beta_{opt} \leqq \beta \leqq 1 \end{cases} \quad \text{for } \xi \geqq 2 + \frac{\sqrt{8\varepsilon}}{1+\varepsilon}, \tag{4.1.8}$$

$$\frac{(\nu-2)^{\nu-2}}{(\nu-1)^{\nu-1}} \qquad \text{else.}$$

$\chi(\beta)$ will attain its minimum in β_{opt}, since $\alpha(1+\beta)/((\alpha-1)^{\nu-1}\delta)$ decreases monotonously, whereas $(1+\beta)/\delta$ increases with β. So for $\beta = \beta_{opt}$ estimate (2.1.29) will be optimal and we may expect this choice to yield a reasonable prediction of the practically best parameter.

Figure 4 shows β_{opt} from (4.1.7) for $\nu = 4, \cdots, 8$.

4.2. Numerical results. To check the validity of our theoretical considerations for § 4.1 we implemented ILU_β for problem (1.1.3) and tested the convergence rate for several parameters h, ε, and β. The results are given in Figs. 5-7.

The curves in Figs. 5-7 show the behavior predicted in § 4.1. So $\kappa(\beta)$ attains a well-defined minimum in $(0, 1)$ depending on ε.

Figure 8 shows the dependence of $\kappa(\beta)$ on ν_{pr}, the number of smoothing steps actually used in practical computations, for $\varepsilon = 10^{-3}$. As predicted by (4.1.7), β_{opt} tends to 0 with increasing ν_{pr}. Comparing $\beta_{opt}(\varepsilon, \nu_{th})$ from (4.1.7) with the optimal value of

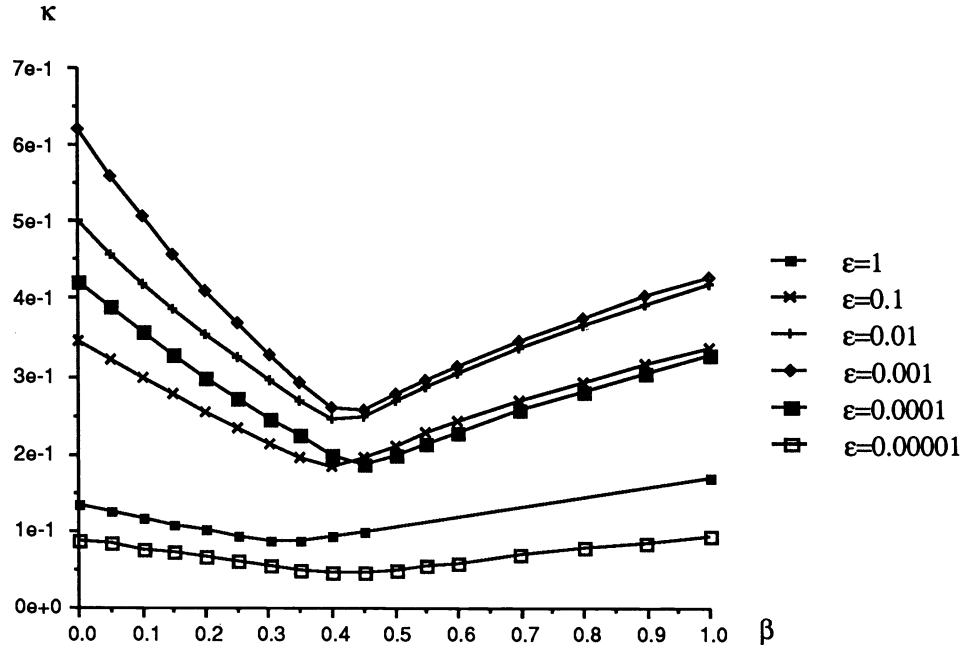


FIG. 5. *Dependence of the convergence rate* κ_{10} *of a multigrid method (W-cycle) with* ILU_β *smoothing on* β *with finest stepsize* $h = 1/256$ *and* $\nu_{\mathrm{pr}} = 1$.

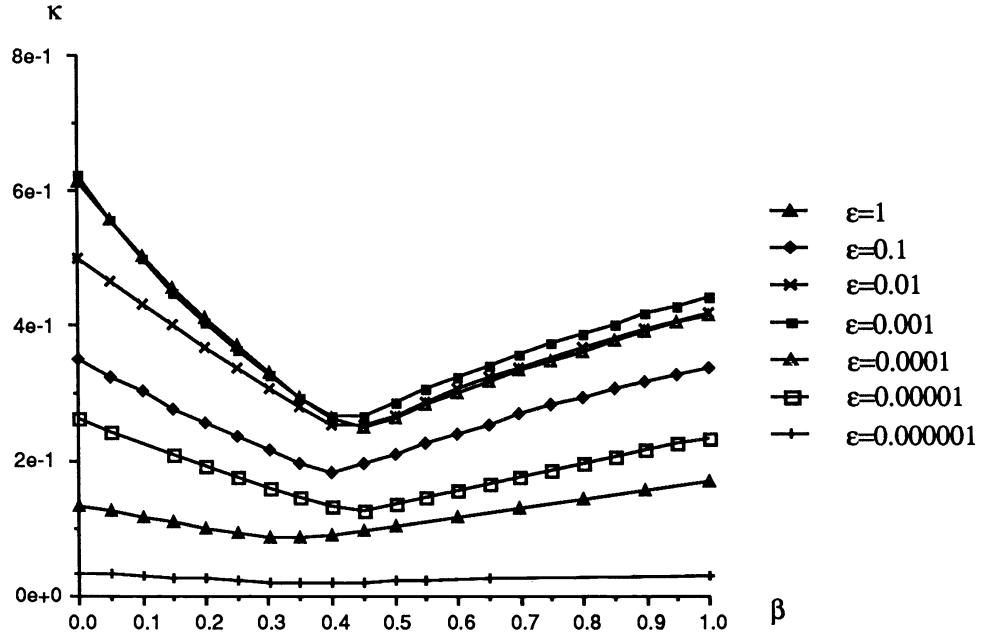


FIG. 6. *Dependence of the convergence rate* κ_{10} *of a multigrid method (W-cycle) with* ILU_β *smoothing on* β *with finest stepsize* $h = 1/512$ *and* $\nu_{\mathrm{pr}} = 1$.

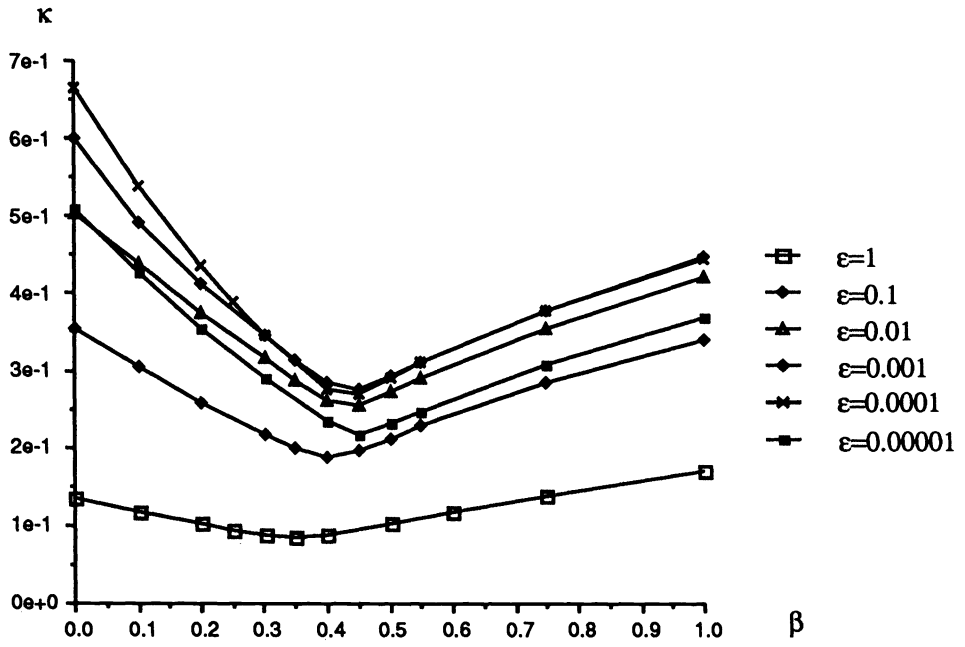


FIG. 7. *Dependence of the convergence rate* κ_{10} *of a multigrid method (W-cycle) with* ILU_β *smoothing on* β *with finest stepsize* $h = 1/1024$ *and* $\nu_{pr} = 1$.

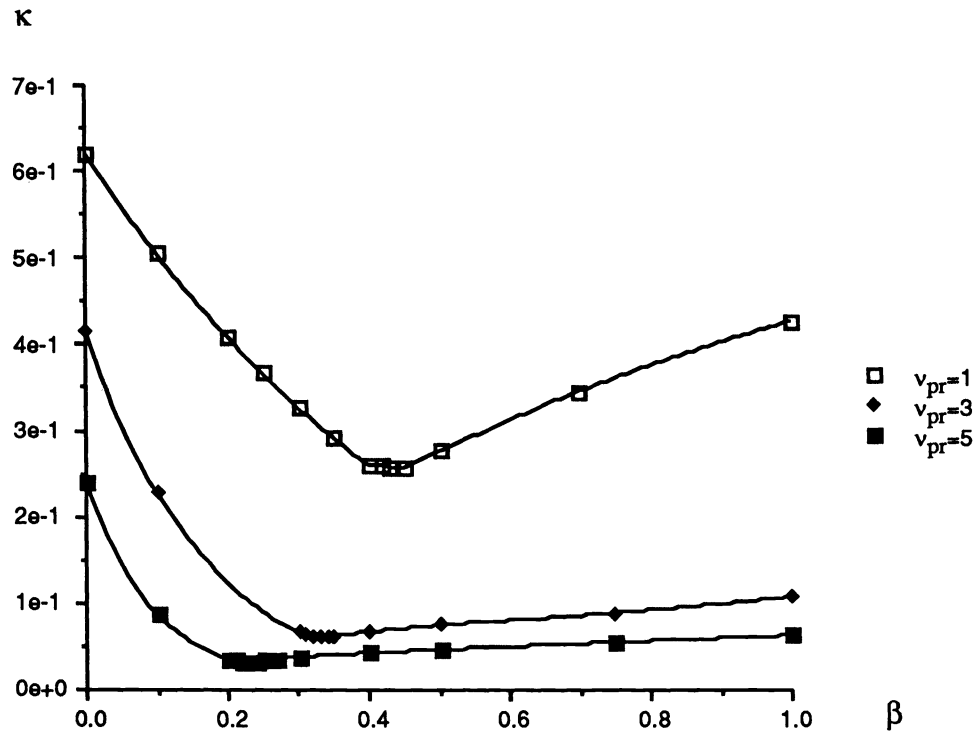


FIG. 8. $\kappa_{10}(\beta)$ *for* $h = 1/512$, $\varepsilon = 10^{-3}$, *W-cycle, and variable* ν_{pr}.

β from Figs. 5-8, however, we have to be careful about the relation of ν_{pr} and $\nu_{th} = \nu$ from the theoretical analysis. It is well known that splitting (2.1.6) of $T_{2,1}(\nu_{th}, 0)$ is primarily constructed to allow a short and well-structured qualitative convergence proof for sufficiently large ν_{th} and tends to give an overestimate for ν_{pr}, the number of smoothing steps required in practical computations, which can be avoided only by constructing special problem-dependent norms (cf. [7]). In addition, we had to pay for estimate (2.1.28) by taking $\eta(\nu_{th}-2)$ instead of $\eta(\nu_{th})$. Hence, we cannot expect ν_{th} in our theoretical considerations to be identical with ν_{pr}, the number of smoothing steps used in practice. A comparison of the practical results as given in Figs. 5-8 with the values of β_{opt} from (4.1.7), shows that $\nu_{th} = 4$ in (4.1.7) corresponds to the results for one smoothing step, $\nu_{pr} = 1$, and a W-cycle. Fixing $\nu_{pr} = \nu_{th} - 3$, our predictions using (4.1.7) for different values of ε and ν turn out to be good approximations to the optimal values of β, as shown in Figs. 9 and 10.

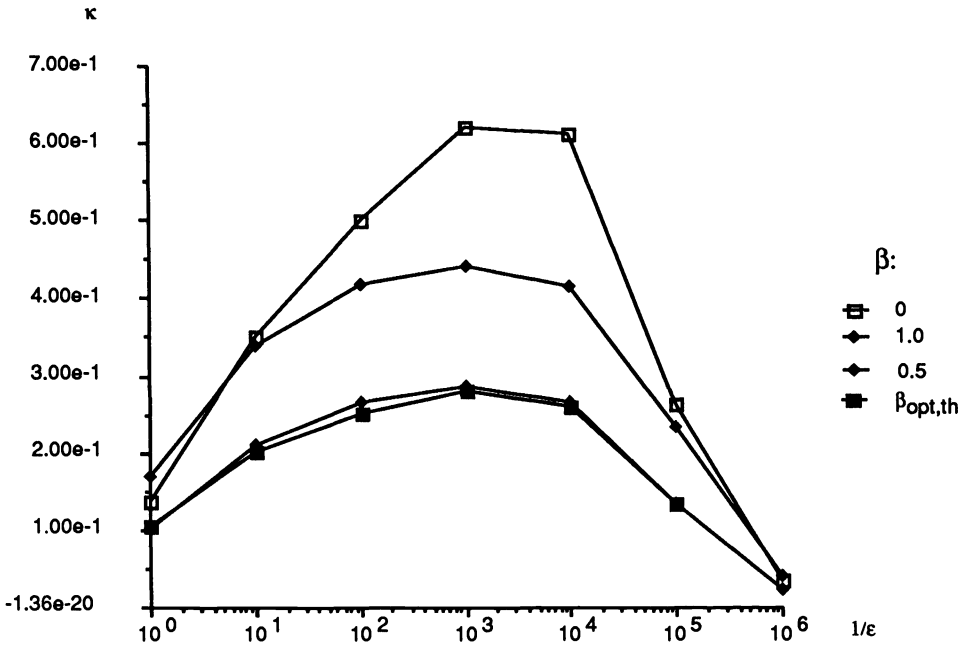


FIG. 9. $\kappa_{10}(\varepsilon)$ *for different values of* β, $h = 1/512$, *W-cycle, and* $\nu_{pr} = 1$. β_{opt} *from* (4.1.7).

From Figs. 9 and 10 we see that β_{opt} from (4.1.7) yields a reasonable prediction for the practically best parameter $\beta_{opt,em}$, as the convergence rates obtained for the empirically determined $\beta_{opt,em}$ differ only approximately 0.02 from the ones obtained with $\beta = \beta_{opt}$. For higher ν, $\beta_{opt,em}$ is not so easy to determine, as the convergence rates are extremely fast and differ very little in a neighborhood of β_{opt}.

Of course, β can also be optimized by local Fourier techniques, which has actually been done for $\varepsilon = 0$ by Khalil [14] and Oertel [17], and for $\varepsilon = 1$ by Hemker [10]. These techniques, however, suffer from certain drawbacks, since they do not include effects arising from the finite dimension of the problem and from the boundary, which are rather important for ILU factorizations, as pointed out in § 3. In addition, it seems impossible to obtain an analytical formulation for β_{opt} depending on ε and ν by local Fourier analysis (cf. [15]). So Oertel [17] and Khalil [14] recommend by applying in

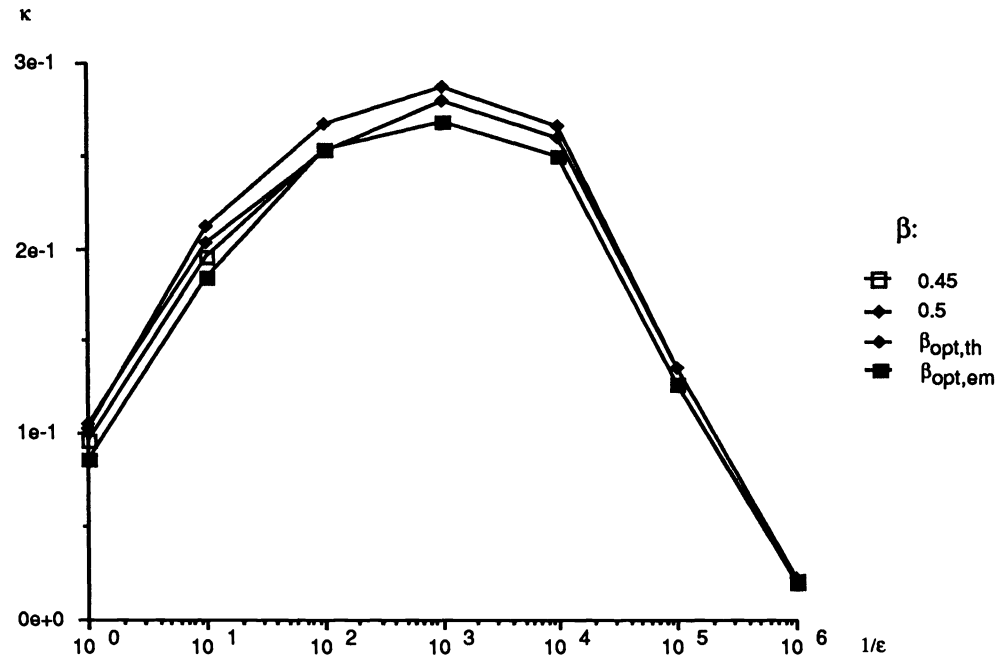


FIG. 10. *Comparison of convergence factors* κ_{10} *for different values of* β *for* $\nu_{pr}=1$, *W-cycle, and* $h=1/512$.

general $\beta=0.5$. This is the correct choice for small ε and $\nu_{pr}=1$. There the practice, local Fourier analysis, and (4.1.7) agree. For $\varepsilon=1$, $\nu_{pr}=1$, and $h=1/512$ we get $\beta_{opt,em}\approx 0.35$ from Fig. 6, while $\beta_{opt}=0.18$ from (4.1.7), and the result of Hemker [10] obtained with local Fourier analysis corresponds to $\beta_{opt,F}=0.14$. However, this is no problem, since the corresponding convergence rates do not differ much, as can be seen from Figs. 5-7. The same holds for the dependence on ν. Hence it is reasonable to use $\beta=0.5$ or, better, $\beta=0.45$ for all ε and ν.

We tested ILU_β also in connection with our transforming approach for the Navier-Stokes equations (cf. [25]). There the results are very similar.

Hence we conclude stating that ILU_β smoothing is robust and should be applied with $\beta=\beta_{opt}(\varepsilon, \nu_{th})$ from (4.1.7) or $\beta=0.45$.

Acknowledgments. The author wishes to thank Professor W. Hackbusch, his colleague Th. Dreyer, and the referees for some valuable advice.

REFERENCES

[1] O. AXELSSON AND V. A. BARKER, *Finite Element Solutions of Boundary Value Problems. Theory and Computation*, Academic Press, New York, 1985.

[2] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333-390.

[3] ———, *Guide to multigrid development*, in Multi-Grid Methods, Proceedings, Lecture Notes in Math. 960, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, 1982.

[4] R. P. FEDORENKO, *A relaxation method for solving elliptic difference equations*, USSR Comput. Math. Math. Phys., 1 (1961), pp. 1092-1096.

[5] I. GUSTAFSSON, *A class of 1st order factorization methods*, Report 77.04, Department of Computer Sciences, Chalmers University, Göteborg, 1977.

[6] W. HACKBUSCH, *Ein iteratives Verfahren zur schnellen Auflösung elliptischer Randwertprobleme*, Report 76-12, Universität Köln, 1976.

[7] W. HACKBUSCH, *Multi-grid Methods and Applications*, Springer-Verlag, Berlin, Heidelberg, 1985.
[8] ———, *Theorie und Numerik elliptischer Differentialgleichungen*, Teubner, Stuttgart, 1986.
[9] ———, *A new approach to robust multi-grid solvers*, Institut für Informatik und Praktische Mathematik, CAU, Kiel, 1987, preprint.
[10] P. W. HEMKER, *The incomplete LU-decomposition as a relaxation method in multigrid algorithms*, in Boundary and Interior Layers—Computational and Asymptotic Methods, Proceedings, J. J. H. Miller, ed., Boole Press, Dublin, 1980.
[11] ———, *Fourier analysis of grid functions, prolongations and restrictions*, NW 93/80, Mathematisch Centrum, Amsterdam, 1980.
[12] A. JENNINGS AND G. M. MALIK, *Partial elimination*, J. Inst. Math. Applic., 20 (1977), pp. 307–316.
[13] R. KETTLER, *Analysis and comparison of relaxation schemes in robust multi-grid and preconditioned conjugate gradient methods*, in Multi-grid Methods, Proceedings, Lecture Notes in Math. 960, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, 1982.
[14] M. KHALIL, *On some modifications of ILU-smoothers*, Lecture given at the Fourth GAMM Seminar, Kiel, January 22–24, 1988.
[15] F. LIEBAU, *Unvollständige LU-Zerlegungen mit angenäherten Koeffizienten als Glätter in Mehrgitterverfahren*, Diplomarbeit, Institut für Informatik und Praktische Mathematik, Kiel, 1987.
[16] 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*, Math. Comp., 31 (1977), pp. 148–162.
[17] K.-D. OERTEL, *ILU-smoothing: Systematic tests and improvements*, Lecture given at the Fourth GAMM Seminar, Kiel, January 22–24, 1988.
[18] K. STÜBEN AND U. TROTTENBERG, *Multigrid methods: Fundamental algorithms, model problem analysis and applications*, Multi-grid Methods, Proceedings, Lecture Notes in Math. 960, W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin, 1982.
[19] C. A. THOLE, *Beiträge zur Fourieranalyse von Mehrigitterverfahren*, Diplomarbeit, Universität Bonn, 1983.
[20] R. S. VARGA, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.
[21] P. WESSELING, *A robust and efficient multigrid method*, in Multi-grid Methods, Proceedings, Lecture Notes in Math. 960, in W. Hackbusch and U. Trottenberg, eds., Springer-Verlag, Berlin 1982.
[22] ———, *Theoretical and practical aspects of a multigrid method*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 387–407.
[23] P. WESSELING AND P. SONNEVELD, *Numerical experiments with a multiple grid and a preconditioned Lanczos type method*, in Approximation Methods for Navier-Stokes Problems, Lecture Notes in Math. 771, R. Rautmann, ed., Springer-Verlag, Berlin, 1980, pp. 543–562.
[24] G. WITTUM, *Distributive Iterationen für indefinite Systeme*, Thesis, Universität Kiel, 1986.
[25] ———, *Multi-grid methods for Stokes and Navier-Stokes equations. Transforming smoothers—Algorithms and numerical results*, Numer. Math., to appear.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 718-741, July 1989
© 1989 Society for Industrial and Applied Mathematics
007

THREE-DIMENSIONAL TRIANGULATIONS FROM LOCAL TRANSFORMATIONS*

BARRY JOE†

Abstract. A new algorithm is presented that uses a local transformation procedure to construct a triangulation of a set of n three-dimensional points that is pseudo-locally optimal with respect to the sphere criterion. It is conjectured that this algorithm always constructs a Delaunay triangulation, and this conjecture is supported with experimental results. The empirical time complexity of this algorithm is $O(n^{4/3})$ for sets of random points, which compares well with existing algorithms for constructing a three-dimensional Delaunay triangulation. Also presented is a modification of this algorithm for the case that local optimality is based on the max-min solid angle criterion.

Key words. three-dimensional triangulation, Delaunay triangulation, max-min solid angle criterion, computational geometry, mesh generation

AMS(MOS) subject classifications. 65N50, 68Q20, 68U05

1. Introduction. The three-dimensional triangulation problem is as follows. Given n three-dimensional points, connect them into nonoverlapping tetrahedrons that fill the convex hull of the points. There are many ways to triangulate the n points. Special triangulations include the Delaunay triangulation and the triangulation satisfying the max-min solid angle criterion. Algorithms for constructing a Delaunay triangulation in k-dimensional space for $k \geqq 2$ are given by Bowyer [2], Watson [14], and Avis and Bhattacharya [1]. For the three-dimensional case, the estimated time complexity is $O(n^{4/3})$ for Bowyer's algorithm and higher for the other two algorithms. Applications of three-dimensional triangulations include finite-element mesh generation (Nguyen [11], Cavendish, Field, and Frey [3]), where it is usually desired to avoid small angles in triangulations, and interpolation and contouring (Petersen, Piper, and Worsey [12]).

In this paper, we investigate a local transformation procedure for three-dimensional triangulations that is analogous to the procedure of Lawson [8] for two-dimensional triangulations, and use this local transformation procedure in a new algorithm for constructing three-dimensional pseudo-locally optimal triangulations, where local optimality is based on either the sphere criterion (satisfied by Delaunay triangulations) or the max-min solid angle criterion. We conjecture that our algorithm always constructs a Delaunay triangulation in the case of the sphere criterion.

In § 2, preliminary definitions and results are given. In § 3, the main theoretical results are presented for the sphere criterion. In § 4, an algorithm and data structure are given for constructing a pseudo-locally optimal triangulation with respect to the sphere criterion, assuming that no four vertices are co-planar. In § 5, this assumption is removed and the algorithm is extended. In § 6, optimal triangulations with respect to the max-min solid angle criterion and their computation are discussed. In § 7, experimental results are presented for a Fortran implementation of the algorithms. In § 8, concluding remarks and open problems are given.

2. Preliminaries. Let S be a set of $n \geqq 4$ three-dimensional points (or vertices) that are not all co-planar. A triangulation of S is valid if and only if (a) the four

* Received by the editors March 21, 1988; accepted for publication (in revised form) September 16, 1988. This work was partially supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

† Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1.

vertices of any tetrahedron are not co-planar; (b) any tetrahedron *abcd* contains no points of $S-\{a, b, c, d\}$; (c) the intersection of the interior of any two tetrahedrons is empty; and (d) a triangular face is either on the boundary of the convex hull of S (and occurs in exactly one tetrahedron), or it is common to exactly two tetrahedrons. For any valid triangulation of S, let V_b and V_i be the number of boundary and interior vertices, respectively; let E_b and E_i be the number of boundary and interior edges, respectively; let F_b and F_i be the number of boundary and interior faces (triangles), respectively; and let T be the number of tetrahedrons. For any triangulation of S, V_b, V_i, E_b, and F_b are the same. V_b is the number of vertices on the boundary of the convex hull of S and $V_i = n - V_b$. E_b and F_b are constant because all two-dimensional triangulations of the same vertices have the same number of edges and triangles.

However, different triangulations of S may have different values for E_i, F_i, and T. The above quantities satisfy the following relations (Fuhring [5]):

(1) (a) $T = (F_b + 2F_i)/4$,
(b) $F_b = 2V_b - 4$,
(c) $F_i = V_b + 2(E_i - V_i) - 4$,
(d) $E_b = 3F_b/2$.

Substituting (b) and (c) into (a) results in

$$T = V_b + E_i - V_i - 3. \tag{2}$$

Clearly, E_i may be at most $n(n-1)/2 = O(n^2)$. From (1 (c)) and (2), it can be seen that F_i and T may be at most $O(n^2)$ as well. Note that if E_i is increased by one, then F_i is increased by two and T is increased by one. It is not too difficult to construct a family of triangulations for which E_i, F_i, and T are all proportional to n^2 (see § 7).

For two-dimensional triangulations, the local transformation procedure is as follows. If two adjacent triangles of the triangulation form a strictly convex quadrilateral, then swap the common edge for the other diagonal edge of the quadrilateral to form two new triangles. Lawson [8] proves that given any two triangulations T_1 and T_2 of a set of two-dimensional points, there exists a finite sequence of local transformations (edge swaps) by which T_1 can be transformed to T_2. Lawson [9] uses this local transformation procedure in an algorithm for constructing a two-dimensional Delaunay triangulation of n points in an estimated average time of $O(n^{4/3})$.

For three-dimensional triangulations, the analogous local transformation procedure is based on the observation that a strictly convex hexahedron formed from five vertices can be triangulated in two ways, the first containing two tetrahedrons and the second containing three tetrahedrons. This is illustrated in Fig. 1, where the five vertices are *a*, *b*, *c*, *d*, and *e*; (i) contains the two tetrahedrons *abcd* and *abce*, and (ii) contains the three tetrahedrons *abde*, *acde*, and *bcde*. Note that (i) contains interior face *abc* and no interior edges while (ii) contains three interior faces *ade*, *bde*, *cde*, and interior edge *de*. The *local transformation procedure* is that if two (three) adjacent tetrahedrons of the triangulation form a strictly convex hexahedron as in Fig. 1, then replace the tetrahedrons by the other possible triangulation of the hexahedron containing three (two) tetrahedrons. This local transformation procedure can be considered to be a face "swap," where one interior face is "swapped" for three interior faces or vice versa. In the next two sections, we describe how this local transformation procedure can be used to construct a (nearly) Delaunay triangulation.

Two special three-dimensional triangulations of n vertices are the Delaunay triangulation and the triangulation satisfying the max-min solid angle criterion. A Delaunay triangulation satisfies the *sphere criterion*: the circumsphere of the four

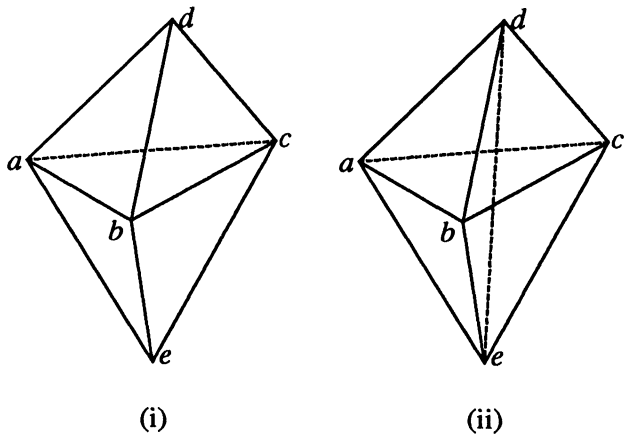


FIG. 1. *Two possible triangulations of strictly convex hexahedron.* (i) *Two tetrahedrons abcd and abce.* (ii) *Three tetrahedrons abde, acde, and bcde.*

vertices of any tetrahedron of the triangulation contains no vertices in its interior. A Delaunay triangulation is unique if no five vertices are co-spherical. The Delaunay triangulation is also the dual of the Voronoi tessellation (Bowyer [2], Watson [14]). The Voronoi tessellation of n vertices is a collection of n convex regions such that each region contains the points closer to one vertex than all the other vertices.

A tetrahedron contains twelve planar angles (three in each of the four triangular faces), six dihedral angles (one at each of the six edges), and four solid or trihedral angles at the vertices. The planar and dihedral angles are straightforward to compute. The definition and computation of a solid angle, e.g., at vertex d of tetrahedron *abcd*, are as follows. The solid angle at d is the surface area on the unit sphere formed by projecting each point on face *abc* to the surface of the unit sphere with d at its centre. In general, a solid angle can be defined as a double integral. In the special case of a tetrahedron, the solid angle (or spherical excess) at d can be computed as $\alpha+\beta+\gamma-\pi$ (Gasson [6]), where α, β, and γ are the dihedral angles at edges *ad*, *bd*, and *cd*, respectively (α, β, and γ are also the spherical angles at the projection of a, b, and c, respectively, on the unit sphere).

A triangulation satisfies the *max-min solid angle criterion* if over all possible triangulations of the vertices, the minimum of the solid angles at all vertices of all tetrahedrons is maximized. For two-dimensional triangulations, the circle and max-min angle criteria are identical, i.e., a Delaunay triangulation satisfies the max-min angle criterion and vice versa (Lawson [9]). Field [4] recently conjectured that the sphere criterion and max-min solid angle criterion are identical for three-dimensional triangulations. However, the following simple example shows that this conjecture is false. Let vertices a, b, c, d, and e have the (x, y, z) coordinates $(0, 0, 0)$, $(2, 0, 0)$, $(2, 2, 0)$, $(1.5, 0.5, 2)$, and $(1.5, 0.5, -0.5)$, respectively. There are two ways to triangulate these five vertices as illustrated in Fig. 1. It is straightforward to verify by calculation that triangulation (i), containing two tetrahedrons, satisfies the max-min solid angle criterion but is not Delaunay, and triangulation (ii), containing three tetrahedrons, is Delaunay but does not satisfy the max-min solid angle criterion. In § 6, we discuss the max-min solid angle criterion further.

3. Theoretical results. In this section, we present some theoretical results for three-dimensional triangulations and the sphere criterion. Some of these results are

three-dimensional versions of those in Lawson [9]. We start with definitions and results concerning the local optimality of interior faces in three-dimensional triangulations. In particular, we show that if every interior face of a triangulation is locally optimal, then it is a Delaunay triangulation. Then we discuss how the local transformation procedure given in the previous section can be used to improve an arbitrary three-dimensional triangulation to a nearly Delaunay triangulation, called a pseudo-locally optimal triangulation. We give results on when this improvement process does not terminate in a Delaunay triangulation, due to the nontransformability of some nonlocally optimal faces (unlike the two-dimensional case). Finally, we give an example of a pseudo-locally optimal triangulation that is not a Delaunay triangulation.

For simplicity, we assume for now that no four vertices are co-planar among the n vertices to be triangulated. (This assumption will be removed in § 5.) This means that the triangulation of five vertices can be three different configurations. The first two configurations are illustrated in Fig. 1 (the boundary of the convex hull contains five vertices). The third configuration occurs when the boundary of the convex hull contains four vertices; in this case, the triangulation of the five vertices consists of four tetrahedrons. For example, if vertex e is not on the boundary of the convex hull of vertices a, b, c, d, and e, the four tetrahedrons are *abce*, *abde*, *acde*, and *bcde*; there are four interior edges and six interior faces.

DEFINITION 1. Let *abcd* and *abce* be two tetrahedrons sharing common face *abc* with d and e on opposite sides of *abc*. Then interior face *abc* is said to be *locally optimal* (with respect to the sphere criterion) if the circumsphere of tetrahedron *abcd* does not contain e in its interior. (Note that the circumsphere of *abcd* contains e in its interior if and only if the circumsphere of *abce* contains d in its interior. This follows from the fact that the intersection of the circumspheres of *abcd* and *abce* is the circumcircle of triangle *abc*.)

LEMMA 1. (a) *Let a, b, c, d, e be five vertices of a convex hexahedron as in Fig.* 1. *Then either the interior face abc is locally optimal or the three interior faces ade, bde, cde are all locally optimal. Only one of these two cases holds if the five vertices are not co-spherical.*

(b) *Let abde, acde, bcde be three tetrahedrons in the configuration of Fig.* 1(ii). *Then the three interior faces ade, bde, cde are either all locally optimal or all not locally optimal.*

(c) *Let abce, abde, acde, bcde be four tetrahedrons in the third configuration. Then the six interior faces, abe, ace, ade, bce, bde, cde are all locally optimal.*

Proof. In part (a), either Fig. 1(i) or Fig. 1(ii) must be a Delaunay triangulation since these are the only two possible triangulations. A Delaunay triangulation satisfies the sphere criterion so all its interior faces are locally optimal. In the case that the five vertices are not co-spherical, only one of the triangulations can be Delaunay; the non-Delaunay triangulation does not satisfy the sphere criterion, and hence the circumsphere of at least one of its tetrahedrons contains a vertex in its interior, implying that at least one of its interior faces is not locally optimal since there are only five vertices. Therefore part (a) holds.

Suppose the non-Delaunay triangulation is Fig. 1(ii). Without loss of generality, let the circumsphere of tetrahedron *abde* contain c in its interior. Then *ade and bde* are not locally optimal. Hence the circumsphere of tetrahedron *acde* contains b in its interior, and *cde* is not locally optimal. Therefore part (b) holds.

Part (c) follows from the fact that there is only one possible triangulation in the third configuration, so it must be a Delaunay triangulation and satisfy the sphere criterion. □

THEOREM 1. *A three-dimensional triangulation T is a Delaunay triangulation* (i.e., *satisfies the sphere criterion*) *if and only if every interior face of T is locally optimal.*

Proof. The "only if" part is clearly true by definition.

Suppose every interior face of T is locally optimal. We will show by contradiction that the sphere criterion is satisfied. Suppose *abcd* is a tetrahedron in T such that the circumsphere of *abcd* contains vertex p in its interior. Without loss of generality, suppose p is on the opposite side of face *abc* from *d*. Then *abc* must be an interior face. Let *abce* be the other tetrahedron with face *abc*. Then *abce* does not contain p and *abc* is locally optimal, i.e., e is not in the interior of the circumsphere S of *abcd*. Let S' be the circumsphere of *abce*. If e is on S, then S' clearly contains p in its interior. Suppose e is exterior to S. Let $R = (\text{interior of } S) \cap H$, where H is the half-space containing p that is determined by the plane containing *a*, *b*, and *c*. Since the intersection of S and S' is the circumcircle of triangle *abc*, S' must contain R in its interior, and thus S' must contain p in its interior.

The above argument can be repeated with tetrahedron *abce* replacing *abcd*, etc. The result is a sequence of connected nonoverlapping tetrahedrons such that the circumsphere of each tetrahedron contains p in its interior. Let this sequence of tetrahedrons be $a_0b_0c_0d_0$, $a_1b_1c_1d_1, \cdots$, where $a_ib_ic_id_i$ and $a_{i+1}b_{i+1}c_{i+1}d_{i+1}$ share common face $a_ib_ic_i$, i.e., $a_{i+1}b_{i+1}c_{i+1}d_{i+1} \equiv a_ib_ic_ie_i$, where e_i is either a_{i+1}, b_{i+1}, or c_{i+1}. From the argument in the previous paragraph, e_i and p are on the same side of $a_ib_ic_i$ for all i. Since there is a finite number of tetrahedrons, the sequence must contain a cycle, i.e., $a_jb_jc_jd_j \equiv a_kb_kc_kd_k$ for some $j < k$. But this results in a contradiction, since it is not possible to have a cycle of connected tetrahedrons such that e_i and p are on the same side of $a_ib_ic_i$ for $i = j, \cdots, k-1$. □

DEFINITION 2. Let *abcd* and *abce* be two tetrahedrons sharing interior face *abc* in a triangulation T. Then face *abc* is said to be *transformable* if either (i) the two tetrahedrons are in the configuration of Fig. 1(i), i.e., line segment *de* intersects the interior of triangle *abc*, or (ii) the boundary of the convex hull of *a*, *b*, *c*, *d*, *e* contains all five vertices, *de* does not intersect triangle *abc* (i.e., *abcd* ∪ *abce* is not convex), and the third tetrahedron needed to fill the convex hull of the five vertices is present in T. If the third tetrahedron in case (ii) is not present in T, then *abc* is *not transformable*, i.e., the local transformation procedure cannot be applied.

The third tetrahedron in case (ii) is either *abde*, *acde*, or *bcde*. It is *abde* if *ab* intersects the interior of triangle *cde*; it is *acde* if *ac* intersects the interior of triangle *bde*; it is *bcde* if *bc* intersects the interior of triangle *ade* (see Fig. 2). If *abcd*, *abce* are in the third configuration, i.e., either *a*, *b*, or *c* is not on the boundary of the convex

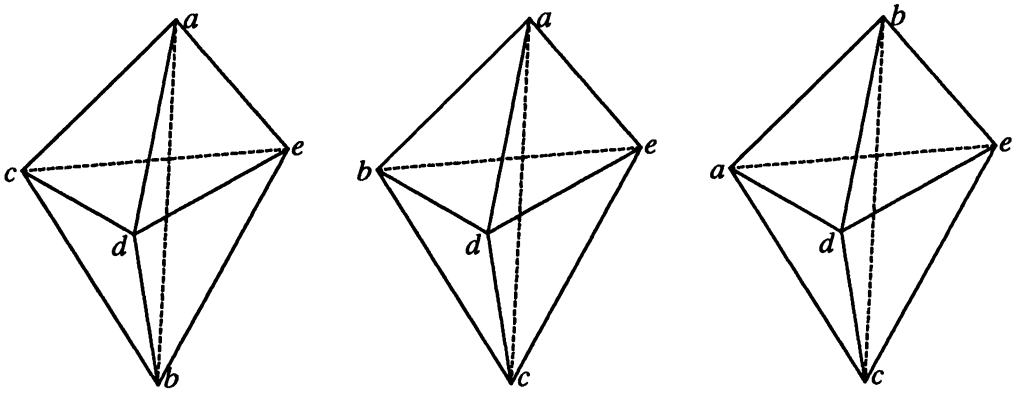


FIG. 2. *Three possible labelings of vertices in Definition* 2(ii).

hull of the five vertices, then *ab* does not intersect triangle *cde*, *ac* does not intersect triangle *bde*, and *bc* does not intersect triangle *ade* (see Fig. 3). Therefore the configuration of *abcd*, *abce* can be determined by line segment and triangle intersection tests.

DEFINITION 3. A triangulation T is said to be *pseudo-locally optimal* (with respect to the sphere criterion) if every nonlocally optimal interior face in T is not transformable. Note that a Delaunay triangulation is pseudo-locally optimal.

LEMMA 2. *Let a, b, c, d, e be the vertices of a strictly convex hexahedron that can be triangulated by tetrahedrons* $T_1 = \{abcd, abce\}$ *or* $T_2 = \{abde, acde, bcde\}$ *as in Fig.* 1. *Let* r_1 (r_2) *be the minimum of the radii of the circumspheres of the tetrahedrons in* T_1 (T_2). *Then* $r_k \leqq r_{3-k}$ *if and only if* T_k *is a Delaunay triangulation of the five vertices, where* $k = 1$ *or* 2.

Proof. Consider spheres expanding at the same rate from centres *a*, *b*, *c*, *d*, and *e*. Let v be the first point (Voronoi vertex) where four or more expanding spheres intersect. If v is the intersection of all five spheres, then the five vertices are co-spherical, $r_1 = r_2$, and both T_1 and T_2 are Delaunay triangulations. Suppose only four expanding spheres intersect at v. Then the tetrahedron formed from the centre of these four spheres has the circumsphere with the smallest radius among the five possible tetrahedrons, and it belongs to the (unique) Delaunay triangulation. □

LEMMA 3. *Let T be a triangulation with m tetrahedrons, and let* $R = (r_1, r_2, \cdots, r_m)$ *be the nondecreasing sequence of circumradii of tetrahedrons in T. Suppose two or three adjacent tetrahedrons in T form a strictly convex hexahedron (as in Fig.* 1) *such that the one or three interior faces are not locally optimal, and the local transformation procedure is applied to these tetrahedrons. Let* $R' = (r'_1, r'_2, \cdots, r'_{m\pm1})$ *be the nondecreasing sequence of circumradii of tetrahedrons in the resulting triangulation T'. Then R' is lexicographically less than R (the shorter sequence can have an arbitrary number added at the end).*

Proof. This lemma follows from Lemma 2. □

LEMMA 4. *Let T be a triangulation containing tetrahedrons abcd and abce such that interior face abc is not locally optimal and is not transformable. Without loss of generality, assume that the "missing" third tetrahedron is abde, i.e., ab intersects the interior of triangle cde. Then there exists another interior face abf,* $f \neq c$, *that is not locally optimal.*

Proof. Let $\bigcirc abcd$ denote the open ball, which is the interior of the circumsphere of tetrahedron *abcd*. Let the tetrahedrons containing edge *ab* be *abcd*, $abdf_1$, abf_1f_2, $\cdots$, abf_ke, *abec*, $k \geqq 1$ in circular order about *ab*. (See the left configuration of Fig. 2 and imagine that there are more vertices between *d* and *e*.) The fact that *abc* is not locally optimal implies that $\bigcirc abcd$ contains *e*. If face *abd* is not locally optimal, then the lemma holds. Hence, suppose *abd* is locally optimal, i.e., $\bigcirc abcd$ does not contain f_1. Then the part of $\bigcirc abcd$ on the opposite side of *abd* from *c* is in $\bigcirc abdf_1$, thus $\bigcirc abdf_1$ contains *e*. This argument can be repeated with face $abf_1, abf_2, \cdots$ in place of *abd*. If faces *abd*, $abf_1, \cdots, abf_{k-1}$ are all locally optimal, then $\bigcirc abcd$, $\bigcirc abdf_1, \cdots,$

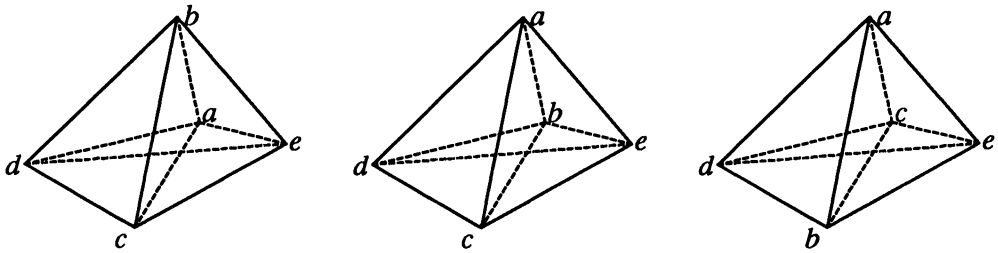


FIG. 3. *Three possible labelings of vertices in third configuration.*

$\bigcirc abf_{k-1}f_k$ all contain e. But that $\bigcirc abf_{k-1}f_k$ contains e implies that face abf_k is not locally optimal. □

DEFINITION 4. Let $abc|de$ denote a pair of adjacent tetrahedrons $abcd$, $abce$ sharing interior face abc such that abc is nonlocally optimal and nontransformable and ab intersects the interior of face cde. We call $abc|de$ a NLONT-*configuration*. Let $C=[a_0b_0c_0|d_0e_0, a_1b_1c_1|d_1e_1, \cdots, a_mb_mc_m|d_me_m]$ denote a sequence of NLONT-configurations. Then C is said to be a *connected* NLONT-*sequence* if $a_ib_ic_i$ and $a_{i+1}b_{i+1}c_{i+1}$ are distinct and share edge a_ib_i (which means that a_ib_i and $a_{i+1}b_{i+1}$ share at least one vertex) for $i=0, 1, \cdots, m-1$; and C is said to be a *connected* NLONT-*cycle* if, in addition, $a_0b_0c_0|d_0e_0=a_mb_mc_m|d_me_m$. Note that the smallest cycle length is $m=2$, which can occur only if $a_0b_0=a_1b_1$.

An example of a connected NLONT-cycle is given below in Fig. 4: $C=[452|67, 475|28, 247|15, 452|67]$, where a vertex is indicated by an integer from 1 to 8. Note that edge 45 intersects the interior of face 267, edge 47 intersects the interior of face 528, and edge 24 intersects the interior of face 715.

LEMMA 5. *Let T be a non-Delaunay triangulation. If T is pseudo-locally optimal, then T contains a connected* NLONT-*cycle.*

Proof. Since T is not Delaunay, it contains at least one nonlocally optimal interior face by Theorem 1. Since T is pseudo-locally optimal, all nonlocally optimal faces are not transformable. So T contains a NLONT-configuration $a_0b_0c_0|d_0e_0$. By Lemma 4, there exists another interior face $a_0b_0f_0$ that is not locally optimal, so T must contain another NLONT-configuration $a_1b_1c_1|d_1e_1$, where $a_1b_1c_1=a_0b_0f_0$. This argument can be repeated for $a_1b_1c_1|d_1e_1$ in place of $a_0b_0c_0|d_0e_0$, then for $a_2b_2c_2|d_2e_2$, etc. The result is a connected NLONT-sequence $[a_0b_0c_0|d_0e_0, a_1b_1c_1|d_1e_1, \cdots]$. Since there is a finite number of faces in T, there must exist j and k such that $0 \leqq j < k$ and $a_jb_jc_j=a_kb_kc_k$, i.e., T contains a connected NLONT-cycle. □

THEOREM 2. *Every non-Delaunay triangulation can be transformed to a pseudo-locally optimal triangulation by a finite sequence of local transformation procedures applied to nonlocally optimal transformable interior faces.*

Proof. Let T_0 be a non-Delaunay triangulation, and let $T_0, T_1, T_2, \cdots$ be a sequence of triangulations where T_{i+1} is obtained from T_i by applying the local transformation procedure to a nonlocally optimal transformable interior face of T_i if such a face exists; otherwise, the sequence terminates at T_i. Let R_i be the nondecreasing sequence of circumradii of tetrahedrons in T_i. From Lemma 3, R_{i+1} is lexicographically less than R_i for all i. Since the R_i are lexicographically decreasing as i increases, it is not possible for the sequence of triangulations to contain a cycle, so the sequence must terminate in a pseudo-locally optimal triangulation T_m. □

COROLLARY 1. *Let T_0 be a non-Delaunay triangulation, and let $T_0, T_1, \cdots, T_m$ be a sequence of triangulations where for $i<m$, T_i is not pseudo-locally optimal and T_{i+1} is obtained from T_i by applying the local transformation procedure to a nonlocally optimal transformable interior face of T_i, and T_m is pseudo-locally optimal. If T_m does not contain a connected* NLONT-*cycle, then T_m is a Delaunay triangulation.*

Proof. This corollary follows from Lemma 5 and Theorem 2. □

If every non-Delaunay triangulation is not pseudo-locally optimal, then it would be straightforward to derive an algorithm to construct a Delaunay triangulation using the local transformation procedure. Unfortunately, we have found an example of a triangulation of eight vertices that is pseudo-locally optimal but not Delaunay. By Lemma 5, this triangulation must contain at least one connected NLONT-cycle. The eight vertices are given in Table 1, the tetrahedrons in the pseudo-locally optimal non-Delaunay and Delaunay triangulations are given in Table 2, and the NLONT-

TABLE 1
Vertex coordinates.

Index	x	y	z
1	0.054	0.099	0.993
2	0.066	0.756	0.910
3	0.076	0.578	0.408
4	0.081	0.036	0.954
5	0.082	0.600	0.726
6	0.085	0.327	0.731
7	0.123	0.666	0.842
8	0.161	0.303	0.975

TABLE 2
Pseudo-locally optimal non-Delaunay triangulation (left) and Delaunay triangulation (right). A tetrahedron is described by its four vertex indices.

1	2	3	5	1	2	3	5
1	2	4	6	1	2	5	6
1	2	4	7	1	2	6	8
1	2	5	6	1	3	4	6
1	2	7	8	1	3	5	6
1	3	4	6	1	4	6	8
1	3	5	6	2	3	5	7
1	4	7	8	2	5	6	8
2	3	5	7	2	5	7	8
2	4	5	6	3	4	6	8
2	4	5	7	3	5	6	7
3	4	6	8	3	6	7	8
3	5	6	7	5	6	7	8
3	6	7	8				
4	5	6	8				
4	5	7	8				
5	6	7	8				

configurations of the former triangulation are given in Table 3. The four tetrahedrons in the connected NLONT-cycle formed from the first three entries of Table 3 are illustrated in Fig. 4.

However, we conjecture that a non-Delaunay triangulation can be transformed to a Delaunay triangulation by a finite sequence of local transformation procedures. From the above example, some of the local transformation procedures may have to be applied to locally optimal transformable interior faces. This is a special case of the following conjecture that holds for two-dimensional triangulations. Unfortunately, the approach of Lawson [8] for proving the two-dimensional version of this conjecture does not extend to the three-dimensional case.

CONJECTURE 1. Given two different triangulations T_1 and T_2 of the same n three-dimensional vertices, T_2 can be obtained from T_1 by a finite sequence of local transformation procedures.

4. Algorithm and data structure. Based on the results of the previous section, we present an algorithm and data structure for constructing a pseudo-locally optimal triangulation of n three-dimensional vertices $v_1, v_2, \cdots, v_n$ (we are still assuming that no four vertices are co-planar). In our algorithm, the n vertices are first sorted in lexicographical order of their coordinates. In the general step, a pseudo-locally optimal

TABLE 3
NLONT-*configurations. The first three form a connected* NLONT-*cycle. There are also three connected* NLONT-*cycles of length* 2.

a	*b*	*c*	*d*	*e*
4	5	2	6	7
4	7	5	2	8
2	4	7	1	5
4	5	6	2	8
4	7	8	1	5
2	4	1	6	7

triangulation T_{i-1} of the first $i-1$ vertices has been constructed, and the ith vertex is added to form a preliminary triangulation of the first i vertices. Note that the ith vertex is outside the convex hull of the first $i-1$ vertices. Then the local transformation procedure is applied to nonlocally optimal transformable faces until a pseudo-locally optimal triangulation T_i of the i vertices is obtained. We now present the pseudocode for our algorithm, called TRSPH1.

In the "for" loop labelled (A) in the pseudocode, a preliminary triangulation is constructed by adding tetrahedrons with vertex v_i to T_{i-1} to fill the convex hull of the first i vertices. The new tetrahedrons are of the form $v_av_bv_cv_i$, where $v_av_bv_c$ is a boundary face of T_{i-1} but is not on the boundary of the convex hull of the first i vertices. This condition holds if and only if v_i and w are on opposite sides of $v_av_bv_c$, where w is any point in the interior of the convex hull of the first $i-1$ vertices, e.g., w is the centroid of the first tetrahedron.

Stack S is used to store the interior faces that are not known to be locally optimal or nontransformable. If $v_av_bv_cv_i$ is a new tetrahedron, then $v_av_bv_c$ may be nonlocally optimal so it is put on S. If $v_av_bv_cv_i$ and $v_av_bv_dv_i$ are adjacent new tetrahedrons, then $v_av_bv_i$ may be a nonlocally optimal interior face, but it is not put on S because the union of $v_av_bv_cv_i$ and $v_av_bv_dv_i$ is not convex, so either $v_av_bv_i$ is not transformable or tetrahedron $v_av_bv_cv_d$ is present in the triangulation, in which case $v_av_bv_cv_i$, $v_av_bv_dv_i$, $v_av_bv_cv_d$ are replaced by the two tetrahedrons $v_av_cv_dv_i$, $v_bv_cv_dv_i$ if either interior face $v_av_bv_c$ or $v_av_bv_d$ is determined to be nonlocally optimal.

It is possible that the face $v_av_bv_c$ referred to in the statement labelled (B) is no longer in the triangulation since three interior faces are replaced by one interior face

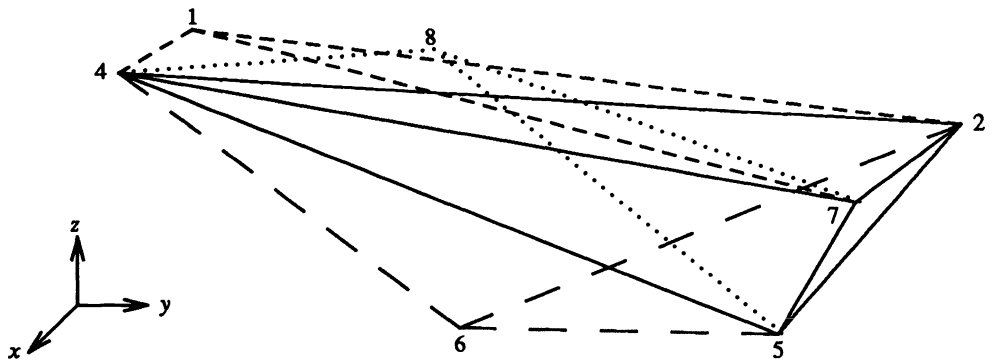


FIG. 4. *Connected* NLONT-*cycle formed from the first three entries of Table* 3. *Four tetrahedrons are*: 2457, 2456, 4578, 1247.

```
ALGORITHM TRSPH1
Sort v_1, v_2, ..., v_n into lexicographical order
Form first tetrahedron v_1v_2v_3v_4
Compute centroid w = (v_1 + v_2 + v_3 + v_4)/4
Initialize S to empty stack
for i := 5 to n do
    Let T_{i-1} = current triangulation of first
      i - 1 vertices (T_{i-1} is pseudo-locally optimal)
(A) for each boundary face v_av_bv_c of T_{i-1} do
        if v_i is on the opposite side of v_av_bv_c from w then
            Add tetrahedron v_av_bv_cv_i to triangulation
            Push interior face v_av_bv_c on stack S
        endif
    endfor
    while stack S is not empty do
        Pop interior face v_av_bv_c from stack S
(B)     if v_av_bv_c is still in triangulation then
            Find the two tetrahedrons v_av_bv_cv_d, v_av_bv_cv_e sharing face v_av_bv_c
(C)         if the circumsphere of v_av_bv_cv_d contains v_e in its interior then
                transform := true
                if v_av_bv_cv_d ∪ v_av_bv_cv_e is a convex hexahedron then
                    Replace v_av_bv_cv_d, v_av_bv_cv_e by the three tetrahedrons
                        v_av_bv_dv_e, v_av_cv_dv_e, v_bv_cv_dv_e
                else
                    if the third tetrahedron needed to fill the convex
                        hull of v_a, ..., v_e is present in the triangulation then
                        Relabel vertices so that three tetrahedrons are v_av_bv_dv_e,
                            v_av_cv_dv_e, v_bv_cv_dv_e
                        Replace v_av_bv_dv_e, v_av_cv_dv_e, v_bv_cv_dv_e by v_av_bv_cv_d, v_av_bv_cv_e
                    else
                        transform := false
                    endif
                endif
                if transform then
(D)                 for each of faces v_av_bv_d, v_av_bv_e, v_av_cv_d,
                        v_av_cv_e, v_bv_cv_d, v_bv_cv_e do
                        Push face on stack S if it is an interior face and it is
                            not yet in S
                    endfor
                endif
            endif
        endif
    endwhile
endfor
```

in the case of two tetrahedrons replacing three tetrahedrons in the local transformation procedure. In the statement labelled (C), a test is made to see whether or not $v_av_bv_c$ is locally optimal. If not, then the local transformation procedure is applied if $v_av_bv_c$ is transformable (the configuration of the two tetrahedrons $v_av_bv_cv_d$, $v_av_bv_cv_e$ can be

determined as described in Definition 2 and the paragraph after it). If the local transformation procedure is applied to $v_a v_b v_c$, then a boundary face of the convex hexahedron formed by the two or three tetrahedrons may no longer be locally optimal, so it is placed on stack S if it is an interior face of the triangulation and it is not yet in S (see statement (D)).

From the above discussion and the pseudocode, it is clear that on completion of the ith step with S empty, the triangulation of the first i vertices, T_i, is pseudo-locally optimal. If T_{i-1} is a Delaunay triangulation of the first $i-1$ vertices, then it seems likely that T_i does not contain a connected NLONT-cycle and is also Delaunay, hence we have the following conjecture.

CONJECTURE 2. Algorithm TRSPH1 constructs a Delaunay triangulation for all sets of three-dimensional vertices.

We now describe the data structure for our three-dimensional triangulation algorithm. The vertex coordinates are stored in an array VC where $VC[i].x$, $VC[i].y$, and $VC[i].z$ are the coordinates of the ith vertex, v_i. The faces and tetrahedrons are changing throughout the algorithm, and there are searching operations on the faces (e.g., find the two tetrahedrons sharing interior face $v_a v_b v_c$). Hence the faces are stored in a hash table HT with direct chaining, where $HT[i]$ is the head pointer of the linked list of faces with hashing function value i. A new face is added at the front of a linked list, since it is more likely to be referenced again. Let $a < b < c$ be the three indices in VC of the three vertices v_a, v_b, v_c of a face. A satisfactory hashing function is $h(a, b, c) = (an^2 + bn + c) \bmod M$, where the hash table size, M, is a prime number. (For descriptions of hashing and linked lists, see any data structure book, e.g., Standish [13].)

We store the elements of the hash table linked lists in an array FC of face records with origin index 1, so that some fields of the face record can be used for two purposes depending on whether the face is an interior or boundary face. The fields of $FC[i]$ are *a, b, c, d, e|flink, stlink|blink, htlink*, where $0 < a < b < c$ are the three vertex indices of a face; v_d and possibly v_e are the fourth vertices of the one or two tetrahedrons with (boundary or interior, respectively) face $v_a v_b v_c$; *stlink* indicates whether or not interior face $v_a v_b v_c$ is in stack S and in the former case, it is also a pointer to the next face in S; *flink* and *blink* are forward and backward pointers for a doubly linked list of boundary faces (since the boundary faces must be traversed and updated in step (A) of the pseudocode); and *htlink* is the pointer to the next element in the hash table linked list.

All four link fields represent positive indices in the FC array or zero for end of list, but due to the double use of some fields, the actual values stored in the *flink*, *blink*, and *stlink* fields are slightly modified. If $ptr \geqq 0$ is the real pointer value, then $-ptr$ is stored in the *flink* or *blink* field and $ptr+2$ is stored in the *stlink* field. For interior faces (with a positive integer in the *e|flink* field), $stlink = 1$ is used to indicate that the face is not in stack S and $stlink > 1$ is used to indicate that the face is in S as well as the pointer to the next element of S. If a face is in stack S but no longer in the triangulation, then b is set to zero to indicate this and the face record is deleted when it reaches the top of S. We also use the a field to maintain an avail linked list of deleted face records (with nonpositive values to indicate pointers as for *flink*), so that a new face record can be obtained from the avail list if it is nonempty or the end of array otherwise.

With this data structure, searching, insertion, and deletion of faces are straightforward, and each operation should take constant time with a sufficiently large hash table size. At the end of the algorithm, the list of tetrahedrons in the triangulation can be obtained by sequentially traversing the array FC. Since each tetrahedron appears four

times in the data structure, duplicates can be avoided as follows. For each new tetrahedron, search for the other three representations and use the *stlink|blink* field to indicate whether the tetrahedron determined by the *d* or *e* field has been listed already.

An example of the *VC* array is given in Table 1. An example of the *FC* and *HT* arrays is given in Fig. 5 for the four tetrahedrons in Fig. 4, where any face appearing in only one tetrahedron is taken to be a boundary face. The number of vertices is $n=8$, the hash table size is $M=5$, and the hashing function is $h(a, b, c) = (an^2+bn+c) \bmod M$. *TOP*, *HEAD*, and *TAIL* are scalar variables that are pointers to the top of stack *S* and the head and tail of the doubly linked list of boundary faces.

An obvious variation of algorithm TRSPH1 is to first construct an initial triangulation T_I as in step (A), then to put all interior faces of T_I in stack *S*, and finally to apply local transformation procedures to nonlocally optimal transformable faces of *S* as in the main "while" loop of TRSPH1. If the interior faces are added to stack *S* by sequentially traversing the array *FC* in the forward (backward) direction, then we call this algorithm TRSPH2 (TRSPH3, respectively). Note that algorithm TRSPH1 can be interpreted as constructing T_I first (although T_I never actually exists during the algorithm) and then processing the interior faces in a different order from algorithms TRSPH2 and TRSPH3. The order of processing the interior faces in TRSPH1 is closer to that in TRSPH3 than TRSPH2, since in TRSPH3 faces created closer to the beginning of the construction of T_I are closer to the top of stack *S* initially. Since T_I is in general not close to a Delaunay triangulation, it seems likely that TRSPH2 and TRSPH3 have a greater chance than TRSPH1 of ending up with a connected NLONT-cycle and a pseudo-locally optimal triangulation that is not Delaunay. In § 7, we report on experiments that compare these algorithms.

5. Degeneracy. In this section, we describe the extensions to the results of § 3 and algorithm TRSPH1 of § 4 when we remove the assumption that no four vertices are co-planar. Definition 1 and Theorem 1 of § 3 (about locally optimal faces) still hold when subsets of four co-planar vertices are allowed.

The degenerate configurations for two tetrahedrons *abcd* and *abce* sharing common face *abc* with *d* and *e* on opposite sides of *abc* are illustrated in Fig. 6, where the vertices of *abc* are labelled so that *a*, *b*, *d*, and *e* are co-planar and *c* lies on a different plane. In Fig. 6(i), quadrilateral *adbe* is strictly convex and the other triangulation of the five vertices contains tetrahedrons *acde* and *bcde*. In Fig. 6(ii), quadrilateral *adbe* degenerates to a triangle. In Fig. 6(iii), quadrilateral *adbe* is nonconvex and tetrahedron *acde* must be added to fill the convex hull of the five vertices. In the latter two cases, there are no other possible triangulations of the five vertices.

Hence, an additional case to the local transformation procedure described in § 2 is as follows. If tetrahedrons *abcd* and *abce* are in the configuration of Fig. 6(i), then replace them by tetrahedrons *acde* and *bcde*, i.e., swap interior face *abc* for face *cde*.

In the three configurations of Fig. 6, the circumcircles of faces *abd* and *abe* (which are on the circumspheres of tetrahedrons *abcd* and *abce*, respectively) are in the same plane. This implies that interior face *abc* is locally optimal if and only if the circumcircle of *abd* does not contain *e* in its interior (i.e., edge *ab* is locally optimal in the two-dimensional triangulation of *a*, *b*, *d*, *e*). Therefore, in Figs. 6(ii) and 6(iii), *abc* is locally optimal, and in Fig. 6(i), either *abc* or *cde* is locally optimal.

DEFINITION 5 (extension of Definition 2). Let *abcd* and *abce* be two tetrahedrons in triangulation *T* that are in the configuration of Fig. 6(i). Then face *abc* is said to be *transformable* if either (i) *abd* and *abe* are boundary faces of *T*, or (ii) *abd* and *abe* are interior faces of *T* and there is a vertex *f* (on the opposite side of *abd* from *c*)

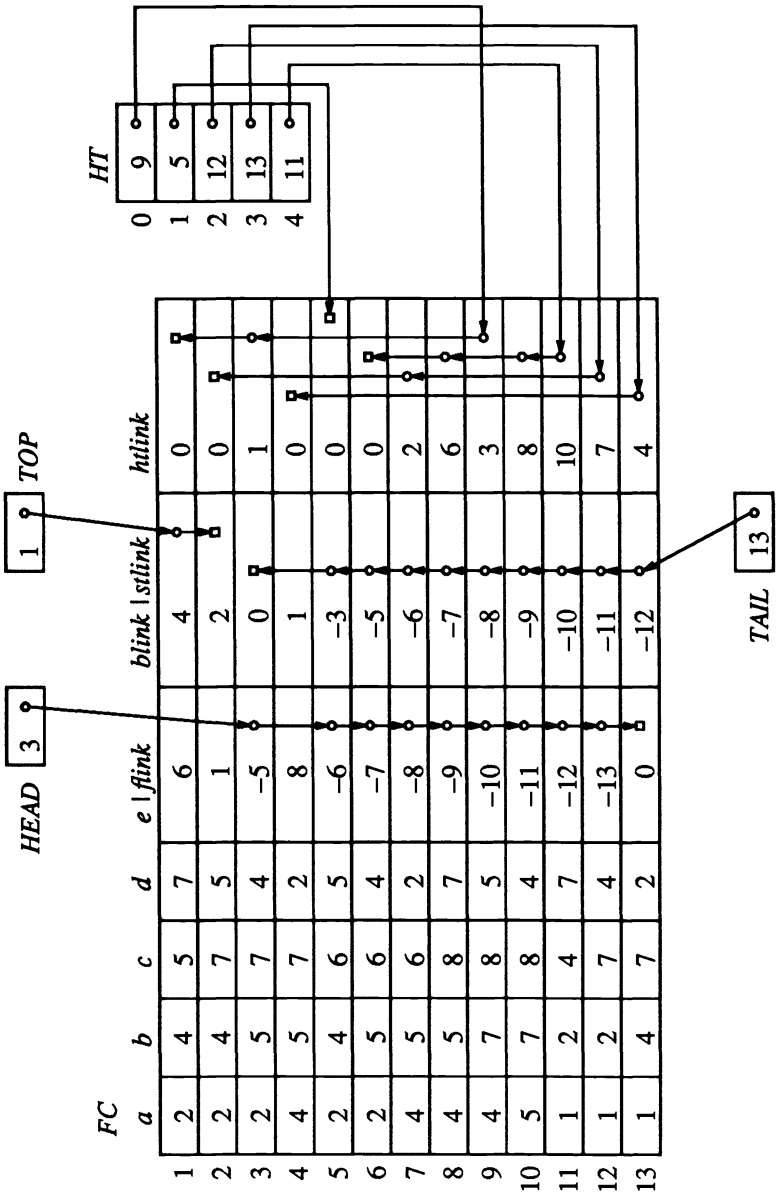


FIG. 5. *Illustration of FC and HT arrays for the four tetrahedrons in Fig. 4. 245, 247, 457 are interior faces, with the former two on stack S; the remaining faces are boundary faces.*

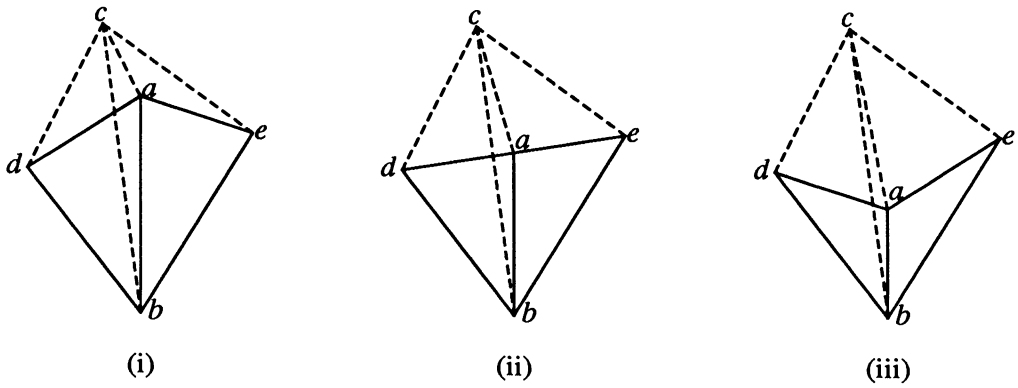


FIG. 6. *Three possible cases of two tetrahedrons with a, b, d, e co-planar.*

such that *abdf* and *abef* are tetrahedrons of T. Face *abc* is *not transformable*, if in case (ii), the other two tetrahedrons of T containing faces *abd* and *abe* are *abdf* and *abeg* where $f \neq g$.

Note that the configuration of Fig. 6(i) can be detected and distinguished from the other configurations of § 3 by the fact that *de* intersects the boundary of triangle *abc*. The main extension to algorithm TRSPH1 is to detect the configuration of Fig. 6(i) when face *abc* from stack S is not locally optimal and to apply the local transformation procedure to *abc* if it is transformable. In the case that *abd* and *abe* are interior faces, the local transformation procedure must be applied to both *abc* and *abf* where f is defined in Definition 5(ii), i.e., tetrahedrons *abcd* and *abce* are replaced by *acde* and *bcde*, and tetrahedrons *abdf* and *abef* are replaced by *adef* and *bdef*. In this case, both *cde* and *def* are locally optimal faces in the new triangulation, and the faces that may have to be put on stack S are *ade*, *bde*, *acd*, *ace*, *bcd*, *bce*, *adf*, *aef*, *bdf*, and *bef*. In the case that *abd* and *abe* are boundary faces, the faces that may have to be put on stack S are *acd*, *ace*, *bcd*, and *bce*.

The only other modification to the algorithm is a possible slight reordering of the sorted vertices to get a valid first tetrahedron (i.e., v_1, v_2, v_3, and v_4 are not co-planar). Let $v_1, \cdots, v_n$ be the sorted vertices. Let v_k, $k \geqq 3$, be the vertex of smallest index such that v_1, v_2, and v_k are not collinear. Let v_m, $m > k$ be the vertex of smallest index such that v_1, v_2, v_k, and v_m are not co-planar. Then shift the vertices to get the new ordering: $v_1, v_2, v_k, v_m, v_3, \cdots, v_{k-1}, v_{k+1}, \cdots, v_{m-1}, v_{m+1}, \cdots, v_n$. Note that with this new ordering, the vertices still satisfy the property that the ith vertex is outside the convex hull of the first $i-1$ vertices. The modifications to algorithms TRSPH2 and TRSPH3 are clearly similar.

All the remaining results in § 3 also extend to the case when subsets of four co-planar vertices are allowed. Lemmas 2 and 3 extend to the two possible triangulations in the configuration of Fig. 6(i). The extensions of Lemma 4 and Definition 4 are as follows.

LEMMA 6. *Let T be a triangulation containing tetrahedrons abcd and abce in the configuration of Fig.* 6(i) *such that interior face abc is not locally optimal and is not transformable, i.e., abdf and abeg are tetrahedrons of T where $f \neq g$. Then there exists another interior face abh, $h \neq c$, that is not locally optimal.*

Proof. The proof is similar to proof of Lemma 4. □

DEFINITION 6. In addition to the NLONT-configuration given in Definition 4, *abc|de* is also a NLONT-*configuration* if *abcd* and *abce* are adjacent tetrahedrons

sharing interior face *abc* as in Fig. 6(i) such that *abc* is nonlocally optimal and nontransformable.

Lemma 5 still holds with the extended definition of NLONT-configuration. Theorem 2, Corollary 1, and Conjecture 1 still hold, provided T_{i+1} is obtained from T_i by two simultaneous applications of the local transformation procedure in Definition 5(ii) as discussed above.

6. Max-min solid angle criterion. In this section, we describe how our algorithms for constructing a pseudo-locally optimal triangulation with respect to the sphere criterion can be modified to construct a locally optimal triangulation with respect to the max-min solid angle criterion. The main modifications are due to the definition of "locally optimal" with respect to the max-min solid angle criterion.

DEFINITION 7. Let *abcd* and *abce* be two tetrahedrons sharing interior face *abc* in a triangulation T, where *abc* is a transformable face (see Definitions 2 and 5). Let T_1 contain the tetrahedrons of T that fill the convex hull of a, b, c, d, e. Let T_2 contain the tetrahedrons in the alternative triangulation of a, b, c, d, e. If *abc* satisfies Definition 5(ii), then let T_1 additionally contain tetrahedrons *abdf* and *abef*, and let T_2 additionally contain tetrahedrons *adef* and *bdef*. Then face *abc* is said to be *locally optimal* with respect to the max-min solid angle criterion if $s(T_1) \geqq s(T_2)$ where $s(T_i) = \min$ {solid angles at vertices of tetrahedrons of T_i}.

DEFINITION 8. A triangulation T is said to be SA-*locally optimal* if every transformable interior face in T is locally optimal with respect to the max-min solid angle criterion. (To avoid confusion with Definition 3, we use SA-*locally optimal* instead of *pseudo-locally optimal.*)

DEFINITION 9. A triangulation T of a set S of three-dimensional vertices is said to be SA-*globally optimal* if over all possible triangulations of S, the minimum of the solid angles at all vertices of all tetrahedrons is maximized in triangulation T.

Note that, unlike the case of the sphere criterion, a SA-locally optimal triangulation may not be SA-globally optimal. We have no theoretical results such as those of Theorem 1 that characterize a SA-globally optimal triangulation. It is possible that the problem of constructing a SA-globally optimal triangulation is NP-hard. The following results, which are similar to Lemma 3 and Theorem 2, indicate how an SA-locally optimal triangulation can be constructed.

LEMMA 7. Let T be a triangulation with m tetrahedrons, let a_i be the minimum of the four solid angles of a tetrahedron, and let $A = (a_1, a_2, \cdots, a_m)$ be the nondecreasing sequence of a_i values of tetrahedrons in T. Suppose the local transformation procedure is applied to a nonlocally optimal interior face (or two faces if they satisfy Definition 5(ii)) in T. Let $A' = (a'_1, a'_2, \cdots, a'_r)$ be the nondecreasing sequence of a_i values of tetrahedrons in the resulting triangulation. Then A' is lexicographically greater than A.

Proof. This lemma follows from Definition 7. □

THEOREM 3. *Every triangulation that is not* SA-*locally optimal can be transformed to a* SA-*locally optimal triangulation by a finite sequence of local transformation procedures applied to nonlocally optimal interior faces.*

Proof. This theorem follows from Lemma 7 using the same approach as in the proof of Theorem 2. □

To construct an SA-locally optimal triangulation, the algorithms given in §§ 4 and 5 must be modified, when processing a face $v_a v_b v_c$ from stack S, to first check whether $v_a v_b v_c$ is transformable before determining whether it is nonlocally optimal. The only other modification is inside the "for" loop labelled (A) in which face $v_a v_b v_i$, $v_a v_c v_i$, or

$v_b v_c v_i$ must be added to stack S if it is an interior face. The reason for this is that a face *abc* in the configuration of Fig. 6(i) may be locally optimal if *abd* and *abe* are boundary faces, but *abc* may become nonlocally optimal if *abd* and *abe* become interior faces. We call the modified algorithms TRMMSA1, TRMMSA2, TRMMSA3, with the obvious correspondence to the earlier algorithms.

7. Experimental results. We implemented algorithms TRSPH1, TRSPH2, TRSPH3, TRMMSA1, TRMMSA2, and TRMMSA3 in Fortran, and ran many test problems to compare these algorithms and to determine the empirical time complexity. It is not possible to obtain an average or worst-case time complexity analytically since the complexity depends on the number of faces tested for local optimality and the number of applications of the local transformation procedure for which we have no general bounds. The implementations used double precision floating point arithmetic and were compiled using the *f*77 compiler without the optimization option (due to compiler bugs). The tests were done on a Sun 3/50 workstation with a MC68881 floating point processor running the Sun Unix 4.2 operating system.

We used 60 test problems in our experiment, consisting of 11 problems each for the five different n values 100, 200, 300, 400, 500, and a 12th problem for the n values 50, 100, 150, 200, 250. For fixed n, the description of the 12 problems is as follows. The first seven problems have vertex coordinates that are pseudorandom numbers from the uniform distribution in $[0, a_x] \times [0, a_y] \times [0, a_z]$. Problem $\mathrm{P1}_n$, $\mathrm{P2}_n$, $\mathrm{P3}_n$, $\mathrm{P4}_n$ have $a_x = a_y = a_z = 1$; problem $\mathrm{P5}_n$ has $a_x = 0.5$, $a_y = a_z = 1$; problem $\mathrm{P6}_n$ has $a_x = 1$, $a_y = 0.5$, $a_z = 1$; and problem $\mathrm{P7}_n$ has $a_x = a_y = 1$, $a_z = 0.5$. The next three problems are used to test the degenerate configurations in which four vertices are co-planar. They consist of approximately $\sqrt{n}$ parallel planes, each with an average of $\sqrt{n}$ vertices. The planes are orthogonal to one of the three axes and are determined by pseudorandom uniform numbers in $[0, 1]$. The vertices on each plane have vertex coordinates that are pseudorandom uniform numbers in $[0, 1] \times [0, 1]$. In problems $\mathrm{P8}_n$, $\mathrm{P9}_n$, and $\mathrm{P10}_n$, the parallel planes have the form $x = c$, $y = c$, and $z = c$, respectively.

The last two problems are not random; one does not have a unique Delaunay triangulation and the other has a Delaunay triangulation containing $O(n^2)$ tetrahedrons. In problem $\mathrm{P11}_n$, the vertex coordinates are on a uniform grid and have the form (i, j, k), where i, j, and k are integers in the ranges 0 to $n_x - 1$, 0 to $n_y = 1$, and 0 to $n_z - 1$, respectively, and $n = n_x n_y n_z$. For $n = 100$, $n_x = 4$, $n_y = n_z = 5$; for $n = 200$, $n_x = n_y = 5$, $n_z = 8$; for $n = 300$, $n_x = 5$, $n_y = 6$, $n_z = 10$; for $n = 400$, $n_x = 5$, $n_y = 8$, $n_z = 10$; and for $n = 500$, $n_x = 5$, $n_y = n_z = 10$. The number of tetrahedrons in a Delaunay triangulation of this problem can range from $5(n_x - 1)(n_y - 1)(n_z - 1)$ to $6(n_x - 1)(n_y - 1)(n_z - 1)$, since a unit cube can be triangulated by five or six tetrahedrons.

Problem $\mathrm{P12}_n$ has $k = \lfloor n/2 \rfloor$ vertices that are equally spaced points on the unit circle centred about the origin in the x-y plane, and $m = n - k$ points that are equally spaced in interval $[0, 1]$ of the z-axis, i.e., $v_i = (\cos(i\alpha), \sin(i\alpha), 0)$ for $i = 1, \cdots, k$ and $v_{i+k} = (0, 0, (i-1)s)$ for $i = 1, \cdots, m$ where $\alpha = 2\pi/k$ and $s = 1/(m-1)$. All tetrahedrons in the Delaunay triangulation of this problem must consist of two vertices with index $\leqq k$ and two with index $> k$, therefore the number of tetrahedrons, faces, and boundary faces in the Delaunay triangulation are $n(n-2)/4$, $n(n-1)/2$, and n, respectively, for even n.

For our experiment, we used the hashing function given in § 4 and a hash table size $M \approx 1.5n$, where M is a prime number and n is the number of vertices, so the storage complexity of the algorithms is proportional to the number of faces in the triangulation. For all algorithms and problems $\mathrm{P1}_n$ to $\mathrm{P11}_n$, the average number of

face records compared when a face is searched in the hash table is a small constant: $\leqq 2.0$ for TRSPH1 and TRMMSA1, $\leqq 4.7$ for TRSPH2 and TRMMSA2, and $\leqq 6.1$ for TRSPH3 and TRMMSA3. For problem $P12_n$, the average number of face records compared during searches is up to 8.2 for TRSPH1, TRSPH2, TRSPH3, TRMMSA1, and 19.7 for TRMMSA2, TRMMSA3 when $n = 250$; the higher numbers are due to a quadratic number of faces (the hash table size should be proportional to the number of faces in order to get a constant number of comparisons, on average).

The following quantities are used to measure the performance of the algorithms (i refers to the algorithm number in TRSPHi or TRMMSAi):

NTETi— number of tetrahedrons in triangulation;
NFACi— number of faces in triangulation;
NBFCi— number of boundary faces in triangulation;
TIMi— CPU time in seconds for constructing triangulation;
TInit— CPU time in seconds for sorting vertices and producing initial triangulation T_I (or producing preliminary tetrahedrons in step (A) of TRSPH1), so TIMi—TInit is the CPU time spent in checking faces for local optimality and transformability, applying the local transformation procedure, and updating stack S;
LOPi— number of faces that are locally optimal when tested for local optimality;
LTPi— number of applications of the local transformation procedure (the degenerate case in which four tetrahedrons are replaced by four other tetrahedrons is counted as one application);
NTFi— number of faces that are nonlocally optimal and nontransformable when tested for local optimality in TRSPHi or that are nontransformable when tested for transformability in TRMMSAi;
MSAi— minimum solid angle in radians at vertices of tetrahedrons of triangulation.

Note that LOPi+LTPi+NTFi is the number of faces on stack S that are tested for local optimality (transformability) in algorithm TRSPHi (TRMMSAi).

We first describe the experimental results from running the 60 test problems for algorithms TRSPH1, TRSPH2, and TRSPH3. TRSPH1 constructed a Delaunay triangulation for all the problems (this is verified by checking that all interior faces are locally optimal). TRSPH3 failed to construct a Delaunay triangulation for only problem $P1_{300}$; for this pseudo-locally optimal triangulation, there are 34 nonlocally optimal nontransformable interior faces. TRSPH2 constructed pseudo-locally optimal non-Delaunay triangulations for 18 of the 60 problems: $P5_{200}$, $P7_{200}$, $P10_{200}$, $P1_{300}$, $P5_{300}$, $P6_{300}$, $P8_{300}$, $P1_{400}$, $P2_{400}$, $P4_{400}$, $P5_{400}$, $P9_{400}$, $P1_{500}$, $P3_{500}$, $P4_{500}$, $P5_{500}$, $P6_{500}$, $P9_{500}$. The number of nonlocally optimal faces in these triangulations are 21, 13, 20, 27, 27, 46, 12, 74, 35, 48, 68, 13, 23, 7, 57, 27, 52, and 33, respectively. It appears that algorithm TRSPH2 is more likely to construct a non-Delaunay triangulation as n increases.

We split the measurements into three categories, the average of $P1_n$ to $P10_n$, $P11_n$, and $P12_n$, since the performance of the algorithms on the last two problems is significantly different from the random problems. The measurements for the 10 random problems are approximately the same, with problems $P5_n$ and $P8_n$ always having the highest CPU times and number of faces tested for local optimality (this is probably because the lexicographical ordering of the vertices causes these two problems to have more long tetrahedrons with small solid angles in the initial triangulation T_I). Tables 4, 5, and 6 contain measurements for the average of $P1_n$ to $P10_n$. Tables 7, 8, and 9

TABLE 4

Average of problems $P1_n$ *to* $P10_n$, *algorithm* TRSPH*i*: *number of tetrahedrons and faces in Delaunay triangulation and* CPU *times.*

n	NTET1	NFAC1	NBFC1	TInit	TIM1	TIM2	TIM3
100	504.0	1039.2	62.4	2.62	22.71	22.19	22.15
200	1129.2	2298.7	80.6	6.59	64.20	62.56	63.07
300	1761.8	3572.7	98.2	11.91	113.97	108.88	111.08
400	2386.9	4834.4	121.2	19.38	167.74	160.96	166.12
500	3042.9	6143.3	115.0	24.11	226.73	219.80	223.91

TABLE 5

Average of problems $P1_n$ *to* $P10_n$, *algorithm* TRSPH*i*: *number of faces tested for local optimality.*

n	LOP1	LTP1	NTF1	LOP2	LTP2	NTF2	LOP3	LTP3	NTF3
100	3,568	1,072	474	3,363	1,012	652	3,440	1,066	494
200	10,278	3,119	1,435	9,415	2,917	1,944	9,854	3,091	1,511
300	18,160	5,557	2,654	16,385	5,105	3,416	17,346	5,484	2,727
400	26,508	8,167	3,957	23,800	7,491	5,102	25,513	8,104	4,081
500	36,098	11,139	5,444	32,867	10,383	6,974	34,749	11,028	5,603

TABLE 6

Average of problems $P1_n$ *to* $P10_n$, *algorithm* TRSPH*i*: *complexity of number of faces, times, and number of tests for local optimality.*

n	$\frac{\text{NFAC1}}{n}$	$\frac{\text{NBFC1}}{n^{1/3}}$	$\frac{\text{TInit}}{n^{4/3}}$	$\frac{\text{TIM1}}{n^{4/3}}$	$\frac{\text{LOP1}}{n^{4/3}}$	$\frac{\text{LTP1}}{n^{4/3}}$	$\frac{\text{NTF1}}{n^{4/3}}$
100	10.4	13.4	.00565	.0489	7.69	2.31	1.02
200	11.5	13.8	.00564	.0549	8.79	2.67	1.23
300	11.9	14.7	.00593	.0567	9.04	2.77	1.32
400	12.1	16.4	.00657	.0569	8.99	2.77	1.34
500	12.3	14.5	.00608	.0571	9.10	2.81	1.37

TABLE 7

Problem $P11_n$, *algorithm* TRSPH*i*: *number of tetrahedrons and faces in Delaunay triangulation and* CPU *times.*

n	NTET1	NFAC1	NBFC1	TInit	TIM1	TIM2	TIM3
100	288	656	160	4.08	12.87	14.18	13.57
200	672	1,488	288	14.90	44.53	50.87	48.23
300	1,080	2,362	404	31.70	97.22	111.03	105.08
400	1,512	3,278	508	53.58	171.73	192.62	182.32
500	1,944	4,194	612	81.52	270.32	296.53	277.92

TABLE 8

Problem $P11_n$, *algorithm* TRSPH*i*: *number of faces tested for local optimality.*

n	LOP1	LTP1	NTF1	LOP2	LTP2	NTF2	LOP3	LTP3	NTF3
100	1,834	474	188	2,149	472	255	1,956	468	211
200	6,051	1,680	643	7,305	1,676	1,064	6,852	1,689	806
300	13,311	3,684	1,500	15,845	3,672	2,633	14,880	3,680	1,863
400	23,870	6,312	3,032	27,417	6,260	4,986	25,653	6,194	3,539
500	38,005	9,796	5,210	42,044	9,620	8,049	39,013	9,370	5,859

TABLE 9

Problem $P11_n$, *algorithm* TRSPHi: *complexity of number of faces, times, and number of tests for local optimality.*

n	NFAC1 / n	NBFC1 / $n^{0.8}$	TInit / $n^{1.8}$	TIM1 / n^2	LOP1 / n^2	LTP1 / n^2	NTF1 / n^2
100	6.56	4.02	.00103	.00129	.183	.0474	.0188
200	7.44	4.16	.00108	.00111	.151	.0420	.0161
300	7.87	4.21	.00110	.00108	.148	.0409	.0167
400	8.20	4.21	.00111	.00107	.149	.0395	.0190
500	8.39	4.24	.00113	.00108	.152	.0392	.0208

contain measurements for $P11_n$. Tables 10, 11, and 12 contain measurements for $P12_n$. The CPU times in these tables are subject to a variation of up to about 5 percent when the program is run at different times.

For the random problems, it can be seen from Table 6 that the number of faces in the Delaunay triangulation is $O(n)$, and the time complexity of algorithm TRSPH1 is approximately $O(n^{4/3})$. From equation (1)(a), the number of tetrahedrons in the

TABLE 10

Problem $P12_n$, *algorithm* TRSPHi: *number of tetrahedrons and faces in Delaunay triangulation and* CPU *times.*

n	NTET1	NFAC1	NBFC1	TInit	TIM1	TIM2	TIM3
50	600	1,225	50	1.25	7.77	8.02	8.38
100	2,450	4,950	100	5.18	31.37	34.62	35.32
150	5,550	11,175	150	11.55	73.30	79.13	73.38
200	9,900	19,900	200	19.12	116.47	129.95	131.68
250	15,500	31,125	250	30.57	184.45	207.70	208.45

TABLE 11

Problem $P12_n$, *algorithm* TRSPHi: *number of faces tested for local optimality.*

n	LOP1	LTP1	NTF1	LOP2	LTP2	NTF2	LOP3	LTP3	NTF3
50	1,385	274	0	1,608	274	0	1,719	274	0
100	5,971	1,199	0	6,959	1,199	0	7,244	1,199	0
150	13,537	2,700	0	16,067	2,700	0	16,421	2,700	0
200	24,151	4,800	0	28,810	4,800	0	29,297	4,800	0
250	38,185	7,624	0	45,521	7,624	0	46,119	7,624	0

TABLE 12

Problem $P12_n$, *algorithm* TRSPHi: *complexity of number of faces, times, and number of tests for local optimality.*

n	NFAC1 / n^2	TInit / n^2	TIM1 / n^2	LOP1 / n^2	LTP1 / n^2
50	.490	.000500	.00311	.554	.110
100	.495	.000518	.00314	.597	.120
150	.497	.000513	.00326	.602	.120
200	.498	.000478	.00291	.604	.120
250	.498	.000489	.00295	.611	.122

Delaunay triangulation has the same complexity as the number of faces. For problem $P11_n$, it can be seen from Table 9 that the number of faces in the Delaunay triangulation is $O(n)$, as expected, and the time complexity of TRSPH1 is approximately $O(n^2)$. We believe that the higher time complexity for this problem is due to the fact that the initial triangulation T_I has many long tetrahedrons with small solid angles and the Delaunay triangulation does not have any of these tetrahedrons. Note that for $P11_n$, the theoretical asymptotic complexities for NBFC1 and TInit are $O(n^{2/3})$ and $O(n^{5/3})$, respectively. For problem $P12_n$, it can be seen from Table 12 that the number of faces in the Delaunay triangulation is $O(n^2)$, as expected, and the time complexity of TRSPH1 is approximately $O(n^2)$.

For the three types of problems, the time complexities of algorithms TRSPH2 and TRSPH3 are the same as that for TRSPH1, as can be seen from calculations similar to those in Tables 6, 9, and 12. Therefore our experiment has shown that there is no advantage to using TRSPH2 or TRSPH3 over TRSPH1, since all three algorithms require approximately the same amount of CPU time and TRSPH2 and TRSPH3 have each failed to construct a Delaunay triangulation for at least one test problem.

Due to the amount of CPU time required, we chose to use a maximum value of $n = 500$ in our experiment. This is large enough to determine the complexity trends given in Tables 6, 9, and 12. We have run TRSPH1 for a few problems with up to $n = 5{,}000$ vertices, and have not found a counterexample to Conjecture 2. The complexities for these larger problems are similar to those given in the above tables. The CPU time required for $P1_{5,000}$ is approximately 73 minutes.

We now describe the experimental results from running the 60 test problems for algorithms TRMMSA1, TRMMSA2, and TRMMSA3. Table 13 contains the minimum solid angles in the triangulations of 30 of the 60 problems (to reduce the table size, we show the results for 20 of the more interesting random problems in which the entries have greater variance). MSAi is the minimum solid angle in the triangulation produced by TRMMSAi (MSA3 is not shown since MSA3 = MSA1 for all the problems). MSADe1 is the minimum solid angle in the Delaunay triangulation produced by TRSPH1, MSA3a is the minimum solid angle in the triangulation produced by a modified version of TRMMSA3 in which the starting triangulation is the Delaunay triangulation produced by TRSPH1 instead of the initial triangulation T_I, so MSA3a $\geqq$ MSADe1 is always satisfied.

The comparison of the MSA values for the 50 random problems is summarized as follows. MSA1 > MSA2 for 27 problems, MSA1 < MSA2 for 7 problems, and MSA1 = MSA2 for 16 problems. MSADe1 > MSA12 for 25 problems, MSADe1 < MSA12 for 10 problems, and MSADe1 = MSA12 for 15 problems, where MSA12 = max (MSA1, MSA2). MSA3a > MSADe1 for 9 of the 50 problems, and MSA12 > MSA3a for 8 of the 50 problems. Therefore none of the algorithms always produces a SA-globally optimal triangulation (we do not know whether a triangulation with the highest MSA value is SA-globally optimal, but we do know that the triangulations with smaller MSA values are not SA-globally optimal), and from Table 13, it can be seen that the MSA value for a SA-locally optimal triangulation can be much smaller than that for a SA-globally optimal triangulation. Since MSA3a $\geqq$ MSA12 for 52 of the 60 problems, it seems that the best approach to constructing an SA-locally optimal triangulation with a "good" MSA value is to improve the Delaunay triangulation by applying the local transformation procedure to nonlocally optimal faces (with respect to the max-min solid angle criterion).

In Tables 14, 15, and 16 we present the counts and times for algorithm TRMMSA1 (the results for TRMMSA2 and TRMMSA3 are similar). From comparison with Tables

TABLE 13
Minimum solid angles in triangulations.

Problem	MSA1	MSA2	MSADe1	MSA3a
$P1_{100}$	.0001568	.0002963	.0012346	.0032361
$P4_{100}$	.0006088	.0001021	.0004123	.0004123
$P5_{100}$	.0001851	.0002813	.0000893	.0000893
$P8_{100}$	.0000115	.0000115	.0001843	.0003123
$P2_{200}$	.0002078	.0001334	.0001774	.0001774
$P4_{200}$	.0001728	.0001023	.0004117	.0004117
$P5_{200}$	.0001794	.0000120	.0000043	.0000043
$P8_{200}$	.0000136	.0000195	.0000414	.0000414
$P4_{300}$	.0000922	.0000071	.0000922	.0003906
$P6_{300}$	.0000913	.0000913	.0000515	.0001215
$P7_{300}$	.0000456	.0000036	.0000619	.0000824
$P8_{300}$	.0000048	.0000071	.0000338	.0000338
$P3_{400}$	.0000742	.0000029	.0001818	.0001818
$P5_{400}$	.0000193	.0000008	.0000205	.0000205
$P7_{400}$	.0000110	.0000096	.0000102	.0000153
$P10_{400}$	.0000008	.0000008	.0000507	.0001061
$P1_{500}$	.0000413	.0000152	.0000762	.0001843
$P4_{500}$	.0000655	.0000213	.0000299	.0000299
$P6_{500}$	.0000097	.0000419	.0001641	.0001641
$P9_{500}$	.0000223	.0000538	.0000858	.0000858
$P11_{100}$	.0050024	.0050024	.1837619	.2617994
$P11_{200}$	.0013148	.0013148	.1837619	.2617994
$P11_{300}$	.0005906	.0005906	.1837619	.2617994
$P11_{400}$	.0004443	.0004443	.1837619	.2617994
$P11_{500}$	.0003192	.0003192	.1837619	.2617994
$P12_{50}$	.0031036	.0031036	.0193708	.0193708
$P12_{100}$	.0003834	.0003834	.0098490	.0098940
$P12_{150}$	.0001134	.0001134	.0066005	.0066005
$P12_{200}$	.0000478	.0000478	.0049631	.0049631
$P12_{250}$	.0000245	.0000245	.0039765	.0039765

TABLE 14
Average of problems $P1_n$ *to* $P10_n$, *algorithm* TRMMSA1: *number of tetrahedrons, faces, tests for transformability, and* CPU *times.*

n	NTET1	NFAC1	TIM1	LOP1	LTP1	NTF1
100	460.2	951.6	12.94	530	207	866
200	1005.0	2050.3	27.84	1,159	406	1,800
300	1571.8	3192.7	45.74	1,836	670	2,920
400	2145.4	4351.4	65.63	2,481	907	4,005
500	2701.9	5461.3	82.08	3,158	1,119	5,011

4 to 12, it can be seen that for TRMMSA1, a smaller percentage of the tests for transformability (local optimality) result in an application of the local transformation procedure and much larger percentage result in a nontransformable face. (This relatively large number of nontransformable faces explains why MSAi can sometimes be much smaller than MSADe1.) Also, for TRMMSA1, there are much fewer tests for transformability in the random problems and problem $P11_n$ than there are for TRSPHi, and about the same number of tests for $P12_n$. Hence, the CPU time required for

TABLE 15
Problem $P11_n$, *algorithm* TRMMSA1: *number of tetrahedrons, faces, tests for transformability, and* CPU *times.*

n	NTET1	NFAC1	TIM1	LOP1	LTP1	NTF1
100	267	614	20.70	707	244	554
200	672	1,488	37.33	1,125	319	804
300	1,080	2,362	69.43	1,929	555	1,308
400	1,512	3,278	101.87	2,599	611	1,565
500	1,938	4,182	154.95	3,854	845	2,215

TABLE 16
Problem $P12_n$, *algorithm* TRMMSA1: *number of tetrahedrons, faces, tests for transformability, and* CPU *times.*

n	NTET1	NFAC1	TIM1	LOP1	LTP1	NTF1
50	532	1,089	20.47	761	199	648
100	2,122	4,294	86.60	3,079	857	2,755
150	4,760	9,595	188.80	6,738	1,891	6,379
200	8,457	17,014	335.58	11,810	3,324	11,522
250	13,201	26,527	528.35	18,644	5,285	18,036

TRMMSA1 is smaller than that for TRSPHi for the random problems and $P11_n$, and is greater for $P12_n$ since a test involving computation of solid angles takes more time than a test for determining whether a point is in a sphere. From calculations similar to Tables 6, 9, and 12, the number of faces in the triangulation produced by TRMMSA1 is $O(n)$ for the random problems, and the time complexity of TRMMSA1 appears to be approximately $O(n^{1.1})$ for the random problems, $O(n^{1.5})$ for $P11_n$, and $O(n^2)$ for $P12_n$.

Finally, to compare the CPU times for the three different types of problems and the two different types of local optimality criteria, the graphs of n versus TIM1 are given in Fig. 7.

8. Concluding remarks. We have presented an algorithm called TRSPH1 for constructing a triangulation of a set of n three-dimensional points that is pseudo-locally optimal with respect to the sphere criterion. Experimental results show that TRSPH1 always constructs a Delaunay triangulation (so far), although variations of TRSPH1 can sometimes fail to construct a Delaunay triangulation. The Delaunay triangulation of n three-dimensional random points (from the uniform distribution) is shown experimentally to contain $O(n)$ tetrahedrons and faces, and the empirical time complexity of TRSPH1 is $O(n^{4/3})$ for sets of random points, which compares well with existing algorithms for constructing a three-dimensional Delaunay triangulation (Bowyer [2], Watson [14], Avis and Bhattacharya [1]). The Delaunay triangulation of n three-dimensional points contains $O(n^2)$ tetrahedrons and faces in the worst case, and we have presented two families of problems for which TRSPH1 requires an empirical time complexity of $O(n^2)$. We have not yet found any problems for which TRSPH1 requires more than $O(n^2)$ time, so $O(n^2)$ appears to be the worst-case time complexity. This is better than the worst-case time complexity of $O(n^3)$ for Avis and Bhattacharya's algorithm (Bowyer and Watson do not discuss the worst case in their papers).

An open problem is to prove that TRSPH1 always constructs a Delaunay triangulation (Conjecture 2), or to find an example for which TRSPH1 fails to construct a

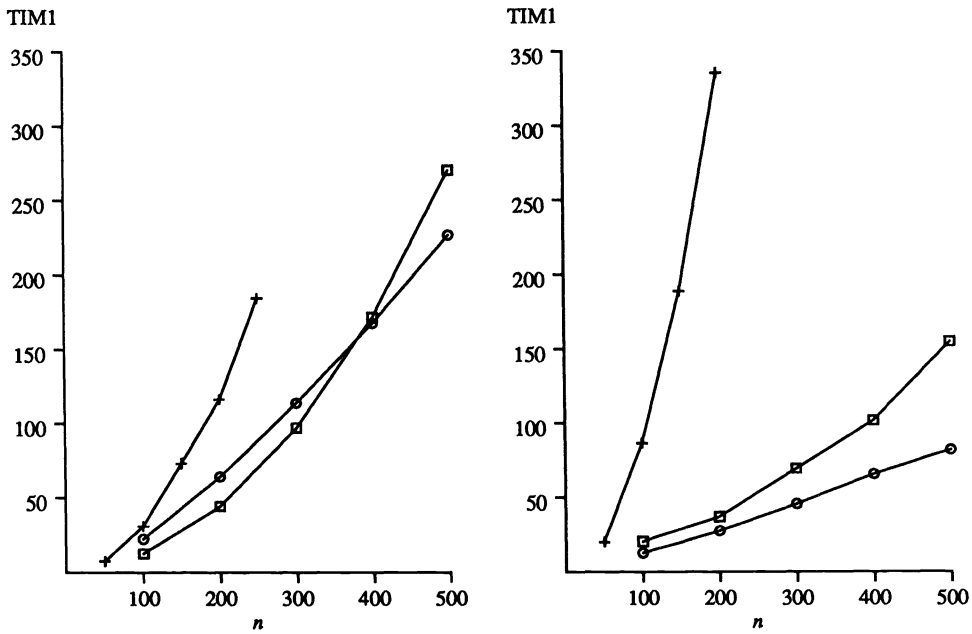


FIG. 7. *Graphs of n versus* TIM1 *(in seconds). Left graph is for* TRSPH1; *right graph is for* TRMMSA1. ○: *average of* $P1_n$ *to* $P10_n$; □: $P11_n$; +: $P12_n$.

Delaunay triangulation. If the latter case occurs, then open problems are to determine sufficient conditions for TRSPH1 to be successful and to determine whether the local transformation procedure can be used in a modified algorithm that always produces a Delaunay triangulation (this may involve applying the local transformation procedure to locally optimal faces, and may be related to Conjecture 1).

We believe that the approach of using the local transformation procedure to improve a triangulation, as in the TRSPH*i* algorithms, is especially useful if an initial triangulation that is nearly Delaunay can be constructed quickly, say in linear time. This may be possible in an application such as finite-element mesh generation in which the vertices as well as the tetrahedrons are generated. Information about the location of the generated vertices can be used to construct a "good" initial triangulation, and then it may be possible to improve this triangulation to a Delaunay triangulation in linear time. This is done in two dimensions in Joe [7] and is a subject of further research in three dimensions.

We have also introduced the max-min solid angle criterion in this paper. This criterion does not seem to have been used before, although Nguyen [11] tries to avoid small solid angles in his three-dimensional triangulation algorithm. Experimental results show that, unlike the case of the sphere criterion, a SA-locally optimal triangulation may be far from being SA-globally optimal due to many nontransformable faces. An approach to constructing a SA-locally optimal triangulation with a "satisfactory" minimum solid angle is to improve the Delaunay triangulation by applying the local transformation procedure to nonlocally optimal faces. A further research problem is to derive more theoretical results for SA-globally optimal triangulations such as determining whether the local transformation procedure can be used in their construction or whether the problem of constructing a SA-globally optimal triangulation is NP-hard.

Finally, we discuss the possibility of extending the local transformation approach to triangulations of higher dimensions. Lawson [10] has recently proved that an arbitrary dimensional version of Theorem 1 is true, shown that a set of $k+2$ points in k-dimensional space may be triangulated in at most two different ways, and characterized the different configurations of $k+2$ points from the point of view of their possible triangulations (the number of configurations increases as k increases). Hence the local transformation procedure can be defined in any dimension, but the number of different cases increases as the dimension k increases. However, we suspect that the use of the local transformation procedure to construct k-dimensional Delaunay triangulations for $k \geqq 4$ will be more difficult than the three-dimensional case (and maybe even not possible), since the configuration containing a facet may have four or more simplices so more facets are likely to be nonlocally optimal and nontransformable.

Acknowledgment. The author would like to thank the referees for their helpful comments.

Note added in proof. The author has now proved that Conjecture 2 is true.

REFERENCES

[1] D. AVIS AND B. K. BHATTACHARYA, *Algorithms for computing d-dimensional Voronoi diagrams and their duals*, in Advances in Computing Research 1, F. P. Preparata, ed., JAI Press, 1983, pp. 159–180.

[2] A. BOWYER, *Computing Dirichlet tessellations*, Comput. J., 24 (1981), pp. 162–166.

[3] J. C. CAVENDISH, D. A. FIELD, AND W. H. FREY, *An approach to automatic three-dimensional finite element mesh generation*, Internat. J. Numer. Methods Engrg., 21 (1985), pp. 329–347.

[4] D. A. FIELD, *Implementing Watson's algorithm in three dimensions*, Proc. 2nd ACM Symposium on Computational Geometry, 1986, pp. 246–259.

[5] H. FUHRING, *The application of node-element rules for forecasting problems in the generation of finite element meshes*, Internat. J. Numer. Methods Engrg., 9 (1975), pp. 617–629.

[6] P. C. GASSON, *Geometry of Spatial Forms*, Ellis Horwood, Chichester, 1983.

[7] B. JOE, *Delaunay triangular meshes in convex polygons*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 514–539.

[8] C. L. LAWSON, *Transforming triangulations*, Discrete Math., 3 (1972), pp. 365–372.

[9] ———, *Software for C^1 surface interpolation*, in Mathematical Software III, J. R. Rice, ed., Academic Press, New York, 1977, pp. 161–194.

[10] ———, *Properties of n-dimensional triangulations*, Computer Aided Geom. Des., 3 (1986), pp. 231–246.

[11] V. P. NGUYEN, *Automatic mesh generation with tetrahedron elements*, Internat. J. Numer. Methods Engrg., 18 (1982), pp. 273–289.

[12] C. S. PETERSEN, B. R. PIPER, AND A. J. WORSEY, *Adaptive contouring of a trivariate interpolant*, in Geometric Modeling: Algorithms and New Trends, G. E. Farin, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 385–395.

[13] T. A. STANDISH, *Data Structure Techniques*, Addison-Wesley, Reading, MA, 1980.

[14] D. F. WATSON, *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes*, Comput. J., 24 (1981), pp. 167–172.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 742-755, July 1989
© 1989 Society for Industrial and Applied Mathematics
008

A CONSERVATIVE ADAPTIVE METHOD FOR FLAME PROPAGATION*

B. LARROUTUROU†

Abstract. An adaptive one-dimensional finite-difference method aimed at the solution of flame-propagation problems is presented. This method uses a classical finite-difference approximation on a moving adaptive mesh. The discrete conservation of the variables is imposed for both the node displacements that occur at each timestep and for the interpolations that are performed at some time levels, when a static grid adaption is done. In particular, an interpolation procedure is presented that is conservative and also has the advantages of preserving the positivity and monotonicity of the interpolated variables.

Key words. adaptive gridding, conservation property, interpolation, flame propagation

AMS(MOS) subject classification. 80A25

1. Introduction. The numerical investigation of one-dimensional flame propagation in a gaseous mixture presents two main difficulties that have received increasing attention in the past 10 years. These two major problems follow from the existence of disparate length and time scales in the physical phenomenon, and have led to development and application to combustion of the following two basic tools: adaptive grids and stiff ODE (ordinary differential equation) solvers.

The first difficulty is related to the presence of scales of different lengths: in particular, the flame propagation exhibits a very narrow zone of sharp gradients, the flame front, which separates two regions where all the variables are almost constant. The size of this zone is very small compared to the dimensions of the vessel inside which the combustion occurs. Moreover, because of the strongly nonlinear dependence of the chemical reaction rates with respect to the temperature, the chemical reactions actually take place in a very thin layer within the flame front: in most of the combustion phenomena, the thickness of this reaction zone is indeed one order of magnitude smaller than the flame thickness itself. Furthermore, in contrast with a purely hydrodynamical shock wave, whose speed is solely influenced by the flow conditions outside the shock, the inner structure of the flame front and of the reaction zone determines the flame speed and the overall burning rate of the mixture. A proper numerical simulation therefore requires an accurate solution (and hence a dense computational grid) in these thin regions of high activity. However, efficiency dictates that the mesh should be sparse where the variables are almost constant. The use of adaptive grids is therefore an essential tool for the numerical investigation of flame-propagation problems, and the development and application of several adaptive gridding techniques to combustion have been very successful in recent years (see among many other works [6], [9], [12], [13], [16], [22], [26], [27]).

The other main difficulty inherent in the numerical treatment of one-dimensional flame-propagation problems follows from the existence of disparate time scales and is referred to as the temporal stiffness of the corresponding system of governing equations (see, e.g., [11]). If a single reaction occurs in the mixture, the timestep used in a numerical simulation may not exceed the order of magnitude of the chemical time

* Received by the editors October 30, 1987; accepted for publication (in revised form) October 19, 1988. This work was supported in part by D.R.E.T. (Direction des Recherches, Etudes et Techniques) under contract 84-209.

† Institut de Recherche en Informatique et Automatique, Sophia-Antipolis, 06560 Valbonne, France.

scale of this reaction, for reasons of accuracy (this is true even if implicit unconditionally stable time-integration schemes are employed, because the chemical transformation of the species need be computed accurately; see § 3.3 below). When a large number of chemical species and reactions are taken into account, a very strong additional stiffness may appear because of the existence of disparate characteristic chemical times (the ratio between the time scales of the slowest reactions and the time scales of the fastest reactions may well be of the order of 10^{-6} for certain problems). This makes it necessary to use implicit time-integration schemes for the solution of these chemically complex problems (see, e.g., [11]).

In this paper we essentially address the first of these difficulties, in the sense that we only consider the case of a simple or a moderately complex chemistry (such as that of the ozone flame). Our method uses a moving adaptive computational grid, whose nodes move at each time t with the same time-dependent velocity $V(t)$, which is chosen so that a certain physical quantity, such as the total energy contained in the computational domain, remains constant during the calculation. Moreover, at some time levels, the grid is changed to a better-adapted mesh; the variables are then interpolated from the old grid to the new one. We show that this interpolation can be made in a conservative way, while preserving the positivity and the monotonicity of the variables.

At the end of the paper, we present the results of several numerical experiments that show the ability of the method to compute unsteady flames, and in particular show how the use of a conservative discrete scheme and of a conservative interpolation improves the convergence of the time-dependent solution to a steady state. We also discuss the question of explicit or implicit time-integration schemes for the considered class of moderately complex flames.

2. The numerical method.

2.1. Governing equations. Let us first briefly recall the equations of the problem under consideration. For the sake of simplicity, we restrict ourselves to the case of a flame propagating in a gaseous mixture in which a single one-step chemical reaction $A \to B$ takes place; the case of a more complex chemistry will be considered later. Using a Lagrangian space coordinate and normalized variables T and Y for the temperature and the mass fraction of the reactant, we can write the governing equations of the flame as follows:

$$T_t = T_{xx} + \Omega(T, Y), \qquad Y_t = \frac{Y_{xx}}{\mathrm{Le}} - \Omega(T, Y), \tag{1}$$

where the normalized reaction rate Ω is given by

$$\Omega(T, Y) = \frac{\beta^2}{2\mathrm{Le}}\, Y \exp\left(-\frac{\beta(1-T)}{1-\alpha(1-T)}\right). \tag{2}$$

The positive constants Le, β, and α denote the Lewis number of the reactant, the reduced activation energy of the reaction, and a nondimensional heat-release parameter, respectively. Equations (1) are associated to initial data:

$$T(x, 0) = T_0(x), \qquad Y(x, 0) = Y_0(x), \tag{3}$$

and to the following boundary conditions:

$$\begin{aligned} &T(-\infty, t) = 0, \quad Y(-\infty, t) = 1 \quad \text{(fresh mixture)}, \\ &T(+\infty, t) = 1, \quad Y(+\infty, t) = 0 \quad \text{(burnt gases)}. \end{aligned} \tag{4}$$

2.2. The numerical problem. As explained in the Introduction, a nonuniform adaptive grid is needed for a low-cost simulation of the flame propagation because of the presence of sharp gradients in the flame front. Moreover, this region of high spatial activity continually moves towards the fresh gases (i.e., towards negative x) during the calculation. For these reasons, (1)-(2) are solved in a *moving* bounded domain $D=[X_0(t), X_N(t)]$, using an adaptive moving grid with a constant number of nodes.

The meshpoint displacement is determined in a very simple way: at each time t, all nodes move with the same velocity, $V(t)$. In other words, the computational grid continually moves as a rigid body in order to follow the flame front during its propagation towards the fresh mixture. This amounts to observing the flame propagation in a non-Galilean reference frame moving with the velocity $V(t)$ with respect to the original fixed reference frame. In this moving frame, the governing equations (1) become

$$\begin{aligned} T_t &= T_{xx}+\Omega(T, Y)+V(t)T_x, \\ Y_t &= \frac{Y_{xx}}{\mathrm{Le}}-\Omega(T, Y)+V(t)Y_x. \end{aligned} \tag{5}$$

Our moving grid procedure therefore simply leads to solving on a *fixed* domain $[X_0, X_N]$ equations (5), in which additional convective terms $V(t)T_x$ and $V(t)Y_x$ appear. The problem to be solved numerically consists of (5), the initial data (3), and the following conditions at the boundaries $x=X_0$ and $x=X_N$:

$$T(X_0, t)=0, \qquad Y(X_0, t)=1, \tag{6a}$$

$$T_x(X_N, t)=Y_x(X_N, t)=0. \tag{6b}$$

We note that the conditions (4) at the cold and hot boundaries are transformed in a different way to give (6a) and (6b). In fact, it is valid to impose Dirichlet conditions for $x=X_0$, since the temperature and the composition of the fresh mixture in which the flame propagates are known. On the other hand, numerical experiments have shown that Neumann conditions are preferable at the boundary $x=X_N$. Of course, if this boundary is located far enough from the flame front, then Dirichlet conditions $T(X_N, t)=1$, $Y(X_N, t)=0$ can be used without appreciably altering the numerical solution.

2.3. The grid velocity and the numerical scheme. We still have to choose how the grid velocity $V(t)$ is evaluated at each timestep. Let us first recall that, from a physical point of view, the instantaneous average flame speed at time t is given by

$$V_0(t)=-\int_{\mathbb{R}} \Omega(T, Y)(t).$$

Moreover, if the parameters Le and β are chosen so that the steady planar flame is stable to one-dimensional perturbations (for instance, if $\mathrm{Le}=1$), the solution of the equations

$$\begin{aligned} T_t &= T_{xx}+\Omega(T, Y)+V_0(t)T_x, \\ Y_t &= \frac{Y_{xx}}{\mathrm{Le}}-\Omega(T, Y)+V_0(t)Y_x, \end{aligned}$$

(with (3)-(4)) is believed to converge for large values of t to a steady solution (this convergence has not been proved mathematically, but is conjectured on the basis of physical arguments and numerical observations; see [13], [19]).

We may therefore think of choosing the grid velocity $V(t)$ close to $V_0(t)$. The choice

$$V(t) = V_0'(t) = -\int_{X_0}^{X_N} \Omega(T, Y)(t) \tag{7}$$

was proposed in [12], [13]. In this study, however, we more specifically evaluate the mesh velocity in order to keep constant the thermal energy contained in the computational domain during the calculation

$$\int_{X_0}^{X_N} T(x, t)\, dx = \text{constant}. \tag{8}$$

Indeed, the equations (5) yield

$$\frac{d}{dt}\int_{X_0}^{X_N} T = T_x(X_N) - T_x(X_0) + V(t)[T(X_N) - T(X_0)] + \int_{X_0}^{X_N} \Omega(T, Y)(t), \tag{9}$$

and (8) is satisfied if we choose $V(t)$ according to the relation

$$V(t) = \frac{-\int_{X_0}^{X_N} \Omega(T, Y)(t) + T_x(X_0) - T_x(X_N)}{T(X_N) - T(X_0)}. \tag{10}$$

We note that the grid velocity $V(t)$ could also be evaluated to keep constant the mass of reactant $\int_{X_0}^{X_N} Y$ or the normalized total enthalpy $\int_{X_0}^{X_N} (T+Y)$ contained in the computational domain.

If the boundaries $x = X_0$ and $x = X_N$ are far enough on both sides of the flame front, the quantities $T_x(X_0)$, $T_x(X_N)$, $T(X_N) - T(X_0) - 1$ are very small, and the velocity $V(t)$ defined by (10) is very close to $V_0'(t)$. But taking these small terms into account in (10) allows us to rigorously impose the conservation property (8) to the numerical solution, as we now show.

Let $(X_i)_{0 \leqq i \leqq N}$ denote the meshpoints; using the boundary conditions (6), we discretize the temperature equations (5) with the following simple explicit scheme:

$$T_0^{n+1} = 0 \tag{11a}$$

for the first meshpoint, and

$$\begin{aligned} \frac{T_i^{n+1} - T_i^n}{\Delta t} &= 2\left(\frac{T_{i+1}^n - T_i^n}{X_{i+1} - X_i} - \frac{T_i^n - T_{i-1}^n}{X_i - X_{i-1}}\right) \cdot (X_{i+1} - X_{i-1})^{-1} \\ &\quad + V(t^n)\frac{T_{i+1}^n - T_{i-1}^n}{X_{i+1} - X_{i-1}} + \Omega(T_i^n, Y_i^n) \end{aligned} \tag{11b}$$

for the interior nodes $(1 \leqq i \leqq N-1)$. For the last node X_N, we may consider two different discrete equations. A first possibility is to adopt a finite-element point of view: setting $T_h = \sum_{i=1}^N T_i \phi_i$, where the $(\phi_i)_{1 \leqq i \leqq N}$ are the usual piecewise linear hat functions, and using a forward Euler mass-lumped approximation (see, e.g., [29]), we obtain the discrete equations by writing

$$\begin{aligned} &\frac{T_i^{n+1} - T_i^n}{\Delta t}\int_{X_0}^{X_N} \phi_i \\ &= -\int_{X_0}^{X_N} \left[\sum_{j=1}^N T_j^n (\phi_j)_x\right](\phi_i)_x + V(t^n)\int_{X_0}^{X_N} \left[\sum_{j=1}^N T_j^n (\phi_j)_x\right]\phi_i \\ &\quad + \Omega(T_i^n, Y_i^n)\int_{X_0}^{X_N} \phi_i, \end{aligned}$$

which is equivalent to (11b) for an interior node, and gives

$$\frac{T_N^{n+1}-T_N^n}{\Delta t}=-2\frac{T_N^n-T_{N-1}^n}{(X_N-X_{N-1})^2}+V(t^n)\frac{T_N^n-T_{N-1}^n}{X_N-X_{N-1}}+\Omega(T_N^n, Y_N^n) \tag{11c}$$

for the last node. On the other hand, using a finite-difference point of view, we could construct the discrete equation for T_N by adding a fictitious point X_{N+1}, assuming $X_{N+1}-X_N=X_N-X_{N-1}$ and $T_{N+1}=T_N$, and using (11b) for $i=N$, which gives

$$\frac{T_N^{n+1}-T_N^n}{\Delta t}=-\frac{T_N^n-T_{N-1}^n}{(X_N-X_{N-1})^2}+V(t^n)\frac{T_N^n-T_{N-1}^n}{2(X_N-X_{N-1})}+\Omega(T_N^n, Y_N^n). \tag{12}$$

We will now only consider the scheme (11a)-(11c), except in Remark 1 below. This scheme is second-order accurate on a uniform mesh (except at the last point X_N), and first-order accurate on a nonuniform grid. It is conservative, as shown by the following relation (where $\Delta x_i=\int_{X_0}^{X_N}\phi_i$, that is, $\Delta x_i=\frac{1}{2}(X_{i+1}-X_{i-1})$ for $1\leqq i\leqq N-1$, $\Delta x_0=\frac{1}{2}(X_1-X_0)$, $\Delta x_N=\frac{1}{2}(X_N-X_{N-1})$):

$$\begin{aligned}&\frac{\sum_{i=0}^N T_i^{n+1}\Delta x_i-\sum_{i=0}^N T_i^n\Delta x_i}{\Delta t}\\&=-\frac{T_1^n-T_0^n}{(X_1-X_0)}+V(t^n)\left(T_N^n-\frac{T_0^n+T_1^n}{2}\right)+\sum_{i=0}^N\Omega(T_i^n, Y_i^n)\Delta x_i.\end{aligned} \tag{13}$$

We can then evaluate the grid velocity as

$$V(t^n)=\frac{-\sum_{i=0}^N\Omega(T_i^n, Y_i^n)\Delta x_i+(T_1^n-T_0^n)\cdot(X_1-X_0)^{-1}}{T_N^n-(T_0^n+T_1^n)/2}. \tag{14}$$

The two last relations are the discrete analogues of (9) and (10). They imply that

$$\sum_{i=0}^N T_i^{n+1}\Delta x_i=\sum_{i=0}^N T_i^n\Delta x_i.$$

The conservation of the energy in the computational domain D is therefore satisfied by the solution of the discrete scheme (11)-(14): the discrete integral of the temperature is constant, up to roundoff errors.

Remark 1. The relation (13) also holds for the scheme (11a)-(11b), (12), if we set $\Delta x_N=X_N-X_{N-1}$.

This moving adaptive method is therefore very cheap, since only one velocity is to be evaluated at each timestep. Moreover, since the grid moves with a velocity close to the instantaneous flame speed, the flame front may be deformed but stays at the same place in the computational domain during the calculation. In particular, the sharp gradients located inside the flame zone remain in the fine-mesh region. This procedure therefore makes it possible to reduce the number of static adaptions performed during the calculation, and also to reduce the length of the computational domain. Last, in many cases, the additional convective term $V(t)T_x$ allows the solution of (11) to converge to a discrete steady state instead of converging towards a traveling wave solution. The investigation of the convergence of the time-dependent solution to steady state and the evaluation of the steady flame speed are then much easier.

It is of course possible to use more sophisticated dynamic rezone methods in which each node moves with its own velocity (see, for instance, [9] for an application of such a method to a flame-propagation problem). But the simple moving method (11)-(14) is altogether adequate and efficient for a free flame-propagation problem, as shown by the results at the end of the paper.

2.4. The conservative interpolation. Let us now consider the static grid adaptions performed at some time levels during the calculation in order to better adapt the spatial-node distribution in the interval $[X_0, X_N]$ to the variable profiles. This static rezone, realized at a fixed time, uses the current "old" grid $(\tilde{X}_i)$ and the "old" solution $(\tilde{T}_i, \tilde{Y}_i)$ to determine a new grid $(\hat{X}_i)$ and the new values $(\hat{T}_i, \hat{Y}_i)$ of the variables at the new nodes.

We do not present in complete detail how the new grid is computed. It is obtained by equidistributing a mesh function $\tilde{w}$, that is, by requiring

$$\int_{\hat{X}_i}^{\hat{X}_{i+1}} \tilde{w} = \text{constant} \quad \forall i, \tag{15}$$

the locations of the first and last nodes being a priori given by $\hat{X}_0 = \tilde{X}_0$, $\hat{X}_N = \tilde{X}_N$. The mesh function $\tilde{w}$ is essentially based on the leading order terms of the spatial truncation error, evaluated on the old mesh (see [12], [13] for more details). The new mesh satisfying (15) is determined using an inverse interpolation method (that is, one seeks for $\hat{X}_i$ satisfying $\int_{\hat{X}_0}^{\hat{X}_i} \tilde{w} = i/N \int_{\hat{X}_0}^{\hat{X}_N} \tilde{w}$; see, for instance, [10] for more details).

Once the new grid is known, it remains to define the values of the variables at the new points. In our previous works (see [12], [13]), as in many studies of other authors (see, e.g., [6], [26], [27]), this is simply done by linearly interpolating the variables on each mesh of the old grid. In addition to its simplicity, this procedure has the advantage of preserving the local positivity and monotonicity of the solution.

We now show that a conservative interpolation can be used instead of the linear one, and that this conservative interpolation also preserves the positivity and monotonicity of the solution.

In this conservative interpolation, the temperature is interpolated to conserve the thermal energy $E = \int T$. Let us first define the old cells $\tilde{C}_i = [\tilde{X}_{i-1/2}, \tilde{X}_{i+1/2}]$, (with $\tilde{X}_{i+1/2} = (\tilde{X}_{i+1} + \tilde{X}_i)/2$ for $0 \leqq i \leqq N-1$, and $\tilde{X}_{-1/2} = \tilde{X}_0$, $\tilde{X}_{N+1/2} = \tilde{X}_N$), and the new cells $\hat{C}_j = [\hat{X}_{j-1/2}, \hat{X}_{j+1/2}]$. Our objective is to apportion the energy $\tilde{E}$, whose "density" (energy per unit length) $\tilde{T}$ is considered to be constant in each cell of the old grid, into the cells of the new grid. For a particular new cell $\hat{C}_j$ and any old cell $\tilde{C}_i$, let $A_i = \text{Length}\,(\hat{C}_j \cap \tilde{C}_i)$ be the overlap length of $\hat{C}_j$ and $\tilde{C}_i$. The contribution of the overlap length $\hat{C}_j \cap \tilde{C}_i$ to the energy contained in $\hat{C}_j$ is $\Delta E_i = A_i T_i$; the final value of the energy $\hat{E}_j$ in the cell $\hat{C}_j$ is $\sum_{i=0}^{N} \Delta E_i$, which gives the new temperature value $\hat{T}_j = \hat{E}_j / \text{Length}\,(\hat{C}_j)$. Clearly, no energy is created or destroyed in this procedure, which is therefore conservative.

This interpolation is therefore the one-dimensional analogue of the two-dimensional conservative rezoning algorithms proposed in [4] and [23]. But its implementation is much simpler: let $\tilde{T}(x)$ be the piecewise constant function defined on the interval $[X_0, X_N]$ by

$$\tilde{T}(x) = \tilde{T}_i \quad \text{if } x \in \tilde{C}_i, \tag{16}$$

and let

$$\tilde{E}(x) = \int_{X_0}^{x} \tilde{T}(s)\, ds \tag{17}$$

($\tilde{E}$ is a continuous piecewise linear function on $[X_0, X_N]$). Then the new temperature value in the cell $\hat{C}_j = [\hat{X}_{j-1/2}, \hat{X}_{j+1/2}]$ is given by

$$\hat{T}_j = \frac{\tilde{E}(\hat{X}_{j+1/2}) - \tilde{E}(\hat{X}_{j-1/2})}{\hat{X}_{j+1/2} - \hat{X}_{j-1/2}}. \tag{18}$$

This interpolation procedure, based on very simple ideas, satisfies the following properties.

LEMMA 2. (a) *Assume that the old temperature values* $\tilde{T}_i$ *satisfy the inequalities*

$$A \leqq \tilde{T}_i \leqq B \quad \text{for } 0 \leqq i \leqq N. \tag{19}$$

Then, the new temperature values $\hat{T}_i$ *also satisfy the same inequalities*

$$A \leqq \hat{T}_j \leqq B \quad \text{for } 0 \leqq j \leqq N. \tag{20}$$

(b) *Assume that the sequence* $(\tilde{T}_i)$ *of the old temperature values is monotone increasing. Then the sequence of the new temperature values* $(\hat{T}_j)$ *is also monotone increasing.*

Proof. These properties are easily checked. Inequalities (20) obviously follow from (17)-(18) when (19) holds. On the other hand, if the sequence $(\tilde{T}_i)$ is increasing, the function $\tilde{E}$ is convex and therefore satisfies

$$\frac{\tilde{E}(\hat{X}_{j+1/2}) - \tilde{E}(\hat{X}_{j-1/2})}{\hat{X}_{j+1/2} - \hat{X}_{j-1/2}} \leqq \frac{\tilde{E}(\hat{X}_{j+3/2}) - \tilde{E}(\hat{X}_{j+1/2})}{\hat{X}_{j+3/2} - \hat{X}_{j+1/2}},$$

whence $\hat{T}_j \leqq \hat{T}_{j+1}$. The analogous property holds for a decreasing sequence $(\tilde{T}_i)$. □

Remark 3. We note that the properties stated in Lemma 2 are local; for instance, if the old temperature is positive in all old cells $\tilde{C}_i$ having a nonzero overlap length with the new cell $\hat{C}_j$, then $\hat{T}_j \geqq 0$.

Remark 4. Let us assume that the old temperature sequence $(\tilde{T}_i)$ is strictly increasing: $\tilde{T}_i < \tilde{T}_{i+1}$. If a linear interpolation is used, the new temperature is also strictly increasing. This is no longer the case with the conservative interpolation (18): we only have $\hat{T}_j \leqq \hat{T}_{j+1}$. This loss of strict monotonicity could be seen as a drawback of the conservative interpolation. But, in fact, it only appears if two adjacent new cells $\hat{C}_j$ and $\hat{C}_{j+1}$ are included in the same old cell $\tilde{C}_i$. In some sense, this loss of strict monotonicity therefore reveals that the interpolation tries to get too much information from a single cell of the old grid.

Thus, the conservative interpolation has the same advantages as the linear interpolation, i.e., it preserves the positivity and monotonicity of the solution. Moreover, its implementation defined by (18) is very simple and cheap. It should be observed however that, in the limit of very fine grids $\Delta x_i \to 0$, this interpolation is less accurate than the linear interpolation. Nevertheless, we have found it preferable in all our calculations to use the conservative interpolation, essentially because of its additional advantage of conserving the energy. Moreover, the same procedure can be used to interpolate the mass fraction: the static rezoning then conserves the mass of reactant and the total enthalpy.

If the conservative scheme (11)-(14) is used at each timestep, and if the conservative interpolation (18) is employed at each static rezone, *the total energy* $E = E(X_N) = \sum_{i=0}^{N} T_i \Delta x_i$ *contained in the computational domain is exactly conserved (to machine accuracy) during the calculation.* This will be shown in the next section to greatly improve the convergence of the time-dependent solution to a steady state.

Remark 5. An interesting way (inspired by [8]) of considering the interpolation (16)-(18) is the following. Using the pointwise values $\tilde{T}_i$, we first define a function $\tilde{T}(x)$ for all x in the computational domain by (16); we then evaluate a primitive $\tilde{E}$ of $\tilde{T}$ by (17), and last we differentiate $\tilde{E}$ to recover pointwise values $\hat{T}_j$ at the new nodes (the right-hand side of (18) appears indeed to be a classical discrete expression for the first derivative of $\tilde{E}$).

We can now see why the interpolation (16)-(18) is only first-order accurate in the limit of a very fine grid $\Delta x_i \to 0$ (whereas the usual linear interpolation is second-order

accurate): this order of accuracy is due to the piecewise constant definition of $\tilde{T}$, and also to the fact that the right-hand side of (18) is only a first-order differentiation approximation of $\tilde{E}_x(\hat{X}_j)$ since $\hat{X}_j/(\hat{X}_{j+1/2}+\hat{X}_{j-1/2})/2$, that is, since the new nodes $\hat{X}_j$ are not located at the center of the new cells $\hat{C}_j$.

The interpolation procedure (16)-(18) can therefore be made second-order accurate if we decide to use those points $(\hat{X}_j)$ whose adaptive distribution is determined by the static rezone algorithm for the cell edges and not for the meshpoints, and if we locate the new gridpoints at $\hat{Z}_j=(\hat{X}_j+\hat{X}_{j+1})/2(=\hat{X}_{j+1/2})$. Then we use for $\tilde{T}$ the piecewise linear interpolant $\tilde{T}=T_h=\sum_{i=1}^{N} T_i\phi_i$, we define $\tilde{E}$ as in (17), and we evaluate the new temperature $\hat{T}_j$ at the node $\hat{Z}_j$ by

$$\hat{T}_j=\frac{\tilde{E}(\hat{X}_{j+1})-\tilde{E}(\hat{X}_j)}{\hat{X}_{j+1}-\hat{X}_j}.$$

It is easy to check that this interpolation is second-order accurate, conservative, and preserves the positivity and the monotonicity (and even the strict monotonicity) of the solution, while remaining very simple to implement. We note in particular that, in contrast with the two-dimensional interpolations of [5], [24], this second-order positivity- and monotonicity-preserving interpolation does not involve slope limiters. Using the same ideas, we could also define higher-order accurate conservative interpolations; but, unless special care is taken (see [3]), these procedures will not preserve positivity and monotonicity.

From a practical point of view, we have seen very little difference in our calculations between the results obtained with the interpolation (16)-(18) and those obtained with the second-order accurate procedure we have just described, probably because of the small mesh spacing inside the flame zone. For this reason, we have used the first-order interpolation (16)-(18) in the computations whose results are now reported.

Remark 6. We have not yet said when this static rezone procedure is used during the calculation. In fact, the first static adaptation is performed after the initialization, before the first timestep. Then, once a new grid has been computed at time t_0, another static rezone is realized only when the solution has been sufficiently changed since the last adaptation, i.e., when some norm of $U(t)-U(t_0)$ is greater than a given quantity (where U denotes the vector whose components are the values of the variables at the nodes). It can then be noted that the number of static rezones performed per unit time decreases to zero when the numerical solution converges to a steady state.

3. Numerical results.

3.1. Flames with one-step chemistry. In this section we briefly present some numerical results obtained with the adaptive conservative numerical method described in the preceding sections for the six test problems proposed in [19]; that is, we consider the solution of problem (1)-(4) for different values of the parameters Le and β. In the conservative scheme (11)-(14), the timestep Δt is evaluated at each time level by imposing that the numerical solution satisfy the inequalities $0\leqq T_i^n$ and $0\leqq Y_i^n\leqq 1$ for all i and n (the analogous inequalities $0\leqq T(x,t)$ and $0\leqq Y(x,t)\leqq 1$ hold for the solution of the continuous problem (1)-(4); see [14]).

As already mentioned, the main advantage of the present method compared with the method developed in [12] and [13] is that the convergence of the discrete solution to a steady state is considerably improved. Defining the residual R^n as

$$R^n=\max_{0\leqq i\leqq N}\left[\frac{|T_i^{n+1}-T_i^n|}{\Delta t}+\frac{|Y_i^{n+1}-Y_i^n|}{\Delta t}\right],$$

we can obtain values of R^n less than 10^{-6} for all stable test problems, and even $R^n < 10^{-12}$ when Le = 1, whereas only residuals of the order of 10^{-2} or 10^{-3} could be obtained in [12] and [13] with the grid velocity $V_0'(t)$ defined by (7). The method therefore allows us to actually obtain a *steady-state discrete solution.* Moreover, since the adaptive mesh is very fine in the reaction zone, as shown by Figs. 1 and 2, we may think that the steady-state flame speeds evaluated with the present method are particularly accurate.

Table 1 shows the steady-state flame velocities V_f computed with 100 adaptive meshpoints distributed in an interval of length $X_N - X_0 = 30$, for the different test problems. The steady-state temperature, mass fraction, and reaction rate profiles for Le = 0.5, $\beta = 20$ are shown in Figs. 1 and 2. Figure 3 shows the quasiperiodic instability observed in the case Le = 2, $\beta = 20$; the pulsations of the flame speed are also satisfactorily computed.

Last, we emphasize that the adaptive procedure used in these calculations is very cheap. Only one velocity is to be evaluated at each timestep in the dynamic rezone algorithm, and a fairly small number of static adaptions is adequate, since the grid

TABLE 1
Flame speeds and residuals.

Test problem		Results	
Le	β	V_f	R^n
2.0	20	Instability	
2.0	10	0.850	10^{-8}
1.0	20	0.954	10^{-8}
1.0	10	0.918	10^{-8}
0.5	20	0.978	10^{-6}
0.5	10	0.959	10^{-6}

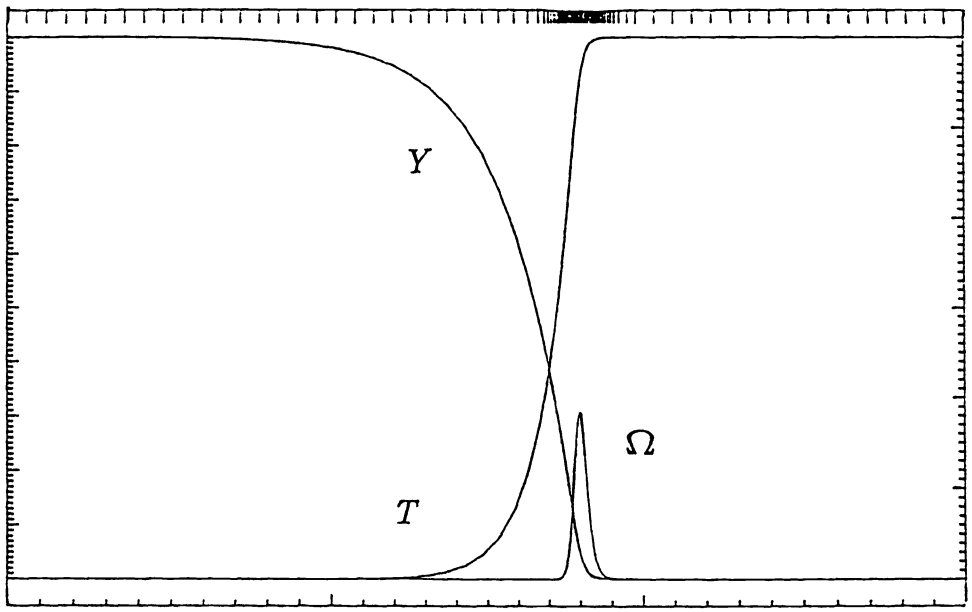


FIG. 1. *Steady-state temperature, mass fraction, and reaction rate profiles for* Le = 0.5, $\beta = 20$.

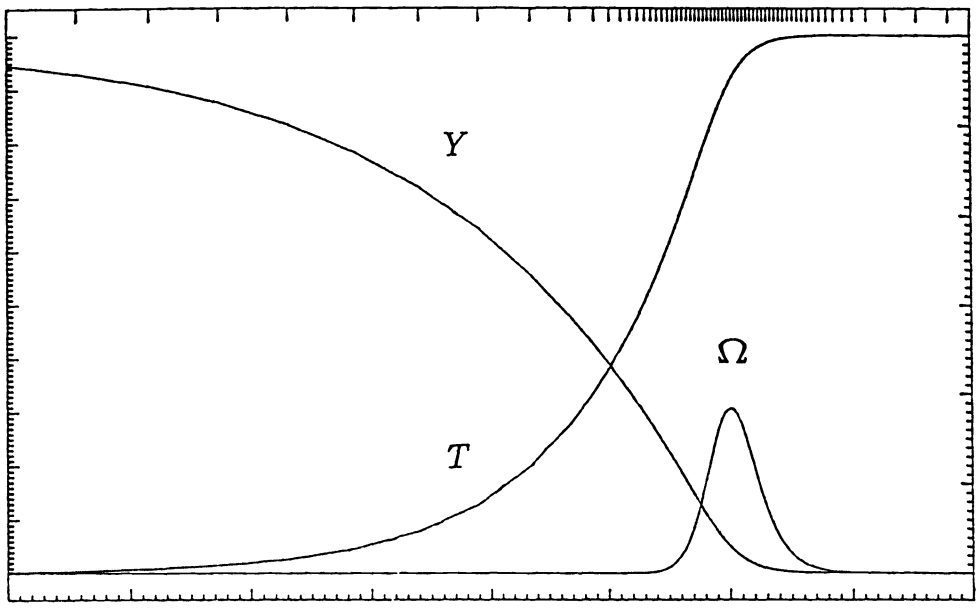


FIG. 2. *Blowup of the steady-state temperature, mass fraction, and reaction rate profiles in the neighborhood of the flame front for* $\mathrm{Le} = 0.5$, $\beta = 20$.

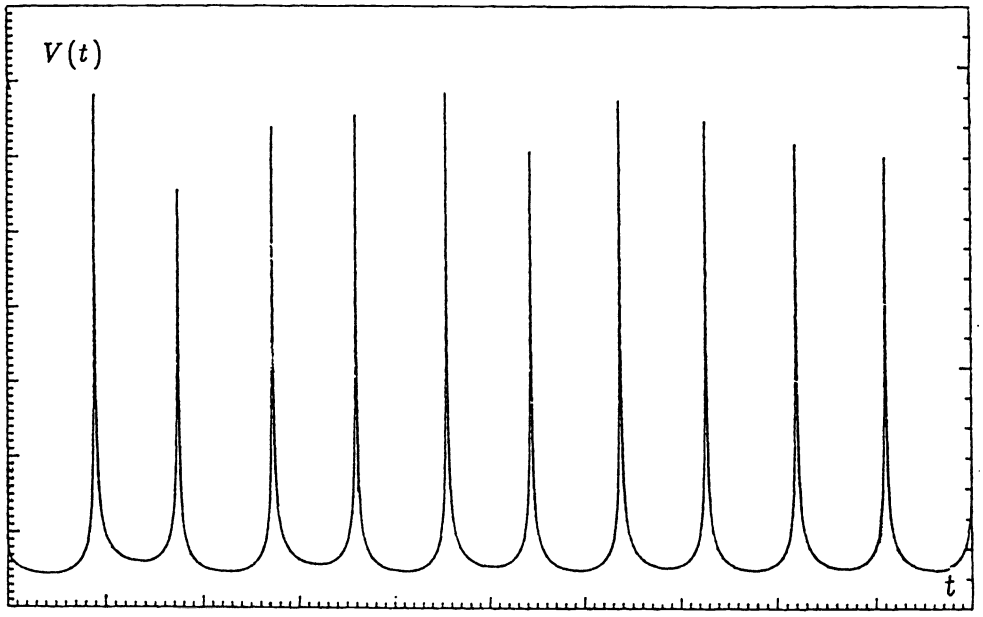


FIG. 3. *Variations of the flame speed as a function of time for* $\mathrm{Le} = 2$, $\beta = 20$.

moves with a velocity close to the flame speed. As a result, less than 3 percent of the computer time is used for the grid adaption itself.

3.2. The ozone decomposition flame. We now investigate a premixed flame with a moderately complex chemical mechanism. More precisely, we consider the propagation of a one-dimensional adiabatic flame in a combustible mixture of monoatomic oxygen (O), oxygen (O_2), and ozone (O_3). This ozone decomposition flame has been computed by many authors (see [2], [15], [17], [18], [21], [22], [25]); it is governed by seven reversible chemical reactions that can be written as follows:

$$O+O_2+X \leftrightarrow O_3+X,$$
$$O+O_3 \leftrightarrow 2O_2,$$
$$O_2+X \leftrightarrow 2O+X,$$

where X represents any of the species O, O_2, or O_3. The detailed formulation of the governing equations describing this flame and the values of the physical constants appearing in these equations are well known (see, for instance, [17]). We simply recall that the Lagrangian form of these equations reads:

$$\begin{aligned} T_t &= T_{xx}+\Omega_T(T, Y, Z), \\ Y_t &= Y_{xx}+\Omega_Y(T, Y, Z), \\ Z_t &= Z_{xx}+\Omega_Z(T, Y, Z), \end{aligned} \tag{21}$$

where T, Y, and Z denote the temperature of the mixture, the mass fraction of O_2, and the mass fraction of O_3, respectively (the mass fraction of O is $1-Y-Z$).

The numerical method presented above can be straightforwardly extended to the solution of (21). We detail only the evaluation of the grid velocity $V(t)$: using only the first equation in (21), we could evaluate as in (7) a velocity $V_T(t)$ satisfying

$$V_T(t)[T(+\infty, t)-T(-\infty, t)]+\int_{\mathbb{R}} \Omega_T(T, Y, Z)(t)=0.$$

In some sense, V_T is the "instantaneous average velocity of the T-profile." We can also define other velocities V_Y and V_Z given by the following relations:

$$V_Y(t)[Y(+\infty, t)-Y(-\infty, t)]+\int_{\mathbb{R}} \Omega_Y(T, Y, Z)(t)=0,$$

$$V_Z(t)[Z(+\infty, t)-Z(-\infty, t)]+\int_{\mathbb{R}} \Omega_Z(T, Y, Z)(t)=0.$$

The three velocities V_T, V_Y, and V_Z coincide when the solution reaches a steady state; in fact, they differ only slightly, even during the transient development of the flame. In the numerical experiment reported below, we solve instead of (21) the following equations:

$$\begin{aligned} T_t &= T_{xx}+\Omega_T(T, Y, Z)+V(t)T_x, \\ Y_t &= Y_{xx}+\Omega_Y(T, Y, Z)+V(t)Y_x, \\ Z_t &= Z_{xx}+\Omega_Z(T, Y, Z)+V(t)Z_x, \end{aligned}$$

where the actual grid velocity $V(t)$ is simply taken equal to an average value of $V_T(t)$, $V_Y(t)$, and $V_Z(t)$:

$$V(t)=\tfrac{1}{3}[V_T(t)+V_Y(t)+V_Z(t)].$$

In fact, a better choice would be to choose the grid velocity $V(t)$ to conserve a physically meaningful combination of T, Y, and Z, such as the total enthalpy.

For the calculation, we use the initial data used in, e.g., [17], [22], and *only* 40 *adaptive meshpoints.* We do not show figures describing the transient development of the flame, which is well computed by our method: these results are well known and can be found in, e.g., [17], [22], [25]. During the calculation, the time-dependent solution rapidly converges to the steady state shown in Fig. 4. Defining now the residual R^n as

$$R^n = \max_i \left[\frac{|T_i^{n+1} - T_i^n|}{t^{n+1} - t^n} + \frac{|Y_i^{n+1} - Y_i^n|}{t^{n+1} - t^n} + \frac{|Z_i^{n+1} - Z_i^n|}{t^{n+1} - t^n} \right],$$

we obtain values of R^n less than 5.10^{-5} at time $t_0 = 600\mu s$. Therefore, the solution again converges to a stationary solution instead of a traveling solution. The computed steady state velocity is $V(t_0) = 49.8$ cm/s (the three velocities $V_T(t_0)$, $V_Y(t_0)$, $V_Z(t_0)$ are in the interval [49.7 cm/s, 49.9 cm/s]). *This velocity is equal to the more accurate values of the ozone flame speed found in the literature* (see the review of these results in [22]).

3.3. Remarks on the time-integration scheme. Because in this paper, we are essentially interested in the adaptive gridding procedure, we have used an explicit time-integration scheme for all calculations whose results are presented above. However, it is not our purpose here to claim that implicit methods are never necessary for an efficient numerical simulation of flame-propagation phenomena. In particular, we have explained in the Introduction that, in the case of a really complex chemical mechanism (for instance, for hydrocarbon flames or even hydrogen-air flames; see [20], [28]), the stiffness due to the disparity of the chemical time scales leads us to treat implicitly the chemical source terms. Moreover, even in the case of (1)-(4) with one-step chemistry, semi-implicit methods are also needed when a high resolution of the flame front is sought; when only 40 meshpoints are used to solve (1)-(4) (as in [12], where the

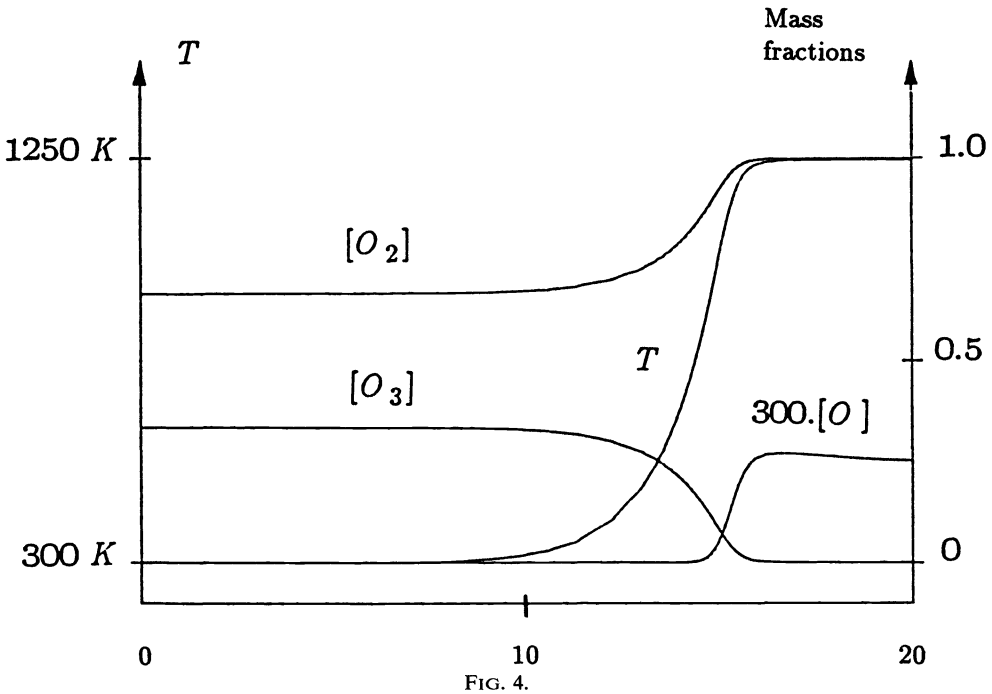


FIG. 4.

computed solutions were satisfactory, but less accurate than those of § 3.1 above), the explicit and implicit timesteps are of the same order of magnitude; when 100 adaptive nodes are used, the diffusive stability condition of the explicit scheme becomes very severe (compared with the stability condition associated with the reactive term), and a semi-implicit scheme, in which at least the diffusive terms are treated implicitly, is more efficient than a fully explicit scheme (see, e.g., [1], [7]).

Nevertheless, it is interesting to note here that, whereas all but one of the authors referenced in § 3.2 above used an implicit method for the problem of the ozone flame (and even sometimes rather sophisticated high-order temporal or spatial approximations), the timestep employed in our explicit calculation is only about half of the timestep used in the implicit method of Lund [15] (unfortunately, the other authors do not give the value of their implicit timestep). The average timestep in the interval $[0, t_0]$ in our calculation is indeed 0.2 μs; it is limited by the diffusive stability restriction, but is very close to the Lipschitz stability restriction for the reaction terms; the corresponding average timestep was 0.4 μs for the implicit method used in [15], and 0.3 μs in the study of Reitz [25], who also used an explicit method, but with a more sophisticated predictor-corrector scheme and with no dynamic rezone procedure. Therefore, the present explicit method operates for this problem with a timestep of the same order of magnitude as the timestep of implicit methods. This shows that the ozone flame problem (21) is certainly less stiff than many other problems involving complex chemistry, and that the combination of a simple explicit schemes with efficient adaptive procedures, which in particular change the convergence to a traveling wave into convergence to steady state, proves to be very competitive for the solution of this moderately complex (i.e., moderately stiff) combustion problem (which however involves 14 elementary reactions).

Acknowledgment. Some of the calculations reported in this paper were done in collaboration with N. Maman, on the CRAY 1S of the C^2VR (Centre de Calcul Vectoriel pour la Recherche).

REFERENCES

[1] F. BENKHALDOUN AND B. LARROUTUROU, *A finite-element adaptive investigation of curved stable and unstable flame fronts*, Comput. Methods Appl. Mech. Engrg., submitted.

[2] L. BLEDJIAN, *Computation of time-dependent laminar flame structure*, Comb. Flame, 20 (1973), pp. 5–17.

[3] R. L. DOUGHERTY, A. S. EDELMAN, AND J. M. HYMAN, *Positivity-monotonicity-, or convexity-preserving cubic and quintic Hermite interpolation*, Report LA-UR-85-2877, Los Alamos National Laboratory, Los Alamos, NM, 1985.

[4] J. K. DUKOWICZ, *Conservative rezoning (remapping) for general quadrilateral meshes*, J. Comput. Phys., 54 (1984), pp. 411–424.

[5] J. K. DUKOWICZ AND J. W. KODIS, *Accurate conservative remapping (rezoning) for arbitrary Lagrangian-Eulerian computations*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 305–321.

[6] H. A. DWYER, R. J. KEE, AND B. R. SANDERS, *Calculation of complex combustion waves with adaptive grids*, in Seventh Internat. Colloquium on Gas Dynamics and Reaction Systems, AIAA, 1978.

[7] H. GUILLARD, B. LARROUTUROU, AND N. MAMAN, *Etude numerique des instabilités cellulaires d'un front de flamme par une méthode pseudo-spectrale*, Report 721, INRIA, 1987.

[8] J. M. HYMAN AND R. J. KNAPP, *High-order conservative difference formulas for nonuniform grids*, Report, Los Alamos National Laboratory, Los Alamos, NM, 1985.

[9] J. M. HYMAN AND B. LARROUTUROU, *On the use of adaptive moving grid methods in combustion problems*, in Complex Chemical Reaction Systems—Mathematical Modelling and Simulation, J. Warnatz and W. Jager, eds., Lecture Notes in Chemistry, Springer-Verlag, Heidelberg, 1987, pp. 222–232.

[10] J. M. HYMAN AND M. J. NAUGHTON, *Static rezone methods for tensor-product grids*, Report LA-UR-83-3245, Los Alamos National Laboratory, Los Alamos, NM, 1983.

[11] R. J. KEE, L. R. PETZOLD, M. D. SMOOKE, AND J. F. GRCAR, *Implicit methods in combustion and chemical kinetics modeling*, in Multiple Time Scales, Brackbill and Cohen, eds., Academic Press, New York, 1985, pp. 113-144.

[12] B. LARROUTUROU, *Utilisation de maillages adaptatifs pour la simulation de flammes monodimensionnelles instationnaires*, in Numerical Simulation of Combustion Phenomena, R. Glowinski, B. Larrouturou, and R. Temam, eds., Lecture Notes in Physics, 241, Springer-Verlag, Heidelberg,1985, pp. 300-312.

[13] ———, *Adaptive numerical methods for unsteady flame propagation*, in Combustion and Chemical Reactors, G. Ludford, ed., Lectures in Applied Mathematics, 24, American Mathematical Society, Providence, RI, 1986, pp. 415-435.

[14] ———, *The equations of one-dimensional unsteady flame propagation: Existence and uniqueness*, SIAM J. Math. Anal., 19 (1988), pp. 32-59.

[15] C. M. LUND, Report UCRL-52504, Lawrence Livermore Laboratory, Livermore, CA, 1978.

[16] A. LUTZ, R. J. KEE, AND H. A. DWYER, *Ignition modeling with grid adaptation*, Report, Sandia National Laboratories, Livermore, CA, 1985.

[17] S. B. MARGOLIS, *Time-dependent solution of a premixed laminar flame*, J. Comput. Phys., 27 (1978), pp. 410-427.

[18] K. MEINTJES, *Predictor-corrector methods for time-dependent compressible flows*, Ph. D. thesis, Princeton University, Princeton, NJ, 1979.

[19] N. PETERS, *Discussion of test problem* A, in Numerical Methods in Laminar Flame Propagation, N. Peters and J. Warnatz, eds., Notes in Numerical Fluid Mechanics, 6, 1982, Vieweg, Wiesbaden, pp. 1-14.

[20] N. PETERS AND J. WARNATZ, EDS., *Numerical methods in laminar flame propagation*, in Notes in Numerical Fluid Mechanics, 6, Vieweg, Wiesbaden, 1982.

[21] J. I. RAMOS, *The application of finite-difference and finite-element methods to a reaction-diffusion system in combustion*, in Numerical Methods in Laminar and Turbulent Flows, Taylor, Johnson, and Smith, eds., Pineridge Press, Swansea, 1983, pp. 1137-1147.

[22] ———, *Development and application of an adaptive finite element method to reaction-diffusion equations*, Internat. J. Numer. Methods Fluids, 5 (1985), pp. 13-23.

[23] J. D. RAMSHAW, *Conservative rezoning algorithm for generalized two-dimensional meshes*, J. Comput. Phys., 59 (1985), pp. 193-199.

[24] ———, *Simplified second-order rezoning algorithm for generalized two-dimensional meshes*, J. Comput. Phys., 67 (1986), pp. 214-222.

[25] R. D. REITZ, *The application of an explicit numerical method to a reaction-diffusion system in combustion*, Courant Institute of Mathematical Sciences, New York University, NY, 1979.

[26] M. D. SMOOKE, *On the use of adaptive grids for premixed combustion*, in Combustion and Nonlinear Phenomena, P. Clavin, B. Larrouturou, and P. Pelce, eds., Editions de Physique, Paris, 1985, pp. 213-228.

[27] M. D. SMOOKE AND M. L. KOSZYKOWSKI, *Fully adaptive solutions of one-dimensional mixed initial-boundary value problems with applications to unstable problems in combustion*, Report SAND-83-8219, Sandia National Laboratories, Livermore, CA, 1983.

[28] J. WARNATZ, *Flame propagation and ignition in hydrocarbon-air mixtures up to octane*, in Numerical Simulation of Combustion Phenomena, R. Glowinski, B. Larrouturou, and R. Temam, eds., Lecture Notes in Physics, 241, Springer-Verlag, Heidelberg, 1985, pp. 391-401.

[29] O. C. ZIENKIEWICZ, *The Finite-Element Method in Engineering Sciences*, McGraw-Hill, London, 1971.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 756-776, July 1989

© 1989 Society for Industrial and Applied Mathematics
009

WAVEFORM ITERATION AND THE SHIFTED PICARD SPLITTING*

ROBERT D. SKEEL†

Abstract. The theme of this paper is that the primary computational bottleneck in the solution of stiff ordinary differential equations (ODEs) and the parallel solution of nonstiff ODEs is the implicitness of the ODE rather than the approximation of the integration process (or in conventional terminology, numerical stability rather than accuracy), and therefore it may be fruitful to apply (at least conceptually) the iterative techniques needed to overcome implicitness in continuous time, before discretization—to waveforms rather than values at a point in time. Several classical iterations, based on splitting, are discussed, but the emphasis is on those not based on a partitioning of the ODE system. The shifted Picard iteration is proposed as a compromise between the cheap but slow Picard iteration and the fast but expensive Newton iteration. By varying the shift parameter from one iteration to the next, a good rate of convergence seems possible. As an alternative, the author also examines the more classical acceleration technique applied to the Picard iteration. Some experimental results are given. However, the practical aspects of discretization are beyond the scope of this paper.

Key words. waveform relaxation, waveform Newton, parallel algorithms, Picard iteration, stiff ODEs, acceleration, operator splitting

C. R. classification. G.1.7

AMS(MOS) subject classifications. 65B99, 65L05

1. Introduction.

Simulations often involve the solution of a system of ordinary differential equations (ODEs):

$$\mathcal{D}_t y^i = f^i(t, y), \quad 0 \leqq t \leqq 1, \quad y^i(0) = \eta^i,$$

$i = 1, 2, \cdots, neqn$, where $\mathcal{D}_t$ indicates differentiation with respect to time. An equivalent, but much better conditioned problem is the integral equation

$$y(t) = \eta + \int_0^t f(s, y(s))\, ds.$$

There are two distinct problems here: the *approximation* (or discretization) of the indefinite integral and the *solution* of this implicit equation for $y(t)$. Traditionally, the problem is discretized first to yield a system of algebraic equations that are solved by Newton-like iterations for stiff problems and by Picard-like iterations ("functional iteration") for nonstiff problems. In the case of methods such as explicit Runge-Kutta methods, the iterations are combined with the quadrature. Solving the equations can be a major part of the effort for stiff initial value problems (IVPs). It is also the major bottleneck in the development of parallel algorithms. (The computation of a system of integrals is inherently a parallel task, although adaptive techniques pose problems.)

It can be argued that since the implicitness rather than the integral approximation is the primary computational challenge, one should first find an iteration to handle the implicitness and then discretize the integration. For, if we discretize first, we run ths risk of making the equations more difficult to solve. Rather, we should use whatever latitude we are given by the error tolerance to make the equations easier to solve. A

* Received by the editors November 5, 1987; accepted for publication (in revised form) October 10, 1988. This work was partially supported by the U.S. Department of Energy under grant DE FG02-85ER25001 and by the National Science Foundation under grant DMS 8703226.

† Department of Computer Science, 240 Digital Computer Laboratory, University of Illinois, 1304 W. Springfield Avenue, Urbana, Illinois 61801.

class of such continuous-time iterative methods, called waveform relaxation methods, are in use in electronic network analysis. A *waveform* is a continuous representation of a solution component on a part of the interval of integration called a *window.* Relaxation can be viewed as a special type of iteration in which a group of unknowns is relaxed so as to satisfy a group of equations. Of course, there are other iterative methods based on different operator splittings, so we use the term "*waveform iteration*" to refer to an iteration for solving ODEs in continuous time in some window on the interval of integration. If a discrete approximation is used for the integral operator and the window is shrunk to a single stepsize, then we get a step-by-step method. However, a method obtained in this way might be better than a conventional method because looking at iterative methods in continuous time broadens possibilities and avoids distracting details. Moreover, progress in this direction may suggest better iterative techniques for solving the nonlinear system of equations that result from the use of conventional difference methods.

There is increasing interest in parallel algorithms for ODEs. If we accept that the bulk of the computational effort is in the evaluation of $f^i(t, y)$, then there are three dimensions for parallelism: the index i, the time t, and the multidimensional state vector y. Parallelism across the index i means that f is partitioned into different subvectors that are evaluated in parallel. This the most natural form of parallelism and was implemented by the original hardware of ENIAC and by the digital differential analyzers that followed. There is, however, the problem of partitioning the evaluation of f among different processors so that each processor shares equally in the computational load. Efficient automated partitioning seems feasible only for application-oriented codes where the structure of f is accessible to the ODE solver. In its simplest form this approach entails the broadcasting of different components of f among the processors after each evaluation, creating significant communication delays. This problem is diminished in waveform relaxation where each subvector of f is evaluated at several different sets of (t, y) values before broadcasting. Parallelism across time t means that each processor evaluates f at a different value of t and at a value of y that approximates $y(t)$. Such is the approach of the block methods of Worland [17], Horiguchi, Kawazoe, and Nara [7], Birta and Abou-Rabia [1], and Chu and Hamilton [2]. The potential for parallelism is limited by the ability to predict future states from the current state without computing intermediate states, and it is one of the aims of this paper to explore these limitations. Parallelism across the state space y seems less useful, except in connection with numerical differencing to evaluate the Jacobian matrix (cf. [13]). Gear [5] uses the terms "parallelism across the problem" for parallelism across i and "parallelism across the method" for parallelism across (t, y).

The next section discusses some fairly classical iterative schemes applied to ODEs in continuous time and reports some numerical results that suggest limitations for the conventional block method approach. Section 3 discusses ways to accelerate such methods. Section 4 proposes an alternative to acceleration that involves a parameterized splitting, as in successive overrelaxation (SOR), except that the parameter varies from one iteration to the next. Numerical results are given for a method of this type. Most of the complicated mathematics has been relegated to three appendices of a technical report [15]. The important issues of window selection and discretization are not addressed by this paper.

The techniques of §§ 3 and 4 seem appropriate for nonstiff problems and for stiff problems where there are relatively few clusters of large eigenvalues. We have no evidence that they are viable for a large class of stiff problems.

2. Splittings for waveform iteration. Iterative methods can be applied to ODEs in continuous time before discretizing. Most iterations can be derived from splittings. Let $F(t, y, z)$ be chosen so that $F(t, y, y) = f(t, y)$ and $\mathscr{D}_t z = F(t, y(t), z)$ is easy to solve given $y(t)$. The iteration is

$$\mathscr{D}_t y_m = F(t, y_{m-1}(t), y_m), \qquad y_m(0) = \eta \tag{2.1}$$

with $y_0(t) \equiv \eta$. In practice $y_m(t)$ is determined on some window $[a_m, b_m]$ on the interval of integration where $[0, a_m]$ is the portion of the interval on which convergence has been achieved. Values of $y_m(t)$ beyond b_m are obtained by extrapolation. Hence it is necessary to store a representation of the solution components on the continuum $[a_m, b_m]$. Typically these "waveforms" would be represented by piecewise polynomial interpolation.

Waveform *relaxation* has received a lot of attention recently for the solution of large-scale networks (e.g., see [9]). It is the special case of an additive splitting

$$F(t, y, z) = g(t, z) + f(t, y) - g(t, y)$$

where g is chosen so that its Jacobian matrix is block diagonal or block triangular, so that the differential equations that must be solved consist of a number of decoupled subsystems that can be solved independently or sequentially, respectively. The original motivation seems to be twofold: (i) reduce the primary storage requirements for the integration, and (ii) permit the use of different discretizations—especially stepsizes—for different subsystems. The bulk of the work that has been done concerns the choice of g for achieving the best rate of convergence of (2.1) subject to certain constraints that deal with the computation time needed to solve the differential equations numerically, at each step of the iteration. See, for example, [16]. Reference [11] examines the convergence of waveform relaxation and extensions of it corresponding to overrelaxation applied over the infinite interval for the linear problem and derives results similar to the results for relaxation methods for linear equations. (The use of an infinite interval places restrictions on g not necessary for convergence on the finite interval, but these are necessary if the *rate* of convergence is to be independent of the stiffness.) More recently [5], there has been interest in this type of method because of its potential for use on parallel computers.

For waveform relaxation each subsystem must generally be solved sequentially. This can be avoided by choosing $g(t, z) \equiv 0$, which yields the classical Picard[1] iteration. The quadrature-type integration is amenable to parallel or vector computation. As an iterative method, Picard is superlinearly convergent on a finite interval $[0, 1]$ because the error is proportional to $L^{m+1}/(m+1)!$ where L is a Lipschitz constant for $f(t, y)$. One can imagine many splittings other than relaxation and Picard; for example, the Newton-Raphson splitting is given by

$$F(t, y, z) = f(t, y) + \frac{\partial f}{\partial y}(t, y) \cdot (z - y).$$

[1] More generally, a Picard iteration is a simplified Newton iteration in which we approximate the Fréchet derivative of a differential (or difference) operator by those terms that determine the character of the operator, that is, the highest-order derivatives with respect to each independent variable. If the difference operator involves just a single step, then Picard iteration becomes the same as "functional iteration." However, the latter term is broad enough to include Newton iteration, which is why the former might be preferred.

In principle, the evaluations of f and the columns of f_y can be performed in parallel, and the values at different time levels can be combined in logarithmic time on a parallel processor by means of odd/even reduction [3], [6]. However, there are a couple of practical difficulties: the need for analytical differentiation and the possible large dimensions of the matrix f_y. There are many ways to avoid analytical differentiation; some of these, such as quasi-Newton methods, involve gleaning information from previous evaluations of f. A quasi-Newton method has the form

$$\mathcal{D}_t y_{m+1} = f(t, y_m(t)) + J_m(t)(y_{m+1} - y_m(t))$$

where for a rank one Jacobian approximation one might use

$$J_m := g_m x_m^T / (x_m^T x_m)$$

with

$$g_m(t) := f(t, y_m(t)) - f(t, y_{m-1}(t))$$

and

$$x_m(t) := y_m(t) - y_{m-1}(t),$$

and for rank two one might use

$$J_m := [g_m, g_{m-1}]([x_m, x_{m-1}]^T[x_m, x_{m-1}])^{-1}[x_m, x_{m-1}]^T.$$

Another way to reduce the linear algebra costs of Newton's method, suggested by B. J. Leimkuhler, is to use the following predictor-corrector process:

$$\mathcal{D}_t \bar{y}_{m+1} = f(t, y_m(t)),$$

$$\mathcal{D}_t y_{m+1} = f(t, y_m(t)) + f_y(t, y_m(t))(\bar{y}_{m+1}(t) - y_m(t)).$$

Again the evaluations of f and f_y can be performed in parallel, and these values can be combined by performing a numerical cubature (double integration), which is cheaper than solving the system of linear equations in Newton's method. Doing more than one correction creates additional terms of higher-dimensional integrals. All of these methods share the superlinear convergence property of Picard.

Discretization reduces analytical Picard iteration to a numerical Picard iteration:

$$y_m(t) = y_{m-1}(t) + \int_{a_m}^{t} \mathcal{I}_m r_m$$

for $t \geqq a_m$ where $\mathcal{I}_m$ is a linear operator that approximates the residual

$$r_m(t) := f(t, y_m(t)) - \mathcal{D}_t y_m(t)$$

from discrete values in the window $[a_m, b_m]$. (Note that this is equivalent to (2.1) if $\mathcal{I}_m$ is not present.) Several explicit Runge-Kutta methods can be derived by appropriate choices for $\mathcal{I}_m$, and, in fact, Moulton [12] uses the Picard iteration to motivate the Adams formulas. Conventional numerical Picard methods, however, do only one new function evaluation per iteration; with parallel computers we can do several function evaluations per iteration and thus do a more accurate integration. This should be very useful if f is not a smooth function of t.

Waveform iteration gives us at least one and possibly two ways to achieve parallelism. First, the system can be partitioned into subsystems, each of which can be integrated independently (with its own error control) on a different processor. It is, of course, important to equidistribute the computational workload. After each iteration it is necessary to communicate the updated waveforms among the processors

and to advance the window. In order to limit the effect of communication delays, it is desirable to use fairly large windows, and this is productive only if the iteration is converging sufficiently fast. For block relaxation the rate of convergence depends on the partitioning. If it is difficult to determine a partitioning that gives both good convergence and a balanced loading of the processors, then it could be more cost effective to use accelerated or shifted Picard (described in the next two sections). Moreover, there is an additional or alternative source of parallelism in a Picard iteration, namely, the SIMD parallelism in the simultaneous evaluation of the derivative function $f(t, y)$ at different time levels. If different processors are used for different time levels, then there is no load balancing problem.

We have performed a numerical experiment for various waveform splittings to get some indication of the potential speedup. The simple circular orbit problem

$$\mathbf{r}(0) = [1, 0]^{\mathrm{T}}, \quad \mathscr{D}\mathbf{r}(0) = [0, 1]^{\mathrm{T}}, \quad \mathscr{D}_t^2\mathbf{r} = -\frac{\mathbf{r}}{\|\mathbf{r}\|_2^3}$$

was integrated for $0 \leqq t \leqq 6$, slightly less than one orbit. As a standard for comparison, we used the state-of-the-art sequential code ODE of Shampine and Gordon [14] with a range of tolerances $ATOL = RTOL$. To simulate the waveform iteration, several dozen iterates were integrated by ODE with $ATOL = RTOL = 10^{-9}$ as a single massive system of partially coupled equations. The results of these experiments are given in Fig. 1. The vertical axis gives the number of function evaluations for ODE and the number of iterations for the waveform methods. Each iteration of a waveform method counts as one parallel function evaluation because the necessary evaluations of f (and the columns of f_y) can be performed in parallel. The horizontal axis gives the digits of accuracy using the one-norm of the relative (global) error at $t = 6$. Data from Fig. 1 has been used to construct Table 1, which gives the speedup of the waveform methods over ODE for various accuracies. The speedup is computed by dividing the number of ODE function evaluations by the number of waveform iterations. The waveform Picard iteration, with its idealized high accuracy numerical integration, probably represents an upper bound on the potential for parallelism with conventional explicit methods, such as block methods.

The limitation on the convergence rate of waveform Picard corresponds to a stability limitation on the stepsize of discrete step-by-step methods, because for the test problem $\mathscr{D}_t y = Ay$ waveform Picard with m iterations is equivalent to the mth order Taylor method. We know from the Jeltsch-Nevanlinna result [8, Thm. 2.1, p. 64] that the extent of the stability region of a one-step method is in some sense determined by the degree of the stability polynomial,[2] which in turn is limited by the number of *sequential* stages of a method. (A stability polynomial of degree m can only result from at least m nested evaluations of f.) Thus, doing stages in parallel may improve accuracy, but it will not enhance stability. That is, parallel versions of conventional explicit methods will permit increases in stepsize only up to the point where stability restrictions become active.

Our results say nothing about the parallel efficiency of these waveform iterations. (The efficiency of our implementation was low because we strived for negligible discretization error.) The results of Birta and Abou-Rabia [1] suggest that speedups close to those given in the tables could be achieved with reasonably efficient use of processors.

[2] For two stability polynomials of the same degree the stability region of one cannot be a proper subset of the stability region of the other.

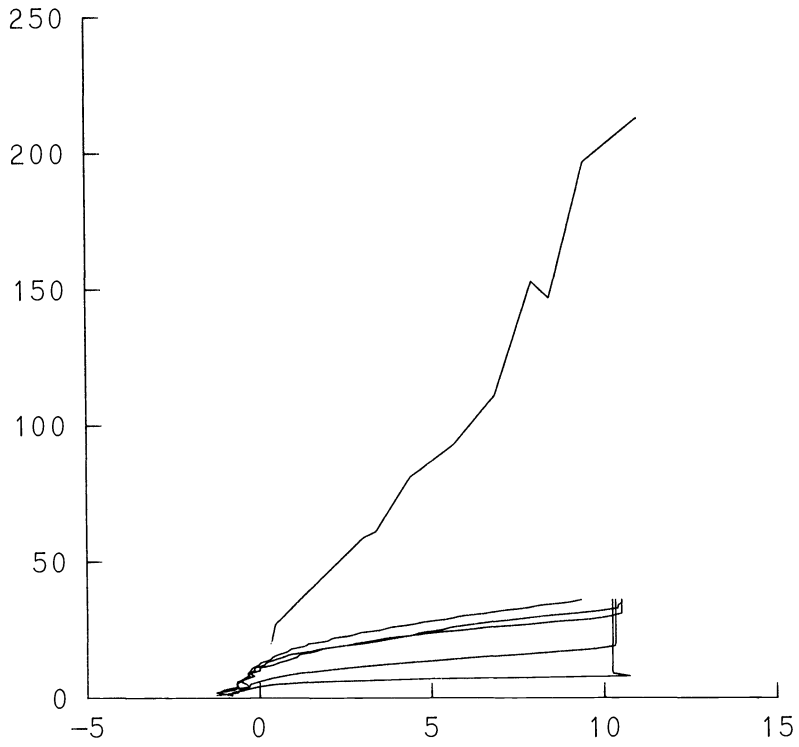


FIG. 1. *Number of parallel function evaluations for* ODE, *Picard, rank one quasi-Newton, rank two quasi-Newton, predictor-corrector, and Newton, respectively, versus digits of accuracy.*

3. Acceleration. Although the parallelism obtained by the Picard iteration is straightforward and problem-independent, the rate of convergence is too slow for stiff problems, and experiments suggest for nonstiff problems a modest speedup compared to the best sequential algorithms (two to five, depending on the accuracy needed). Therefore one seeks ways to accelerate the convergence. The waveform iteration (2.1) can be expressed as follows:

$$\mathcal{D}y_m - F(t, y_{m-1}, y_m) = \mathcal{D}y_{m-1} - F(t, y_{m-1}, y_{m-1}) + r_{m-1}.$$

A suitable acceleration might be achieved by replacing r_{m-1} by $\alpha_m r_{m-1} + \beta_m r_{m-2}$.

TABLE 1
Speedup of waveform methods over ODE.

Method	Digits of accuracy								
	1	2	3	4	5	6	7	8	9
Picard	1.9	2.2	2.4	2.8	3.1	3.2	3.6	4.5	4.8
rank one quasi-Newton	2.1	2.6	2.9	3.4	3.5	3.6	4.1	5.2	5.5
rank two quasi-Newton	2.3	2.6	2.9	3.4	3.7	4.0	4.4	5.6	6.1
predictor-corrector	3.8	4.6	5.1	5.8	6.5	6.7	7.5	9.3	9.7
Newton	6.0	7.2	8.9	11.0	12.6	13.3	15.8	20.2	21.2

A choice of acceleration parameters could be based on a study of the model problem $\mathcal{D}_t y = Ay + \phi(t)$, where A is identified with the Jacobian matrix evaluated at $t = a_m$. The solution of the given problem departs from this by $O((t-a_m)^2)$. The model is even more accurate for semidiscretized quasi-linear partial differential equations (PDEs) and is exact for some problems of interest in control theory.

To see why acceleration should be possible, consider the fact that the Picard iteration generates the truncated Taylor series approximation to $\exp(tA)$. There are, of course, other polynomials of the same degree that may give a better approximation if we have information about the extreme eigenvalues of A. Thus there is the possibility of a Chebyshev-type acceleration based on minimizing the norm of the iteration operator. Estimates of extreme eigenvalues can be obtained dynamically during unaccelerated or less-than-optimally accelerated iterations (see, for example, [10]).

To be specific, an accelerated Picard might have the form

$$\mathcal{D}y_m = \mathcal{D}y_{m-1} + \alpha_m r_{m-1}$$

or

$$y_m = y_{m-1} + \alpha_m \mathcal{D}^{-1} r_{m-1} \tag{3.1}$$

where

$$\mathcal{D}_t^{-1} = \int_0^t .$$

Introducing the error

$$e_m := y_m - y,$$

we have for the model problem that

$$r_m = (A - \mathcal{D}) e_m$$

and from (3.1) that

$$\begin{aligned} e_m &= e_{m-1} + \alpha_m \mathcal{D}^{-1}(A - \mathcal{D}) e_{m-1} \\ &= (1 - \alpha_m (1 - A\mathcal{D}^{-1})) e_{m-1}. \end{aligned} \tag{3.2}$$

Solving the recurrence, we get

$$e_m = Q_m(A\mathcal{D}^{-1}) e_0$$

where Q_m is a polynomial of degree at most m satisfying $Q_m(1) = 1$. It is easy to show that any such Q_m can be obtained with an appropriate choice of real and complex conjugate pairs of acceleration parameters α_m. Moreover, Q_m can be expressed as

$$Q_m(z) = 1 + zP_{m-1}(z) - P_{m-1}(1)$$

where P_{m-1} could be any polynomial of degree at most $m-1$. It can be shown by induction that

$$(A\mathcal{D}_t^{-1})^{j+1} e_0(t) = \int_0^t \frac{A^j (t-s)^j}{j!} A\, e_0(s)\, ds,$$

and hence we get

$$e_m(t) = (1 - p_{m-1}(0) - \cdots - p_{m-1}^{(m-1)}(0))\, e_0(t) + \int_0^t p_{m-1}(A(t-s)) A\, e_0(s)\, ds$$

where $p_{m-1}(z)$ is obtained from $P_{m-1}(z)$ by dividing the z^j term in $P_{m-1}(z)$ by $j!$.

If A is a scalar,

$$-L \leqq A \leqq 0,$$

then after a change of variable we have

$$\frac{\|e_m\|_\infty}{\|e_0\|_\infty} \leqq |p_{m-1}(0) + \cdots + p_{m-1}^{(m-1)}(0) - 1| + \int_{-L}^{0} |p_{m-1}(x)|\, dx \tag{3.3}$$

where

$$\|z\|_\infty := \max_{0 \leqq t \leqq 1} |z(t)|.$$

A fairly obvious good choice for p_{m-1} would be the Chebyshev polynomial of degree $m-1$ transformed to the interval $[-L, 0]$ and scaled so that the first term in (3.3) vanishes:

$$p_{m-1}(x) = \frac{T_{m-1}(1 + (2/L)x)}{T_{m-1}(1) + \cdots + (2/L)^{m-1} T_{m-1}^{(m-1)}(1)}.$$

With this choice, it is shown in Appendix A of [15] that

$$\frac{\|e_m\|_\infty}{\|e_0\|_\infty} \leqq \frac{L^{3/2}}{m} \exp\left(\frac{-m^2}{L} + O(1)\right), \tag{3.4}$$

assuming $m = O(L^{3/4})$. Then it is straightforward to show that a number of iterations

$$m = L^{1/2}\left(\log \frac{1}{\varepsilon} + \log L\right)^{1/2}$$

is sufficient to have

$$\frac{\|e_m\|_\infty}{\|e_0\|_\infty} = O(\varepsilon), \tag{3.5}$$

assuming $L \geqq 1$ and $\exp(-L^{1/2}) \leqq \varepsilon \leqq \exp(-1)$.

Note. It is likely that iteration (3.1) would lead to problems of numerical instability and that an alternative formulation might be preferable. However, there is no obvious three-term recurrence satisfied by Q_m that would result in a stable second-order iteration.

On the other hand, with no acceleration ($\alpha_m \equiv 1$) an error reduction of $O(\varepsilon)$ requires

$$m = eL - \frac{1}{2}\log L + \log \frac{1}{\varepsilon} \tag{3.6}$$

with the same restriction on L and ε as before. The details are in Appendix A of [15].

Alternatively, the choice of parameters could be based on a dynamic minimization of some measure of the error as in the conjugate gradient (CG) method. Nonlinearity may complicate the selection of parameters. It may be necessary to use a guess $\bar{\alpha}_m$ (suppose $\beta_m \equiv 0$) for which the residual is $\bar{r}_m$ and then to choose α_m under the assumption that

$$r_m = \bar{r}_m + \left(\frac{\alpha_m}{\bar{\alpha}_m} - 1\right)(\bar{r}_m - r_{m-1})$$

is true, which is correct for the model problem.

Remark. The relationship between Q_m and p_{m-1} can be simply expressed as

$$Q_m(z) = \int_0^\infty e^{-s}\{zp_{m-1}(zs) - p_{m-1}(s) + 1\}\, ds,$$

which for our special choice of p_{m-1} becomes

$$Q_m(z) = \frac{z\displaystyle\int_0^\infty e^{-s} T_{m-1}\left(1+\frac{2}{L}zs\right) ds}{\displaystyle\int_0^\infty e^{-s} T_{m-1}\left(1+\frac{2}{L}s\right) ds}.$$

4. Dynamic splittings. Unaccelerated waveform iteration increases the formal order of accuracy by one for each iteration because the error is $O(t^{m+1})$. This property is lost if the iteration is accelerated, which may or may not be important. In any case, there is an alternative approach that preserves this property. We restrict our discussion to the Picard iteration, although the technique is applicable in general (at a greater cost in *local* computation). The basic idea is to approximate the Jacobian matrix by a scalar μ_m times the identity:

$$y_m' - \mu_m y_m = f(t, y_{m-1}) - \mu_m y_{m-1}.$$

This we call the "*shifted Picard splitting.*" Computationally, this is just a weighted quadrature:

$$\begin{aligned} y_m(t) - y_{m-1}(t) &= (\mathcal{D}_t - \mu_m)^{-1} r_{m-1}(t) \\ &= \int_0^t e^{\mu_m(t-s)} r_{m-1}(s)\, ds. \end{aligned} \tag{4.1}$$

In the case of the model problem, it will be shown that the exact solution can be obtained by having μ_m cycle through the eigenvalues of the Jacobian matrix (more precisely, the roots of the minimal polynomial of the Jacobian matrix).

Since it is desirable to use eigenvalues of f_y as shift parameters, the need will arise to do a complex conjugate pair of shifts μ_1, μ_2:

$$y_1^* = y_0 + (\mathcal{D} - \mu_1)^{-1} r_0, \qquad y_2 = y_1^* + (\mathcal{D} - \mu_2)^{-1} r_1,$$

where for simplicity we are using subscripts 0, 1, and 2. It may be useful to be slightly more general and permit

$$\mu_1, \mu_2 = \nu \pm \sqrt{\omega}$$

where ν and ω are both real, and $\sqrt{\ }$ denotes the square root on either the positive real or the positive imaginary axis. The residual can be expressed as

$$r_1 = \mathcal{F}\{y_1\}$$

where

$$\mathcal{F}_t\{z\} := f(t, z(t)) - z'(t).$$

For nonlinear problems it may not be meaningful (or at least not convenient) to compute $\mathcal{F}$ for an imaginary function and so some way should be found to avoid this. Therefore, we want instead to compute

$$\begin{aligned} y_1 &= y_0 + \tfrac{1}{2}[(\mathcal{D} - \mu_1)^{-1} + (\mathcal{D} - \mu_2)^{-1}] r_0 \\ &= y_0 + (\mathcal{D} - \nu)(\mathcal{D} - \mu_1)^{-1}(\mathcal{D} - \mu_2)^{-1} r_0 \end{aligned} \tag{4.2}$$

(just the real part) and $r_1 = \mathcal{F}\{y_1\}$, and then to construct r_1^* from r_0 and r_1 in such a way that *for the model problem*

$$r_1^* = \mathcal{F}\{y_1^*\}$$

where

$$y_1^* = y_0 + (\mathcal{D} - \mu_1)^{-1} r_0.$$

Using (4.2) to eliminate r_0, we have

$$y_1^* = y_1 + \sqrt{\omega}(\mathcal{D} - \nu)^{-1}(y_1 - y_0).$$

For the model problem, $\mathcal{F}$ is affine and it commutes with $\mathcal{D}$, so that we have

$$\mathcal{F}(y_1^*) = \mathcal{F}(y_1) + \sqrt{\omega}(\mathcal{D} - \nu)^{-1}(\mathcal{F}(y_1) - \mathcal{F}(y_0)),$$

and hence

$$r_1^* := r_1 + \sqrt{\omega}(\mathcal{D} - \nu)^{-1}(r_1 - r_0).$$

Therefore,

$$y_2 = y_1^* + (\mathcal{D} - \mu_2)^{-1} r_1^*,$$

and so after some algebra and the use of (4.2) to eliminate r_0, we get that

$$y_2 = y_1 + (\mathcal{D} - \nu)^{-1} r_1 + \omega(\mathcal{D} - \nu)^{-2}(y_1 - y_0). \tag{4.3}$$

Expressing the iteration (4.2), (4.3) in more familiar terms, we have

$$y_1(t) = y_0(t) + \int_0^t e^{\nu(t-s)} \cosh \sqrt{\omega}(t-s) \cdot r_0(s)\, ds,$$

and

$$y_2(t) = y_1(t) + \int_0^t e^{\nu(t-s)} r_1(s)\, ds + \omega \int_0^t (t-s)\, e^{\nu(t-s)}(y_1(s) - y_0(s))\, ds.$$

The error term turns out to have a simple form in the case of the model problem. From (4.1) we see that

$$\begin{aligned} e_m &= e_{m-1} + (\mathcal{D} - \mu_m)^{-1}(A - \mathcal{D}) e_{m-1} \\ &= (\mathcal{D} - \mu_m)^{-1}(A - \mu_m) e_{m-1}. \end{aligned} \tag{4.4}$$

Repeated application of this recursion is shown, in Apendix B of [15], to lead to

$$e_m(t) = (A - \mu_m) \cdots (A - \mu_1) \int_0^t (t-s)^{m-1} \exp[(t-s)\mu_1, \cdots, (t-s)\mu_m] e_0(s)\, ds, \tag{4.5}$$

where the square brackets denote an $(m-1)$th order divided difference.

Also in Appendix B of [15] it is shown that if $A =: x$ is a scalar and $\mu_1, \cdots, \mu_m$ are real, then

$$\frac{\|e_m\|_\infty}{\|e_0\|_\infty} \leqq |(x - \mu_1) \cdots (x - \mu_m) \exp[0, \mu_1, \cdots, \mu_m]|. \tag{4.6}$$

Thus if x is known to satisfy $-L \leqq x \leqq 0$, we should choose the $\{\mu_j\}$ that minimizes

$$\exp[0, \mu_1, \cdots, \mu_m] \max_{-L \leqq x \leqq 0} |(x - \mu_1) \cdots (x - \mu_m)|. \tag{4.7}$$

The second factor is minimized by choosing the Chebyshev abscissas

$$\mu_j = \frac{L}{2}\left(\cos\frac{j-\frac{1}{2}}{m}\pi - 1\right),$$

and it is shown in Appendix B of [15] that the product of the two factors is minimized *locally* for this same choice. How this bound compares with that for acceleration is not known.

The error analysis could be used as a basis for choosing the shift parameters if one can estimate the extreme eigenvalues of the Jacobian matrix f_y.

Alternatively, one might attempt an adaptive approach based on minimizing the actual error in some sense. This will require a guess $\bar{\mu}_m$ to get

$$\bar{y}_m := y_{m-1} + (\mathscr{D} - \bar{\mu}_m)^{-1} r_{m-1}$$

and $\bar{r}_m := \mathscr{F}\{\bar{y}_m\}$. It is desirable to obtain estimates of the residual for other choices of μ_m without further evaluations of $\mathscr{F}$. Define

$$y_m := y_{m-1} + (\mathscr{D} - \mu_m)^{-1} r_{m-1}$$

and we have

$$y_m = \bar{y}_m + (\mu_m - \bar{\mu}_m)(\mathscr{D} - \mu_m)^{-1}(\bar{y}_m - y_{m-1})$$

and

$$\mathscr{F}\{y_m\} \approx \bar{r}_m + (\mu_m - \bar{\mu}_m)(\mathscr{D} - \mu_m)^{-1}(\bar{r}_m - r_{m-1}) =: r_m$$

with equality holding for the model problem. This is nonlinear in μ_m, but one could do a sequence of one or more minimizations based on local linearization. Alternatively, several values of μ_m could be tried in parallel.

We have tested this idea on some simple problems. The shift μ_m was chosen to minimize

$$(\mathscr{D}^{-1} r_m, \mathscr{D}^{-1} r_m)$$

where

$$(u, v) := \int_0^1 u \cdot v,$$

because it was felt that $\mathscr{D}^{-1} r_m$ was close to the error in the sense that the ratio $\|\mathscr{D}^{-1} r_m\| / \|e_m\|$ is bounded away from zero and infinity by quantities that depend only on the Lipschitz constant for f. (Beginning with $\mu_{(0)} = \bar{\mu}_m$, an inner iteration was used to determine a sequence $\mu_{(1)}, \mu_{(2)}, \cdots$ converging to μ_m. Let

$$r := \bar{r}_m + (\mu - \mu_{(0)})(\mathscr{D} - \mu)^{-1}(\bar{r}_m - r_{m-1}).$$

Linearizing about the current approximation $\mu_{(k)}$ gives

$$\mathscr{D}^{-1} r = \mathscr{D}^{-1} \bar{r}_m + (\mu_{(k)} - \mu_{(0)})\sigma + (\mu - \mu_{(k)})(\sigma + \tau) + O((\mu - \mu_{(k)})^2)$$

where

$$\sigma := \mathscr{D}^{-1}(\mathscr{D} - \mu_{(k)})^{-1}(\bar{r}_m - r_{m-1})$$

and

$$\tau := (\mu_{(k)} - \mu_{(0)})(\mathscr{D} - \mu_{(k)})^{-1}\sigma.$$

Solving the linearized minimization problem gives

$$\mu_{(k+1)} = \mu_{(k)} - \frac{(\sigma+\tau, \mathscr{D}^{-1}\bar{r}_m + (\mu_{(k)} - \mu_{(0)})\sigma)}{(\sigma+\tau, \sigma+\tau)}.$$

Integration was accomplished by applying high accuracy piecewise cubic interpolation to the integrand and by applying the lemma in Appendix B of [15] to eliminate the multiple integrations indicated in the expressions for σ and τ.)

The following four problems were solved:

(I) Single nonlinear equation:

$$y(0) = 1,$$

$$\mathscr{D}_t y = -\frac{1}{3}y^3, \qquad 0 \leqq t \leqq 5,$$

with solution $y(t) = (1+t)^{-1/2}$.

(II) Linear system of four equations:

$$\mathbf{y}(0) = X[1, 1, 1, 1]^T,$$

$$\mathscr{D}_t y = X \operatorname{diag}(-20, -15, -5, -1)X^{-1}\mathbf{y}, \qquad 0 \leqq t \leqq 1,$$

where

$$X = \begin{pmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{pmatrix}.$$

(III) Diagonal system of 10 equations:

$$\mathbf{y}(0) = [1, 1, \cdots, 1]^T,$$

$$\mathscr{D}_t \mathbf{y} = \operatorname{diag}(-1, -2, \cdots, -10)\mathbf{y}, \qquad 0 \leqq t \leqq 1.$$

(IV) Artificial nonlinear problem of Krogh (see [4]) with $\beta_1 = 20$, $\beta_2 = 15$, $\beta_3 = 5$, $\beta_4 = 1$.

In Figs. 2-5 we plot for each of Problems (I)-(IV) the value of the shift parameter μ_k as a function of k. Although it is not too evident from the plots, one can discern in Fig. 3 a clustering of the shift parameters about the eigenvalues -20, -15, -5, -1 of the Jacobian matrix for Problem (II).

Performance plots similar to that for the orbit problem are given in Figs. 6-9 for each of Problems (I)-(IV). The dotted graph represents the number of function evaluations for the sequential code ODE, and the dashed and solid lines represent the number of waveform iterations for Picard and adaptively shifted Picard, respectively. No plot is given for Picard on Problem (IV) because of overflow on an intermediate iteration. Data from these figures is used to construct Tables 2-5, which gives the speedup of the waveform methods over ODE for various accuracies.

If we want to do a double shift adaptively, then the procedure could be quite involved. The general case is described in Appendix C of [15]. Here we describe a somewhat more special case: Assume that we have obtained y_0 and $r_0 \approx \mathscr{F}\{y_0\}$. Using guesses $\tilde{\nu} \pm \tilde{\omega}^{1/2}$, determine

$$\tilde{y}_1 := y_0 + \tfrac{1}{2}[(\mathscr{D} - \tilde{\nu} + \tilde{\omega}^{1/2})^{-1} + (\mathscr{D} - \tilde{\nu} - \tilde{\omega}^{1/2})^{-1}]r_0$$

and $\tilde{r}_1 := \mathscr{F}\{\tilde{y}_1\}$. An estimate of what we would have obtained had we used ν instead

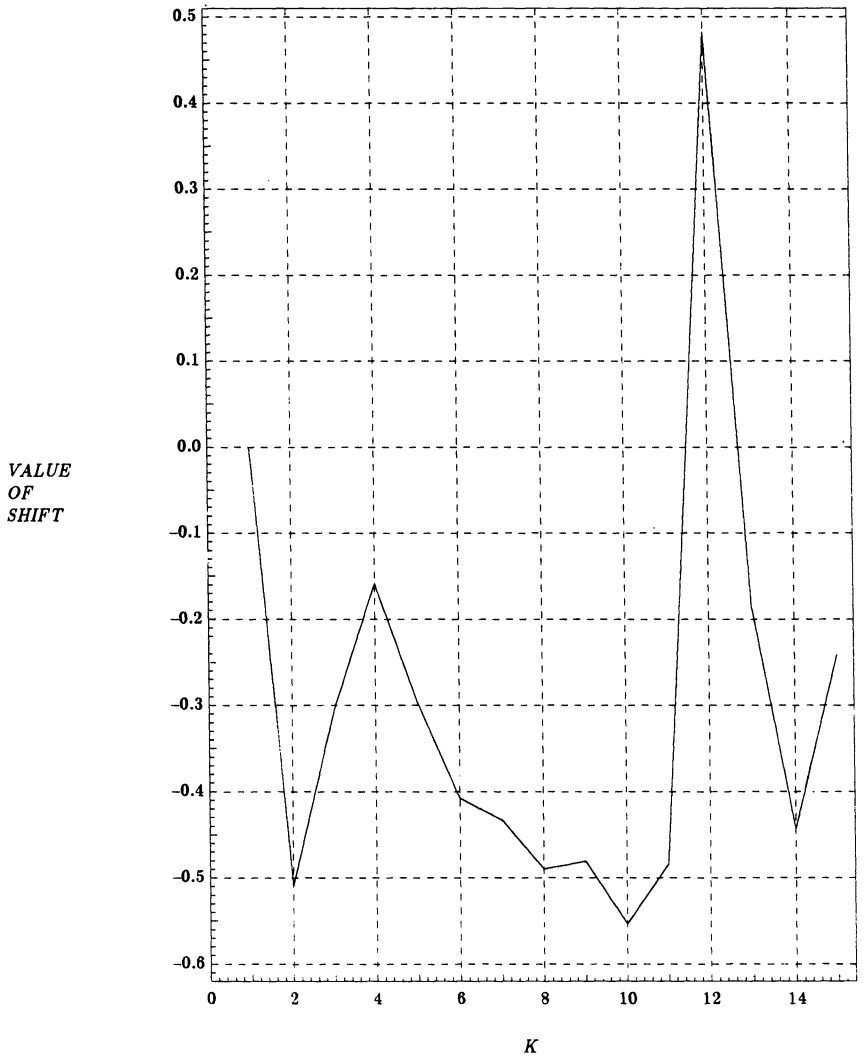


FIG. 2. *Adaptive shift value for single nonlinear equation.*

TABLE 2
Speedup for single nonlinear equation.

	Digits of accuracy						
Method	2	3	4	5	6	7	8
Picard	2.0	2.0	2.6	3.1	3.6	3.8	5.3
shifted Picard	4.0	4.0	5.4	5.9	6.3	6.4	9.0

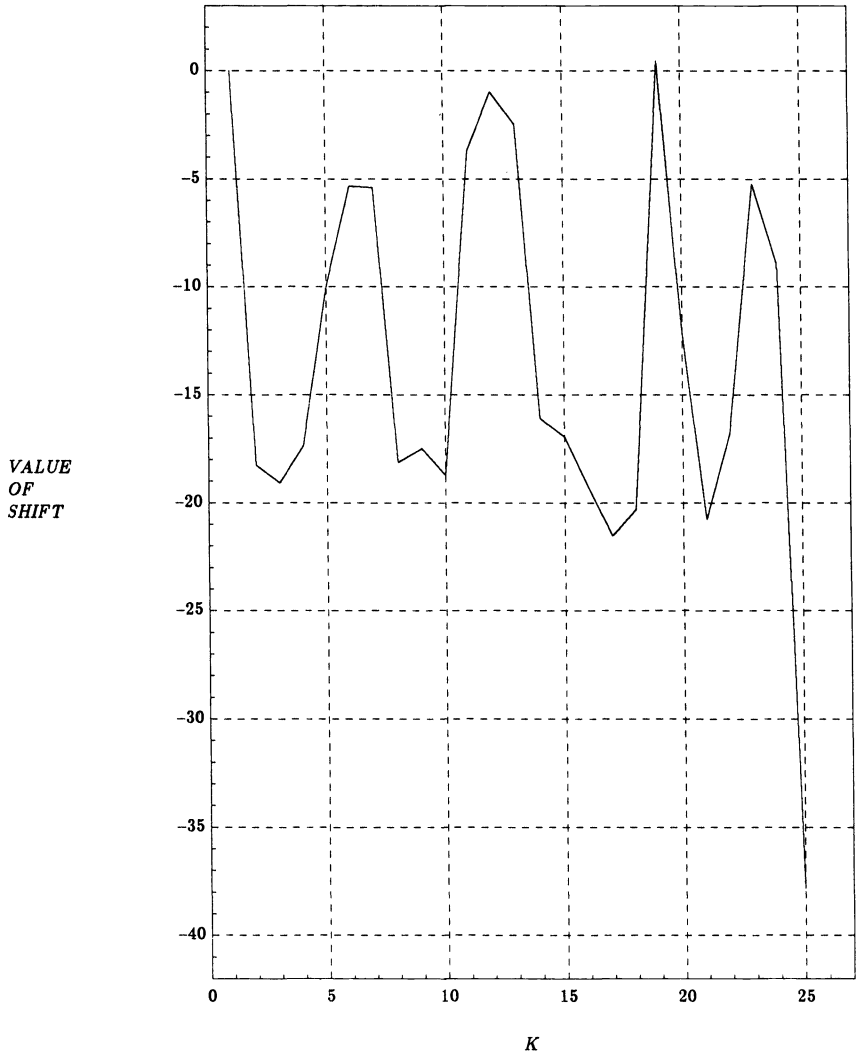


FIG. 3. *Adaptive shift value for linear system of* 4 *equations.*

TABLE 3
Speedup for linear system of 4 *equations.*

Method	Digits of accuracy					
	2	3	4	5	6	7
Picard	0.9	0.9	1.1	1.4	1.7	1.8
shifted Picard	4.7	4.7	5.0	5.7	6.6	5.9

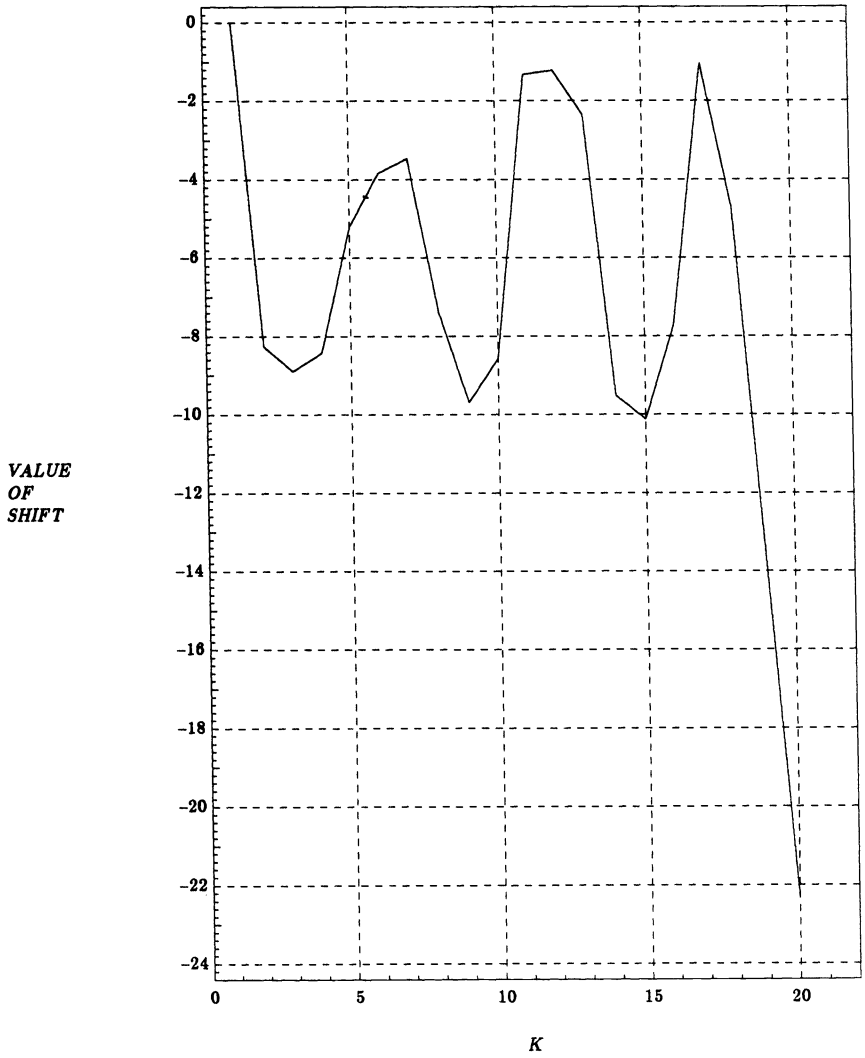


FIG. 4. *Adaptive shift value for diagonal system of* 10 *equations.*

TABLE 4
Speedup for diagonal system of 10 *equations.*

	Digits of accuracy						
Method	2	3	4	5	6	7	8
Picard	0.9	1.0	1.2	1.7	1.8	2.3	2.4
shifted Picard	3.6	3.2	3.5	5.0	4.8	6.3	6.2

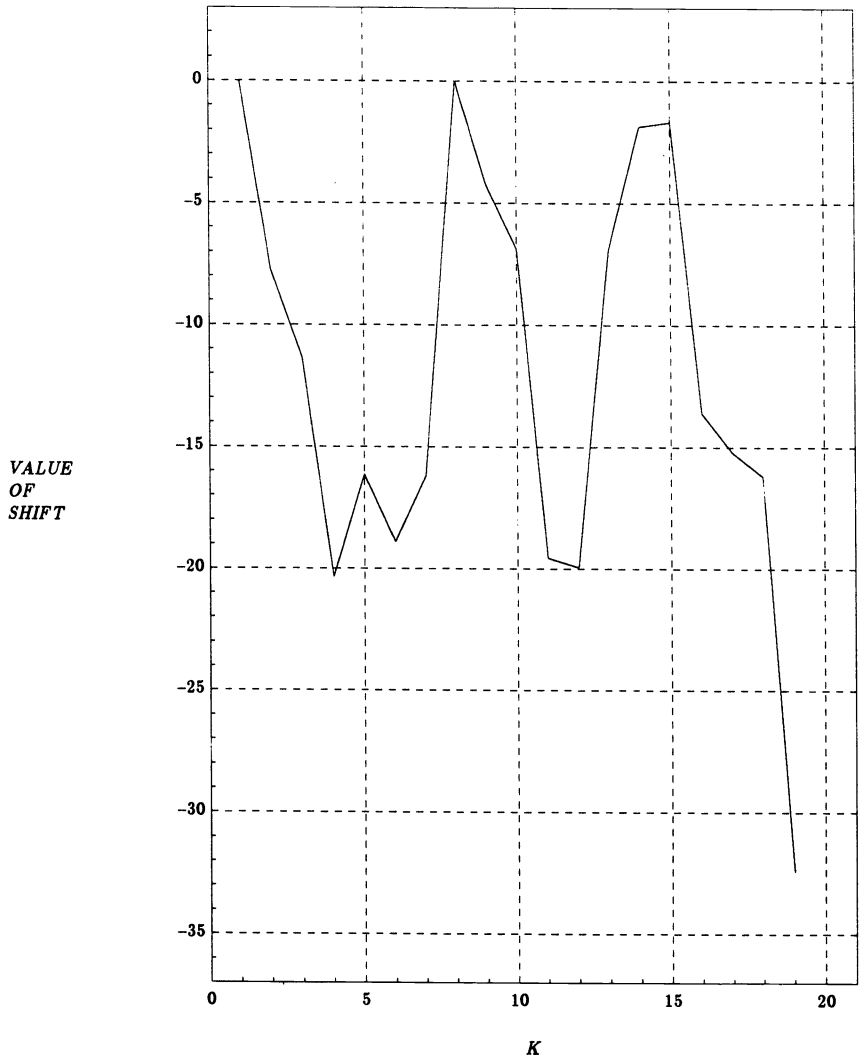


FIG. 5. *Adaptive shift value for Krogh's nonlinear system.*

TABLE 5
Speedup for Krogh's nonlinear system.

	Digits of accuracy					
Method	1	2	3	4	5	6
shifted Picard	3.9	3.5	3.8	5.0	5.4	6.2

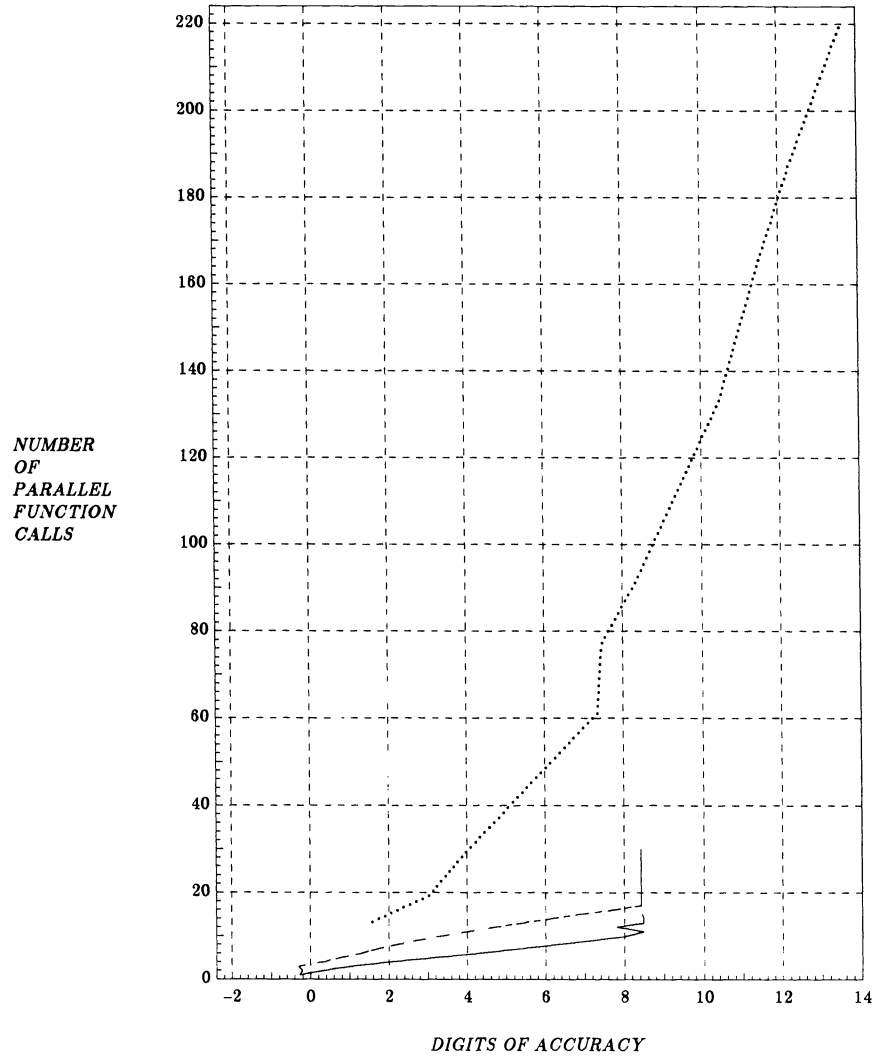


FIG. 6. *Number of function evaluations for single nonlinear equation for* ODE, *Picard, and shifted Picard, respectively.*

of $\tilde{\nu}$ is given by

$$\bar{r}_1 := \tilde{r}_1 + (\nu - \tilde{\nu})[2\mathscr{L}_\nu^{-1} - (\mathscr{D} - \tilde{\nu})^{-1} - (\nu - \tilde{\nu})\mathscr{L}_\nu^{-1}(\mathscr{D} - \tilde{\nu})^{-1}](\tilde{r}_1 - r_0)$$

where

$$\mathscr{L}_\nu^{-1} := \tfrac{1}{2}[(\mathscr{D} - \nu + \tilde{\omega}^{1/2})^{-1} + (\mathscr{D} - \nu - \tilde{\omega}^{1/2})^{-1}].$$

Choose ν to minimize $\bar{r}_1$ in some sense, and then use $\nu \pm \tilde{\omega}^{1/2}$ in a double shift, obtaining

$$\bar{y}_1 := y_0 + \mathscr{L}_\nu^{-1} r_0,$$

$$\bar{y}_2 := \bar{y}_1 + (\mathscr{D} - \nu)^{-1}\bar{r}_1 + \tilde{\omega}(\mathscr{D} - \nu)^{-2}(\bar{y}_1 - y_0),$$

and $\bar{r}_2 =: \mathscr{F}\{\bar{y}_2\}$. An estimate of what would have been obtained had ω been used

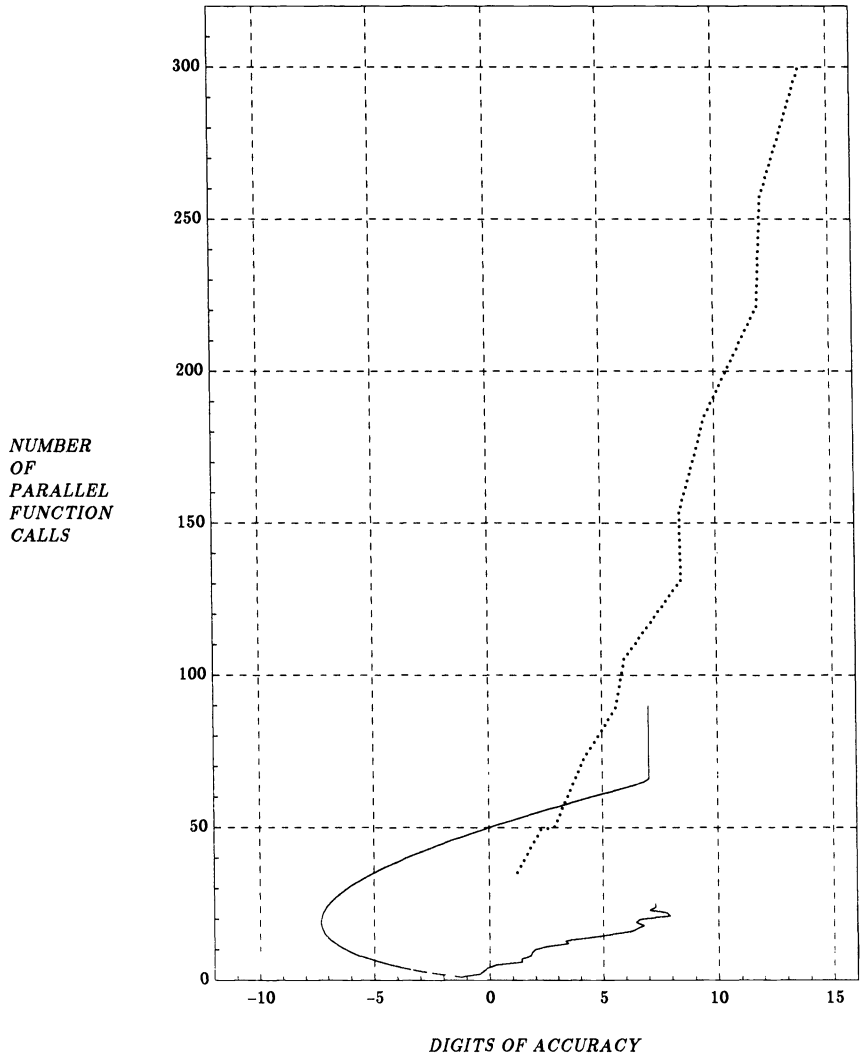


FIG. 7. *Number of function evaluations for linear system of* 4 *equations for* ODE, *Picard, and shifted Picard, respectively.*

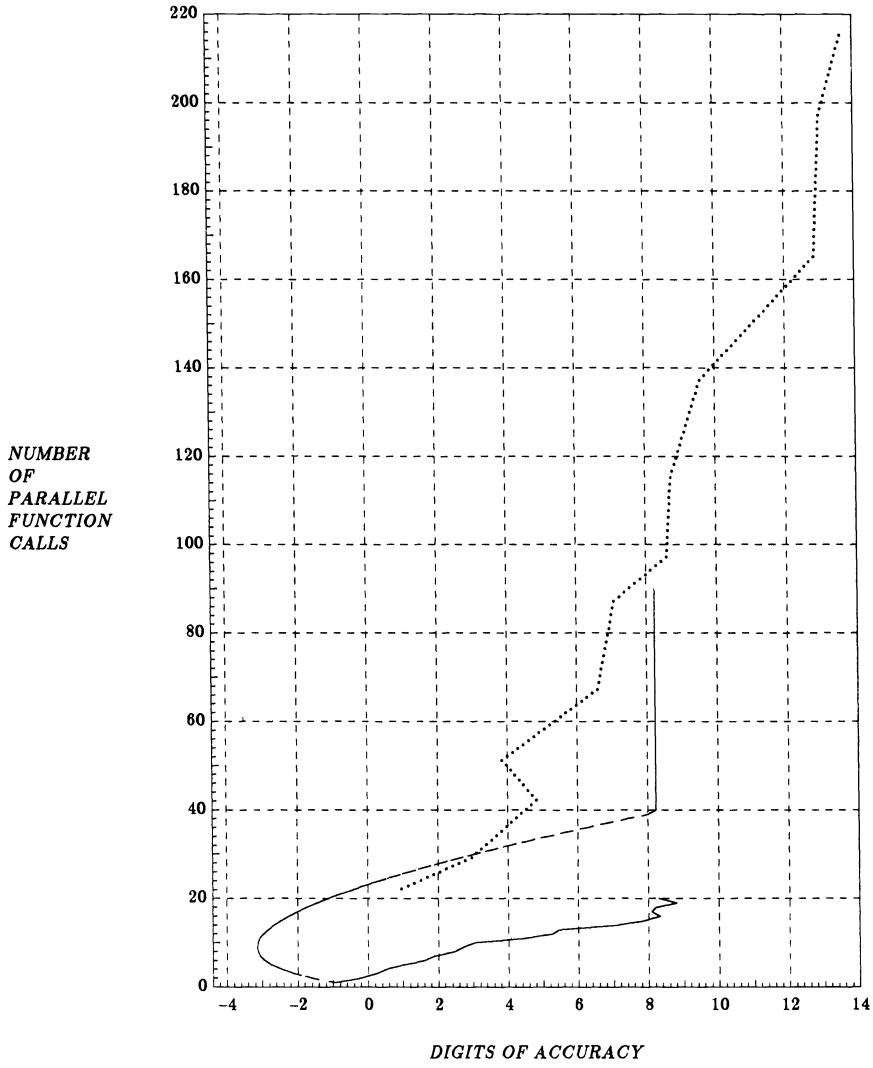


FIG. 8. *Number of function evaluations for diagonal system of* 10 *equations for* ODE, *Picard, and shifted Picard, respectively.*

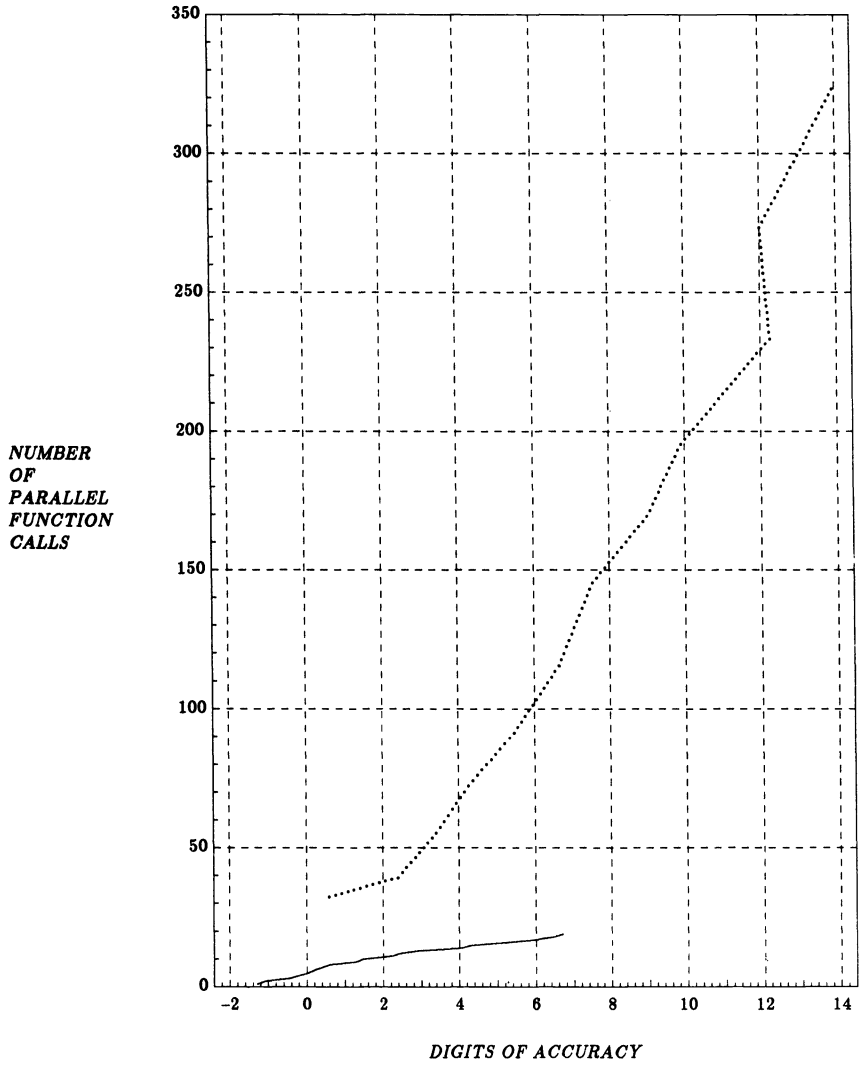


FIG. 9. *Number of function evaluations for Krogh's nonlinear system for* ODE *and shifted Picard, respectively.*

instead of $\tilde{\omega}$ is given by

$$r_2 = \bar{r}_2 + (\omega - \tilde{\omega})(\mathcal{D}^2 - 2\nu\mathcal{D} + \nu^2 - \omega)^{-1}(\bar{r}_2 - r_0).$$

Choose ω to minimize this in some sense and set

$$y_2 = \bar{y}_2 + (\omega - \tilde{\omega})(\mathcal{D}^2 - 2\nu\mathcal{D} + \nu^2 - \omega)^{-1}(\bar{y}_2 - y_0).$$

Acknowledgment. The author is grateful to H.-W. Tam for implementing the adaptively shifted waveform Picard iteration and performing the numerical experiments on the Alliant FX/8 at the Center for Supercomputing Research and Development.

REFERENCES

[1] L. G. BIRTA AND O. ABOU-RABIA, *Parallel block predictor-corrector methods for* ODEs, IEEE Trans. Comput., 36 (1987), pp. 299-311.

[2] M. T. CHU AND H. HAMILTON, *Parallel solutions of* ODE's *by multi-block methods*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 342-353.

[3] T. N. GAMBILL AND R. D. SKEEL, *Logarithmic reduction of the wrapping effect with application to ordinary differential equations*, SIAM J. Numer. Anal., 25 (1988), pp. 153-162.

[4] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[5] ———, *The potential for parallelism in ordinary differential equations*, Report R-86-1246, Department of Computer Science, University of Illinois, Urbana, IL, 1986.

[6] ———, *Parallel methods for ordinary differential equations*, Report R-87-1369, Department of Computer Science, University of Illinois, Urbana, IL, 1987.

[7] S. HORIGUCHI, Y. KAWAZOE, AND H. NARA, *A parallel algorithm for the integration of ordinary differential equations*, in Proc. Internat. Conference on Parallel Processing, 1984, pp. 465-469.

[8] R. JELTSCH AND O. NEVANLINNA, *Stability of explicit time discretizations for solving initial value problems*, Numer. Math., 37 (1981), pp. 61-91.

[9] E. LEHARASMEE, A. E. RUEHLI, AND A. L. SANGIOVANNI-VINCENTELLI, *The wavefront relaxation method for time domain analysis of large integrated circuits*, IEEE Trans. on CAD of IC and Syst. 1, 131-141 (1983).

[10] T. A. MANTEUFFEL, *An adaptive procedure for estimating parameters for the nonsymmetric Chebyshev iteration*, Numer. Math., 31 (1978), pp. 183-208.

[11] U. MIEKKALA AND O. NEVANLINNA, *Convergence of dynamic iteration methods for initial value problems*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 459-482.

[12] F. R. MOULTON, *New Methods in Exterior Ballistics*, University of Chicago, Chicago, IL, 1926.

[13] J. NIEVERGELT, *Parallel methods for integrating ordinary differential equations*, Comm. ACM, 7 (1964), pp. 731-733.

[14] L. F. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations*, W. H. Freeman, San Francisco, CA, 1975.

[15] R. D. SKEEL, *Waveform iteration and the shifted Picard splitting*, Report No. 700, Center for Supercomputing Research and Development, University of Illinois, Urbana, IL, 1987.

[16] J. WHITE AND A. L. SANGIOVANNI-VINCENTELLI, *Partitioning algorithms and parallel implementations of waveform relaxation algorithms for circuit simulation*, in Proc. Internat. Symposium on Circuits and Systems, 1985, pp. 221-224.

[17] P. B. WORLAND, *Parallel methods for the numerical solution of ordinary differential equations*, IEEE Trans. Comput. 25 (1976), pp. 1045-1048.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 777–786, July 1989
©1989 Society for Industrial and Applied Mathematics
010

THE REDUCED BASIS METHOD FOR INCOMPRESSIBLE VISCOUS FLOW CALCULATIONS*

JANET S. PETERSON†

Abstract. The reduced basis method is a type of reduction method that can be used to solve large systems of nonlinear equations involving a parameter. In this work, the method is used in conjunction with a standard continuation technique to approximate the solution curve for the nonlinear equations resulting from discretizing the Navier–Stokes equations by finite–element methods. This paper demonstrates that the reduced basis method can be implemented to approximate efficiently solutions to incompressible viscous flows. Choices of basis vectors, issues concerning the implementation of the method, and numerical calculations are discussed. Two fluid flow calculations are considered, the driven cavity problem and flow over a forward facing step.

Key words. reduced basis method, Navier–Stokes, continuation, finite elements

AMS(MOS) subject classifications. 76D05, 65H10, 65N30

1. Introduction.

The numerical treatment of nonlinear problems that arise in areas such as structural and fluid mechanics often requires solving large systems of nonlinear equations that involve a parameter. Many methods have been proposed that attempt to solve these systems efficiently; one such class of methods are reduction methods that reduce the degrees of freedom.

The discretization of nonlinear partial differential equations usually involves functions that have little to do with the differential equation. For example, piecewise polynomials are used in finite–element methods, grid functions are used in finite–difference methods, and Chebyshev polynomials are used in spectral methods. The reduced basis method is a reduction-type method that uses approximating functions that are closely related to the solution of the differential equation. For example, two choices of reduced spaces are the Taylor space and the Lagrange space, i.e., the solution at a point along with its derivatives with respect to the parameter and the solution to the system at various values of the parameter.

Apparently, the reduced basis method was proposed first in [1] and [2] and used for the structural analysis of beams and arches. The idea of using the Taylor basis functions was introduced in [3], and the method was implemented for structural problems in [4] and [5]. The error behavior of the technique was investigated in [6] and [7]. High Reynolds number solutions to the driven cavity problem were obtained in [8] by using the technique. In addition, the applicability of the method for solving differential algebraic equation systems was studied in [9].

The goal of this paper is to demonstrate that the reduced basis method can be implemented to approximate high Reynolds number solutions for incompressible viscous flow problems. Possible choices for the basis vectors are given in §2. The reduced basis method is explained in §3 in the context of solving the Navier–Stokes equations. The equations are written in terms of the streamfunction and vorticity and then discretized by using finite elements. Neither the choice of the streamfunction–vorticity formulation nor the method of discretization was mandatory. The equations that must be solved to obtain the Taylor basis vectors are given as well as the reduced

* Received by the editors July 18, 1988; accepted for publication September 28, 1988.

† Computing and Communications Division, Los Alamos National Laboratory, Los Alamos, New Mexico 87545.

basis system of equations. In §4 the method is applied to two fluid flow calculations, the driven cavity problem and flow over a forward facing step. Various numerical aspects of the calculations are discussed.

2. Choices of basis vectors. A system of discretized nonlinear equations that involve a parameter can be written in the form

$$\mathcal{F}(u,\lambda)=0 \quad \text{for } \lambda\in\mathbb{R}^1,\ u\in \mathrm{X}, \quad \text{and} \quad \mathcal{F}:\mathrm{X}\times\mathbb{R}^1\to\mathrm{X}\,, \tag{2.1}$$

where X is a finite–dimensional space. If standard finite–element techniques were used to discretize this problem, then the choice of X would be, for example, a piecewise polynomial space. The reduced basis method makes a different choice for the approximating space. Let X_R be the reduced basis space that satisfies $\mathrm{X}_R\subset\mathrm{X}$. In the remainder of this section, three choices for X_R are discussed.

The first choice is the Taylor space, or what is often called the moving frame. This choice has been used almost exclusively in the literature [3]–[5]. It is assumed that for some value of λ, say λ_0, the solution to (2.1) is known; call this solution u_0. Next, equations for $\partial^k u/\partial\lambda^k$ are obtained by differentiating (2.1) with respect to the parameter and then evaluating at λ_0. If u_k denotes $\partial^k u/\partial\lambda^k$ for $k=1,\cdots,p$, then u_k is obtained by solving an equation of the form

$$\mathcal{F}_u(u_0,\lambda_0)u_k=\mathcal{G}_k(u_0,u_1,\cdots,u_{k-1},\lambda_0)\,. \tag{2.2}$$

For example, u_1 satisfies the equation

$$\mathcal{F}_u(u_0,\lambda_0)u_1=-\mathcal{F}_\lambda(u_0,\lambda_0)\,.$$

The space X_R is the $(p+1)$-dimensional space defined by $\operatorname{span}\{u_0,u_1,\cdots,u_p\}$. It is important to note that even though the computation of u_0 requires the solution of a large nonlinear system, the evaluation of each u_k, $k=1,\cdots,p$, requires solving a linear system. Moreover, the coefficient matrix is the same, so that the computation of u_k, $k=2,\cdots,p$, requires only backsolves.

A second choice of X_R is the Lagrange space. In this case u_k, $k=0,\cdots,p$, are solutions to (2.1) for various values of λ; hence, all basis vectors are generated by solving the large nonlinear system (2.1). However, after the vectors have been initially generated, the process of updating can be accomplished by updating one vector at a time instead of generating $(p+1)$ new vectors. A possible advantage of using this space over the Taylor space is that for some equations generating the right-hand sides of (2.2) can be quite complicated.

Another possible choice for the reduced basis subspace is to use discrete least squares. As in the choice of the Lagrange subspace, the solution to the nonlinear system must be known for different values of λ. See [7] for details.

3. Reduced basis method for the Navier–Stokes equations. In this section the reduced basis method is formulated for a specific problem, i.e., incompressible viscous flows. The Navier–Stokes equations govern the flow of an incompressible viscous fluid. When written in terms of the streamfunction ψ and the vorticity ς, the time-independent equations are

$$\Delta\psi=-\varsigma \qquad \text{in } \Omega, \tag{3.1}$$

$$-\Delta\varsigma+\mathcal{R}\left(\frac{\partial\psi}{\partial y}\frac{\partial\varsigma}{\partial x}-\frac{\partial\psi}{\partial x}\frac{\partial\varsigma}{\partial y}\right)=\operatorname{curl}\mathbf{f} \qquad \text{in } \Omega\,. \tag{3.2}$$

Here $\mathcal{R}$ is the Reynolds number, $\mathbf{f}$ is the body force, and Ω is a bounded region in $\mathbb{R}^2$ whose boundary is given by Γ. In addition to satisfying (3.1) and (3.2), ψ must also satisfy certain boundary conditions. For example,

$$\psi = q \qquad \text{on} \quad \Gamma \tag{3.3}$$

and

$$\frac{\partial \psi}{\partial n} = a \qquad \text{on} \quad \Gamma, \tag{3.4}$$

where q and a are prescribed. In this treatment $q = 0$; inhomogeneous conditions can be handled in the usual way. For a discussion of other boundary treatments see [10].

In this work a finite–element discretization of (3.1)–(3.4) is chosen; however, other methods of discretization can be implemented with the reduced basis technique. Before stating the weak formulation of (3.1)–(3.4) under consideration, it is necessary to give several definitions. $H^1(\Omega)$ is the usual Sobolev space of real-valued functions having one derivative belonging to $L^2(\Omega)$; $H_0^1(\Omega)$ is the constrained space given by $\{\eta \in H^1(\Omega) : \eta = 0 \text{ on } \Gamma\}$. See [11] for details. The bilinear forms $a(\cdot,\cdot)$ and $b(\cdot,\cdot)$ and the trilinear form $c(\cdot,\cdot,\cdot)$ are defined by

$$a(\varsigma,\eta) = -\int_\Omega \varsigma\eta \, d\Omega \qquad \forall\, \varsigma,\eta \in H^1(\Omega),$$

$$b(\psi,\eta) = \int_\Omega \mathbf{grad}\, \psi \cdot \mathbf{grad}\, \eta \, d\Omega \qquad \forall\, \psi \in H_0^1(\Omega),\ \eta \in H^1(\Omega),$$

and

$$c(\phi,\psi,\varsigma) = \int_\Omega \left(\frac{\partial \psi}{\partial y}\frac{\partial \varsigma}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \varsigma}{\partial y} \right) \phi \, d\Omega \qquad \forall\, \phi,\psi \in H_0^1(\Omega),\ \varsigma \in H^1(\Omega).$$

In addition, let $g(\cdot)$ and $f(\cdot)$ be the linear functions given by

$$g(\eta) = \int_\Gamma a\eta \, d\Gamma \qquad \forall\, \eta \in H^1(\Omega)$$

and

$$f(\phi) = \int_\Omega \phi \, \text{curl}\, \mathbf{f} \, \mathbf{d\Omega} \qquad \forall\, \phi \in \mathbf{H_0^1(\Omega)}.$$

A weak formulation of (3.1)–(3.4) is the following. Seek $\psi \in H_0^1(\Omega)$ and $\varsigma \in H^1(\Omega)$ such that

$$a(\varsigma,\eta) + b(\psi,\eta) = g(\eta) \qquad \forall\, \eta \in H^1(\Omega), \tag{3.5}$$

$$b(\varsigma,\phi) + \mathcal{R}\, c(\phi,\psi,\varsigma) = f(\phi) \qquad \forall\, \phi \in H_0^1(\Omega). \tag{3.6}$$

The standard finite–element discretization of (3.5)–(3.6) is to seek the solution in subspaces of $H_0^1(\Omega)$ and $H^1(\Omega)$; i.e., choose $V^h \subset H_0^1(\Omega)$ and $S^h \subset H^1(\Omega)$ and seek $\psi^h \in V^h$, $\varsigma^h \in S^h$ satisfying

$$a(\varsigma^h,\eta^h) + b(\psi^h,\eta^h) = g(\eta^h) \qquad \forall\, \eta^h \in S^h, \tag{3.7}$$

$$b(\varsigma^h,\phi^h) + \mathcal{R}\, c(\phi^h,\psi^h,\varsigma^h) = f(\phi^h) \qquad \forall\, \phi^h \in V^h. \tag{3.8}$$

Now the reduced basis method uses reduced spaces, say $V_R^h \subset V^h$ and $S_R^h \subset S^h$. The various choices for these spaces were discussed in § 2. The Lagrange space consists of solutions to (3.7)–(3.8) for different values of the Reynolds number. The Taylor space consists of the solution $(\psi_0^h, \varsigma_0^h)$ to (3.7)–(3.8) at a given Reynolds number, $\mathcal{R}_0$. The remaining vectors are obtained by solving the equations that result by differentiating (3.7)–(3.8) with respect to the Reynolds number and evaluating at $\mathcal{R}_0$. If $\psi_k^h = \partial^k \psi^h / \partial \mathcal{R}^k$ and $\varsigma_k^h = \partial^k \varsigma^h / \partial \mathcal{R}^k$ for $k = 1, \cdots, p$, then ψ_k^h and ς_k^h satisfy the following equations. Seek $\psi_k^h \in V^h$ and $\varsigma_k^h \in S^h$, $k = 1, \cdots, p$, satisfying

$$a(\varsigma_k^h, \eta^h) + b(\psi_k^h, \eta^h) = 0 \qquad \forall\, \eta^h \in S^h\,, \tag{3.9}$$

$$\begin{aligned} &b(\varsigma_k^h, \phi^h) + \mathcal{R}\left(c(\phi^h, \psi_k^h, \varsigma_0^h) + c(\phi^h, \psi_0^h, \varsigma_k^h)\right) \\ &\qquad = \int_\Omega \mathcal{G}_k\left(\psi_0^h, \cdots, \psi_{k-1}^h, \varsigma_0^h, \cdots \varsigma_{k-1}^h\right) \phi^h \, d\Omega \qquad \forall\, \phi^h \in V^h\,. \end{aligned} \tag{3.10}$$

For example,

$$\mathcal{G}_1 = \frac{\partial \psi_0^h}{\partial x}\frac{\partial \varsigma_0^h}{\partial y} - \frac{\partial \psi_0^h}{\partial y}\frac{\partial \varsigma_0^h}{\partial x}\,.$$

In obtaining these equations, we assume that the boundary value a and the body force $\mathbf{f}$ do not depend on the Reynolds number. The reduced basis space $V_R^h \times S_R^h$ is defined as the span of the vectors $\{(\psi_0^h, \varsigma_0^h), \cdots, (\psi_p^h, \varsigma_p^h)\}$. The reduced basis solution $(\psi_R^h, \varsigma_R^h)$ is the linear combination of these vectors found by solving the following nonlinear equations. Seek $(\psi_R^h, \varsigma_R^h) \in V_R^h \times S_R^h$ satisfying

$$a(\varsigma_R^h, \eta^h) + b(\psi_R^h, \eta^h) = g(\eta^h) \qquad \forall\, \eta^h \in S_R^h\,, \tag{3.11}$$

$$b(\varsigma_R^h, \phi^h) + \mathcal{R}\, c(\phi^h, \psi_R^h, \varsigma_R^h) = f(\phi^h) \qquad \forall\, \phi^h \in V_R^h\,. \tag{3.12}$$

For a given Reynolds number, the solution of the nonlinear equations (3.7)–(3.8) is obtained by applying an iterative method such as Newton's method. This yields a large, sparse linear system that must be solved for each iteration. For example, if the problem is posed on the unit square, which has been divided into quadrilaterals that are then divided into two triangles, and continuous linear functions over the triangles are chosen for V^h and S^h, then the resulting system for a 25×25 mesh has 1154 unknowns with a bandwidth of 103. The solution of (3.11)–(3.12) results in a dense system of equations. This is due to the fact that the reduced basis vectors are *global* as opposed to the *local* basis vectors used in standard finite elements. Hence the reduced basis method is feasible only if a very small number of basis vectors are needed. This is shown to be the case in §4.

4. Numerical results. In this section two viscous incompressible flow problems are solved by using the reduced basis method. The first is the driven cavity problem. Although this problem is not a physical one, it is chosen for its popularity in the literature. The second problem is flow over a forward facing step.

For the driven cavity problem the physical region is the unit square, and the boundary conditions are

$$\psi = 0 \quad \text{on } \Gamma \qquad \text{and} \qquad \frac{\partial \psi}{\partial n} = \begin{cases} -1 & \text{for } y = 1 \\ 0 & \text{elsewhere.} \end{cases}$$

The region Ω is the unit square, which is subdivided into quadrilaterals that are then divided into two triangles by drawing a diagonal. In this example, the finite–element spaces V^h and S^h are chosen as the space of continuous piecewise linear functions over the triangles. In all computations a nonuniform 25×25 mesh is used.

Initially, the solution at a low Reynolds number was generated from (3.7)–(3.8), and then the remaining vectors in the Taylor basis space were generated from (3.9)–(3.10). Once these basis vectors were obtained, then the solution curve for the reduced problem, given by (3.11)–(3.12), was traced out to a prescribed value of the Reynolds number by using a continuation method such as PITCON [12]. Last, the solution to the reduced problem was used as a starting guess to solve the nonlinear equations (3.7)–(3.8). Usually, the reduced solution was a good enough initial guess that Newton's method took only a very few iterations to converge. The procedure begins again by using this solution to the full set of equations to generate a new set of basis vectors. An alternative to using the reduced basis vectors would be to solve the full set of nonlinear equations (3.7)–(3.8) by a continuation method. Of course, this would involve the solution of many large nonlinear systems as opposed to the solution of the large nonlinear system at only a few values of the Reynolds number, which the reduced basis method requires.

In Table 1 the L^2-norms of the solution and the first four Taylor basis vectors are given for a few values of the Reynolds number. Here χ is the solution $(\psi_0^h, \varsigma_0^h)$ at the given Reynolds number, χ' is the derivative of the solution with respect to the Reynolds number, etc. As Table 1 illustrates, the norms of the corresponding basis vectors are decreasing with the Reynolds number as well as with the number of derivatives taken.

TABLE 1

L^2-norms of the Taylor basis vectors for the driven cavity problem.

Re	$\|\chi\|_2$	$\|\chi'\|_2$	$\|\chi''\|_2$	$\|\chi'''\|_2$	$\|\chi''''\|_2$
100	5.71	$.979\times10^{-2}$	$.999\times10^{-4}$	$.176\times10^{-5}$	$.475\times10^{-7}$
500	6.27	$.398\times10^{-2}$	$.156\times10^{-4}$	$.680\times10^{-7}$	$.577\times10^{-9}$
1000	6.75	$.161\times10^{-2}$	$.161\times10^{-5}$	$.295\times10^{-8}$	$.790\times10^{-11}$

Since the reduced basis method results is a dense system of equations, it is efficient only if a small number of basis vectors are needed. Figures 1 and 2 compare the reduced solution using three and five Taylor basis vectors to the full solution, i.e., the solution to (3.7)–(3.8), at Reynolds number 400 and 1000, respectively. Specifically, the u-velocity profile, i.e., $\partial\psi^h/\partial y$, through the geometric center for the full solution is compared to the reduced basis solution, i.e., the solution to (3.11)–(3.12) found by continuing in the Reynolds number. The reduced solution was generated by using three and five basis vectors. For a comparison, the figures also have a plot of the full solution at the point where the basis vectors were generated. For Fig. 1 the basis vectors were generated at Reynolds number 100, and for Fig. 2 the basis vectors were generated at Reynolds number 400.

In [8] Peterson obtained solutions to the driven cavity problem for Reynolds numbers as high as 5000, by using a 19×19 nonuniform mesh and five basis vectors. These results were compared to calculations done in [13] in which a uniform 257×257 grid was used. The reduced basis results agreed favorably with these fine mesh calculations.

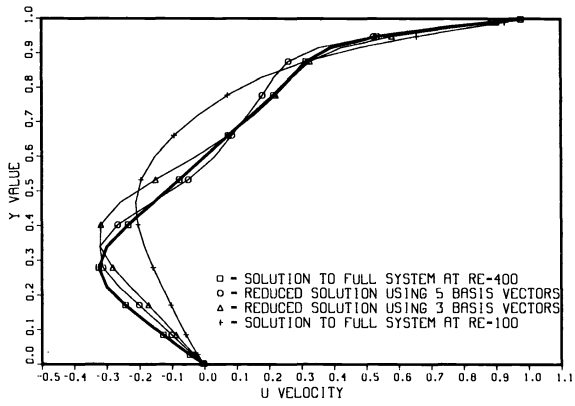


FIG. 1. *Comparison of reduced solution to full solution at* Re=400.

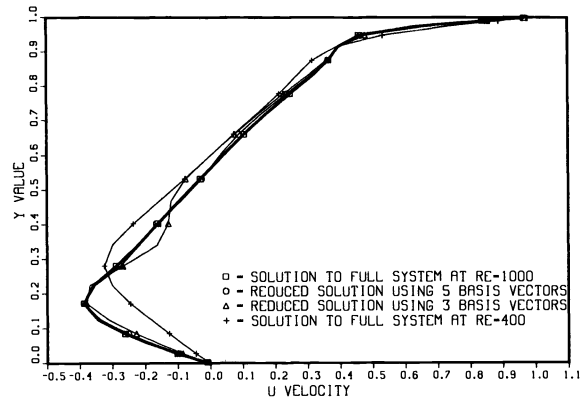


FIG. 2. *Comparison of reduced solution to full solution at* Re=1000.

A natural question that arises when the Taylor basis vectors are used is how the reduced solution compares with the Taylor polynomial. Note that for the Taylor polynomial the coefficients in the expansion of the basis vectors are fixed; i.e., the kth coefficient is given by $(\mathcal{R} - \mathcal{R}_0)^k/k!$. The reduced basis solution chooses the coefficients so that (3.11)–(3.12) are satisfied. Figure 3 plots the accuracy of a few Taylor polynomials along with the accuracy of the reduced basis solution generated by using five basis vectors. The accuracy is defined as $\|\chi - T\|_2/\|\chi\|_2$, where χ represents the solution to the full system at the given Reynolds number and T represents the Taylor polynomial. The reduced basis solution and Taylor polynomials are generated at Re=100.

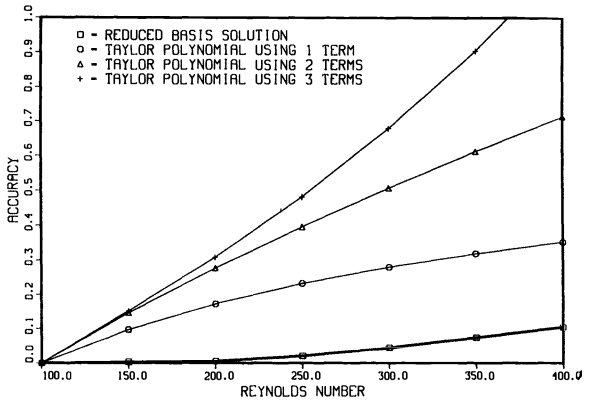


FIG. 3. *Comparison of accuracy of reduced solution and Taylor polynomials.*

The second problem is flow over a forward facing step. Figure 4 gives the physical domain of the problem along with the inflow and outflow conditions. In this example, the Dirichlet boundary condition for ψ is no longer homogeneous. This can be handled in the usual manner by modifying the right-hand sides in the weak formulations. However, when the reduced solution is computed, the coefficient of $(\psi_0^h, \varsigma_0^h)$ must be forced to be 1 so that the reduced solution will satisfy the correct boundary conditions. In the computations a nonuniform 70×35 mesh was used. Figures 5 and 6 give streamline plots for the reduced basis solution generated by using five Taylor basis vectors and the streamline plots for the full solution, respectively. These plots were generated for Reynolds number 1500. Table 2 gives the L^2-norms of the Taylor basis vectors for the step problem and illustrates that the decreasing size of the vectors is not just an artifact of the driven cavity problem. Figure 7 compares the accuracy of reduced basis solutions generated by using two through five basis vectors. The basis vectors were generated at Reynolds number 100. The accuracy is defined as the L^2-norm of the difference in the full and reduced solutions normalized by the L^2-norm of the full solution. Table 3 gives the accuracy of the reduced solution using between two and five basis vectors that were generated at Reynolds number 100. In addition, the accuracy of the L^2-projection is given in parenthesis for each case. The L^2-projection is the best approximation in the reduced space in an L^2 sense; it is defined by the following. Seek $\chi_P \in V_R^h \times S_R^h$ satisfying

$$\int_\Omega \chi_P \chi_R \, d\Omega = \int_\Omega \chi \chi_R \, d\Omega \qquad \forall \chi_R \in V_R^h \times S_R^h,$$

where χ is the solution to (3.7)–(3.8) at the given Reynolds number.

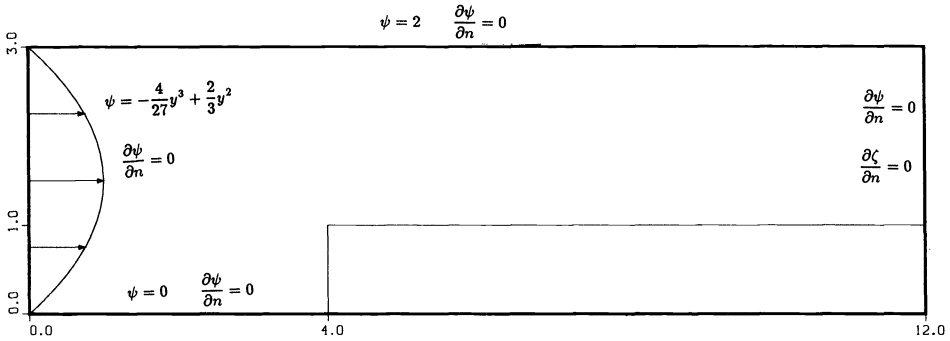


FIG. 4. *Geometry of forward facing step in a two-dimensional channel.*

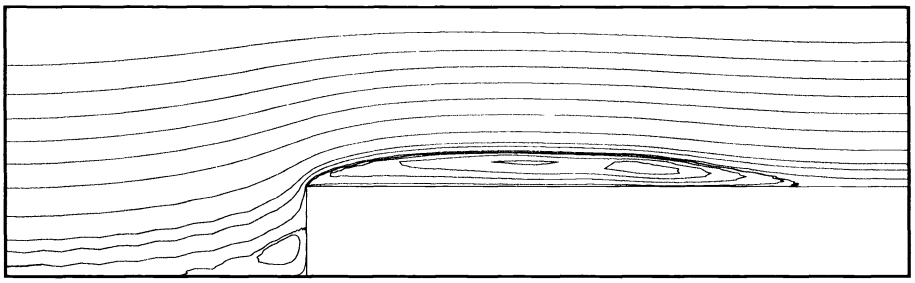


FIG. 5. *Streamline plots for reduced solution at Reynolds number* 1500.

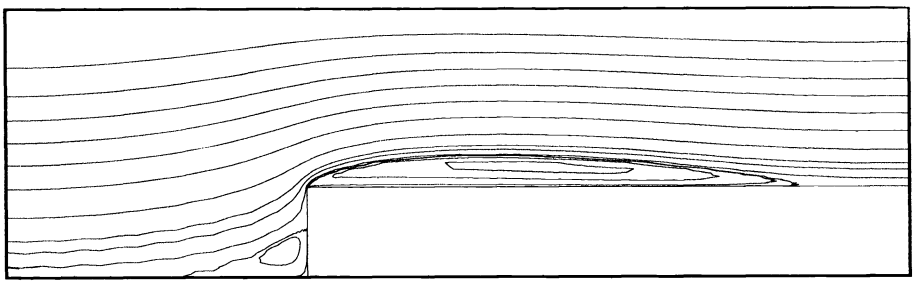


FIG. 6. *Streamline plots for full solution at Reynolds number* 1500.

TABLE 2

L^2-norms of the Taylor basis vectors for the forward facing step.

Re	$\|\chi\|_2$	$\|\chi'\|_2$	$\|\chi''\|_2$	$\|\chi'''\|_2$	$\|\chi''''\|_2$
100	10.2	.0172	$.173\times10^{-3}$	$.394\times10^{-5}$	$.127\times10^{-6}$
500	11.2	.0082	$.277\times10^{-4}$	$.163\times10^{-6}$	$.152\times10^{-8}$
1000	11.9	.0073	$.184\times10^{-4}$	$.924\times10^{-7}$	$.669\times10^{-9}$

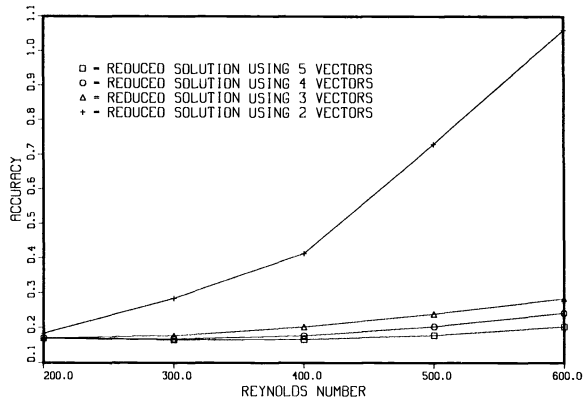


FIG. 7. *Comparison of accuracy of the reduced solutions.*

TABLE 3
Accuracy of reduced solution and L^2-projection.

Re	Number of basis vectors							
	5		4		3		2	
200	.169	(.169)	.170	(.169)	.170	(.170)	.184	(.176)
300	.165	(.164)	.167	(.166)	.177	(.172)	.283	(.201)
400	.166	(.163)	.177	(.171)	.202	(.189)	.413	(.241)
500	.178	(.172)	.203	(.188)	.239	(.219)	.730	(.287)
600	.203	(.192)	.242	(.217)	.283	(.257)	1.06	(.333)

REFERENCES

[1] B. O. ALMROTH, *Automatic choice of global shape functions in structural analysis*, AIAA J., 16 (1978), pp. 525–528.

[2] D. A. NAGY, *Modal representation of geometrically nonlinear behavior by the finite element method*, Comput. & Structures, 10 (1979), pp. 683–688.

[3] A. K. NOOR, *Reduced basis technique for nonlinear analysis of structures*, AIAA J., 18 (1980), pp. 455–462.

[4] ———, *Reduced basis technique for collapse analysis of shells*, AIAA J., 19 (1981), pp. 393–397.

[5] A. K. NOOR AND J. M. PETERS, *Recent advances in reduction methods for instability analysis of structures*, Comput. & Structures, 16 (1983), pp. 1–14.

[6] J. P. FINK AND W. C. RHEINBOLDT, *On the error behavior of the reduced basis technique for nonlinear finite element approximations*, Z. Angew. Math. Mech., 63 (1983), pp. 21–28.

[7] T. A. PORSCHING, *Estimation of the error in the reduced basis method solution of nonlinear equations*, Math. Comput., 45 (1985), pp. 487–496.

[8] J. S. PETERSON, *High Reynolds number solutions for incompressible viscous flow using the reduced basis technique*, Report 83–49, Institute for Computational Mathematics and Applications, University of Pittsburgh, 1983.

[9] M. L. LEE AND T. A. PORSCHING, *The reduced basis method for initial value problems*, Report 86-95, Institute for Computational Mathematics and Applications, University of Pittsburgh, 1986.

[10] M. D. GUNZBURGER AND J. S. PETERSON, *Finite-element methods for the streamfunction-vorticity equations: Boundary-condition treatments and multiply connected domains*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 650–668.

[11] P. ADAMS, *Sobolev Spaces*, Academic Press, New York, 1975.

[12] W. C. RHEINBOLDT AND J. BURKARDT, *A program for a locally-parametrized continuation process*, ACM–TRANS. on Math. Software, 9 (1983), pp. 215–235.

[13] U. GHIA, K. N. GHIA, AND C. T. SHIN, *High*–Re *solutions for incompressible flow using the Navier–Stokes equations and a multigrid method*, J. Comput. Phys., 48 (1982), pp. 387–395.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 4, pp. 787–815, July 1989
©1989 Society for Industrial and Applied Mathematics
011

COMPUTATIONAL STRUCTURE OF THE N-BODY PROBLEM*

JACOB KATZENELSON†

Abstract. This work considers tree algorithms for the N-body problem where the number of particles is on the order of a million. The main concern of this work is the organization and performance of these computations on parallel computers.

This work introduces a formulation of the N-body problem as a set of recursive equations based on a few elementary functions. It is shown that both the algorithm of Barnes–Hut and that of Greengard–Rokhlin satisfy these equations using different elementary functions. The recursive formulation leads directly to a computational structure in the form of a pyramid-like graph, where each vertex is a process, and each arc a communication link.

The pyramid is mapped to three different processor configurations: (1) a pyramid of processors corresponding to the processes pyramid graph; (2) a hypercube of processors, e.g., a connection-machine-like architecture; and (3) a rather small array, e.g., $2 \times 2 \times 2$, of processors faster than the ones considered in (1) and (2) above.

The main conclusion is that simulations of this size can be performed on any of the three architectures in reasonable time. Approximately 24 seconds per timestep is the estimate for a million equally distributed particles using the Greengard-Rokhlin algorithm on the CM-2 connection machine. The smaller array of processors is quite competitive in performance.

Key words. N-body problem, particle simulation, tree algorithms for particle simulation, parallel computing

AMS(MOS) subject classifications. 70-08, 70F15

1. Introduction.

The study of physical systems by particle simulation is called the "many-body" or the "N-body" problem. Such studies are conducted in celestial mechanics, plasma physics, and fluid mechanics, as well as in semiconductor device simulation [1]. The simulation finds the trajectories of each particle over some time interval, given the initial position, the initial velocity, the external force, and the nature of the forces that the particles exert on each other.

The number of particles used for such studies is quite large. It is estimated that to get insight into three-dimensional turbulent flow about one million particles are needed [2]. Thus, such simulations require intensive and prolonged computations, and the question of efficiency is of great interest.

This work considers a family of algorithms for calculating the force or the potential Φ which can be called "tree algorithms." These algorithms reduce the asymptotic computational complexity from $O(N^2)$ in the naive approach, where each particle interacts with each other particle, to $O(N \log N)$ in [5],[6]; down to $O(N)$ in [7] (for particles in two dimensions); and in [8] and [10] (for particles in three dimensions). The main goal of this work is the organization and performance of such algorithms on parallel computers. In particular, it considers how such computers should be organized in order to handle tree algorithms efficiently when the number of particles is very large, and how time is divided between computation time and time for communication between processors.

* Received by the editors June 20, 1988; accepted for publication (in revised form) October 28, 1988. This paper describes research done in the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Project Agency of the Department of Defense under Office of Naval Research contract N00014-86-K-0180.

† Artificial Intelligence Laboratory and Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. On sabbatical leave from the Department of Electrical Engineering, Technion — Israel Institute of Technology, Haifa 32000, Israel.

The work introduces a formulation of the many-body problem as a set of recursive equations based on a few elementary functions. It is shown that the algorithms of both [6] and [7] satisfy these equations using different elementary functions. The recursive formulation leads directly to a computational structure in the form of a pyramid-like graph, where each vertex is a process, and to a single-instruction-multiple-data (SIMD) computational algorithm.

The pyramid is mapped to three different processor configurations:

- A pyramid of processors corresponding to the processes pyramid graph, where each vertex is assigned a process and each arc is assigned a communication "wire";
- A hypercube of processors, i.e., a connection-machine-like architecture [11],[12];
- A smaller array of larger processors, e.g., $2 \times 2 \times 2$ processors.

Computation and communication time are computed for the first two cases, and estimates are derived for the third case.

The main qualitative conclusion is that large simulations (about one million particles) can be performed on any of the above architectures in a reasonable amount of time. This is particularly interesting with respect to the connection machine — an existing, ready-to-use system — but also noteworthy with respect to the array of processors, which is quite competitive in performance.

2. A recursive formulation of the N-body problem. We consider N point particles distributed in two- or three-dimensional space (2D or 3D). We are interested in the forces that these particles exert on each other. We use the terminology of celestial mechanics (point masses and Newtonian gravitational fields), although the method is equally applicable to electrostatics (electrical charges and electrostatic fields) and fluid mechanics (vortex particle and vortex field) [1].

The force that a particle at a point x, $x \in R^3$, exerts on a particle y, $y \in R^3$, is given by

$$f_{xy} = G\frac{M_x M_y}{\| x - y \|^2} 1_r, \tag{1}$$

where 1_r is a unit vector in the $x - y$ direction. The forces are additive, i.e., the total force on any particle p, F_p, is the sum of the forces due to all other particles:

$$F_p = \sum_{q \in particles, q \neq p} f_{pq}. \tag{2}$$

An additional valuable property of these forces is the symmetry, $f_{pq} = -f_{qp}$. It is also convenient to express the forces in terms of a potential function, Φ, such that the force on a unit mass at x is

$$F_x = -grad\ \Phi. \tag{3}$$

The tree algorithm proceeds by dividing the space into "computational boxes." The method is illustrated by its 2D version, which is simpler to explain; the generalization to 3D is straightforward.

We choose a (2D) rectangle such that all the particles are included in it. This is called the *top* rectangle. It is partitioned into four equal rectangles, called the *children* of the *top* rectangle. *Top* is called the *parent* of the *children* rectangles. Each child is partitioned to four equal rectangles. This process continues to an arbitrary level, say

h. For compatibility with prior work [10], the level of *top* is named the 0-level, so the bottom level becomes the hth level. (See Fig. 1.)

The set of *neighbors* of a computational box b comprises all boxes of the same size whose boundary has at least one point in common with the boundary of b. Thus, in 2D a box has at most eight neighbors; in 3D a box has 26 neighbors.

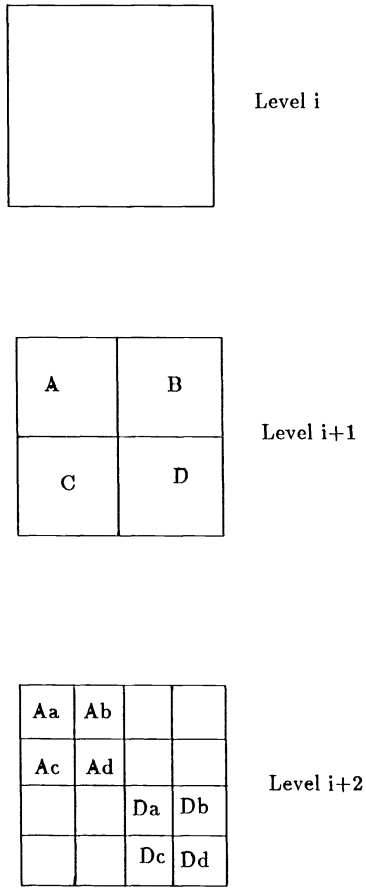


FIG. 1. *The computational box, its parent, children, and neighbors. Aa, Ab, Ac, Ad are children of A; A is a parent of Aa, etc.; the neighbors of Ad are as follows: Aa, Ab, Ba, Bb, Da, Cc, Ca, and Ac.*

It is convenient to partition the force, or the potential, into the *far field* and the *near field* so that $\Phi_p = \Phi_{p,nearfield} + \Phi_{p,farfield}$. The reasons for this partition are twofold: (1) approximation formulas valid for the far-field potential are not valid for the near-field potential; and (2) considerations related to the finite computer word size and to global accuracy necessitate modification of the near field at small distances (smoothing [3], [4]). In partitioning into far and near fields we follow [7], [8] by taking the field at computational box due to particles in itself and in its neighbor boxes to be the near-field potential; the field in the box due to all boxes that are not neighbors is the far-field potential.

Let Φ_n be the far-field potential function due to particles in the box n. For any

x, $x \notin n$ and $x \notin neighbors(n)$, $\Phi_n(x)$ is the contribution of particles in n to the potential at x.

Let Ψ_n be the far-field potential in n; i.e., for any x, $x \in n$, $\Psi_n(x)$ is the contribution of all particles q, $q \notin n$, and $q \notin neighbors(n)$ to the potential at the point x.

A recursive relation among Ψ_n and $\Psi_{parent(n)}$ is the heart of the tree methods:

$$\Psi_n = \Psi_{parent(n)} + \sum_{i \in I_n} \Phi_i, \tag{4}$$

where I_n, the set on which the summation of Φ_i is done, is called the *interaction set* of n and is defined by

$$I_n = \{j | j \in children(neighbors(parent(n))), j \notin neighbors(n)\}. \tag{5}$$

A solution of (4) is defined by its boundary conditions

$$\Psi_{top} = 0, \tag{6}$$

and by

$$\Phi_n = \sum_{i \in children(n)} \Phi_i, \tag{7}$$

where n is not a bottom-level rectangular; for a bottom-level n,

$$\Phi_n = \Phi_n^0, \tag{8}$$

where Φ_n^0 is the far-field potential function given in terms of the particles in n.

3. The Barnes–Hut and the Greengard–Rokhlin methods. The Barnes–Hut (BH in the sequel) [6] and the Greengard–Rokhlin (GR) [7], [8] methods both solve the above recursive equations. Their differences are in how the potential functions are *represented* and *approximated.*

For presenting the material in this section it is useful to distinguish between a function, say, f; its approximation, $\hat{f}$; and the representation of this function or its approximation, $rep(f)$ or $rep(\hat{f})$. Usually, we want to represent the function itself; however, for various reasons we carry only enough information to define the approximation completely. Thus, by design, $rep(\hat{f})$ represents f.

In the BH method the function Φ_n^h is represented by two quantities $(m,\vec{c})$, where m is the sum of the masses of the particles in the box n, and $\vec{c}$ is the center of mass vector. Thus, Φ_n^h at a point y, $y \notin neighbors(n)$ and $y \notin n$, is approximated by $\hat{\Phi}_n^h$, which is the potential of a mass m at $\vec{c}$ evaluated at the point y, i.e.,

$$\hat{\Phi}_n^h(y) = G\frac{m}{\| y - c \|}. \tag{9}$$

Similarly, $\hat{\Phi}_n$, for level l, $l \neq h$, approximates Φ_n. It is calculated by summing the masses of the children and finding a center of mass of the masses of the children. The representation of $\hat{\Phi}_n$ is

$$rep(\hat{\Phi}_n) = \left(\sum_{i \in children(n)} m_i, \frac{\sum_{i \in children(n)} m_i \vec{c_i}}{\sum_{i \in children(n)} m_i} \right). \tag{10}$$

The approximation to Ψ_n, $\hat{\Psi}_n$, is derived from (4); $\hat{\Psi}_n$ is represented as a *list* of pairs; the pairs $(m_i, \vec{c_i})$ of all the members of the interactive set are appended to form a list which, in turn, is appended with a similar list representing $\hat{\Psi}_{parent(n)}$.

The actual potential, or the force at a point p, is calculated by evaluating the appropriate $\hat{\Psi}_n$, n at level h, by evaluating each member of the list at p, and by adding the near-field effect to the result.

The 3D version of the GR method is described in [8] and in [10]. The much simpler 2D version suffices for showing how the algorithm fits the above computational structure.

In the sequel we identify R^2 with the complex plane C, which simplifies the notations. Let n be a box on some level. For all z, $z \notin n$ and $z \notin neighbors(n)$, the GR method considers Φ_n as its multipole expansion around the center of n

$$\Phi_n(z) = a_0 \log(z) + \sum_{k=1}^{\infty} \frac{a_k}{z^k}. \tag{11}$$

For all z, $z \in n$, the GR method considers Ψ_n as its local (Taylor) expansion around the center of n,

$$\Psi_n(z) = \sum_{l=0}^{\infty} b_l z^l. \tag{12}$$

In both cases the functions are approximated and represented by their first $p+1$ coefficients, where p is chosen according to the accuracy required.

To derive $\hat{\Phi}_n$ from the $\hat{\Phi}$'s of its children (see (7)), the child representation is "translated", i.e., each child Φ_j, $j \in children(n)$, is expanded to a multipole expansion around the center of n. Once this is done, the Φ's of all children are expanded around the same origin and Φ_n is found by summing the coefficients of the translated Φ_j.

To derive Ψ_n (see (4)), the Φ_i, $i \in$ *interactive set of* n, are converted to a *local* expansion around the center of the box n. $\Psi_{parent(n)}$ is in this local form and, therefore, the remaining operation is to translate (shift) the local expansion from the center of $parent(n)$ to the center of n. Now, all these functions are represented in n by their local expansions around the same point, and Ψ_n is obtained by summing up the coefficients of all these local expansions.

Details of the GR algorithm follow. Let n be a box of level h (bottom level). Let the center of the box be the origin of the coordinate system; let the box have m point masses $\{q_i,\ i = 1, \cdots, m\}$ located at points $\{z_i,\ i = 1, \cdots, m\}$ in the box. Let r be the diameter of the box; thus, $|z_i| \leq r$. Then, for any $z \in C$ with $|z| > r$,

$$\Phi_n^h(z) = a_0 \log(z) + \sum_{k=1}^{\infty} \frac{a_k}{z^k}, \tag{13}$$

where

$$a_0 = \sum_{i=1}^{m} q_i \qquad \text{and} \qquad a_k = \sum_{i=1}^{m} \frac{-q_i\,(z_i)^k}{k}.$$

Φ_n is approximated by the first p terms of the infinite series. It can be shown that for any $p \geq 1$,

$$\left| \Phi_n(z) - a_0 \log(z) - \sum_{k=1}^{p} \frac{a_k}{z^k} \right| \leq \frac{A}{1 - \left|\frac{r}{z}\right|} \left|\frac{r}{z}\right|^{p+1}, \tag{14}$$

where

$$A = \sum_{i=1}^{m} |q^i|.$$

Clearly, $\hat{\Phi}_n$ is represented by $\{ a_k, k = 0, \cdots, p \}$ and the coordinates of the center of the nth box.

To find Φ_n, n not of level h, one uses (7). To sum the Φ_j, $j \in children(n)$, the expansion representing Φ_j is "translated"; i.e., each Φ_j is expanded to a multipole expansion around the center of n. Once this is done, the Φ's of all children are expanded around the same origin, and Φ_n is found by summing the coefficients of the translated Φ_j.

Let the coefficients $a_i, i = 0, \cdots, \infty$, specify the multipole expansion of Φ_n around the point z_0. The translation of this expansion to the origin is given by

$$\Phi_n(z) = a_0 log(z) + \sum_{l=1}^{\infty} \frac{b_l}{z^l}, \tag{15}$$

where

$$b_l = -\frac{a_0 z_0^l}{l} + \sum_{k=1}^{l} a_k z_0^{l-k} \binom{l-1}{k-1} \tag{16}$$

with $\binom{l}{k}$ the binomial coefficients.

$\hat{\Psi}_n$ is formed by retaining the first $p+1$ terms of the series of (12). It is represented by $\{ b_k, k = 0, \cdots, p \}$ and the coordinates of the center of the nth box.

Ψ_n (see (3)) is computed by converting the Φ_i, $i \in$ *interactive set of* n, to a *local* expansion around the center of the box n. This conversion is given as follows.

Let $\{ a_k, k = 0, \cdots \}$ be the representation of some Φ_j, where j is not a neighbor of n. Let n's center be the origin, and let the center of j be z_0. The contribution of Φ_j to Ψ_n at a point z in n is

$$\sum_{l=0}^{\infty} b_l z^l, \tag{17}$$

where

$$b_0 = a_0 log(-z_0) + \sum_{k=1}^{\infty} \frac{a_j}{(-z_0)^k}, \tag{18}$$

and

$$b_l = \frac{a_0}{l.z_0^l} + \frac{1}{z_0^l} \sum_{k=1}^{\infty} \frac{a_k}{(-z_0)^k} \binom{l+k-1}{k-1}. \tag{19}$$

The shifting of $\Psi_{parent(n)}$ to the center of n is given by the following equation: For any z_0, z, and a_k, $k = 0, 1, \cdots, p$,

$$\sum_{k=0}^{p} a_k (z - z_0)^k = \sum_{i=0}^{n} \left(\sum_{k=l}^{n} a_k \binom{l}{k} (-z_0)^{k-l} \right) z^l. \tag{20}$$

Several functions can be defined to summarize the GR algorithm. Let "+" be an operator that maps representations of the same type which are expansions around the same point z into a representation around z each of whose its coefficient is the sum of the appropriate coefficients of the "+"'s arguments.

$Translate_z$ maps a representation of a multipole expansion to a multipole expansion around the center of box z (15).

$Shift_z$ maps a representation of a local expansion to a local expansion around the center of box z (20).

$Convert_z$ maps a representation of a multipole expansion to a local expansion around the center of box z (17).

Under these conventions, the computation process by which the GR algorithm computes (4) and (7) can be written in the following way:

$$rep(\hat{\Phi}_n) = \sum_{i \in children(n)} Translate_n(rep(\hat{\Phi}_i)) \tag{21}$$

$$rep(\hat{\Psi}_n) = Shift_n(rep(\hat{\Psi}_{parent(n)})) + \sum_{\substack{i \in \{j \mid j \in children(neighbors(parent(n))), \\ j \notin neighbors(n)\}}} Convert_n(rep(\hat{\Phi}_i)). \tag{22}$$

The operations of the BH algorithm can be described with functions bearing the same names with one minor difference: $Translate_z$ has all the representations of the children of z as arguments, and the summation is omitted. That is to say, (21) becomes

$$rep(\hat{\Phi}_n) = Translate_n(rep(\hat{\Phi}_1), rep(\hat{\Phi}_2), \cdots, rep(\hat{\Phi}_i), \cdots) \quad all\ i,\ i \in children(n). \tag{23}$$

4. The computational structure. This section considers a network of processes, each associated with a region of space (a cell, a rectangale for 2D, a box for 3D). Each process communicates with its parents, its children, and its neighbors to compute the solution to the N-body problem according to the above algorithm. If a processor is assigned to each process and all processors operate in a SIMD style, this architecture is "connection-machine-like" [11] in the sense that there are many equal processors operating in SIMD style that communicate via many connections to "neighbors." The difference between the connection machine and this network is that this network is not a hypercube; instead, its connections are derived from the structure of the equations. Our purpose is to find bounds on the computation time of this network architecture in order to compare it with the connection machine and other architectures.

A look at the equations shows us the following communication paths for each process (cell) C:

(a) C to C's parents (8);
(b) C's parents to C (4);
(c) C to each of its children (4);
(d) Each child of C to C (8);
(e) C to C's interactive set (4);
(f) Each member of C's interactive set to C (4);
(h) C to its neighbors (h level only for the calculation of the near field);
(i) Each neighbor of C to C (h level only for the calculation of the near field).

If we abstain from sending messages in both directions at the same time (as long as the SIMD discipline is maintained this does not occur), a and b, c and d, and h and i can be satisfied by the same "wire." Moreover, the interactive set is symmetric, i.e., for any box a, b, $a \in$ *interactive set*(b) implies $b \in$ *interactive set*(a). Therefore, e and f can also be united.

Thus, we need the following connections at each process C:

(a) C to/from C's parents (8);

(b) C to/from each of its children (4);

(c) C to/from C's interactive set (4);

(d) C to/from its neighbors (h level only for the calculation of the near field).

If each connection is implemented, connection machine style, by one wire, we have 32 wires in 2D (1 parent, 4 children, up to 27 members in the interactive set). In 3D we have 198 wires (1 parent, 8 children, up to 189 members in the interactive set). This number seems high; certainly it cannot be implemented as one physical wire per connection. However, we can reduce the number of connections as follows.

The information sent from C to its interactive set members is the same as that sent to its parents. C's interactive set members can be reached by having C's parents forward the information to their 26 neighbors, who, in turn, forward it to their children. Since each message carries its source, the children can check if the sender, C, is in the child's interaction set and discard the information on a negative answer.

This argument replaces connection to the interactive set members with connections to neighbors. The number of connections per cell becomes 13 for 2D and 35 (1 parent, 8 children, 26 neighbors) for 3D, which, although not low, is much better. In such an arrangement there are connections to parents, to children, and to neighbors for all processors, including those at the h level. Figure 2 illustrates the connection structure in 2D, the equivalent 3D structure is somewhat difficult to draw.

Figure 3 shows that the computation structure has the form of a graph in which each node describes a cell and each branch describes a wire connecting two cells. The graph has the form of a pyramid, with the top node corresponding to the top cell and the bottom nodes to the h level cells. Levels, or cells of equal size, appear distinctly in the graph; they are nodes of equal height from the bottom or the top of the pyramid.

Consider an algorithm that performs the far-field computation in the following steps:

Algorithm:

Step 1. $rep(\hat{\Phi}_n^0)$: Compute $rep(\hat{\Phi}_n^0)$, for all n, n in level h. (The exact representation and method of computation depends on the method used and will be elaborated on in the sequel.)

Step 2. Calculate $rep(\hat{\Phi}_n)$ going up the tree: Given $rep(\hat{\Phi}_n^0)$, and using (8) and (21) compute $rep(\hat{\Phi}_n)$ for all n, level by level, starting in level $h-1$ and ending in level 0.

Step 3. For all nodes of the tree calculate the effect of the interactive set members: For all direction d in a level do

Step 3.1. Each node n sends to its neighbor that lies in direction d from it the $rep(\hat{\Phi}_p)$ for all p, p a child of n;

Step 3.2. Each node n receiving such a message plays the role of a neighbor and retransmits it to all its children;

Step 3.3. Each node n (playing the role of a child) computes $Convert_z(rep(\hat{\Phi}_i))$ for all messages received from one of its interactive set members and discards the rest of the messages. The result is accumulated, awaiting

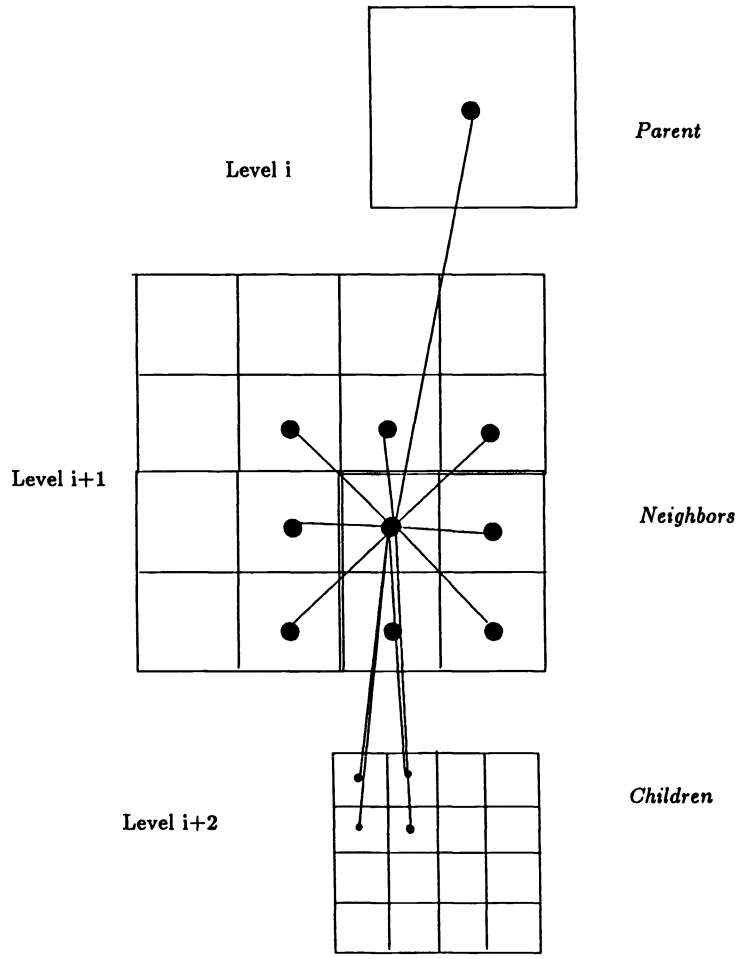


FIG. 2. *A* 2D *cell (node) and its connections.*

the arrival of $rep(\hat{\Psi}_i)$ from n's parent.

Step 4. $rep(\hat{\Psi}_n)$: The values of $rep(\hat{\Psi}_n)$ are propagated down the tree to compute $rep(\hat{\Psi}_n)$ for all children according to (22).

Step 5. Evaluation: For all n in level h, $rep(\hat{\Psi}_n)$ is used to calculate the far-field potential at each mass-point in the cell.

Note that all the h-level nodes can perform step 1 simultaneously. Similarly, step 2 is simultaneously performed by all nodes of the same level. Step 3 is simultaneously performed in parallel by all nodes of the entire network. This holds if we understand that a node without children does not send them messages, etc.

Some additional consideration of the algorithm shows that our tree consists of three kinds of nodes: the top node, the intermediate nodes, and the bottom level nodes. Figure 4 illustrates the functions computed and the information flow in each node.

The above algorithm differs from the algorithms described in [8] and in [10] in several aspects. First, the algorithm is really a frame in which different functions may be substituted for Φ_n, $Translate_z$, etc. The algorithm is described as a "tree" of

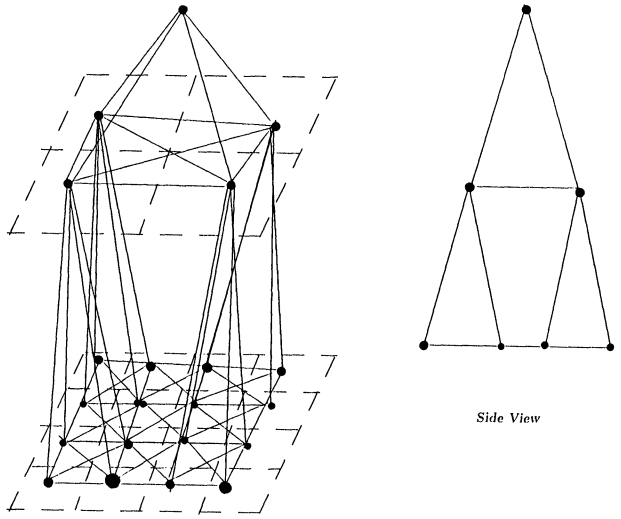


FIG. 3. *The computation "tree."*

processes operating in parallel rather than a single process algorithm as in [8], or the level-by-level parallel algorithm as in [10, p. 50]. More importantly, this algorithm handles the information transfer explicitly and so reduces both communication and computation times. The main tool for accomplishing this is the parallel structure. Information is sent in parallel so that messages do not conflict, and computing is done in parallel by all processes of the tree.

The network of processes can operate also in the "data flow" mode, where each process performs the computation once the data is available. The operation of the network in this mode has not been fully investigated as yet.

5. Communication time. The time required to solve the N-body problem by the above computation structure consists of the time that each cell spends computing plus the time required to transfer the information from process to process. The latter component is named *communication time.* The communication time is bounded under several assumptions:

(1) Time is sampled and divided into intervals. During one time-interval a process can both send one message and receive one message;

(2) Messages are of bounded length;

(3) The time to send a message from a cell to an adjacent cell (neighbor, child, or parent cell) is one "message time" unit.

When the messages are small, one can add the assumption that processes can combine several messages into one message. It turns out that in our applications, sizes are such that this assumption should be be omitted.

It is further assumed that each function representation is transferred by one message (this assumption will be modified later on). Under the above assumptions, we get the following communication time bound. (Notice that under the SIMD assumptions all processes are doing the same operation at the same time.)

Consider the algorithm for far-field computation in the previous section. Let us assume that in 2D there are $n \times n$ regions ($n \times n \times n$ regions for 3D) with $h = \lceil \log n \rceil$.

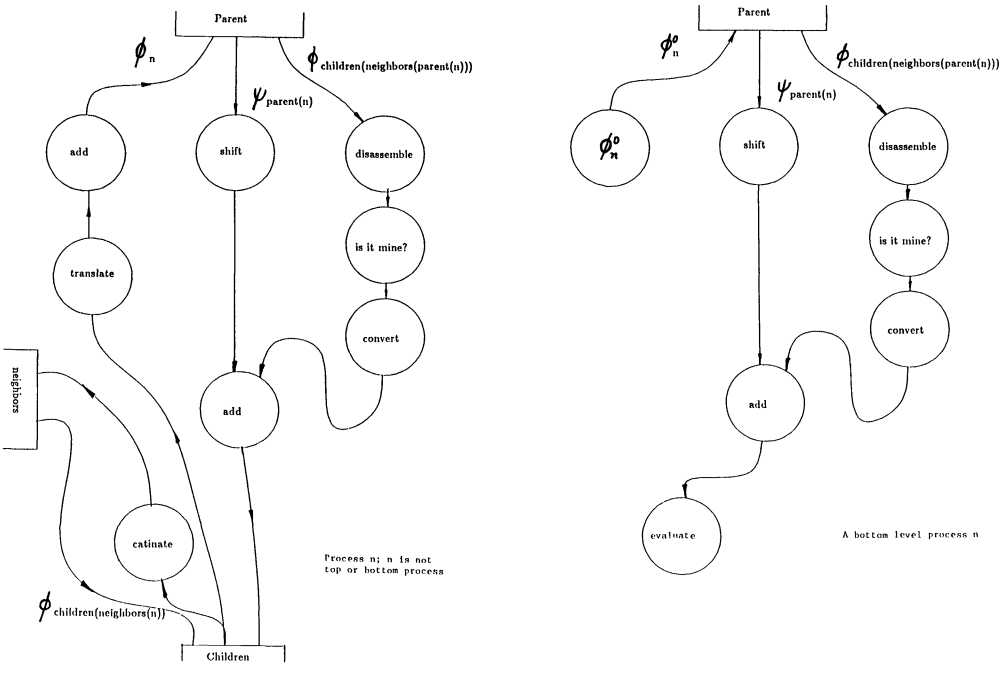


FIG. 4. *Computation and information transfer in tree nodes.*

Denote by $n_{children}$ the number of children of a processor (processor level is not h). Denote by n_n the maximum number of neighbors of a processor. Denote by n_{set} the maximum number of members in an interactive set. The following lists the *communication time* in message-time units for each step of the algorithm:

Step 2. Since each processor can send one message and receive one message at a time, the time is $n_{children} \times h$.

Step 3. Here each processor sends $n_{children}$ messages to each neighbor which, in turn, sends one copy to each of its children. The time is $n_n(n_{children} + n^2_{children})$. Note that messages can be sent and received simultaneously by all processors without interference, provided that each processor sends its message in the same "direction"; i.e., all processors send to the left and receive from the right, etc.

Step 4. Here each processor sends a message to each of its children. The time is $n_{children} \times h$.

Thus, the total communication time in message-time units is

$$\text{(24)} \qquad \textit{communication time} \leq 2h.n_{children} + n_n \times (n_{children} + n^2_{children}).$$

In terms of actual time, what is "message time"?

The GR algorithm specifies a function by q coefficients and by the coordinates of the center of the box around which the function is expanded (the source of the message). Since messages are function descriptions, all messages are of equal length and the concept of message time is well defined as the time required to send a function description. For example, if $q = 10$ and each coefficient and the source are represented by 32 bits, the length of the message is about 352 bits. The connection-machine messages, for example, are 190 bits long; therefore, on that machine, two actual messages are needed to implement our 352-bit message, and the "message time" becomes the time for two connection-machine messages.

The case of the BH messages is somewhat more complicated. Fixed-size messages, each consisting of $(center\ of\ mass, mass, source)$, travel up the tree. Therefore, the contribution of steps 2 and 3 of the algorithm is as above, totaling

$$h.n_{children} + n_n \times (n_{children} + n^2_{children}). \tag{25}$$

However, on the way down, the number of such triplets that pass each node of the tree equals the number of interactive set members of all the node ancestors. Moreover, this function description is not a message of fixed length, and the estimation of communication time has to be done differently.

The number of triplets that a node at the bottom of the tree (pyramid) receives is (less than)

$$n_{set} \times h. \tag{26}$$

If messages are transmitted starting from the bottom of the tree, the above is the number of messages step 4 requires. It is an upper bound since some nodes have small interactive sets (e.g., *top* has zero members). Thus, the total number of messages in the BH algorithm is

$$h \times n_{children} + n_n \times (n_{children} + n^2_{children}) + n_{set} \times h. \tag{27}$$

With k triplets per message, the communication time is the above expression over k.

The following argument supports the above expression. Choose an arbitrary node, say A. A sends down the tree the triplets of its interactive set. As soon as the last triplet clears A's child, the child can send its own interactive set triplet down the tree. Thus, for a tree of height h and a bound on the number of members in an interactive set, we get the above expression as a bound on the communication time.

6. Computation time. This section estimates the computation time resulting from the arithmetic operation only. It does not include the communication time, and it does not include data transfers inside processors. The estimation is made for 3D using both GR and BH algorithms. This section is an abbreviated version of the corresponding section in [19]. The reader is referred to that for complete details.

The time to perform floating point addition, multiplication, etc., is denoted by t_+, t_*, etc.

The computation time is bounded under the assumption that the number of particles in a cell is bounded by a number m. Often, physical systems exhibit sparsity, that is, certain regions of space have many particles, while others are empty or nearly so. Thus, the uniform bound m may suggest particles that are fairly uniformly distributed in space, but it certainly does not imply it. The algorithms and the derived bounds apply to sparse systems, but they do not take advantage of sparsity to save either time or space or both.

6.1. GR algorithm in 3D. The following calculations follow the 3D expansion by Zhao [10]. Zhao expands the various functions to a series of the derivatives of $\frac{1}{R}$, where R is the distance from the center of a cell (whose function is being computed), to the point at which the potential is desired. To avoid evaluation of (direction) cosines, they have been expressed in terms of the Cartesian coordinates (x_1, x_2, x_3). In the sequel, p is the order of the highest derivative retained. The number of terms in the expansion, denoted by n_p, is derived from p.

6.1.1. $\hat{\Phi}_n^0$. This calculation follows Theorem 3.2.1 in [10].

It is assumed that in each cell there are m mass points q_i at points (x_{1i}, x_{2i}, x_{3i}), $i = 1 \cdots m$.

$$a_{ijk} = \sum_{l=1}^{m} q_l (-1)^{i+j+k} \frac{1}{i!j!k!} x_{1l}^i x_{2l}^j x_{3l}^k. \tag{28}$$

If we retain only terms of $i + j + k \leq p$ then the number of terms is

$$n_p = \sum_{k=0}^{p} \sum_{j=0}^{p-k} \sum_{i=0}^{p-k-j} 1 = \frac{(p+1)(p+2)(p+3)}{6}, \tag{29}$$

which results in $n_1 = 4$, $n_2 = 10$, $n_3 = 20$, $n_4 = 35$, etc.

Using the following recursive relations between the coefficients,

$$a_{0,0,0} = 1 \tag{30}$$

$$a_{i,j,k} = a_{i-1,j,k} \frac{(-1)x_i}{i} = a_{i,j-1,k} \frac{(-1)x_j}{j} = a_{i,j,k-1} \frac{(-1)x_k}{k} \tag{31}$$

results in the following bound

$$\tau(\hat{\Phi}_n^0) = n_p(3mt_* + (m-1)t_+). \tag{32}$$

6.1.2. *Translate*$_z$. This calculation follows from Lemma 3.2.1 in [10]. The coefficients of the translated expansion c_{ijk} are

$$c_{ijk} = \sum_{\alpha=0}^{i} \sum_{\beta=0}^{j} \sum_{\gamma=0}^{k} a_{\alpha\beta\gamma} a'_{i-\alpha,j-\beta,k-\gamma}. \tag{33}$$

Here a'_{ijk} are the coefficients in the expansion for $1/R$, measured from the old cell center with respect to the new center.

The a'_{ijk} are constants that can be precomputed. The number of different sets of such constants equals the number of children of a cell; there are eight in 3D. The constants at different levels can be derived by multiplying by appropriate powers of 2. Retaining only terms of $i + j + k \leq p$ results in

$$\tau(Translate_z) = k_p(2t_* + t_+), \tag{34}$$

where

$$k_p \stackrel{\text{def}}{=} \frac{1}{720}(p^6 + 21p^5 + 175p^4 + 735p^3 + 1624p^2 + 1764p + 720). \tag{35}$$

(The above expression was calculated by Zhao using MACSYMA [18].)

6.1.3. *Convert*$_z$. This calculation follows from Lemma 3.2.2 in [10]. The coefficients of the converted expansion c_{ijk} are

$$c_{ijk} = \frac{1}{i!j!k!} \sum_{\alpha,\beta,\gamma=0}^{\infty} a_{\alpha\beta\gamma} b_{i+\alpha,j+\beta,k+\gamma} (i+\alpha)!(j+\beta)!(k+\gamma)!. \tag{36}$$

b_{ijk} are the coefficients of the local expansions for $1/R$, measured from the old cell center, with respect to the new center.

The b_{ijk} are constants that can be precomputed. The number of different sets of such constants in the number of interactive set members of a cell — 189 in 3D. The constants of different levels can be derived by multiplying by appropriate powers of 2. Both a_{ijk} and b_{ijk} are zero if one of their coefficients is larger then p. The amount of time is

$$\tau(Convert_z) = (2t_* + t_+)c_p, \tag{37}$$

where

$$c_p \stackrel{\text{def}}{=} \frac{1}{24}(3p^4 + 22p^3 + 57p^2 + 62p + 24). \tag{38}$$

6.1.4. *Shift*$_z$. This calculation follows from Lemma 3.2.3 in [10]. The coefficients of the converted expansion d_{ijk} are

$$d_{ijk} = \sum_{\alpha,\beta,\gamma=0}^{\infty} c_{i+\alpha,j+\beta,k+\gamma} \binom{i+\alpha}{\alpha} \binom{j+\beta}{\beta} \binom{k+\gamma}{\gamma} \Delta x_1^{\alpha} \Delta x_2^{\beta} \Delta x_3^{\gamma}. \tag{39}$$

Here $(\Delta x_1, \Delta x_2, \Delta x_3)$ are the coordinates of the old center in terms of the new center.

The Δx_i are constants that can be precomputed. The number of different sets of such constants is the number of children of a cell; there are eight in 3D. The constants of different levels can be derived by multiplying by appropriate powers of 2. The result is

$$\tau(Shift_z) = c_p(3t_* + t_+), \tag{40}$$

where c_p is defined in (38).

6.1.5. Evaluation. The evaluation of the potential at each mass point requires the evaluation of Ψ_n, n at level h, using equation

$$\Psi_n(x) = \sum_{i,j,k=0}^{p} d_{ijk} x_1^i x_2^j x_3^k$$

for the potential. The force can be obtained by taking the partial derivatives. Thus,

$$F_{x_1}(x) = \sum_{i,j,k=0}^{p} d_{ijk} i x_1^{i-1} x_2^j x_3^k.$$

The potential can be evaluated with two multiplications and one addition per term (keep $x_1^i x_2^j x_3^k$ for each ijk and proceed by increasing the indices one at a time). For m mass points in z, the evaluation of the potential requires

$$\tau(evaluate_{potential}) = mn_p(t_+ + 2t_*). \tag{41}$$

For m mass points in z, the evaluation of the force requires about

$$\tau(evaluate_{force}) = m(n_p(t_+ + 2t_*) + 3n_p t_*). \tag{42}$$

6.1.6. Total GR time. Using the algorithm of §4 and denoting by n_n the number of neighbors and by $n_{children}$ the number of children, the total time is

$$\tau_{totalGR} = \tau(\Phi n^0) + h.\tau(Translate_z) + (n_n).n_{children}.\tau(Convert_z) + h.\tau(Shift_z) + \tau(Evaluation) \tag{43}$$

Under the simplifying assumption $t_+ = t_*$, the above equation becomes

$$\tau_{totalGR} = (n_p(7m-1) + (3k_p + 4c_p)h + 3n_n n_{children} c_p)t_+. \tag{44}$$

6.2. BH algorithm in 3D. Only the calculation of Φ_z^0 and $Translate_z$, as well as the final evaluation, have significant operation counts; $Convert_z$ and $Shift_z$ amount to concatenation of lists.

6.2.1. $\hat{\Phi}_n^0$. The following time is obtained from (10) (in 3D):

$$\tau(\Phi n^0) = 3mt_* + 4(m-1)t_+ + 3t_/. \tag{45}$$

6.2.2. $Translate_z$. Translation is the above calculation with $m = 4$:

$$\tau(Translate_z) = 12t_* + 12t_+ + 3t_/. \tag{46}$$

6.2.3. Evaluation. The evaluation of the potential, or force, at each mass point requires the evaluation of each member of the list representing Ψ_n, n at level h, using (9) for the potential and (1) for the force. Ψ_n , n at level h, has at most

$$(h-2) * (maximum\ number\ of\ members\ in\ an\ interactive\ set) \tag{47}$$

elements. That maximum is 189 in 3D.

The force between two points (1) is evaluated according to

$$F_{x_i} = G\frac{M_x M_y}{\| x-y \|^2}\frac{x_i - x_i}{\| x-y \|}, \tag{48}$$

and requires $(3t_* + 2t_+ + 3t_-) + 15t_* + 3t_- + 3t_/$, where the 15 is an estimate of the time for evaluating a square root. With $t_- = t_+$, the result is $18t_* + 8t_+ + 3t_/$.

Thus, for m elements and h levels, the time is

$$189m(h-2)(18t_* + 8t_+ + 3t_/ + 1). \tag{49}$$

6.2.4. Total BH time. Using the algorithm of §4, the total time is

$$\tau_{totalBH} = \tau(\Phi n^0) + h.\tau(Translate_z) + \tau(Evaluation). \tag{50}$$

Under the simplifying assumptions $t_+ = t_*$ and $t_/ = 5t_+$, the above becomes

$$\tau_{totalBH} = 189m(h-2)41t_+ + 39ht_+ + (7m+11)t_+. \tag{51}$$

7. A comparison. In order to gain insight into the above results, this section maps the pyramid of processes to three different processor configurations:

- A pyramid of processors corresponding to the possessive pyramid graph, where each vertex is assigned a process and each arc a communication "wire";
- A hypercube of processors, i.e., a connection-machine-like architecture [11];
- A smaller array of larger processors, e.g., $2 \times 2 \times 2$ processors.

TABLE 1
Pyramid's far-field communication and computation time— BH algorithm.

n	m	particles	h	communication millisec.	$Translate_z$	Φ_n^0	evaluation millisec.
32	1	32K	5	714	-	-	1150
32	10	320K	5	714	-	-	11500
32	20	640K	5	714	-	-	23250
38	1	62712	6	763.5	-	-	1550
38	10	627120	6	763.5	-	-	15500
38	20	1250K	6	763.5	-	-	31000

The performance of the BH and the GR algorithms are compared by calculating their communication and computation times in these configurations. The calculation is done for several typical values of N, m, p, and n_p, where N is the total number of particles; m is the number of particles in a cell; p and n_p determine the number of coefficients that the expansion carries. To get some idea of actual time in seconds, the speed of communication and floating point operations of the CM2 model connection machine is used. To simplify the discussion, it is assumed that $t_+ = t_*$; $t_/ = 5t_+$.

7.1. The BH algorithm — comparative speed. This section considers the performance of the BH algorithm for the pyramid of processors.

Messages. It is assumed that the BH algorithm triplet consists of four single precision 32-bit word plus one 32-bit word for specifying the origin of the cell. Thus, the 160-bit message fits into the 190-bit connection-machine message. The message-time-unit becomes the time for one message on the CM2 connection machine, which is about 0.250 millisecond.

Communication time. In 3D we have 8 children, 26 neighbors, and 189 as the maximum size of an interactive set. Thus, the communication time is $197h + 1872$ (27).

Computation time. Computation time is calculated using (51).

Cells and particles. Let the number of cells in the bottom level, h, be $N = n \times n \times n$; then $h = \lceil \log_2 n \rceil$. The total number of cells is

$$N(1 + \frac{1}{8} + \frac{1}{64} + \cdots) \leq \frac{8}{7}N. \tag{52}$$

Since the number of processors in the connection machine is 64K, we would like to restrict the number of cells to this number.

The number of particles is mN.

Summary. Table 1 summarizes communication and computation times for selected values of n and m. To get some insight into these numbers, the table is presented using 20 Kflops per second for t_+ (CM2 model connection machine measured time [13]), and 250 microseconds per message between adjacent processors (CM2 model connection machine message time [14]). Time in Table 1 is expressed in milliseconds and is rounded.

Table 1 illustrates several characteristics of this computation scheme. While the communication time is substantial, it does not increase with N so long as the height of the tree is constant. Neither, in fact, do any of the quantities associated with the "tree." These quantities are a function of the size of the tree, i.e., of n and h, but not the number of particles.

With a low m, the number of particles in a box, a significant portion of the total time is communication time. As m increases, the evaluation time takes over.

For a given, large number of particles the computation time is roughly proportional to m. This time could be reduced significantly by increasing the number of processors so that m remains one. This cannot be done in the connection machine that has a fixed number of processors.

The increase in m, while the number of processors remains fixed, shifts the computation load to the evaluation and to the bottom level (h level) of processors. They do most of the work while the rest of the processors are idle.

7.2. The GR algorithm — comparative speed. This section considers the performance of the GR algorithm for the pyramid of processors.

Messages. The GR algorithm uses n_p coefficients to represent a function; each coefficient is represented by a single precision 32-bit word plus one 32-bit word for specifying the origin of the cell. Thus, the length of a message is $32(n_p + 1)$ bits. A message-time unit is the time for one such message; the exact time on any particular hardware, say, the connection machine, depends on n_p.

Communication time. In 3D we have 8 children and 189 neighbors as the maximum size of an interactive set. Thus, the communication time is $16h + 1872$ (25).

Computation time. Computation time is calculated using (44).

Cells and particles. As in the BH calculation above, the number of cells in the bottom level, h, is denoted by $N = n \times n \times n$; then $h = \lceil \log_2 n \rceil$. The total number of cells is less then $\frac{8}{7}N$.

We would like to restrict the number of cells to 64K, which is the number of processors in the connection machine.

The total number of particles is mN.

Summary. The following table summarizes communication and computation times for selected values of n, m, and p. The table is given in "real" numbers: the communication time assumes a 190-bit message and coefficients of 32 bits; the computation time assumes 20 Kflops per second, typical of the CM2 model connection machine [13]. The communication time is assumed to be 250 microseconds per message between adjacent processors [14].

The time for $Translate + Convert + Shift$ is denoted by Q. This time is independent of m. In the table this time is separated from the time that depends on m and that represents operations done by the bottom-level processes.

Table 2 illustrates several characteristics of this computation scheme. As in the BH case, the communication time does not increase with m. Neither do, in fact, any of the quantities associated with the "tree." These quantities are a function of the size of the tree, i.e., n and h, but not of the number of particles. Communication time, however, does increase with n_p and is roughly proportional to it.

Unlike the BH case, most of the time is spent in the "body" of the tree performing $Convert$ — calculating the effect of the interactive set. The evaluation stage is relatively fast.

The time of the GR algorithm depends on p, a parameter that does not appear in the BH algorithm. Accuracy is improved with the increase of p, and so does communication time. For the values of p considered above, and for a large number of particles, the GR algorithm is several times faster than the BH algorithm.

Unlike the BH algorithm, the GR algorithm utilizes all the processors more or less equally.

TABLE 2
Pyramid's far-field communication and computation time — GR algorithm.

n	m	particles	h	p	n_p	comm. millisec.	Q $3(2h+104)n_p p$ millisec.	Φn^0 + evaluation $(9m+7p-4)n_p$ millisec.
32	1	32K	5	1	4	488	228	1
32	1	32K	5	2	10	488*3	786	3
32	1	32K	5	3	20	488*6	2135	6
32	10	320K	5	1	4	488	228	14
32	10	320K	5	2	10	488*3	786	35
32	10	320K	5	3	20	488*6	2135	69
32	20	640K	5	1	4	488	228	28
32	20	640K	5	2	10	488*3	786	70
32	20	640K	5	3	20	488*6	2135	139
38	1	62712	6	1	4	492	231	1
38	1	62712	6	2	10	492*3	796	3
38	1	62712	6	3	20	492*6	2159	6
38	10	627120	6	1	4	492	231	14
38	10	627120	6	2	10	492*3	796	35
38	10	627120	6	3	20	492*6	2159	69
38	20	1250K	6	1	4	492	231	28
38	20	1250K	6	2	10	492*3	796	70
38	20	1250K	6	3	20	492*6	2159	139

An accurate comparison of the algorithm requires comparing performance for the same global error. To my knowledge, no error analysis was done for the BH algorithm. References [9] and [10] present the error analysis of each step of the GR algorithm, but the effects are not summed to form the global error.

7.3. Mapping the "pyramid" into a hypercube. The mapping of the network of processes into an hypercube is a preparation for executing the algorithm on a connection-machine-like computer. The interest here is twofold: How fast can a hypercube machine simulate the pyramid network, and how can the N-body computation be organized on the hypercube? The definitions in this section follow [16]. The actual mapping is a rather straightforward generalization of [17].

An embedding $< \phi, \rho >$ of a graph $G = (V_G, E_G)$ into a graph $H = (V_H, E_H)$ is defined by an injective mapping ϕ from V_G to V_H, together with a mapping ρ that maps $(u, v) \in E_G$ onto a path $\rho(\phi(u), \phi(v))$ in H that connects $\phi(u)$ and $\phi(v)$.

The quality of an embedding is measured by three cost functions — *expansion, dilation, and load factor.* The *expansion* of an embedding $< \phi, \rho >$ of G into H is the ratio of the size of V_H to the size of V_G. The *dilation* of an edge (u, v) under $< \phi, \rho >$ is the length of the path $\rho(\phi(u), \phi(v))$ in H. The *dilation* of an embedding $< \phi, \rho >$ is the maximum edge dilation, over all edges in G, under $< \phi, \rho >$. The *load factor* $\lambda(e)$ of an edge e in H is the number of paths that pass through e that are images of edges in G. The load factor of an embedding is defined to be the maximum load-factor over all edges in H.

Consider the 2D case first. An $n \times n$ pyramid graph is formed from meshes of size $n \times n$, $\frac{n}{2} \times \frac{n}{2}$, $\frac{n}{4} \times \frac{n}{4}$, $\frac{n}{8} \times \frac{n}{8}$, $\cdots$, 1×1 connected as implied by Fig. 5. Note that n is a power of 2 and that if each node is one of our cells, this pyramid does not contain the connection to the "diagonal" neighbors. (Omission of the diagonal link does not change d in either the 2D or the 3D cases. As it is shown in the sequel, d is the important quantity in our case.)

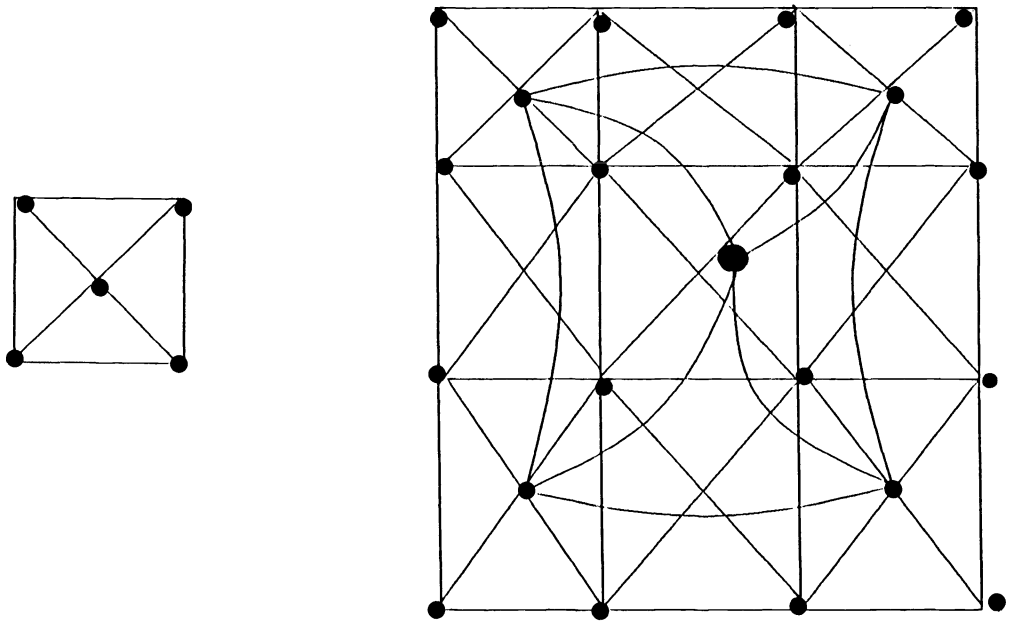


FIG. 5. *A* 2×2 *and a* 4×4 *pyramid.*

LEMMA 7.1. *An* $n \times n$ *pyramid can be embedded in a* $2n^2$ *node hypercube so that dilation of the embedding is* 3 *and the load factor is* 2.

See [17] for a proof. Figure 6 illustrates a simple 2D embedding.

LEMMA 7.2. *An* $n \times n \times n$ *pyramid can be embedded in a* $2n^3$ *node hypercube so that dilation of the embedding is* 4 *and the load-factor is* 3.

The lemma can be proved similarly to the proof in [17]. Figure 7 illustrates a simple 3D embedding.

Clearly, we are interested in a mapping that simulates the tree well. Since each process is mapped to a different processor, the issue is an issue of communication time rather then computation time. Maps differ from each other in the amount of communication overhead they introduce as compared to the tree. While there is no claim that the above mapping is optimal, it is attractive since neighboring nodes remain close together. Moreover, we can bound the number of cycles of the hypercube machine needed to simulate a cycle of the pyramid machine.

Reference [16] considers the case of simulating G by H, where all nodes send messages at the same time on all adjacent wires. The resulting bounds are $\max\{d, \lambda\} \leq$

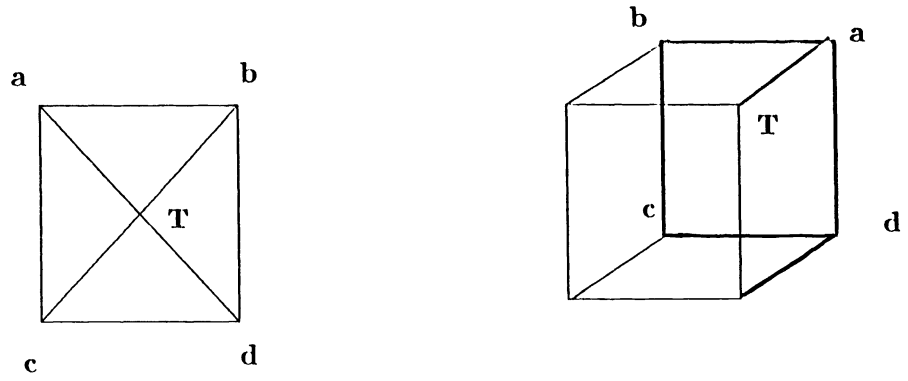


FIG. 6. *A simple* 2D *embedding.*

$T \leq d\lambda$, where T is the number of cycles of H required to simulate any cycle of G. These bounds are too pessimistic for our case. In our case an upper bound is simply the dilation d. This is indeed not the bound on a general computation performed on a pyramid network, but a bound on our own algorithm with the above mapping. The bound results from the fact that in this mapping the only overlapping paths are those from a node to its children. Since only one child at a time sends messages to its parent, no conflict arises. Since in most parts of the algorithms the information consists of several messages that can be queued inside a process, it is believed that d is actually a conservative upper bound.

The conclusion is that hypercube communication can be bounded by d multiplied by the communication performance of the pyramid. We believe that this is a rather conservative estimation of performance; with the pyramid mapped as above, the average performance of the connection machine will be very close to that of the pyramid machine.

The expansion story is less attractive. A $32 \times 32 \times 32$ base (32K cells in the bottom level) requires 64K processors, $\frac{3}{7}$ of them not participating in the computation.

7.4. A small number of larger processors. This section considers the implementation of the algorithm by a small number of larger processors. Both "small number" and "larger" are with respect to the connection machine. We consider either one processor, or, for 3D, arrays of $2 \times 2 \times 2$, $4 \times 4 \times 4$, etc., of processors. What we have in mind for a processor is a board with fast arithmetic chips, memory, and a controller. Currently, a board like this can deliver about 20M floating point operations per second and has a memory of several million words of 32 bits each. The interconnection among neighboring processors is via 32-bit buses. To facilitate simple programming and symmetric information transfer between processors, the buses

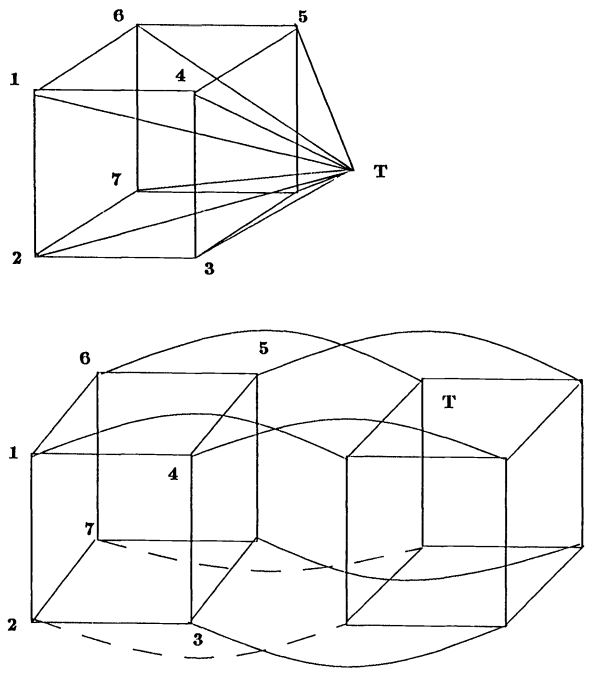


FIG. 7. *A simple* 3D *embedding.*

are interconnected as illustrated by Fig. 8. The processors are associated with grid points. Each processor has an input bus and an output bus. The input bus of a processor is connected (via tri-state gates) to the output buses of all its neighbors. The output bus of a processor is connected (via tri-state gates) to the input buses of all its neighbors.

The pyramid of processes has to be partitioned and assigned to individual processors. It is assumed that this partition is symmetric. The base of the pyramid is equally divided among the processors, each getting a smaller pyramid; whatever is left is divided again, or assigned to one processor. For example, a 32×32 base pyramid of processors is divided among four processors. Each processor gets a 16×16 base pyramid; the top process remains and has to be assigned to one of the processors.

There are several differences between this computation mode and the ones discussed before. First, no messages have to be sent between two processes that reside on the same processors (*local* processes). (A process accesses the results of another process directly.) Thus, in this case, the communication time is negligible. Second, a processor evaluates its processes sequentially; parallelism exists between processors only. Thus, the evaluation order has to be such that no processor will be waiting for data from another processor.

The transfer of information between processors is different. It is assumed that

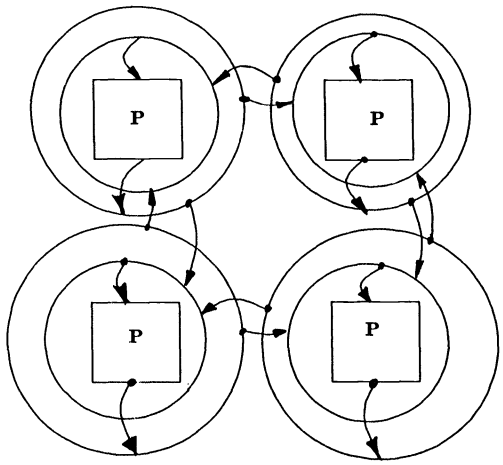


FIG. 8. *Interconnection among neighboring processors.*

each processor has a direct memory access (DMA) channel so that data transfer to and from buses can be done simultaneously with computing. In the pyramid case, to save "wires," information was routed to interactive set members through parents. Clearly, this does not make any sense for local processes; for processes in different processors the issue is how not to send the same information more than is necessary. Therefore, when a cell has a result that is needed by a cell in another processor, the result is sent over; the other processor has a memory cell for each cell on another processor that is in one of its own cell's interactive sets. The arriving information is stored there to be used by all cells that need it.

Let us consider a pyramid of processes with a base of $n \times n \times n$ (e.g., $n = 32$) and an array (of larger processors) of size $k \times k \times k$, where $k = 2, 4, 8$, etc.

The pyramid of processes is partitioned among the array's processors so that each processor gets a pyramid of base $\frac{n}{k} \times \frac{n}{k} \times \frac{n}{k}$. The top of the original pyramid is not covered; in the case of base 32 pyramid, which is used here for comparison, and arrays of base 2, 4, and 8, the top is a pyramid with base $1 \times 1 \times 1$, $2 \times 2 \times 2$, and $4 \times 4 \times 4$, respectively. It is assumed that this tip is either assigned to one of the processors or divided among them. For the cases $k = 2, 4, 8$ the contribution of the tip pyramid to the total time can be ignored, and this is what is done in the sequel.

It is assumed that the system of processors uses the same algorithm as in §4, except that each processor services all processes in its own pyramid. The processors work in parallel and in synchronization so that no processor has to wait for the results of its fellow processors.

It is not difficult to see that each processor implements somewhat fewer than $\frac{8}{7}(\frac{n}{k})^3$ processes. Similarly, a processor has to implement the communication between the processes; the time for communication between processors residing on the same processor can be ignored. The communication time between processors has to be accounted for. This communication is represented by "wires" from processes on the

pyramid faces to neighboring processes on neighboring processors. In 3D the number of these wires is proportional to $(\frac{n}{k})^2$, and the sequel considers their effect on the communication time.

The calculation of the time computation is based on 20M floating point operations per second, and the communication time is based on transfer time of 80 nanoseconds per 32 bits. (See reference [15] for an example.)

Communication time. It is assumed that each processor has pseudo-cells, or p-cells, one p-cell for each cell z, such that z is a member of an interactive set of a cell in the processor's pyramid, z not in the pyramid. The number of such cells can be calculated. For a cell y, y a cell on the pyramid face, these are the *children*(*neighbors*(*parent*(y))) that are not in the pyramid. (Note that *parent*(y) is in the pyramid and that we are interested in *neighbors*(*parent*(y)) that are not in the pyramid.) If we "cover" the pyramid faces with such cells, this cover includes all the interactive set members of cells in the pyramid where the members are not in the pyramid. The number of cells in this cover is

$$(53) \qquad (\textit{number of children}) \times (\textit{number of cells in the faces of a } \frac{n}{2k} \times \frac{n}{2k} \times \frac{n}{2k} \textit{ pyramid}).$$

The number of cells in a face of a $n \times n \times n$ pyramid is

$$(54) \qquad n^2(1 + \frac{1}{4} + \frac{1}{16} + \cdots) < \frac{4}{3}n^2.$$

Thus, the number of cells in the above cover becomes

$$(55) \qquad 8\left(\frac{4}{3}\left(\frac{\frac{n}{k}}{2}\right)^2 \times 6 + 8\log\left(\frac{n}{2k}\right)\right).$$

The last term is the contribution of cells that cover the corners. This last term can be ignored.

Thus, the number of p-cells is about $16(\frac{n}{k})^2$, and this is the number of messages the processor has to send or to receive (ignoring the fact that some processors do not have neighbors in all directions).

The GR algorithm defines the content of such a cell by $n_p + 3$ coefficients (the 3 represents the coordinates of the source). Thus, the communication time is

$$(56) \qquad (n_p + 3) \times 32 \times 16(\frac{n}{k})^2(\frac{80 \times 10^{-9}}{32}).$$

Similarly, the BH algorithm defines the content of such a cell by $1 + 3$ coefficients (the 3 represents the coordinates of the source). Thus, the communication time is

$$(57) \qquad 4 \times 32 \times 16(\frac{n}{k})^2(\frac{80 \times 10^{-9}}{32}).$$

Computation time. A processor that implements a $\frac{n}{k} \times \frac{n}{k} \times \frac{n}{k}$ base pyramid has to perform all the operations that its processes do.

For the 3D GR algorithm the computation time is

$$(58) \qquad (\frac{n}{k})^3(\tau(\Phi n^0) + \frac{8}{7}\tau(Translate_z) + \frac{8}{7}n_{set}\tau(Convert_z) + \frac{8}{7}\tau(Shift_z) + \tau(Evaluation)).$$

TABLE 3
Array of processors' far-field communication and computation time — GR algorithm.

n	m	par ticles	p	n_p	k_p	c_p	k=2 sec.		k=4 sec.		k=8 sec.	
							comm. time	comp. time	comm. time	comp. time	comm. time	comp. time
32	20	640K	1	4	7	7	2.3	1.05	0.57	0.132	0.15	0.016
32	20	640K	2	10	28	25	4.27	3.6	1.07	0.46	0.27	0.057
32	20	640K	3	20	84	65	7.53	9.3	1.87	1.16	0.47	0.146
38	20	1250K	1	4	7	7	3.25	1.76	0.8	0.22	0.21	0.027
38	20	1250K	2	10	28	25	6.0	6.1	1.51	0.764	0.37	0.095
38	20	1250K	3	20	84	65	10.6	15.6	2.63	1.95	0.67	0.244

TABLE 4
Far-field ratios — pyramid to array — GR algorithm.

n	m	particles	p	ratio k=2	ratio k=4	ratio k=8
32	20	640K	3	0.32	1.75	8.61
38	20	1250K	3	0.20	1.16	5.23

Using (32), (34), (38), (40), and (41), the expression becomes

$$(\frac{n}{k})^3 \left(n_p(4m-1) + 3\frac{8}{7}k_p + 3n_{set}c_p + 4\frac{8}{7}c_p + 3mn_p \right) t_+, \tag{59}$$

which results in

$$(\frac{n}{k})^3 \left(n_p(7m-1) + \frac{8}{7}(3k_p + (3n_{set}+4)c_p) \right) t_+. \tag{60}$$

The computation time of the 3D BH algorithm almost entirely consists of the evaluation time. Thus, for a $\frac{n}{k} \times \frac{n}{k} \times \frac{n}{k}$ base pyramid the computation time is

$$(\frac{n}{k})^3 (\textit{evaluation time for one processor of the pyramid}) \frac{\textit{pyramid processor speed}}{\textit{array processor speed}}. \tag{61}$$

GR algorithm summary. Table 3 summarizes the computation and communication time for the GR algorithm and for various values of n, p, and k.

The above numerical results can be compared with the performance of the GR algorithm on the pyramid (§7.2). Table 4 compares the performance for $p = 3$. Define *Ratio*:

$$\textit{Ratio} \stackrel{\text{def}}{=} \frac{(\textit{total time pyramid})}{(\textit{total time } k \times k \times k \textit{ array})}. \tag{62}$$

BH algorithm summary. Table 5 summarizes the computation and communication time for the BH algorithm.

The above numerical results can be compared with the performance of the BH algorithm on the pyramid (§7.1). The ratios are given in Table 6.

Summary. It follows from the previous section that

TABLE 5
Array of processors' far-field communication and computation time — BH algorithm.

n	m	particles	k=2 sec.		k=4 sec.		k=8 sec.	
			comm. time	comp. time	comm. time	comp. time	comm. time	comp. time
32	20	640K	1.3	126.4	0.33	16.0	0.08	2.0
38	20	1250K	1.84	179.2	0.46	22.4	0.11	2.4

TABLE 6
Far-field ratios — pyramid to array — BH algorithm.

n	m	particles	Ratio k=2	Ratio k=4	Ratio k=8
32	20	640K	0.15	1.21	9.8
38	20	1250K	0.14	1.25	9.12

$$\textit{computation time pyramid} + \textit{communication time pyramid} \leq$$

(63)
$$\textit{total time hypercube} \leq$$

$$\textit{computation time pyramid} + d \times (\textit{communication time pyramid}),$$

where $d = 4$. Bounds for the ratio of the total time hypercube, to the total time array can be derived from this relation. Since I believe that $d = 4$ is a rather conservative bound, I am giving the hypercube the benefit of the doubt. For the sake of comparison, consider its performance equal to the pyramid.

The above numerical results indicate that the array can comfortably compete with either the pyramid or the hypercube (both with the CM2 connection-machine parameters). A general comparison of the array and the hypercube has to weigh two additional factors: First, the hypercube, in the form of the connection machine, is an existing commercially available equipment; the array has to be built. Second, the smaller arrays, e.g., $2 \times 2 \times 2$, can be expected to be relatively inexpensive, probably more than an order of magnitude less expensive than the connection machine.

8. Near-field computation. This section estimates the computation and communication time required for near-field computation. These estimates are added to the estimates of the far-field computation to yield an estimate for the total time required.

Let q be a point mass at some cell n. The near-field force on q is the force exerted on q by particles in n and in $neighbors(n)$. This force is computed using (48).

This calculation is rather straightforward, and the only issue is how to take advantage of the symmetry of the force ($F_{x_i} = -F_{x_i}$), which, in parallel computation, requires some elaboration.

Consider the pyramid's computational structure. Only the bottom layer processes participate in the near-field computation. Consider a cell c and its neighbors in this layer. c and each of its neighbors, say e, determine a *direction.* This direction is determined by $x_e - x_c$, where x_e is the center of the neighbor cell and x_c is the center

TABLE 7
Near-field communication and computation time.

m	communication time $n_n m$ millisec.	computation time $36(\frac{n_n}{2}+1)m(m-1)t_+$ millisec.
10	65	2,268
20	130	9,576

of c. Clearly, if $x_e - x_c$ is a direction, so is $x_c - x_e$. Let D be a set of directions such that if d is a direction, then either $d \in D$ or $-d \in D$, but not both.

For all directions d in D, each bottom-layer cell c sends the value and coordinates of each of its mass points to its neighbor in the d direction. The neighbor calculates the forces on each of these mass points due to its own mass points and sends these m vectors back, each with the coordinates of the mass point.

The total time is

$$\frac{36}{2}m(m-1)t_+ + \frac{n_n}{2}\left(36m(m-1)t_+ + 2mt_{message\ unit}\right). \tag{64}$$

(See details in [19].)

Since the communication time is small compared to the computation time, we (almost) get the $\frac{1}{2}$ factor discussed in the beginning of this section. This is achieved by having one processor calculate the force and send the results back to its neighbor.

Using the CM2 connection-machine parameters once again (20K floating point operations per second, 250 microseconds per message) the time for evaluating the near-field by the pyramid is given in Table 7; (n_n is 26 for 3D).

The array of larger processors is faring just as well in the near-field computation. As seen from the above, communication time is negligible. Since only the bottom-layer processors of the pyramid participate in the calculation, we get for an $n \times n \times n$ base pyramid

$$\frac{pyramid\ time}{k \times k \times k\ processor\ array\ time} = \frac{1}{(\frac{n}{k})^3 \frac{1}{speedup}}, \tag{65}$$

where

$$speedup = \frac{t_+\ for\ a\ pyramid\ processor}{t_+\ for\ a\ larger\ processor}. \tag{66}$$

The speedup is 1000 for the values used in §7.4. Thus, the value of

$$\frac{pyramid\ time}{k \times k \times k\ processor\ array\ time}$$

is given in Table 8 for various values of k.

9. Summary and conclusions. The work formulates the N-body problem as a set of recursive equations with boundary conditions. The numerical solution to this set of equations can be viewed as a pyramid structure of processes. There are three kinds of processes: the bottom-level processes, the top-level processes, and all the rest. All processors are implemented by means of five functions: $\hat{\Phi}_n^0$, $Translate_z$,

TABLE 8
Near-field ratios — pyramid time to processor array time.

k	n=32	n=38
2	0.25	0.146
4	2	1.17
8	16	9.33

Convert$_z$, *Shift*$_z$, *Evaluate*. Both the BH and the GR algorithms have the same structure. They differ from each other in the above functions. This formulation of the N-body problem simplifies the understanding of the algorithms, their analysis, and programming.

The pyramid structure is mapped into three different architectures that can be viewed as possible hardware implementations. For each mapping the amount of time for arithmetic computations and the time for communication between processors is estimated. Out of these estimates several qualitative conclusions are derived.

Parallel execution reduces *both* the BH and the GR algorithms to $O(\log N)$, where N is the number of particles; this bound covers both computation and communication time. This is somewhat of a surprise, because in the serial computation the BH algorithm is considered an $O(N \log N)$ algorithm while the GR one is $O(N)$.

The algorithms exhibit both common and differing patterns of behavior. In both cases, given the height of the tree h, the time spent in the tree is independent of m, the number of particles in a cell. The evaluation stage, however, is hard on BH; for large m most of the time is spent in it, and computation time far exceeds communication time. For large m the GR algorithm performs exceedingly well; the evaluation of the potential function for each particle is straightforward. Clearly, using a $38 \times 38 \times 38$ pyramid, the GR algorithm can handle 10M equally distributed particles almost as fast as it handles 1M. At these numbers, the increase of m from 20 to 200 increases the total time by about 25 percent only because the algorithm is bounded by the communication time rather than by the evaluation time at the bottom cell.

Since the BH algorithm is much simpler to program, it is worthwhile noting that its performance for 62K particles, one at a cell, is about equal to that of the GR algorithm when $p = 2$, using the same number of particles. A complete comparison of BH and GR requires comparing the algorithms' performance for the same global error. As far as I know, no analysis of the global error as function of the main parameters exists for any of the two algorithms. (References [9] and [10] present the error analysis of each step of the GR algorithm, but the effects are not summed to form the global error. However, by increasing p, GR can arbitrarily reduce the error.) Thus, a calculation of the performance for a given global error is yet to be done.

A hypercube machine can simulate the pyramid structure without conflict of messages and with a slowdown equal to or less than $d = 4$.

Using the GR algorithm with $p = 3$ and a connection machine of the present size, it is possible to calculate a solution to the 1.25M particle (static) many-body problem in about 24 seconds. This number is based on multiplying $d = 4$ by the communication time computed for the tree and adding the computation times for the far field and for the near field. It is considered a rather conservative estimate. This number is also the estimate for one timestep for the dynamic many-body problem.

For running the N-body algorithm, the connection machine has a clear advantage:

it is an existing product that can be readily used. The analysis, however, points out some possible disadvantages; while 64K processors seem quite a large number, the 3D N-body problem reaches the limit rather fast. One has to increase m , assigning more particles per cell. Clearly, at least for the BH case, we would rather increase the number of processors, a feature the hardware does not support.

Rough performance estimates for an array of faster processors indicate that with an array of modest size it is possible to get performance competitive to that of the connection machine.

The above time calculations use the same value of m for all cells. This suggests particles that are fairly uniformly distributed in space. Often, physical systems exhibit sparsity, certain regions of space have many particles while others are empty or nearly so. The algorithms' implementations as presented here can handle sparse systems, but they do not take advantage of sparsity to save either time or space. While taking advantage of sparsity awaits further work (see, however, reference [5] for an "adaptive" cell), some of its direction is clear from this work.

Clearly, the GR algorithm has an advantage in the sparse case as the penalty for having cells with a large m is small. In the connection machine, whenever the number of particles for any cell exceeds the available storage, one must refine the grid, and since the number of cells goes way up, give up the uniform map of the pyramid to the hypercube (§7.3) and, perhaps, the uniform partition of the domain to cells. In this case, assign to processors only those cells that have particles and rely on a map for sending the messages to their destinations. In the dynamic N-body problem one must also provide for the allocation and deallocation of cells as the particles move.

The case of the array of processors is somewhat simpler since each processor has a sizable memory and empty cells do not occupy space. Since the array can work in an multiple-instructions-multiple-data (MIMD) mode, a processor can finish "earlier" and give a hand to its neighbor. This, however, raises the questions of the distribution of load and of the migration of processes among processors, questions whose absence made the SIMD scheme tractable.

The computation and the communication times computed should be considered initial estimates, as data-moving operations were not taken into account. Moreover, these estimates await experimental verification, preferably by programs simulating the above processes. This work is currently under way.

Acknowledgments. Thanks are due to Gerald J. Sussman for introducing me to astronomy; Alexandre J. Chorin for bringing the work of Greengard and Rokhlin to my attention and for commenting on the manuscript; Harold Abelson and Feng Zhao for detailed comments and discussions; Joshua Barnes, Andrew A. Berlin, Piet Hut, and J. Makino for discussions; and Charles E. Leiserson for pointers to the graph work.

REFERENCES

[1] R. W. HOCKNEY AND J. W. EASTWOOD, *Computer Simulation Using Particles,* McGraw Hill, New York, 1981.

[2] A. J. CHORIN, private communication.

[3] ———, *Numerical studies of slightly viscous flow,* J. Fluid Mech., 57(1973), pp. 785-796.

[4] C. ANDERSON AND C. GREENGARD, *On vortex methods,* SIAM J. Numer. Anal., 22(1985), pp. 413-440.

[5] A. APPEL, *An efficient program for many-body simulation,* SIAM. J. Sci. Statist. Comput., 6(1985), pp. 85-103.

[6] J. BARNES AND P. HUT, *A hierarchical $O(N \log N)$ force calculation algorithm*, Nature, 324(1986), pp. 446-449.
[7] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, J. Comp. Phys., 73(1987), pp. 325-348.
[8] L. GREENGARD, *The rapid evaluation of potential fields in particle systems*, Ph.D. thesis, Yale University, New Haven, CT, April 1987; MIT Press, Cambridge, MA, 1988.
[9] L. GREENGARD AND V. ROKHLIN, *The rapid evaluation of potential fields in three dimensions*, Res. Report YALEU/DCS/RR-518, Yale University, New Haven, CT, January 1987.
[10] F. ZHAO, *An $O(N)$ algorithm for three-dimensional N-body simulations*, Master's thesis , Dept. of Electrical Engineering and Computer Science, Mass. Inst. of Technology, Cambridge, MA, October 1987.
[11] D. HILLIS, *The Connection Machine,* MIT Press, Cambridge, MA, 1985.
[12] D. HILLIS AND G. STEELE, JR., *Data parallel algorithms,* Comm. ACM, 29(1986), pp. 1170–1183.
[13] J. MAKINO, Unpublished report, School of Advanced Study, Princeton, NJ, 1987.
[14] B. KAHLE, private communication.
[15] Bit Bipolar Integrated Technology, Inc., *Designing a micro-sequenced* CPU *with the* B3110/B3120, AN-1, August 1987.
[16] S. BHATT, F. CHUNG, T. LEIGHTON, AND A. ROSENBERG, *Optimal simulation of tree machines,* Proc. 27th Annual IEEE Conference on Foundations of Computer Science, October 1986, pp. 274-282.
[17] T. LEIGHTON, 18.435 course handouts, Mass. Inst. of Technology, Cambridge, MA, November 1987.
[18] Macsyma Reference Manual, version 12, Symbolics, Inc., 1986.
[19] J. KATZENELSON, *Computational structure of the N-body problem,* Mass. Inst. of Technology, Cambridge, MA, Artificial Intelligence Laboratory, AI Memo 1042, April 1988.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 817-845, September 1989

MATHEMATICAL STRUCTURE OF COMPOSITIONAL RESERVOIR SIMULATION*

JOHN A. TRANGENSTEIN† AND JOHN B. BELL†

Abstract. In this paper multicomponent two-phase isothermal fluid flow in petroleum reservoirs is described. The fluid-flow model consists of component conservation equations, Darcy's law for the volumetric flow rates, balance between the fluid volume and the rock void, and the conditions of thermodynamic equilibrium that determine the distribution of the chemical components into phases. Thermodynamic equilibrium is described by means of a mathematical model for the chemical potentials of each component in each phase of the fluid. The flow equations are manipulated to form a pressure equation and a modified component-conservation equation; these form the basis for the sequential method. It is shown that the pressure equation is parabolic under reasonable assumptions on the thermodynamic equilibrium model, and that the component-conservation equations are hyperbolic in the absence of diffusive forces such as capillary pressure and mixing. A numerical method based on the sequential formulation of the flow equations is outlined and used to illustrate the kinds of flow behavior that occur during miscible gas injection.

Key words. reservoir simulation, hyperbolic conservation law, thermodynamic equilibrium

AMS(MOS) subject classification. 76T05

1. Introduction.

1.1. Overview. In this paper we develop and analyze a sequential method for isothermal fluid flow in a petroleum reservoir. We use the compositional model for the fluid flow, since it provides a more realistic description of the reservoir fluid than does the more conventional black-oil model. The compositional model is suitable for modeling several important enhanced oil recovery techniques, such as condensing gas drive and miscible gas injection.

The sequential method developed in this paper represents an extension of the sequential method developed for the black-oil model in [32]. (Some discussion of sequential methods can be found in that work and its references.) The compositional and black-oil models share a number of common features. Darcy's law, the volume balance, and the component conservation equations are virtually identical. The primary difference between the two models lies in the formulation of thermodynamic equilibrium. The compositional model uses a more detailed description of the fluid thermodynamics through a phase equation of state. As a result, thermodynamic equilibrium is described by a constrained minimization problem for the Gibbs free energy of the fluid. This characterization of phase equilibrium plays a crucial role in the analysis of the flow equations. Consequently, a separate analysis of the sequential method is needed for the compositional model, even though the results of the analysis are similar to those for black-oil.

1.2. Compositional flow equations. Before discussing the details of the compositional model, let us introduce the flow equations in very general terms. We consider the reservoir fluid to be made of at least two but often as many as a dozen different chemical compounds called *components*. The moles (mass divided by molecular weight) per pore volume of each of these components in the total fluid at any point in space is denoted by the vector n. We assume that at each point in space and time the components associate to form at most two *phases*, called liquid and vapor, in order to

* Received by the editors March 7, 1988; accepted for publication (in revised form) September 19, 1988.
† Exxon Production Research Company, P.O. Box 2189, Houston, Texas 77252-2189. Present address, Lawrence Livermore National Laboratory, Computing and Mathematics Research Division, P.O. Box 808, L-316, Livermore, California 94566.

reach thermodynamic equilibrium. Given n and the pressure p, the compositional model describes thermodynamic equilibrium by determining the vectors n_l and n_v of moles of each component in each phase divided by the pore volume. This is done in such a way that the fluid mass is balanced between the phases, or

$$n = n_l + n_v. \tag{1.1}$$

Stable thermodynamic equilibrium is attained by requiring that the Gibbs free energy of the total fluid be minimized over all possible nonnegative phase compositions n_l and n_v.

In addition, the compositional model uses n_l, n_v, and p to compute the ratios of the phase volumes to the pore volume denoted by u_l and u_v. From this, we derive the fluid equation of state

$$u_l + u_v = 1, \tag{1.2}$$

which states that the sum of the phase volumes is the pore volume.

The third assumption about reservoir fluid flow is that the volumetric flow rates (that is, phase volume per flow area per time) of each of the phases is given by Darcy's law:

$$\nu_l = -\lambda_l \left(\frac{\partial p}{\partial x} - \rho_l \frac{\partial d}{\partial x} \right) \kappa, \qquad \nu_v = -\lambda_v \left(\frac{\partial p}{\partial x} - \rho_v \frac{\partial d}{\partial x} \right) \kappa. \tag{1.3}$$

Here λ_l and λ_v are the phase mobilities, ρ_l and ρ_v are the phase mass densities, κ is the rock permeability, and d is the reservoir depth multiplied by a gravitational constant.

Finally, we require that the mass of each of the components is conserved. Using the Darcy velocities, we see that the flux of each component is the sum over phases of the density of that component times its flow rate; the vector of total fluxes of each of the components is therefore $n_l(\nu_l/u_l) + n_v(\nu_v/u_v)$. Thus the conservation law is written in the form

$$\frac{\partial (n\phi)}{\partial t} + \frac{\partial}{\partial x} \left(n_l \frac{\nu_l}{u_l} + n_v \frac{\nu_v}{u_v} \right) = 0. \tag{1.4}$$

We can use (1.1) and (1.3) to reduce (1.2) and (1.4) to a system of equations in the pressure p and the fluid composition n. Of course, the flow equations (1.1)–(1.4) have been presented only in outline form. A more complete description and detailed discussion of the functional dependencies will appear in the text below.

1.3. Outline. Our first task in this paper is to describe the compositional phase behavior model in § 2. This will involve examining the mathematical properties of equation-of-state models for the pressure versus molar density behavior of individual phases to develop the thermodynamic description of phase equilibrium. The key fact in this stage of the work is that the phase behavior is determined by the minimization of the Gibbs free energy. The mathematical characterization of the minimum is essential to the analysis of both the pressure equation and the component conservation law.

Afterward, in § 3 we examine those aspects of Darcy's law that are different from the black-oil model. Then we examine the fluid flow equations and the sequential approach to their solution in § 4. In § 5 we analyze the pressure equation and show that it is parabolic, and in § 6 we analyze the linearized wave structure of the component conservation law. The analysis of the wave structure is constructive; that is, the method we use to analyze the eigenvalues and eigenvectors is the same as the method we use to compute them. Finally, in § 7, we outline the numerical method and present some numerical results for one-dimensional flows.

2. The compositional equation of state.

2.1. Components and phases. In compositional reservoir simulation we consider the reservoir fluid to consist of a variety of chemical compounds, especially hydrocarbons (methane, ethane, and so on). Various enhanced recovery techniques involve injection of additional chemicals such as carbon dioxide, nitrogen, and water. Typical hydrocarbon reservoir fluids actually involve hundreds of different chemicals, far too many to be modeled in reservoir simulation. Instead, the heavier components are lumped together in groups, each of which is treated as a single pseudocomponent [7], [20]. Thus, we include in our notion of *component* both individual chemical species and pseudocomponents. The amounts of these components are defined in terms of moles per pore volume. The term "mole" is a familiar idea in chemistry, where the number of moles of a component is equal to the mass divided by the molecular weight.

These components are allowed to associate as *phases* in the reservoir. Here "phase" refers to any homogeneous fluid separated from other phases by an interface. We are concerned in this paper with reservoirs containing at most two phases, which we label liquid and vapor. For the purposes of this paper, we assume that every component in the fluid actually appears in every phase that is formed.

Since components flow in phases but are conserved by moles (or equivalently by mass), it is important to determine how the moles of components in the fluid are distributed into the phases. This phase behavior problem can be partially stated as follows. Given the pressure p and the vector n of moles per pore volume of each of the components in the fluid, we must find the vectors n_l and n_v of moles per pore volume for each of the components in the liquid and vapor phases. We require that the mass be balanced between the phases; that is, that (1.1) holds. Of course, the phase compositions n_l and n_v must be nonnegative.

We adopt the notation

$$N \equiv [n_l, n_v].$$

Then the mass-balance condition (1.1) can be written in the form

$$n = Ne. \tag{2.1}$$

Here and throughout the remainder of the paper, e is a vector of the appropriate length with all entries equal to one.

Of course, we have not yet provided enough information to determine unique values for the unknown phase compositions n_l and n_v. This additional information is contained in the fact that the components will distribute themselves in order to reach stable thermodynamic equilibrium. For this reason we examine the thermodynamic properties of the phases.

2.2. Phase behavior overview. Before we examine the mathematical structure of compositional phase behavior, let us take time to examine the important physical aspects of the fluid model. For a fixed fluid composition, the effects of pressure and temperature can be illustrated by a phase envelope, shown in Fig. 1. The phase envelope is the curve that separates the two-phase region from the single-phase region. Points along this curve are either bubble points, dew points, or critical points. In order to label the points on the phase envelope properly, let us consider two thought experiments. For a sufficiently low temperature (labeled T_1 on Fig. 1), let us begin at a high pressure, so that only one phase is formed. Intuitively, this phase should be a liquid. As we drop the pressure, keeping the temperature fixed, we reach a point at which a vapor phase is formed. This pressure is called the *bubble point* for this temperature. As we drop the pressure further, the vapor phase increases in volume, as does the total fluid

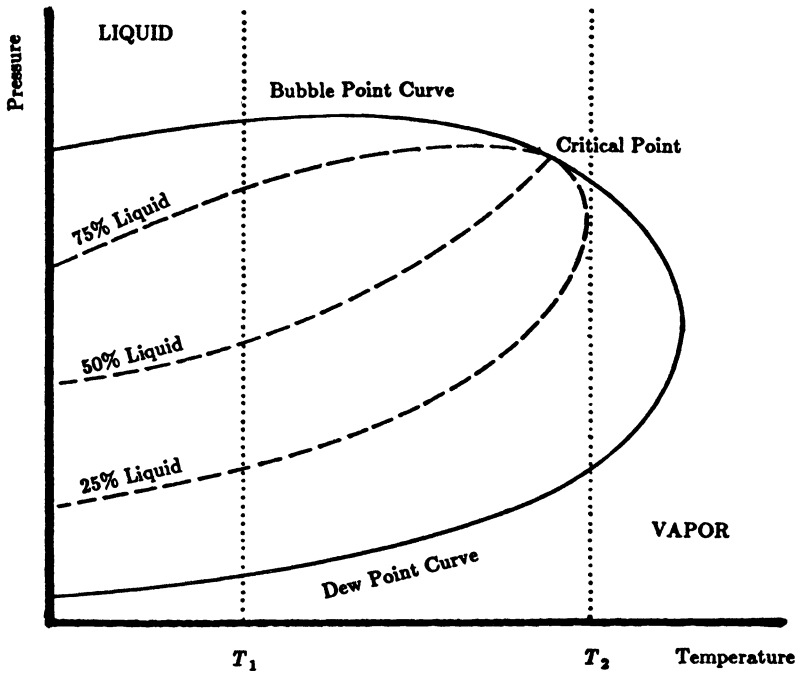


FIG. 1. *Phase envelope.*

volume. The liquid phase may or may not increase in volume initially, due to competing effects of the drop in pressure and the release of gas. Finally, at a sufficiently low pressure the liquid completely volatilizes and only a vapor phase remains. This point is called a *dew point.* For pressures below the dew point, only a vapor phase is formed.

For sufficiently high temperatures, only a vapor phase is formed at any practical pressure. However, for temperatures (such as T_2 in Fig. 1) in the region of *retrograde condensation,* the effect of pressure changes is counterintuitive. For low pressures, only a vapor phase is formed, as might be expected. As the pressure is increased, a dew point is reached and a liquid phase is formed. However, it is surprising to note that as the pressure is increased further the liquid volatilizes until a second dew point is reached. Above this pressure, the fluid forms only a single phase, normally considered to be a vapor.

The point of intersection between the bubble point curve and the dew point curve is the *critical point.* At the critical point, liquid and vapor have identical compositions. This is a very desirable regime for the operation of enhanced oil recovery methods because the phases will flow without inhibiting each other. This allows for a greater recovery of the original reservoir components. However, the critical point is almost never approached for a given reservoir fluid composition by varying the pressure and temperature. Instead, the reservoir engineer has better control of the fluid composition by means of his choice of injected fluids. To view the effects of variations in the fluid composition, we need a different picture.

In Fig. 2 we have drawn a ternary diagram for a fluid consisting of three components. Here the pressure, temperature, and total moles in the fluid are fixed while the moles of the individual components are allowed to vary. (The constraint that the total number of moles is fixed means that there are only two remaining degrees of freedom; the constraints that the moles of each component are nonnegative mean that

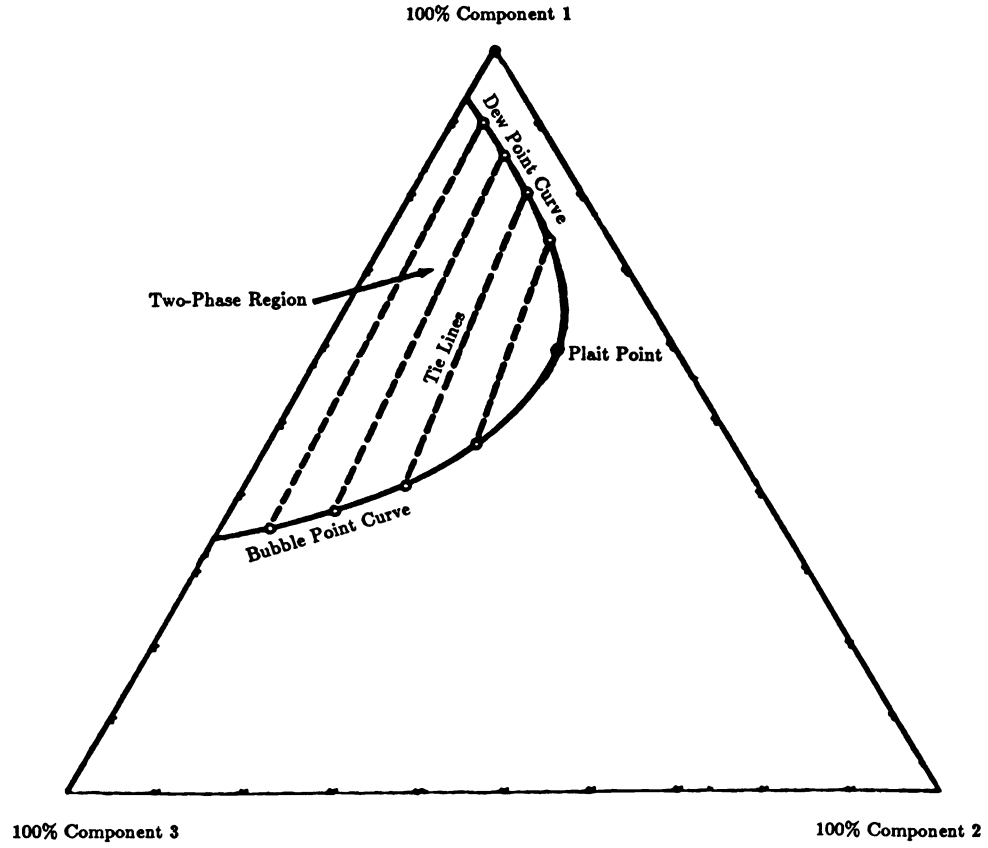


FIG. 2. *Ternary diagram.*

the allowable fluid compositions form a triangle.) Again we note that there is a curve separating the two-phase region from the one-phase region. There are two parts to this curve: a bubble-point curve representing liquid phase compositions for two-phase mixtures and a dew-point curve representing vapor phase compositions. Every fluid composition inside the two-phase region is associated with its equilibrium liquid phase composition on the bubble-point curve and its equilibrium vapor phase composition on the dew-point curve. The two-phase fluid compositions with the same vapor and liquid phase compositions lie along the *tie-line* between these two points. In other words, points on a tie line in the two-phase region correspond to fluids with the same liquid and vapor phase compositions but with varying phase volumes. At the *Plait point*, the tie-lines become infinitesimally short, and the equilibrium vapor and liquid phases become identical. These ternary diagrams and associated tie-lines are often used to design miscible gas injections for enhanced oil recovery [30].

2.3. Thermodynamic equilibrium for two-phase fluids. Now that we have examined the physical aspects of multicomponent phase equilibrium, let us examine its mathematical aspects. In order to determine the equilibrium distribution of the components in the phases, we need to employ the *Gibbs free energy*

$$\gamma \equiv n_l^T f_l + n_v^T f_v.$$

Here, the phase potential functions f_l and f_v are nonlinear functions of the pressure,

and the phase compositions n_l and n_v, respectively, and are outlined in § 2.3.1 below. (We have avoided the use of the symbol μ, which is standard in thermodynamics, in order to avoid conflicts with the notation for viscosity in § 3 below.) Stable equilibrium [29] for a fixed pressure p occurs when the phase compositions n_l and n_v lead to the smallest possible value of the Gibbs free energy among all phase compositions that are nonnegative and that balance mass by satisfying (1.1).

The pair of phase compositions n_l and n_v are uniquely specified by the condition that the Gibbs free energy be minimized. The labels for the individual phases must be determined by the condition that liquid has greater mass density than vapor (see § 3.5 below). However, the numerical determination of the equilibrium phase compositions is not a simple matter; for more details of the mathematical structure of the problem and an efficient, robust algorithm, see [31]. In this paper, we are content with analyzing the mathematical properties of the equilibrium phase compositions.

2.3.1. Equation of state for phases. It is common in compositional reservoir simulation to use an equation of state to model the dependence of the pressure on the temperature, composition, and molar specific volume of the phase. The Peng–Robinson model [23] is the equation of state used most often in reservoir simulation. We do not describe this model in detail, but concentrate on its mathematical properties instead. For a survey of related equations of state for phases, see [33]. For a discussion of some of the inherent inadequacies of this class of equations of state for phases, see [21].

The equation of state for a phase can be manipulated and applied to thermodynamic computations to develop *phase potential* functions [22]. These functions represent the potential energy per mole of each component in the phase. It is important to note that these functions depend on the pressure, temperature, and composition of the phase, but not on the amount of mass in the phase. In other words, the liquid phase potentials actually depend on the liquid phase mole fractions

$$x_l \equiv \frac{n_l}{e^T n_l}, \tag{2.2}$$

and the vapor phase potentials depend on the vapor phase mole fractions x_v. Thus, we write

$$f_l = f_l(x_l, p) \quad \text{and} \quad f_v = f_v(x_v, p)$$

and use the chain rule to compute molar derivatives of the phase potentials:

$$\frac{\partial f_l}{\partial n_l} = \frac{\partial f_l}{\partial x_l}\frac{\partial x_l}{\partial n_l} = \frac{\partial f_l}{\partial x_l}(I - x_l e^T)\frac{1}{e^T n_l}. \tag{2.3}$$

We have suppressed the dependence on temperature since we assume the fluid flow to be isothermal. Often we use the same equation of state for both phases, and thereby obtain the same functional form for both vectors of phase potentials. (There is a subtle point regarding the choice of the solution of the phase equation of state so that it is independent of the phase labels; for more details, see [31].)

2.3.2. Gibbs–Duhem equation. Because the phase potential functions are derived from thermodynamic principles, they satisfy some important physical constraints. One of these constraints is the Gibbs–Duhem equation [28], which says that the phase potentials are the derivatives of the phase energy with respect to the phase composition. For the liquid phase, we have

$$f_l = \frac{\partial \gamma_l}{\partial n_l};$$

a similar equation holds for vapor. The following constraint on the derivatives of the phase potentials is an equivalent statement of the Gibbs–Duhem equation:

$$n_l^T \frac{\partial f_l}{\partial n_l} = 0 = n_v^T \frac{\partial f_v}{\partial n_v} \tag{2.4}$$

for all nonnegative vectors n_l and n_v. It is straightforward to verify that the phase potentials determined by the Peng–Robinson equation of state satisfy the Gibbs–Duhem equation (2.4).

The Gibbs–Duhem equation has a variety of useful consequences. One is that $\partial f_l/\partial n_l$ and $\partial f_v/\partial n_v$ are symmetric. To see this for liquid, let us consider the scalar function $\gamma_l(n_l) \equiv n_l^T f_l(x_l, p)$, where x_l is given by (2.2). Then the Gibbs–Duhem equation shows that

$$\frac{\partial \gamma_l}{\partial n_l} = f_l^T + n_l^T \frac{\partial f_l}{\partial n_l} = f_l^T,$$

and

$$\frac{\partial}{\partial n_l}\left(\frac{\partial \gamma_l}{\partial n_l}\right)^T = \frac{\partial f_l}{\partial n_l}.$$

Since the matrix of second derivatives of a scalar is symmetric, we must have that $\partial f_l/\partial n_l$ is symmetric. In an identical fashion we can prove that $\partial f_v/\partial n_v$ is symmetric.

Another immediate consequence of the Gibbs–Duhem equation is that $\partial f_l/\partial n_l$ and $\partial f_v/\partial n_v$ are singular. In fact, the symmetry of $\partial f_l/\partial n_l$ shows that n_l lies in the null space of this matrix. This fact can be derived independently from (2.3).

2.3.3. Kuhn–Tucker conditions. Following [31], we use the shorthand notation

$$\bar{n} \equiv \begin{bmatrix} n_l \\ n_v \end{bmatrix} \quad \text{and} \quad \bar{f} \equiv \begin{bmatrix} f_l \\ f_v \end{bmatrix}.$$

We also define the matrix

$$B \equiv \begin{bmatrix} I \\ -I \end{bmatrix}.$$

Then if $\bar{n}_0$ is any particular solution to the mass balance condition, the general solution to this equation is

$$\bar{n} = \bar{n}_0 + Bz, \tag{2.5}$$

where z is undetermined. We shall use this notation to enforce the equality constraints in the Gibbs free energy minimization problem.

If we assume that both phases are formed, then we can use the Gibbs–Duhem equation to find the first three terms in the Taylor series approximation to $\gamma = \bar{n}^T \bar{f}$:

$$\gamma(\bar{n}_0 + Bz) \approx \gamma(\bar{n}_0) + g^T z + \tfrac{1}{2} z^T G z,$$

where

$$\begin{aligned} g &\equiv B^T \bar{f}(\bar{n}_0, p) = f_l - f_v, \\ G &\equiv B^T \frac{\partial \bar{f}}{\partial \bar{n}}(\bar{n}_0, p) B = \frac{\partial f_l}{\partial n_l} + \frac{\partial f_v}{\partial n_v}. \end{aligned} \tag{2.6}$$

The Gibbs free energy γ has a local minimum at $\bar{n}_0$ whenever $g = 0$ and G is positive definite. These are the familiar Kuhn-Tucker sufficient conditions (see [17] or any modern text on constrained optimization). The positivity condition on the Hessian matrix G serves the purpose of avoiding stationary points of the energy that are not local minima and are therefore physically unstable under infinitesimal perturbations. The condition that $g = 0$ merely says that for each component in the two-phase fluid, the potentials in each phase must be equal.

2.3.4. Fluid equation of state. Recall from thermodynamic principles that the molar density of a phase is the derivative of the Gibbs free energy per mole with respect to the pressure. As a result, the phase equation of state can be manipulated to obtain a formula for the molar density of the phase. (Some authors have advocated modifications to the Peng-Robinson equation of state that improve its prediction of molar densities without changing its predictions of phase equilibrium compositions; see [15].) Like the phase potentials f_l, the molar density ξ_l is a function of the pressure p and the mole fractions x_l, rather than the component moles n_l. A similar statement holds for vapor. Using these functions, we can compute the ratios of phase volumes to pore volume

$$u_l = \frac{e^T n_l}{\xi_l}, \qquad u_v = \frac{e^T n_v}{\xi_v}.$$

These equations can be written in the matrix-vector form

$$u^T = e^T N \Xi^{-1}, \tag{2.7}$$

where Ξ is the diagonal matrix

$$\Xi \equiv \operatorname{diag}(\xi_l, \xi_v).$$

Then the fluid equation of state says that the sum of the phase volumes must equal the pore volume, or

$$e^T u = 1.$$

This equation is equivalent to (1.2).

Let us take time to discuss the mathematical implications of the fluid equation of state. Equation (2.7) shows that the vector u of ratios of phase volumes to pore volumes is a function of the phase compositions n_l, n_v and the pressure p, partially as a result of the dependence of the molar densities ξ_l, ξ_v on the mole fractions and pressure. In turn, the phase compositions n_l and n_v depend on the fluid composition n and the pressure p through the condition that they minimize the Gibbs free energy γ. Another way to express this dependence of n_l and n_v on n and p is to say that the reduced gradient of the Gibbs free energy, $g = f_l - f_v$, must be zero. Thus the fluid equation of state (1.2) actually relates the fluid pressure p to the fluid composition n.

2.4. Single-phase fluids. In the preceding §§ 2.3.3 and 2.3.4, we have assumed that the solution to the phase behavior problem leads to the formation of two phases. This is not always the case, however. Recall the discussion and figures in § 2.2. For sufficiently large pressures and low temperatures, we expect all of the components to be contained in a single liquid phase. On the other hand, for low pressures and high temperatures we expect the components to form a single vapor phase. Thus, we cannot assume that both a liquid phase and a vapor phase will always be formed simultaneously.

If the minimum of the Gibbs free energy is given by taking either n_l or $n_v = 0$, then the fluid forms a single phase. In such a case, the mole fractions of the missing

phase are of no importance for the results of this paper. The mole fractions of the existing phase are identical to those of the total fluid, and are only used for the computation of the molar density. The phase potentials in either phase will also be of no interest in this work.

Let us assume that the pressure, temperature, and fluid composition n are such that only a liquid phase is formed. Then the solution to the phase behavior problem is

$$N = [n, 0].$$

We compute the phase volumes as in (2.7). Here the vector of liquid phase mole fractions x_l is determined as in (2.2), and x_v is not needed. This formula also shows that the vapor phase volume is zero, independent of the value of the vapor phase molar density ξ_v. In this case, the fluid equation of state (1.2) says that

$$1 = u_l \equiv \frac{e^T n}{\xi_l((n/e^T n), p)}.$$

This equation makes it very clear that the pressure is related to the fluid composition by the fluid equation of state.

All of the results in this section can be restated in a similar fashion for the case when the fluid forms a single vapor phase.

3. Darcy's law. In the previous section we examined the compositional model for thermodynamic equilibrium. We saw how the pressure and fluid composition determine the phase compositions and the phase volumes. At this point, we are in a position to determine how fast the fluid flows in the phases through the reservoir.

A variety of simplifying assumptions are required in justifying the use of Darcy's law for multiphase flow with mass transfer (see [1], [6], [8], [24]). In one spatial dimension with gravitational effects, the vector of Darcy phase velocities can be written componentwise, as in (1.3), or in matrix-vector form as follows:

$$\nu = -L\left(e\frac{\partial p}{\partial x} - \rho\frac{\partial d}{\partial x}\right)\kappa. \tag{3.1}$$

Here,

$$L \equiv \operatorname{diag}\left(\frac{\kappa_l}{\mu_l}, \frac{\kappa_v}{\mu_v}\right)$$

is the diagonal matrix of phase mobilities (i.e., phase relative permeability divided by phase viscosity),

$$\rho \equiv \begin{bmatrix} \rho_l \\ \rho_v \end{bmatrix}$$

is the vector of phase densities, and κ is the total rock permeability. (We have used L, instead of Λ, to denote the mobilities in order to avoid a conflict in notation during the characteristic analysis in § 6 below.) Also, d is the depth of the reservoir times a gravitational constant. Both $\partial d/\partial x$ and κ are allowed to be functions of the spatial coordinate x. We have ignored capillary pressure so that the pressure in each phase is p.

In order to complete the description of the flow equations, we need to specify the phase mobilities L and the phase densities ρ. We have two goals in this discussion. The primary goal, of course, is to specify the functional dependence of the phase velocities ν, so that we can perform a characteristic analysis in § 6. The other goal is

to completely specify the model for the reader. Nevertheless, we shall be brief in our discussion of these points so that we can quickly return to our analysis of the mathematical structure of the flow equations.

3.1. Saturations. Saturations are the fractions of the fluid volume occupied by each of the phases. We define the vector of phase saturations by

$$s \equiv u \frac{1}{e^T u}. \tag{3.2}$$

With this definition, the saturations are all nonnegative, and sum to one even if the fluid equation of state (1.2) is not satisfied exactly. Thus, the saturations s must be considered to be distinct from the vector u of ratios of phase volumes to pore volume. This distinction plays an important role in the volume-discrepancy sequential method described in the next section because we will not require the equation of state to be satisfied exactly at each step of the algorithm.

3.2. Interfacial tension. The interface between liquid and vapor has properties intermediate between those of liquid and vapor. The unequal forces between the molecules on either side of this interface have a direct effect on the flow of the phases. This effect is modeled through the interfacial tension σ. A variety of empirical correlations are available for modeling σ [13], [25], [27]. In the remainder of this paper, we will not be concerned with the specific form of the interfacial tension. Instead, we merely assume that it is a function of the mole fractions and molar densities in the two phases. The specific mathematical formula for σ used in the numerical examples for this paper is given in Appendix A.

The inclusion of interfacial tension in the model is crucial to the modeling of multiple-contact miscible floods. As the fluid composition approaches the critical point, the interfacial tension between the phases becomes very small. This in turn has an important effect on the relative permeabilities of the phases.

3.3. Relative permeability. Because of the differences in the ways the phases occupy the pore volume, the phases adversely affect the flow of each other in a complicated fashion. The dependence of the relative permeabilities on rock and fluid properties is very complicated (see, for example, [2], [3], [9]). For our purposes, we consider the relative permeabilities to be nonnegative functions of the saturations s and the interfacial tension σ.

One important requirement of the saturation dependence is that the relative permeability of a phase must tend to zero as its saturation tends to zero. (It is often the case that the phase will, in fact, have zero relative permeability for some range of positive saturations.) The interfacial tension has an important effect in enhanced oil recovery. If miscible gas is injected to reduce the interfacial tension, then the phases tend not to affect the flow of each other, and the relative permeabilities tend to be linear functions of their individual phase saturations. The formulas for the relative permeabilities as functions of saturation and interfacial tension are given in Appendix A.

3.4. Viscosity. Viscosity represents the resistance of a phase to flow under the influence of a pressure gradient. Viscosities are always positive; furthermore, the viscosities of liquid and vapor should become identical when their compositions become identical near the critical point. Empirical correlations [12], [19] typically consider the viscosity of a phase to be a function of the mole fractions and molar density for the phase. For the numerical examples in this paper, we have chosen to use the Lohrenz correlation [19] for the viscosities.

3.5. Mass density. In order to determine the effects of gravity on the flow of the phases through the reservoir, we need to compute the phase mass densities. By definition, liquid is chosen to have the greater mass density of either of the two phases found to exist as a result of solving the phase behavior problem (i.e., minimizing Gibbs free energy).

The definition of the mass densities requires the molecular weights w. This vector consists of the mass per mole of each of the components. Then the liquid mass density is

$$\rho_l = \frac{w^T n_l}{u_l},$$

and the vapor phase mass density has a similar definition. Note that when the phase behavior problem has a single-phase solution, the mass density of the missing phase can be given an arbitrary value. This is because the relative permeability functions and Darcy's law (3.1) are such that the density of the missing phase will be multiplied by zero in forming the phase velocity.

It is a simple matter to show that the matrix $X\Xi$ of mole fractions times molar densities consists of the array of moles of each component in each phase divided by the volume of the phase. The matrix of mole fractions in each phase is obtained from (2.2):

$$X \equiv [x_l, x_v] = N \begin{bmatrix} e^T n_l & 0 \\ 0 & e^T n_v \end{bmatrix}^{-1},$$

and the diagonal matrix of phase volumes divided by pore volumes is obtained from (2.7):

$$D_u \equiv \begin{bmatrix} u_l & 0 \\ 0 & u_v \end{bmatrix} = \begin{bmatrix} e^T n_l & 0 \\ 0 & e^T n_v \end{bmatrix} \Xi^{-1}.$$

Then if both phases are formed,

$$ND_u^{-1} = X\Xi. \tag{3.3}$$

We will use this result a number of times in the remainder of the paper. Our first use of this result is to write the vector of phase mass densities in the form

$$\rho^T = w^T X \Xi.$$

3.6. Phase velocity functional dependence. Now that we have examined all of the terms in Darcy's formula for the phase velocities, we shall summarize the variables on which they depend. The phase velocities depend on the mobilities L, the pressure gradient $\partial p/\partial x$, the mass densities ρ, and various functions of the spatial coordinate x. The mass densities are functions of the mole fractions and the molar densities, while the viscosities are functions of the pressure, mole fractions, and molar densities. The relative permeabilities contain the only dependence on the sizes of the phases through their dependence on the saturations. In addition, the relative permeabilities are functions of the interfacial tension, which is a function of the mole fractions and the molar densities. The molar densities are functions of the pressure and the mole fractions. We can condense all these relationships and consider the phase velocities to be functions of the saturations s, mole fractions x_l and x_v, the pressure p, and the pressure gradient $\partial p/\partial x$.

4. Sequential formulation of the compositional flow equations. In §§ 2 and 3 we have discussed the basic equations governing the compositional thermodynamic model and compositional fluid flow. Using these equations, we can now derive the formula for the flux appearing in the molar conservation equation (1.4). Note that if liquid is formed, then the vector of moles of each component flowing in the liquid phase through a given flow area per unit time is $n_l(\nu_l/u_l) = x_l\xi_l\nu_l$. Thus by using (3.3) we can write the total component flux vector in the following form:

$$h \equiv n_l \frac{\nu_l}{u_l} + n_v \frac{\nu_v}{u_v} = X\Xi\nu. \tag{4.1}$$

Here, the sum in the middle is understood to ignore terms involving zero phase composition over zero phase volume; these terms are correctly handled in the right-hand expression. The minimization of Gibbs free energy and the phase equation of state provide us with the necessary information to evaluate the quantities required for computing the mole fractions X, the molar densities Ξ, and the phase velocities ν.

As in the previous paper [32] on the black-oil flow equations, we introduce a sequential splitting of the equations into a pressure equation and a system of component conservation equations. In the subsequent sections we show that the pressure equation is parabolic and that the component conservation equations are hyperbolic.

4.1. Pressure equation. Our first step in the sequential approach is to develop a differential equation for pressure. The purpose of the pressure equation is to represent the total volume balance. More specifically, given the pressure p and fluid composition n at the current time level, we want to predict the value of the pressure at the advanced time level $t+\Delta t$ so that the equation of state (1.2) will be satisfied. If we linearize (1.2) about time level t we obtain

$$\frac{1-e^T u}{\Delta t} = e^T \frac{\partial u}{\partial p}\frac{\partial p}{\partial t} + e^T \frac{\partial u}{\partial n}\frac{\partial n}{\partial t}.$$

Note that we have not assumed that the equation of state (1.2) is satisfied at the current time t; in fact, we want to choose the pressure at the advanced time $t+\Delta t$ to correct any existing volume discrepancy errors. Next, we multiply by the porosity ϕ, and use the mass conservation equation to replace the time derivative of n:

$$\frac{e^T u - 1}{\Delta t}\phi = \left[-\phi e^T \frac{\partial u}{\partial p} + e^T \frac{\partial u}{\partial n} n \frac{\partial \phi}{\partial p}\right]\frac{\partial p}{\partial t} + e^T \frac{\partial u}{\partial n}\frac{\partial}{\partial x}(X\Xi\nu). \tag{4.2}$$

Equation (4.2) establishes the basic form of the pressure equation for the sequential method. In § 5 we shall show that it is actually a parabolic equation for pressure.

4.2. Component conservation equations. Next we want to derive the system of component conservation equations used in the second step of the sequential method. The basic idea here is to define a total velocity describing total net flow of fluid and use it to eliminate the pressure gradient term appearing in Darcy's law. The motivation for this step is that for incompressible flow a total velocity splitting correctly decouples the elliptic and hyperbolic behavior inherent in the equations.

The total fluid velocity is defined simply as the sum of the phase velocities; that is,

$$\nu_T \equiv e^T\nu = -e^T L e \frac{\partial p}{\partial x}\kappa + e^T L\rho \frac{\partial d}{\partial x}\kappa.$$

We use this expression for ν_T to solve for the pressure gradient and substitute into expression (3.1). This gives us the following equation for the sequential formulation of the phase velocities:

$$\nu = Le\frac{1}{e^T Le}\nu_T + \left[I - Le\frac{1}{e^T Le}e^T\right]L\rho\frac{\partial d}{\partial x}\kappa. \tag{4.3}$$

The development of the sequential component flux vector is more subtle. If the equation of state were satisfied exactly, it would make no difference if we were to write the flux as $ND_u^{-1}\nu$ or $ND_s^{-1}\nu$ because the saturations s would be equal to the ratios of phase volume to pore volume u. Since the equation of state is not satisfied exactly in the sequential formulation of the flow equations, these two ways of writing the component flux lead to slight variations in the conservation law. Both choices were discussed for the black-oil model in [32], and both were found to lead to hyperbolic conservation laws. Here we arbitrarily choose the sequential component flux vector in the form (4.1), where ν is given in the form (4.3) instead of (3.1). As a result, the sequential component conservation law takes the form

$$\frac{\partial(n\phi)}{\partial t} + \frac{\partial}{\partial x}(X\Xi\nu) = 0. \tag{4.4}$$

In § 6 we present a characteristic analysis of the molar conservation law (4.4) to show that it is hyperbolic, provided the pressure and total fluid velocity are considered to be independent of n for this step of the sequential method.

5. Analysis of the pressure equation. In the previous section we developed the pressure equation (4.2) from a volume balance. The analysis of the pressure equation depends on two properties of the fluid model. First, there are certain homogeneity relationships of the equilibrium phase compositions that allow us to simplify the form of (4.2). Second, a condition describing the overall compressibility of the total rock-fluid system will guarantee that the pressure equation is parabolic.

5.1. Partial derivatives of the phase moles. The pressure equation involves pressure and molar derivatives of the phase volumes. The phase volumes in turn involve the phase moles and phase mole fractions. Thus our first step in differentiating the phase volumes with respect to pressure and component moles is to compute the derivatives of the phase moles.

Recall from § 2.3.3 that we can write any vector of phase moles satisfying the mass-balance constraints (2.1) in the form (2.5). For the purposes of this discussion, we take

$$\bar{n}_0 = \begin{bmatrix} 0 \\ n \end{bmatrix} \quad \text{and} \quad z = n_l.$$

Also recall that the Kuhn–Tucker conditions (2.6) show that if $\bar{n}$ minimizes the Gibbs free energy, then $B^T\bar{f} = 0$. If we differentiate this Kuhn–Tucker condition with respect to the vector n of the component moles in the total fluid, we obtain

$$0 = B^T\frac{\partial\bar{f}}{\partial\bar{n}}\left(\frac{\partial\bar{n}}{\partial\bar{n}_0}\frac{\partial\bar{n}_0}{\partial n} + \frac{\partial\bar{n}}{\partial z}\frac{\partial z}{\partial n}\right) = -\frac{\partial f_v}{\partial n_v} + G\frac{\partial n_l}{\partial n}.$$

Thus $\partial n_l/\partial n = M$, where M is given by solving the following symmetric, positive-definite system of linear equations:

$$GM \equiv \frac{\partial f_v}{\partial n_v}. \tag{5.1}$$

Note that this equation, together with the Gibbs-Duhem equation (2.4), shows that

$$Mn_v = 0, \qquad (I - M)n_l = 0. \tag{5.2}$$

Now that we know how to evaluate $M = \partial n_l / \partial n$, we can use this result to substitute into the expression for the molar derivatives of the phase moles:

$$\frac{\partial \bar{n}}{\partial n} = \frac{\partial \bar{n}_0}{\partial n} + BM = \begin{bmatrix} M \\ I - M \end{bmatrix}. \tag{5.3}$$

The pressure derivatives of $\bar{n}$ are computed in a similar fashion. From the Kuhn-Tucker conditions (2.6) we have

$$0 = B^T \left(\frac{\partial \bar{f}}{\partial p} + \frac{\partial \bar{f}}{\partial \bar{n}} \frac{\partial \bar{n}}{\partial p} \right) = \frac{\partial f_l}{\partial p} - \frac{\partial f_v}{\partial p} + G \frac{\partial n_l}{\partial p}.$$

First, we solve another linear system with the same positive-definite matrix,

$$Gm = \frac{\partial f_v}{\partial p} - \frac{\partial f_l}{\partial p},$$

for $m \equiv \partial n_l / \partial p$. Then we have the following formula for the pressure derivatives of the phase moles:

$$\frac{\partial \bar{n}}{\partial p} = Bm = \begin{bmatrix} m \\ -m \end{bmatrix}.$$

5.2. Partial phase volumes. In order to form the pressure equation (4.2), we need the partial derivatives of the phase volumes with respect to the component vector n. In order to avoid confusion during partial differentiation, we shall define

$$[c_l, c_v] \equiv C = N \Xi^{-1}.$$

Then $u = C^T e$ so the partial molar derivatives of the phase volumes involve the molar derivatives of the columns of C. Note that

$$\frac{\partial c_l}{\partial n_l} = \left[I - c_l \frac{\partial \xi_l}{\partial x_l} \frac{\partial x_l}{\partial n_l} \right] \frac{1}{\xi_l}. \tag{5.4}$$

Since n_l is in the null space of $\partial x_l / \partial n_l$ (see (2.3), for example), (5.4) implies that

$$\frac{\partial c_l}{\partial n_l} n_l = c_l. \tag{5.5}$$

Similar results hold for vapor.

It follows that

$$\frac{\partial u_l}{\partial n} = e^T \frac{\partial c_l}{\partial n_l} M. \tag{5.6}$$

Equations (5.2), (5.5), and (5.6) show that

$$\frac{\partial u_l}{\partial n} N = [u_l, 0].$$

A similar result holds for vapor. We can combine the results for the two phases into the following equation:

$$\frac{\partial u}{\partial n} N = D_u \equiv \operatorname{diag}(u_l, u_v). \tag{5.7}$$

This equation essentially says that each phase volume is a homogeneous function of the first degree in its own phase composition. It holds no matter how many phases are actually formed. The most important application of this result is to prove that

$$\frac{\partial u}{\partial n} X \Xi \nu = \nu \tag{5.8}$$

in both the one-phase and two-phase cases.

5.3. Total fluid compressibility. In order to guarantee that the pressure equation is parabolic, we also need to guarantee that the coefficient of the time derivative of pressure in (4.2) is positive. Since the porosity ϕ is an increasing function of pressure, we have that $\partial\phi/\partial p \geqq 0$. All that remains is to examine the pressure derivatives of u.

In the two-phase case, we have that

$$\frac{\partial u_l}{\partial p} = e^T \frac{\partial c_l}{\partial n_l} \frac{\partial n_l}{\partial p} - \frac{u_l}{\xi_l} \frac{\partial \xi_l}{\partial p}.$$

A similar result holds for vapor. Thus the pressure derivative of the total fluid volume is

$$e^T \frac{\partial u}{\partial p} = e^T \left(\frac{\partial c_l}{\partial n_l} - \frac{\partial c_v}{\partial n_v} \right) m - \left(\frac{u_l}{\xi_l} \frac{\partial \xi_l}{\partial p} + \frac{u_v}{\xi_v} \frac{\partial \xi_v}{\partial p} \right).$$

We want this quantity to be negative:

$$e^T \frac{\partial u}{\partial p} < 0. \tag{5.9}$$

This condition guarantees the mechanical stability of the fluid and is a firm consequence of standard thermodynamic analysis of real fluids. We do not, however, know how to prove (5.9) for the Peng–Robinson model applied to a multiphase mixture. In the case when only the vapor phase is formed, we have that

$$\frac{\partial u_l}{\partial p} = -\frac{u_l}{\xi_l} \frac{\partial \xi_l}{\partial p}.$$

For the Peng–Robinson model, (5.9) can be ensured by preventing the choice of the middle root of the cubic.

5.4. Parabolicity of the pressure equation. We can use the calculations in §§ 5.1 and 5.2 to simplify and analyze the pressure equation (4.2). Using the first-degree homogeneity of the phase volumes (5.7), we obtain

$$\begin{aligned} \frac{e^T u - 1}{\Delta t} \phi &= \left[e^T \frac{\partial u}{\partial n} n \frac{\partial \phi}{\partial p} - \phi e^T \frac{\partial u}{\partial p} \right] \frac{\partial p}{\partial t} + \left[\frac{\partial}{\partial x} \left(e^T \frac{\partial u}{\partial n} X \Xi \nu \right) - \frac{\partial}{\partial x} \left(e^T \frac{\partial u}{\partial n} \right) X \Xi \nu \right] \\ &= \left[e^T u \frac{\partial \phi}{\partial p} - \phi e^T \frac{\partial u}{\partial p} \right] \frac{\partial p}{\partial t} + \frac{\partial e^T \nu}{\partial x} - \frac{\partial}{\partial x} \left(e^T \frac{\partial u}{\partial n} \right) X \Xi \nu. \end{aligned}$$

Next, we use Darcy's law (3.1) to write the pressure equation in the form

$$\begin{aligned} &\frac{e^T u - 1}{\Delta t} \phi - \frac{\partial}{\partial x} \left(e^T L \rho \frac{\partial d}{\partial x} \kappa \right) + \frac{\partial}{\partial x} \left(e^T \frac{\partial u}{\partial n} \right) X \Xi L \rho \frac{\partial d}{\partial x} \kappa \\ &\quad = \left[e^T u \frac{\partial \phi}{\partial p} - \phi e^T \frac{\partial u}{\partial p} \right] \frac{\partial p}{\partial t} - \frac{\partial}{\partial x} \left(e^T L e \frac{\partial p}{\partial x} \right) + \frac{\partial}{\partial x} \left(e^T \frac{\partial u}{\partial n} \right) X \Xi L e \frac{\partial p}{\partial x}. \end{aligned} \tag{5.10}$$

Note that the coefficient of $\partial p/\partial t$ in (5.10) is the sum of two positive terms since $\partial\phi/\partial p \geqq 0$, and (5.9) requires that $e^T \partial u/\partial p < 0$. Typically, the rock and fluid compressibility, which combine to form the coefficient of $\partial p/\partial t$, are very small. Also note that the total mobility $e^T L e$, which occurs in the term involving the second-order spatial derivative of pressure, is positive. Thus the pressure equation (5.10) is parabolic. Because of the small compressibilities, we expect the pressure to reach a steady state very rapidly.

Note that we could easily have considered multidimensional fluid flow and used the same arguments to show that the pressure equation remains parabolic. We restricted our analysis to one dimension for simplicity only.

6. Characteristic analysis. In this section, we analyze the component conservation equations to show that they are hyperbolic. This analysis is similar in spirit to that in [32] in that we use the structure of the various equations of state and the functions in Darcy's law to prove the hyperbolicity of the conservation law. This analysis is in contrast to that in [11], [14] where the authors merely assume the hyperbolicity of the conservation law and proceed to analyze its structure using the theory of multicomponent chromatography. The analysis in this section bears some similarity to that in [18], but that author is interested in "quasistatic" composition changes that maintain equilibrium between phases and is not concerned with flow in porous media.

6.1. One-phase fluids. If we write the sequential formulation of the conservation law in quasilinear form, we have

$$\frac{\partial n\phi}{\partial t} + \frac{\partial h}{\partial n}\frac{\partial n}{\partial x} = 0.$$

Here, we are considering pressure to be independent of n, so ϕ is independent of n. This equation is hyperbolic if and only if $\partial h/\partial n$ has real eigenvalues for all admissible values of n.

As we shall now see, this condition is straightforward to verify when the fluid forms a single phase. Equation (4.4) shows that the component flux vector for a fluid forming only a liquid phase is

$$h = X\Xi\nu = n\frac{\nu_T}{u_l}.$$

Since the total fluid velocity is considered to be constant in this step of the sequential method, we can differentiate this equation to obtain

$$\frac{\partial h}{\partial n} = \left(I - n\frac{1}{u_l}\frac{\partial u_l}{\partial n}\right)\frac{\nu_T}{u_l}.$$

Equation (5.7) shows that

$$\frac{\partial h}{\partial n}n = 0 \quad \text{and} \quad \frac{\partial u_l}{\partial n}\frac{\partial h}{\partial n} = 0.$$

Thus u_l is a Riemann invariant with zero characteristic speed, and the rarefaction curves for this wave are aligned with the direction of the total fluid composition n.

The remaining characteristic speeds are all equal to ν_T/u_l, and the rarefaction curves all lie in directions orthogonal to $\partial u_l/\partial n$. Thus, all of the wave speeds are essentially linearly degenerate (since the fluid equation of state (1.2) shows that u_l is expected to be very close to one), and Riemann problem solutions for one-phase flow can possess only constant states separated by contact discontinuities.

6.2. Two-phase fluids. The characteristic analysis of two-phase flow is more complex. Recall that the two-phase flux is given by (4.4) in terms of the mole fractions X, molar densities Ξ, and phase velocities ν. Also recall that the phase velocities ν depend on the saturations s, the mole fractions X, and the molar densities Ξ. In turn, the saturations depend on the phase compositions N and on the mole fractions X through the molar densities Ξ. Of course, the mole fractions X are straightforward functions of the phase compositions N. Thus, our first step is to analyze the structure of the molar derivatives of N. Afterward, we will be able to examine the form of the molar derivatives of ν. Finally, this will allow us to study the molar derivatives of the component flux vector h.

6.2.1. Transformation to a symmetric eigenvalue problem. Recall that the Gibbs-Duhem equation (2.4) shows that both $\partial f_l/\partial n_l$ and $\partial f_v/\partial n_v$ are singular. Next recall that the Hessian matrix G of the Gibbs free energy, defined by (2.6), is symmetric and positive definite. Finally, recall that the molar derivatives of the liquid phase composition vector n_l are given by the matrix M, which solves (5.1). We used these facts to find two eigenvectors and associated eigenvalues of M in (5.2).

We can use this information to deflate these eigenvalues out of the eigenproblem for M. The first step in this deflation process is identical to the transformation used in [31] to handle the near-singularity of the Hessian matrix G near the phase envelope. Let H_1 be a symmetric orthogonal matrix so that

$$H_1 e_1 = n_l \frac{1}{\|n_l\|_2}.$$

That is, the first column of H_1 is a unit vector in the direction of n_l; in particular, H_1 could be chosen to be a Householder reflector [34]. Since G is positive-definite, $H_1 G H_1$ is also positive-definite and has a Cholesky factorization [10]:

$$H_1 G H_1 = U^T U,$$

where U is upper-triangular. We use the Cholesky factorization and (5.1) to perform a similarity transformation on M:

$$U H_1 M H_1 U^{-1} = U^{-T} H_1 \frac{\partial f_v}{\partial n_v} H_1 U^{-1} \equiv K.$$

This equation shows that M is similar to the symmetric matrix K. From (5.2) we know two eigenvectors of M; thus, we can use the transformations to obtain two eigenvectors of K. The first eigenvector has eigenvalue equal to one, and is given by

$$U H_1 n_l \frac{1}{u_l} = U e_1 \frac{\|n_l\|_2}{u_l} \equiv e_1 \delta_l,$$

where the scalar δ_l is defined to be

$$\delta_l \equiv e_1^T U e_1 \frac{\|n_l\|_2}{u_l}.$$

Another eigenvector of K is

$$U H_1 n_v \frac{1}{u_v},$$

with corresponding eigenvalue zero. Since eigenvectors of symmetric matrices corresponding to distinct eigenvalues must be orthogonal, we have

$$\delta_l e_1^T U H_1 n_v \frac{1}{u_v} = 0.$$

This suggests that we define a second symmetric orthogonal matrix H_2 so that

$$H_2 e_1 = e_1, \qquad H_2 U H_1 n_v \frac{1}{u_v} = e_2 \delta_v,$$

where the scalar δ_v is defined by

$$\delta_v \equiv \pm \frac{\| U H_1 n_v \|_2}{u_v}.$$

Here the sign of δ_v is chosen to avoid rounding errors in forming H_2. We can use the second transformation H_2 to obtain the deflated form of K

$$H_2 K H_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \bar{K} \end{bmatrix}.$$

This similarity transformation shows that one and zero are eigenvalues of K, while the remaining eigenvalues of K are eigenvalues of $\bar{K}$. Since $\bar{K}$ is symmetric, we solve its algebraic eigenvalue problem to find an orthogonal matrix $R_{\bar{K}}$ and a diagonal matrix $\Lambda_{\bar{K}}$ such that

$$\bar{K} R_{\bar{K}} = R_{\bar{K}} \Lambda_{\bar{K}}.$$

Next, we define

$$R_K \equiv H_2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & R_{\bar{K}} \end{bmatrix}, \qquad D \equiv \begin{bmatrix} \delta_l & 0 & 0 \\ 0 & \delta_v & 0 \\ 0 & 0 & I \end{bmatrix},$$

$$\Lambda_M \equiv \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \Lambda_{\bar{K}} \end{bmatrix}, \qquad R_M \equiv H_1 U^{-1} R_K D.$$

Then we have found the eigenvectors and eigenvalues of M, since

$$M R_M = R_M \Lambda_M. \tag{6.1}$$

In particular, note that the matrix of eigenvectors has been defined so that the first two eigenvectors of M are the mole fractions times molar densities:

$$R_M = [N D_u^{-1}, \bar{R}_M] = [X \Xi, \bar{R}_M]. \tag{6.2}$$

6.2.2. Velocity derivatives. Recall that the vector of phase velocities ν depends on the saturations and the mole fractions. Also recall that for the purposes of the sequential method, we assume that the total fluid velocity and the pressure are independent of n. Thus

$$\frac{\partial \nu}{\partial n} = \frac{\partial \nu}{\partial x_l} \frac{\partial x_l}{\partial n_l} \frac{\partial n_l}{\partial n} + \frac{\partial \nu}{\partial x_v} \frac{\partial x_v}{\partial n_v} \frac{\partial n_v}{\partial n} + \frac{\partial \nu}{\partial s} \frac{\partial s}{\partial u} \frac{\partial u}{\partial n}. \tag{6.3}$$

Here $\partial\nu/\partial x_l$ involves derivatives of ν with respect to viscosity, mass density, and interfacial tension. Of course, $e^T\,\partial\nu/\partial n = 0$ since the total fluid velocity is constant. Finally, we note that (5.2) and (5.7) imply that

$$\frac{\partial\nu}{\partial n}\,ND_u^{-1} = \frac{\partial\nu}{\partial s}\frac{\partial s}{\partial u} = \frac{\partial\nu}{\partial s}[I - s\,e^T]\frac{1}{e^T u}. \tag{6.4}$$

6.2.3. Similarity transformation of the flux derivatives. Using the definition (4.1) of the sequential component flux h, we can differentiate h to compute

$$\frac{\partial h}{\partial n} = M\frac{\nu_l}{u_l} + (I - M)\frac{\nu_v}{u_v} + n_l\frac{1}{u_l^2}\left(u_l\frac{\partial\nu_l}{\partial n} - \nu_l\frac{\partial u_l}{\partial n}\right) + n_v\frac{1}{u_v^2}\left(u_v\frac{\partial\nu_v}{\partial n} - \nu_v\frac{\partial u_v}{\partial n}\right).$$

Next, we use the eigenvectors R_M of M to perform a similarity transformation on $\partial h/\partial n$:

$$R_M^{-1}\frac{\partial h}{\partial n}R_M = \Lambda_M\frac{\nu_l}{u_l} + (I - \Lambda_M)\frac{\nu_v}{u_v} + e_1\frac{1}{u_l}\left(u_l\frac{\partial\nu_l}{\partial n}R_M - \nu_l\frac{\partial u_l}{\partial n}R_M\right)$$
$$+ e_2\frac{1}{u_v}\left(u_v\frac{\partial\nu_v}{\partial n}R_M - \nu_v\frac{\partial u_v}{\partial n}R_M\right).$$

Then the transformed flux derivatives can be written in partitioned form as follows:

$$R_M^{-1}\frac{\partial h}{\partial n}R_M = \begin{bmatrix} V & Q \\ 0 & \bar{\Lambda} \end{bmatrix}, \tag{6.5}$$

where

$$V \equiv \begin{bmatrix} \dfrac{v_l}{u_l} & 0 \\ 0 & \dfrac{\nu_v}{u_v} \end{bmatrix} + e_1\frac{\partial\nu_l}{\partial n}ND_u^{-1} + e_2\frac{\partial\nu_v}{\partial n}ND_u^{-1} - e_1\frac{\nu_l}{u_l}\frac{\partial u_l}{\partial n}ND_u^{-1} - e_2\frac{\nu_v}{u_v}\frac{\partial u_v}{\partial n}ND_u^{-1},$$

$$Q \equiv e_1\frac{\partial\nu_l}{\partial n}\bar{R}_M + e_2\frac{\partial\nu_v}{\partial n}\bar{R}_M - e_1\frac{\nu_l}{u_l}\frac{\partial u_l}{\partial n}\bar{R}_M - e_2\frac{\nu_v}{u_v}\frac{\partial u_v}{\partial n}\bar{R}_M, \tag{6.6}$$

$$\bar{\Lambda} \equiv \Lambda_{\bar{K}}\frac{\nu_l}{u_l} + (I - \Lambda_{\bar{K}})\frac{\nu_v}{u_v}.$$

6.2.4. Deflation of the eigenproblem for the flux derivatives. The block structure of the matrix in (6.5) shows that the eigenvalues of $\partial h/\partial n$ are either an average of particle velocities, given by a diagonal entry of $\bar{\Lambda}$, or an eigenvalue of the 2×2 matrix V. Equations (5.7), (6.4), and (6.6) show that

$$V = \frac{\partial\nu}{\partial s}(I - s\,e^T)\frac{1}{e^T u}. \tag{6.7}$$

This matrix also occurred in the characteristic analysis of the black-oil model [32]; its eigenvectors and eigenvalues are given by

$$VR_V = R_V\Lambda_V,$$

where

$$R_V \equiv \begin{bmatrix} s_l & -1 \\ s_v & 1 \end{bmatrix},$$

$$\Lambda_V \equiv \begin{bmatrix} 0 & 0 \\ 0 & \left(\dfrac{\partial\nu_v}{\partial s_v} - \dfrac{\partial\nu_v}{\partial s_l}\right)\dfrac{1}{e^T u} \end{bmatrix}.$$

6.2.5. Eigenvectors of the flux derivatives. Since R_V is always nonsingular, we can use it to transform the flux derivatives a final time:

$$\begin{bmatrix} R_V^{-1} & 0 \\ 0 & I \end{bmatrix} R_M^{-1} \frac{\partial h}{\partial n} R_M \begin{bmatrix} R_V & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} \Lambda_V & R_V^{-1} Q \\ 0 & \bar{\Lambda} \end{bmatrix}.$$

If the entries of Λ_V are distinct from those of $\bar{\Lambda}$, then we can solve

$$\Lambda_V A - A\bar{\Lambda} = R_V^{-1} Q$$

for A. In this case the eigenvectors and eigenvalues of $\partial h/\partial n$ are given by

$$\frac{\partial h}{\partial n} R = R\Lambda,$$

where

$$R \equiv R_M \begin{bmatrix} R_V & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & -A \\ 0 & I \end{bmatrix},$$

$$\Lambda \equiv \begin{bmatrix} \Lambda_V & 0 \\ 0 & \bar{\Lambda} \end{bmatrix}.$$

This completes the characteristic analysis for two-phase flow. Since the characteristic speeds (i.e., eigenvalues of Λ) are all real, the sequential conservation law is hyperbolic.

It is not hard to decipher the rarefaction directions corresponding to the first two eigenvalues of $\partial h/\partial n$. By computing the first column of R, we see that rarefaction waves in the family of the zero wave speed lie in the direction of the total fluid composition n. On the other hand, the second column of R shows that the rarefaction corresponding to the Buckley–Leverett wave lies in the direction of $x_v \xi_v - x_l \xi_l$. This result is similar to that of Helfferich [14], who claimed that this rarefaction must point along tie lines, the directions of which are given by $x_v - x_l$. Also note that the total fluid volume $e^T u$ is constant along this rarefaction curve since the first-degree homogeneity of the phase volumes (5.7) shows that

$$e^T \frac{\partial u}{\partial n} (x_v \xi_v - x_l \xi_l) = e^T \frac{\partial u}{\partial n} N D_u^{-1} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = e^T I \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 0.$$

In the case of two-component two-phase flow, the total fluid volume is a Riemann invariant corresponding to the zero wave speed. This is essentially because (5.7) and (6.2) show that $\partial u/\partial n = R_M^{-1}$ in this case.

6.3. Continuity of the eigenstructure. Generally speaking, the eigenvalues and eigenvectors of the flux derivatives should be continuously varying as long as the flow produces two phases. This is under the assumption that the relative permeabilities, viscosities, molar densities, and phase potentials are continuously differentiable. (At those points where the Peng–Robinson cubic equation of state changes from having one real root to three, these assumptions may be violated.) Similarly, if the flow continues to have the same phase missing due to undersaturation, the eigenstructure of the conservation law will be continuous. However, when the flow undergoes a phase change, then the eigenstructure exhibits certain discontinuities.

The cause of these discontinuities is essentially due to the fact that we cannot write $\partial u/\partial n \; N D_u^{-1} = I$ for one-phase flow. For two-phase flow, (3.3) and (5.7) allow us to rewrite (6.6) in the form of (6.7); for single-phase flow the terms cancel in a different way so that an eigenvalue of V changes from a Buckley–Leverett mode in two-phase flow to a particle velocity for one-phase flow. Note that the direction of one of the

right eigenvectors of $\partial h/\partial n$ is unchanged in crossing from a two-phase region to a one-phase region. In particular, the total fluid composition n always lies in the direction of the rarefaction corresponding to the zero wave speed.

6.4. Eigenvector deficiency. In the course of computing the eigenvectors of the flux derivatives in § 6.2.5 above, we assumed that the eigenvalues in Λ_V are distinct from those in $\bar{\Lambda}$. This assumption allowed us to compute the matrix A in order to reduce the transformed flux derivatives to diagonal form. On the other hand, if the entries of Λ_V and $\bar{\Lambda}$ are not distinct, then we have the possibility of an eigenvector deficiency.

Note that an eigenvector deficiency is possible whenever an average of particle velocities is equal to either zero or the Buckley–Leverett wave speed. These eigenvector deficiencies can greatly complicate the solution of Riemann problems required in the Godunov method. For the numerical examples in this paper, which are derived from simple slim tube simulations, the Riemann problem solutions can ignore the eigenvector deficiencies because all of the characteristic speeds are positive. However, it is necessary to provide appropriate techniques to treat eigenvector deficiencies in characteristic tracing, and in approximate Riemann problem solutions for problems involving countercurrent flow. For more details on how to handle these problems, the reader can consult [5].

7. Numerical examples. The bulk of this paper has been concerned with the development and analysis of a sequential approximation to the compositional flow equations. The characteristic analysis of § 6 is very useful for qualitative analysis of the flow patterns in the absence of strong pressure effects. However, in order to obtain a quantitative description of the types of fronts that can form and propagate, one is essentially forced to use a numerical approximation. Of course, some information can be obtained by freezing coefficients in the component conservation equations and studying the associated Riemann problems. However, in realistic flow situations the fluid velocity depends on the pressure gradient so that if pressure is constant nothing flows. All of our numerical examples involve very small pressure gradients so that the theory of hyperbolic conservation laws can be expected to be of use in the interpretations.

The objective of this section is to study the behavior of the compositional system and not to advocate any particular numerical method. Our numerical methods for both the pressure equation and the component conservation equations are the same as that in [32]. We refer the interested reader to that paper for further details.

The numerical examples are chosen to illustrate the differences between immiscible and multiple-contact miscible flow. We only consider injection of a vapor phase into a liquid reservoir. We have avoided numerical examples involving miscible flow. This is because purely miscible flow problems involve single phase flow with very little of interest in the wave structure (since all characteristic wave speeds would be identically equal). Furthermore, these problems have been extensively studied in the mathematics literature for very simple thermodynamic models by a number of authors [4], [26] and the references cited therein.

The computations are performed for a linear reservoir 250 feet in length with no dip. The reservoir is homogeneous with permeability 2,000 md and porosity identically equal to one. The problems are all adaptations of the fluid composition in Coats [7] in that we consider mixtures of methane, normal butane, and decane. The reservoir temperature is fixed at 160° Fahrenheit, and the reservoir is initially at a pressure of 2,000 p.s.i.; these are the conditions specified by Coats.

Unlike the example in Coats' paper, our fluids are always injected at 2,002 p.s.i. and produced at 2,000 p.s.i. Since the pressure gradient is so small, the dominant effects in the flows are due to composition gradients. Our relative permeability model is different from that in Coats' paper, and we have ignored the immobile water in that paper as well. Unless otherwise noted, the simulation results are obtained with a discretization of the reservoir into 1,000 grid cells of equal size. The phase equilibrium computations are required to obtain seven digits of accuracy through tests on the differences in the phase potentials and the change in the solution to phase equilibrium. Since the phase equilibrium computations are vectorized, most of these iterations were continued beyond this desired level of accuracy until the worst in the vector of problems reached acceptable accuracy. Since the minimization method used to solve the phase equilibrium computations has a quadratic convergence rate, and the computer has essentially fourteen digits of accuracy, most of the phase equilibrium problems were solved to the limiting accuracy of the machine.

7.1. Immiscible injection into a liquid reservoir. For our first set of numerical results, we have chosen to simulate an immiscible flood into the reservoir fluid used by Coats. His reservoir fluid is 20% methane, 20% normal butane, and 60% decane. At the left-hand side of the reservoir, we inject a very lean gas consisting of 95% methane and 5% ethane. In Fig. 3(a) we show the plots of the component densities as functions of position in the reservoir. Fig. 3(b) illustrates the vapor phase saturation and characteristic wave speeds, also as functions of position in the reservoir. The simulation results show that the injected vapor mixes with the resident liquid to form a two-phase mixture. Note that the leading front only sweeps out half of the decane. The remaining decane is swept out very slowly.

The plot of the eigenvalues of the molar derivatives of the component flux in Fig. 3(b) illustrate many of the features of the characteristic analysis in § 7. Note that in the single-phase regions on the left and right, both of the positive wave speeds are equal to the particle velocities in the respective phases. As the fluid progresses from left to right, there is an initial discontinuity in the eigenvalues due to the phase transition. From this discontinuity, the flow forms a rarefaction in the Buckley–Leverett family (denoted by the solid curve), which connects to a contact discontinuity in the particle velocity family (denoted by the dotted curve). (The third wave speed is identically zero and is not plotted.) Since the Buckley–Leverett wave speed and the particle wave speed cross, the flow problem is not strictly hyperbolic. This type of compound wave is predicted by the theory in [16]. To the right of the contact discontinuity, a constant state separates the initial compound wave from the faster rarefaction-shock in the Buckley–Leverett family. The combination of rarefaction and shock in the faster wave is due to the lack of genuine nonlinearity in the Buckley–Leverett wave family.

7.2. Multiple-contact miscible injection into a liquid reservoir. For our next example, we inject a richer gas into the same liquid reservoir. The new injection mixture is 75% and 25% normal butane, and the simulation results are shown in Figs. 4(a) and 4(b). Note that this injection mixture is different from that suggested by Coats. Coats's injection gas is very near the Plait point in the ternary diagram, and the effects of numerical diffusion on the simulation results are less pronounced with a near-critical injection than with the injection we have chosen.

As in the previous example, the injection of the vapor phase into the reservoir liquid results in the formation of a two-phase mixture in a portion of the reservoir. Unlike the previous example, this injection accomplishes a very efficient sweep of the decane from the reservoir. The characteristic feature of this multiple-contact miscible

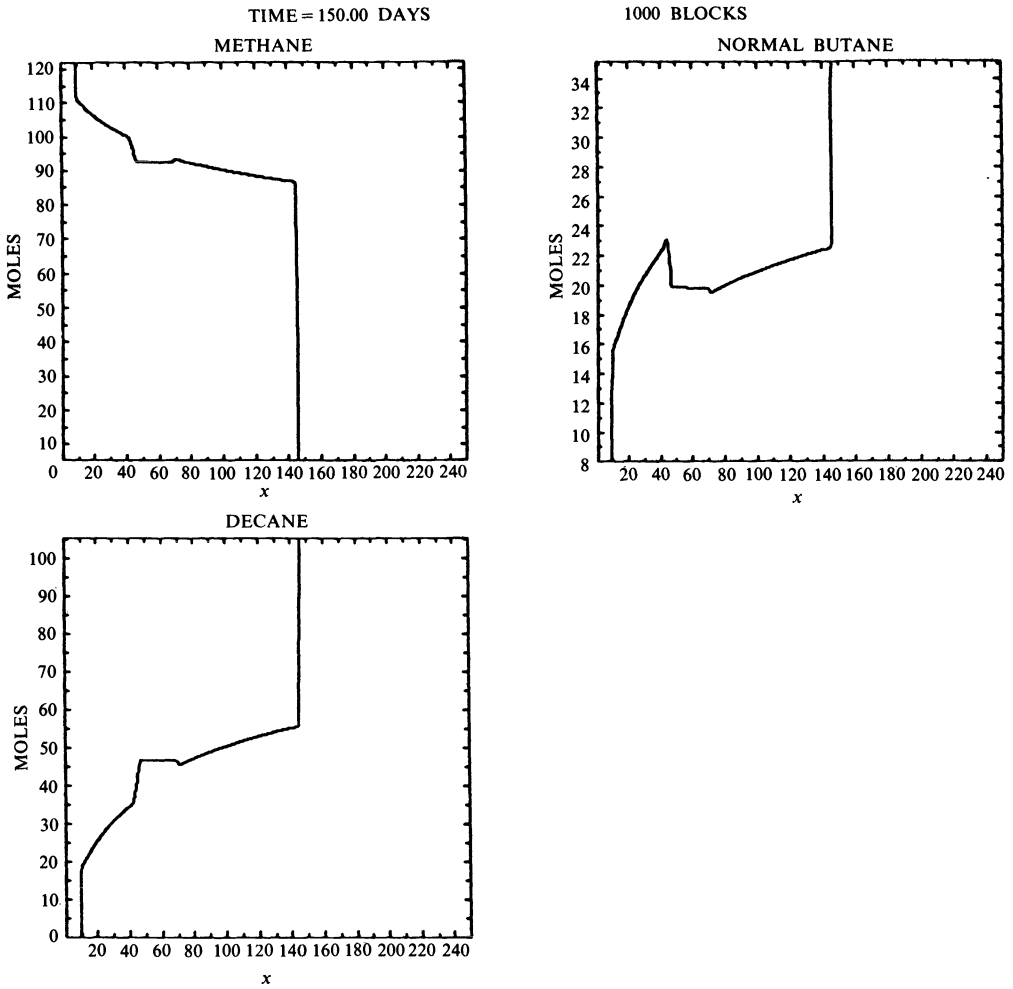


FIG. 3(a). *Component densities for immiscible flow example.*

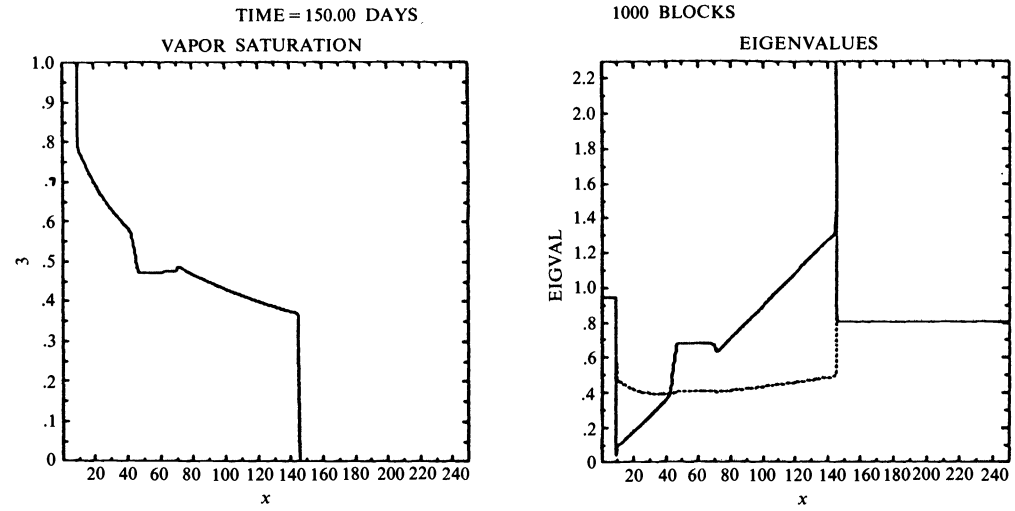


FIG. 3(b). *Saturations and wave speeds for immiscible flow example.*

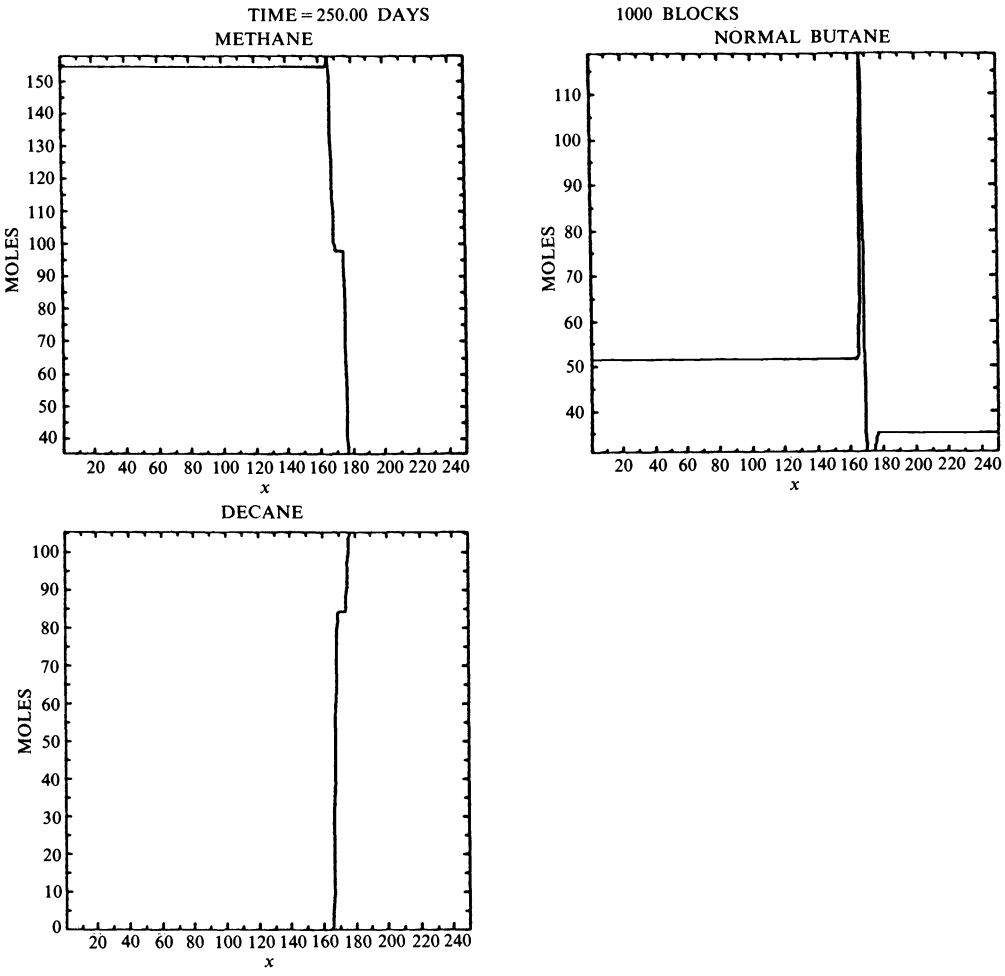


FIG. 4(a). *Component densities for multiple-contact miscible flow example.*

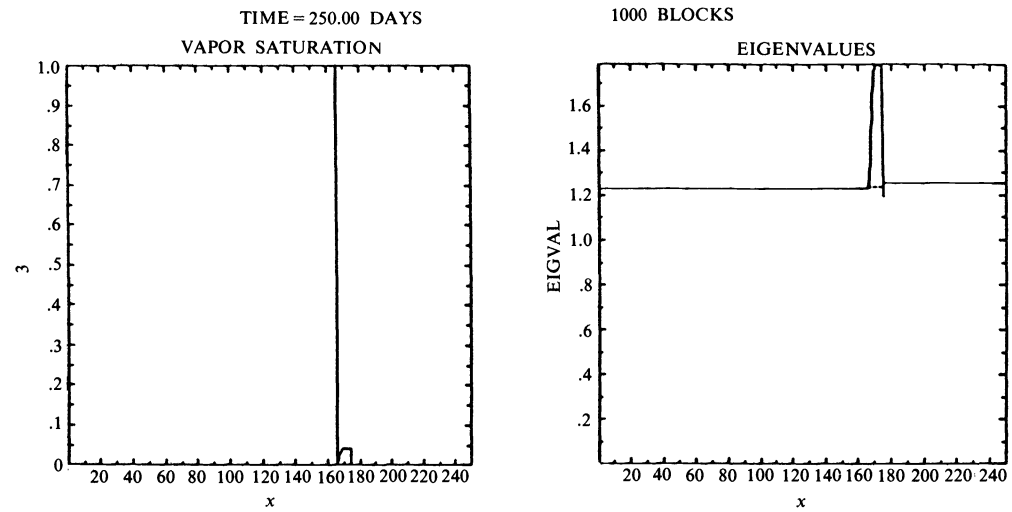


FIG. 4(b). *Saturations and wave speeds for multiple-contact miscible flow example.*

injection is the very steep butane bank formed in the two-phase region. If the phase compositions were plotted on a ternary diagram for each point in the butane bank, the resulting curve would trace a path essentially on top of the bubble-point and dew-point curves just inside the two-phase region. The multiple-contact miscible injection achieves its efficient recovery of decane because of the close approach of the flow path to the Plait point, where the liquid and vapor phases are identical and do not inhibit the flow of each other.

We are not certain of the correct interpretation of the eigenvalue plots in Fig. 4(b) for this multiple-contact miscible injection. In the two-phase region of Fig. 4(b), the two wave speeds are much closer in value than in Fig. 3(b) for the previous immiscible problem. Since the wave speeds are so nearly equal, it is difficult to attribute effects to phase transitions, contact discontinuities, and shocks in the two families of waves.

7.3. Grid refinement. In Figs. 5 and 6 we have shown the results of several different simulations of the previous immiscible and multiple-contact miscible injections for grids of varying size. We have plotted the component density of normal butane for each of these simulations, since this component has the most interesting behavior in

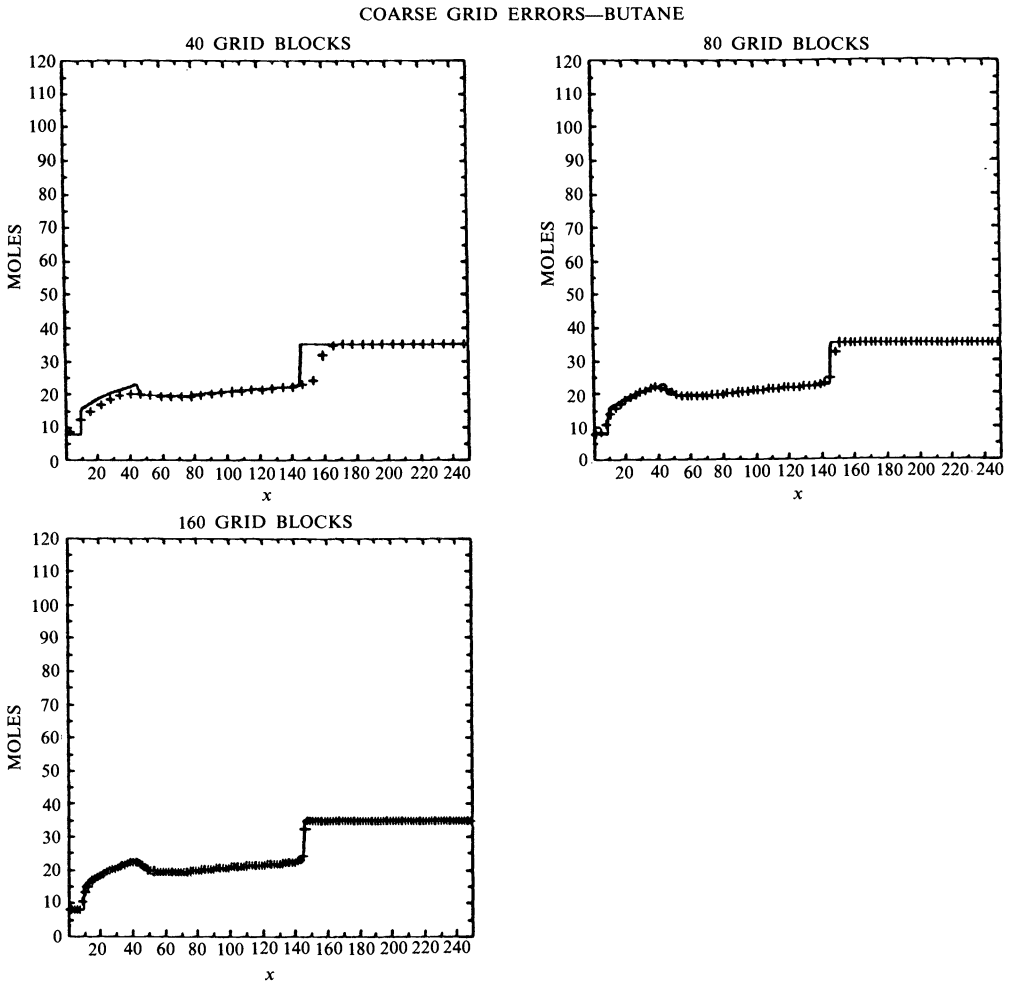


FIG. 5. *Grid refinement study for immiscible flow example.*

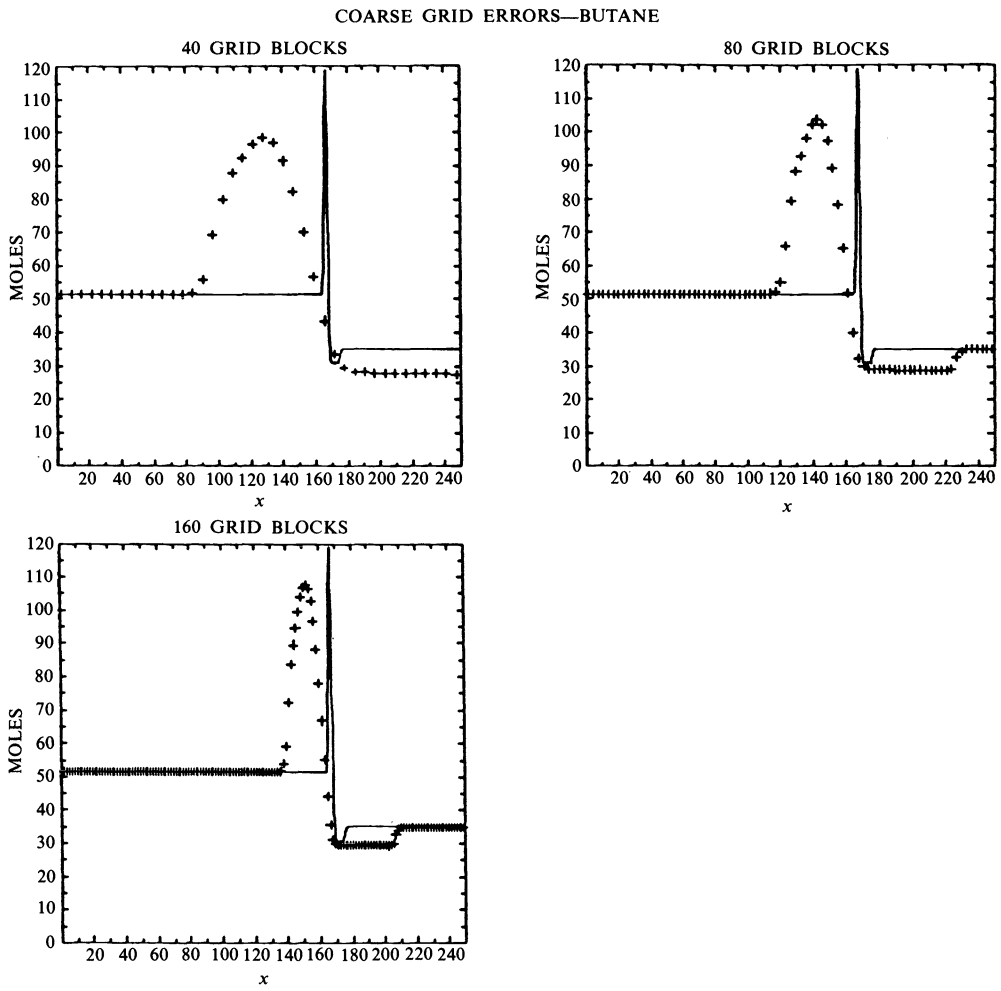


FIG. 6. *Grid refinement study for multiple-contact miscible flow example.*

both simulations. Fig. 5 shows the grid refinement study for the immiscible flood shown in Fig. 3, and Fig. 6 shows the results of the grid refinement for the multiple-contact miscible flood in Fig. 4. Note that a fairly coarse grid of only 80 grid blocks achieves very respectable accuracy for the immiscible flow. On the other hand, 160 grid blocks are not sufficient to properly place the location or height of the butane bank in the multiple-contact miscible flood. Clearly, it is much more difficult to simulate multiple-contact miscible displacement than immiscible displacement. A contributing factor to this difficulty is the fact that the contact discontinuities are not inherently self-steepening, and any numerical dissipation added by the numerical scheme contributes to the spreading of the discontinuity.

8. Concluding remarks. In this paper we have presented and analyzed a volume discrepancy splitting of the equations governing compositional reservoir simulation. This splitting is, essentially, an extension of a splitting previously developed in [32] for black-oil simulation to the compositional model. It decouples the system into a parabolic pressure equation and a system of hyperbolic conservation laws.

Using the splitting as a basis for a sequential numerical method, we have studied the types of fronts that can propagate through the system. As in the black-oil case, the hyperbolic structure of the component conservation equations provides the key to understanding the propagation of fronts through the reservoir. Thus, in addition to providing a basis for developing sequential numerical algorithms, the analysis in this paper also provides a qualitative analysis of the solution structure.

Appendix A. Functional forms. In this section we describe several of the functional forms of the compositional flow equations. In particular, we describe the Sugden-Macleod correlation for the interfacial tension, the relative permeability used in the computations, and the effect of the interfacial tension on the relative permeabilities. The Peng-Robinson model is described in detail in [23], and the viscosity model is described in [19]. We have omitted the description of these two models because they are somewhat complex.

The relative permeability functions were taken to be the following:

$$\kappa_{rl}(s_l) = s_l^2, \qquad \kappa_{rv}(s_v) = s_v^2.$$

The interfacial tension was computed by

$$\sigma = [q^T(x_l\xi_l - x_v\xi_v)]^4$$

where the vector q consists of the component parachor numbers. Then the interfacial tension was used to modify the relative permeabilities as follows:

$$\alpha = \left(\frac{\sigma}{\sigma_0}\right)^{0.1},$$

$$\kappa_{rl}(s_l, \sigma) = \alpha\kappa_{rl}(s_l) + (1-\alpha)s_l,$$

$$\kappa_{rv}(s_v, \sigma) = \alpha\kappa_{rv}(s_v) + (1-\alpha)s_v.$$

Appendix B. Equation of state parameters. There are several parameters needed to specify the various models used in the compositional flow equations. For example, the Peng-Robinson model requires several parameters for its successful prediction of fluid behavior. These parameters include the acentric factor, critical pressure, and critical temperature for each component. In addition, the model requires interaction parameters to model mixtures of components. The Lohrenz viscosity model requires the critical molar volumes in addition to these parameters. The computation of mass density uses the molecular weights, and the Sugden-Macleod interfacial tension model needs the parachor numbers.

The numbers used in Coats' model are provided in Tables 1 and 2. Note that the units of the parachor numbers are of no interest because of the way in which the

TABLE 1
Fluid characterization parameters.

	Methane	Normal butane	Decane	(Units)
Acentric factor	1.0104	0.2010	0.4885	(dimensionless)
Parachor	77.0	189.0	431.3	
Molecular weight	16.043	58.124	142.286	(gm-mass/gm-mole)
Critical pressure	667.757	550.659	303.987	(p.s.i.)
Critical temperature	190.589	425.189	617.661	(degrees Kelvin)
Critical molar volume	0.0035051	0.0089955	0.021299	(cubic ft/gm-mole)

TABLE 2
Interaction parameters.

	Methane	Normal butane	Decane
Methane	0.0	0.024	0.025
Normal butane	0.024	0.0	0.0
Decane	0.025	0.0	0.0

interfacial tension is used to modify the relative permeabilities. We have chosen the initial interfacial tension σ_0 in the relative permeability model (see Appendix A) so that the initial value of α in the reservoir is slightly less than one.

REFERENCES

[1] M. ALLEN, *Collocation Techniques for Modelling Compositional Flows in Oil Reservoirs*, Lecture Notes in Engineering, 6, Springer-Verlag, New York, 1984.

[2] J. O. AMAEFULE AND L. L. HANDY, *The effect of interfacial tensions on relative oil/water permeabilities of consolidated porous media*, Soc. Petroleum Engrg J., 22 (1982), pp. 371-381.

[3] C. BARDON AND D. G. LONGERON, *Influence of very low interfacial tension on relative permeability*, Soc. Petroleum Engrg. J., 15 (1980), pp. 391-401.

[4] J. B. BELL, C. N. DAWSON, AND G. R. SHUBIN, *An unsplit, higher-order Godunov method for scalar conservation laws in multiple dimensions*, J. Comput. Phys., 74 (1988), pp. 1-24.

[5] J. B. BELL, P. COLELLA, AND J. A. TRANGENSTEIN, *Higher-order Godunov methods for general systems of hyperbolic conservation laws*, J. Comput. Phys., to appear.

[6] R. M. BOWEN, *Compressible porous media models by use of the theory of mixtures*, Internat. J. Engrg. Sci., 20 (1982), pp. 697-735.

[7] K. H. COATS, *Simulation of gas condensate reservoir performance*, in Proc. 6th SPE Symposium on Reservoir Simulation, New Orleans, LA, 1982.

[8] R. E. COLLINS, *Flow of Fluids through Porous Materials*, Petroleum Publishing Co., Tulsa, OK, 1961.

[9] H. T. DAVIS AND L. E. SCRIVEN, *The origins of low interfacial tensions for enhanced oil recovery*, in Proc. 55th Annual Fall Technical Conference and Exhibition of the Society of Petroleum Engineers of AIME, Dallas, TX, 1980.

[10] 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.

[11] J. M. DUMORÉ, J. HAGOORT, AND A. S. RISSEEUW, *An analytical model for one-dimensional, three-component condensing and vaporizing gas drives*, Soc. Petroleum Engrg. J., 24 (1984), pp. 169-179.

[12] J. F. ELY AND H. J. M. HANLEY, *Prediction of transport properties*: 1. *Viscosity of fluids and mixtures*, Ind. Engrg. Chem. Fundam., 20 (1981), pp. 323-332.

[13] K. E. GUBBINS AND J. M. HAILLE, *Molecular theories of interfacial tension*, in Improved Oil Recovery by Surfactant and Polymer Flooding, Academic Press, New York, 1977, pp. 119-159.

[14] F. G. HELFFERICH, *Theory of multicomponent, multiphase displacement in porous media*, Soc. Petroleum Engrg. J., 21 (1981), pp. 51-62.

[15] B. S. JHAVERI AND G. K. YOUNGREN, *Three-parameter modification of the Peng-Robinson equation of state to improve volumetric predictions*, in Proc. 59th Annual Technical Conference and Exhibition, Houston, TX, 1984.

[16] B. L. KEYFITZ AND H. C. KRANZER, *A system of non-strictly hyperbolic conservation laws arising in elasticity theory*, Arch. Rational Mech. Anal., 72 (1980), pp. 220-241.

[17] H. W. KUHN AND A. W. TUCKER, *Nonlinear Programming*, in Proc. 2nd Berkeley Symposium on Mathematical Statistics and Probability, University of California Press, Berkeley and Los Angeles, CA, 1961, pp. 481-492.

[18] E. KVAALEN, L. NEEL, AND D. TONDEUR, *Directions of Quasi-static mass and energy transfer between phases in multicomponent open systems*, Chem. Engrg. Sci., 40 (1985), pp. 1191-1204.

[19] J. LOHRENZ, B. G. BRAY, AND C. R. CLARK, *Calculating viscosities of reservoir fluids from their compositions*, J. Petroleum Tech., 16 (1964), pp. 1171-1176.

[20] R. K. MEHRA, R. A. HEIDEMANN, AND K. AZIZ, *A statistical approach for combining reservoir fluids into pseudo components for compositional model studies*, in Proc. 6th SPE Symposium on Reservoir Simulation, New Orleans, LA, 1982.

[21] M. A. J. MICHELS AND H. MEIJER, *The performance of cubic equations of state on the critical isotherms of pure fluids*, Fluid Phase Equilibria, 17 (1984), pp. 57-75.

[22] H. C. VAN NESS AND M. M. ABBOTT, *Classical Thermodynamics of Nonelectrolyte Solutions*, McGraw-Hill, New York, 1982.

[23] D. Y. PENG AND D. B. ROBINSON, *A new two-constant equation of state*, Ind. Engrg. Chem. Fundam., 15 (1976), pp. 59-64.

[24] J. H. PREVOST, *Mechanics of continuous porous media*, Internat. J. Engrg. Sci., 18 (1980), pp. 787-800.

[25] R. REID, J. PRAUSNITZ, AND T. SHERWOOD, *The Properties of Gases and Liquids*, McGraw-Hill, New York, 1977.

[26] T. F. RUSSELL, *Finite Elements With Characteristics for Two-Component Incompressible Miscible Displacement* (SPE paper 10500), in Proc. 6th SPE Symposium on Numerical Reservoir Simulation, New Orleans, LA, February 1-3, 1982, pp. 123-135.

[27] M. SAHIMI, H. T. DAVIS, AND L. E. SCRIVEN, *Thermodynamic modeling of phase and tension behavior of CO_2/hydrocarbon systems*, Soc. Petroleum Engrg. J., 25 (1985), pp. 235-254.

[28] J. M. SMITH AND H. C. VAN NESS, *Introduction to Chemical Engineering Thermodynamics*, McGraw-Hill, New York, 1975.

[29] W. E. SMITH AND R. W. MISSEN, *Chemical Reaction Equilibrium Analysis: Theory and Algorithms*, John Wiley, New York, 1982.

[30] F. I. STALKUP, JR., *Miscible Displacement*, Society of Petroleum Engineers of AIME, New York, Dallas, TX, 1984.

[31] J. A. TRANGENSTEIN, *Customized minimization techniques for phase equilibrium computations in reservoir simulation*, Chem. Engrg. Sci., 42 (1987), pp. 2847-2863.

[32] J. A. TRANGENSTEIN AND J. B. BELL, *Mathematical structure of black-oil reservoir simulation*, SIAM J. Appl. Math., 49 (1989), pp. 749-783.

[33] C. TSONOPOULOS AND J. L. HEIDMAN, *From Redlich-Kwong to the present*, Fluid Phase Equilibria, 24 (1985), pp. 1-23.

[34] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 846-879, September 1989
© 1989 Society for Industrial and Applied Mathematics
002

A RIEMANN SOLVER FOR A TWO-PHASE MULTICOMPONENT PROCESS*

THORMOD JOHANSEN†, ASLAK TVEITO‡, AND RAGNAR WINTHER‡

Abstract. A Riemann problem solver for an $(n+1)\times(n+1)$ system of nonstrictly hyperbolic conservation laws is presented in an algorithmic form. The system models polymer flooding as an enhanced oil-recovery process. The Riemann problem solver is implemented and some examples of solutions are given. The exact solution of the Riemann problem is also used to study the behaviour of the classical Godunov method.

Key words. Riemann problem, polymer flooding

AMS(MOS) subject classifications. 35L65, 76S05

1. Introduction. The purpose of this paper is to discuss the structure and implementation of a Riemann solver for a multicomponent model of the form

$$\begin{aligned} &s_t + f(s, c_1, \cdots, c_n)_x = 0, \\ &(sc_i + a_i(c_i))_t + (c_i f(s, c_1, \cdots, c_n))_x = 0, \qquad i = 1, \cdots, n \end{aligned} \tag{1}$$

where $(s, c_1, \cdots, c_n) = (s, c) \in \mathbf{I}^{n+1}$ is the unknown state vector and where $f: \mathbf{I}^{n+1} \to \mathbf{I}$ and $a_i: \mathbf{I} \to \mathbb{R}$, $i = 1, \cdots, n$, are given functions. $\mathbf{I}$ denotes the unit interval. By a Riemann solver we mean an algorithm that solves the Riemann problem for the hyperbolic system of conservation laws (1) with general Riemann data. The algorithm is derived from the analysis given in [13], where existence and uniqueness of the solution of the Riemann problem are established. The Riemann solver has been implemented and we use it to give some examples of solutions of Riemann problems. We also compare the solution of a Riemann problem generated by the classical Godunov method with the exact solution given by the Riemann solver.

In oil recovery, water is frequently used to displace the oil in the porous rock. Assume that the fluid in the medium consists of two immiscible phases, an aqueous phase and an oleic phase, and that the rock and the fluid are incompressible. Then, neglecting diffusive effects, the displacement of oil by water in a one-dimensional homogeneous medium can be modelled by the Buckley-Leverett equation

$$s_t + f(s)_x = 0 \tag{2}$$

where $s = s(x, t)$ is the saturation of the aqueous phase ($1 - s$ is the saturation of the oleic phase) and where f is the fractional flow function given by

$$f(s) = \frac{k(s)}{k(s) + k^o(s)\dfrac{\mu}{\mu^o}}. \tag{3}$$

Here $k(s)$ and $k^o(s)$ are the relative permeabilities of the aqueous phase and the oleic phase respectively, and μ and μ^o are the corresponding viscosities. This displacement process is called a water flooding.

* Received by the editors June 20, 1988; accepted for publication (in revised form) October 12, 1988. This research was supported by VISTA, a research cooperation between the Norwegian Academy of Science and Letters and Den norske stats oljeselskap a.s. (Statoil).

† Institute for Energy Technology, P.O. Box 40, Kjeller, Norway.

‡ Department of Informatics, P.O. Box 1080 Blindern, University of Oslo, 0369, Oslo, 3, Norway.

To enhance the oil-recovery process, some polymer can be dissolved in the water to increase the viscosity and thereby improve the displacement effect. This is called polymer flooding. The model (2) can be extended to include this kind of flooding.

Assume that a small amount of n different polymer components is added to the water; the amount is supposed to be so small that the volumes are unchanged. Further we assume that all the polymer is contained in the aqueous phase and that the polymer is totally miscible in water.

Let $c_i = c_i(x, t)$ denote the concentration of the ith polymer component in the aqueous phase, then the system (1) models polymer flooding. The fractional flow function is given by

$$f(s, c) = \frac{k(s)}{k(s) + k^o(s)\dfrac{\mu(c)}{\mu^o}} \tag{4}$$

where $c = (c_1, \cdots, c_n)$. Observe that the viscosity of the aqueous phase is a function of the polymer concentrations. The function $\mu = \mu(c_1, \cdots, c_n)$ is supposed to be an increasing function in all c_i. The model is often simplified by choosing $n = 1$. But in practical flooding situations, the number of polymer components is usually quite large.

The term a_i in (1) models the adsorption of polymer on the rock. The assumption that a_i is only a function of c_i means that the adsorption of the ith component is independent of the other components. This simplification is often made in the modelling of polymer flooding (cf. [18]). By allowing the more general adsorption term $a_i = a_i(c)$, the differential equations (1) also model some other processes in enhanced oil recovery (cf. [14]). A more precise formulation of our model will be given in the next section. For a derivation of the model (1) we refer to [9], [12], or [17].

The model (1) is an example of a system of hyperbolic conservation laws. In general an $m \times m$ system of hyperbolic conservation laws can be written in the form

$$u_t + F(u)_x = 0 \tag{5}$$

where $u = u(x, t) \in \mathbb{R}^m$ and $F: \mathbb{R}^m \to \mathbb{R}^m$ is a smooth function and where the eigenvalues of the Jacobian dF of F are real. The Riemann problem for (5) is the pure initial value problem with initial data of the form

$$u(x, 0) = \begin{cases} u^L, & x < 0, \\ u^R, & x > 0. \end{cases} \tag{6}$$

If $u(x, t)$ is a solution of (5), (6) then so is $u(ax, at)$, for all $a > 0$. Hence, to be unique, the solution of a Riemann problem has to be self-similar, i.e.,

$$u(x, t) = \bar{u}(x/t). \tag{7}$$

To solve the Riemann problem, a path connecting the left state $\bar{u}(-\infty) = u^L$ and the right state $\bar{u}(\infty) = u^R$ must be computed. The solution is composed of constant states and two types of elementary waves: rarefaction and shock.

Rarefaction waves are smooth solutions of the Riemann problem. By rewriting the system (5) in the form

$$u_t + dF(u)u_x = 0 \tag{8}$$

and substituting (7) into (8), we obtain the equation

$$\left(dF - \frac{x}{t} I\right)\bar{u}'\left(\frac{x}{t}\right) = 0. \tag{9}$$

Hence, in this case the value x/t equals an eigenvalue of dF and $\bar{u}'$ equals the corresponding eigenvector. Let λ be an eigenvalue of dF and let r be the corresponding eigenvector. If $\bar{u}$ is an integral curve of r, i.e., if

$$\bar{u}'(\xi) = r(\bar{u}(\xi)),$$

connects the states u^L and u^R and the eigenvalue $\lambda = \lambda(\bar{u})$ increases as $\bar{u}$ goes from u^L to u^R, then the path $\bar{u}$ traverses is called a rarefaction curve. The rarefaction curves are directed towards increasing values of $\lambda(\bar{u})$. Furthermore, the function

$$u(x, t) = \begin{cases} u^L, & x/t < \lambda(u^L), \\ \bar{u}(x/t), & x/t = \lambda(\bar{u}), \\ u^R, & x/t > \lambda(u^R) \end{cases}$$

is called a simple rarefaction solution of (5). The speed of the rarefaction wave is given by $\lambda(\bar{u})$. In particular the initial speed is $\lambda(u^L)$ and the final speed is $\lambda(u^R)$.

Shock waves are weak solutions of (5), (6) of the form

$$u(x, t) = \begin{cases} u^L, & x/t < \sigma, \\ u^R, & x/t > \sigma, \end{cases} \tag{10}$$

where σ is the shock speed. In order to be a weak solution u^L, u^R, and σ have to satisfy the Rankine-Hugoniot condition

$$F(u^R) - F(u^L) = \sigma(u^R - u^L) \tag{11}$$

(cf. [16], [19]). A weak solution is not necessarily unique. Therefore an additional entropy condition is introduced to obtain uniqueness. The entropy conditions used in this paper are derived from a travelling wave analysis (cf. [12], [19]).

The solution of the Riemann problem consists of a sequence of elementary waves connecting the left state, u^L, and the right state, u^R. We introduce the following notation: $u^1 \xrightarrow{w} u^2$ means that the state u^1 can be connected to the state u^2 by an elementary wave w. Let $u^1 \xrightarrow{w_1} u^2$ and $u^2 \xrightarrow{w_2} u^3$ be two elementary waves, and let v_i^I, v_i^F, $i = 1, 2$, denote the initial and final speed of the waves, respectively. If w_i is a shock, then $v_i^I = v_i^F = \sigma$. The two waves are said to be compatible if

$$v_1^F \leqq v_2^I. \tag{12}$$

We impose strict inequality in (12) if both waves are shocks. The notation $u^1 \xrightarrow{w_1} u^2 \xrightarrow{w_2} u^3$ means that w_1 and w_2 are compatible, and that u^1 can be connected to u^3 through the intermediate state u^2. In order to solve the Riemann problem a sequence

$$u^L = u^1 \xrightarrow{w_1} u^2 \xrightarrow{w_2} \cdots \xrightarrow{w_k} u^R$$

of compatible elementary waves must be computed.

There are several reasons for studying Riemann problems. First, it is of great interest to know the analytical solution of an initial value problem, even for a small class of initial data. This gives a better understanding of the physical process that is modelled. Second, the solution of a Riemann problem gives a unique opportunity to test numerical methods. The solution of Riemann problems can also be used in numerical methods. The basis of the Random Choice Method, developed by Glimm [7] and used as a numerical method in [2], [3], and [4], is the solution of the Riemann problem. Also, the Godunov method [8] depends on the solution of the Riemann problem, at least for the value $x/t = 0$. Furthermore, solutions of Riemann problems are used in front-tracking methods (cf. [5]).

The Riemann problem for the system (1) with $n=1$ and with $a_1(c_1)\equiv 0$ was analyzed by Keyfitz and Kranzer [15] and by Isaacson [9]. The model was also studied by Barkve [1], who allowed a more general form of the fractional flow function. In [12] the adsorption term was included. Isaacson and Temple [10] studied the system (1) for $n>1$ and with $a_i(c_i)\equiv 0$, $i=1,\cdots,n$. In [13] the adsorption functions of the form $a_i=a_i(c_i)$ have been included.

In this paper we will give an algorithm to compute the solution of the Riemann problem for (1). In § 2 we will give a precise formulation of the mathematical model. The simple rarefaction waves are derived in § 3, while § 4 is devoted to the shock waves. The algorithm to solve the Riemann problem for (1) is given in § 5 and it is used to produce some examples in § 6. In § 7 we study the behaviour of the Godunov method by applying it to a Riemann problem for which we have the exact solution.

2. The mathematical model. In this section we briefly review the mathematical model stated in [13] for $n>1$ components. Let $s=s(x,t)\in \mathbf{I}=[0,1]$ be the water saturation, and let $c_i=c_i(x,t)\in\mathbf{I}$, $i=1,\cdots,n$ be the concentration of the polymer components in the aqueous phase. Then the model is given by a nonstrictly hyperbolic $(n+1)\times(n+1)$ system of conservation laws of the form

$$\begin{aligned} &s_t+f(s,c_1,\cdots,c_n)_x=0,\\ &(sc_i+a_i(c_i))_t+(c_i f(s,c_1,\cdots,c_n))_x=0, \qquad i=1,\cdots,n \end{aligned} \tag{13}$$

where $f:\mathbf{I}^{n+1}\to\mathbf{I}$ is the fractional flow function and $a_i:\mathbf{I}\to\mathbb{R}$, $i=1,\cdots,n$, are the adsorption functions. The fractional flow function is assumed to be a smooth function such that for each $c=(c_1,\cdots,c_n)\in\mathbf{I}^n$, f has the following properties:

(i) $f(0,c)\equiv 0$, $f(1,c)\equiv 1$.

(ii) $f_s(s,c)=\partial f(s,c)/\partial s>0$ for $0<s<1$.

(iii) There exists $s^I=s^I(c)\in\mathbf{I}$ such that $f_{ss}(s,c)>0$ for $s\in(0,s^I(c))$ and $f_{ss}(s,c)<0$ for $s\in(s^I(c),1)$.

(iv) $f_i(s,c)=\partial f(s,c)/\partial c_i<0$ for $0<s<1$, $i=1,\cdots,n$.

The derivatives of the adsorption functions $a_i(c_i)$ will be denoted by $h_i(c_i)=da_i(c_i)/dc_i$. The adsorption functions are assumed to be smooth functions such that

(i) $a_i(0)=0$,

(ii) $h_i(c_i)>0$,

(iii) $dh_i(c_i)/dc_i<0$

for all $i=1,\cdots,n$. We assume no relation between the a_i's. Figure 1 shows a typical fractional flow function for a fixed $c=(c_1,\cdots,c_n)$, and Fig. 2 shows an adsorption function.

The purpose of this paper is to discuss solutions of the differential equation (13) with Riemann initial data, i.e.,

$$(s(x,0),c(x,0))=\begin{cases}(s^L,c^L), & x<0,\\ (s^R,c^R), & x>0.\end{cases} \tag{14}$$

The main result of [13] is a theorem which states that for arbitrary left state $u^L=(s^L,c^L)\in\mathbf{I}^{n+1}$ and right state $u^R=(s^R,c^R)\in\mathbf{I}^{n+1}$ there exists a unique solution of this Riemann problem. Here we will discuss how to implement an algorithm to compute the entropy solution of the Riemann problem (13), (14) to any desired degree of accuracy. Some examples of solutions will also be given.

3. Rarefaction waves. Let $u=(s,c)\in\mathbf{I}^{n+1}$ be the state vector; then the system (13) can be written as

$$u_t+A(u)u_x=0 \tag{15}$$

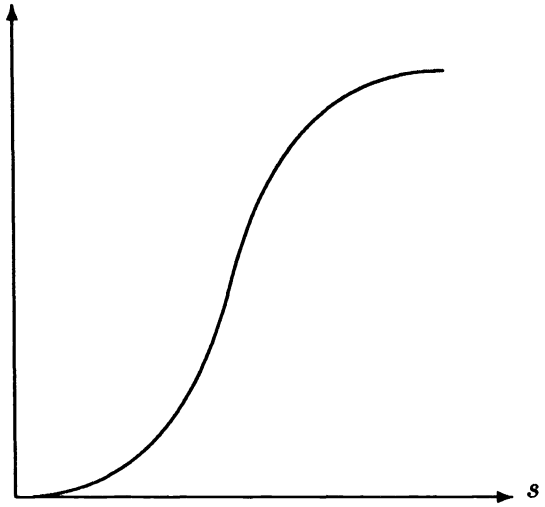


FIG. 1. *The fractional flow function f as a function of s for a fixed c.*

where $A(u)$ is the $(n+1)\times(n+1)$ matrix

$$A(u)=\begin{pmatrix} f_s & f_1 & f_2 & \cdots & f_n \\ 0 & f/(s+h_1) & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & f/(s+h_n) \end{pmatrix}.$$

Here $f_s=\partial f(s, c)/\partial s$ and $f_i=\partial f(s, c)/\partial c_i$. The eigenvalues of A are

$$\lambda^s=\lambda^s(s, c)=f_s(s, c),$$

$$\lambda^i=\lambda^i(s, c)=\frac{f(s, c)}{s+h_i(c_i)}, \qquad i=1, \cdots, n.$$

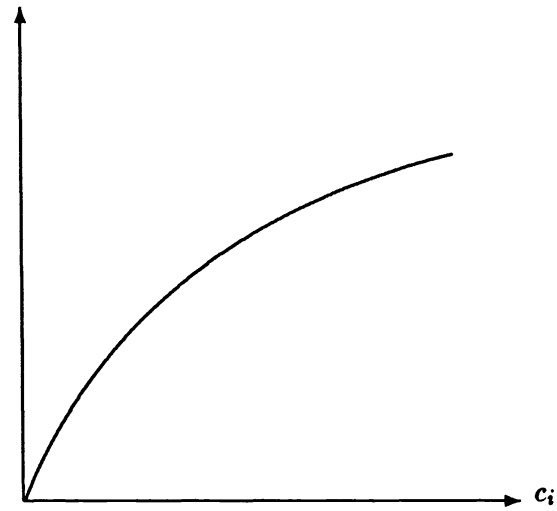


FIG. 2. *A typical adsorption function* $a_i(c_i)$.

The eigenvalues may coincide. For a fixed i and a given $c \in \mathbf{I}^n$ we assume that there is exactly one value of $s > 0$ such that $\lambda^s(s, c) = \lambda^i(s, c)$ (cf. Fig. 3). Since we assume

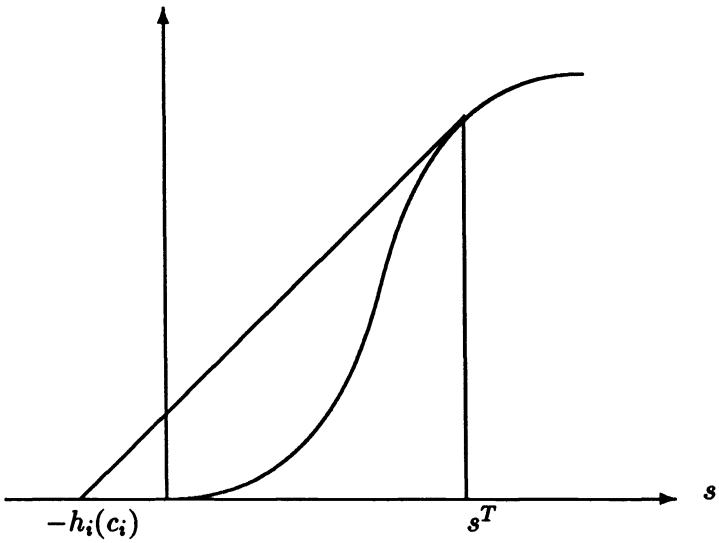


FIG. 3. *There exists exactly one* $s = s^T > 0$ *such that* $f_s(s^T, c) = f(s^T, c)/(s^T + h_i(c_i))$ *for fixed* c *and* i.

no relation between the a_i's, the family $\{\lambda^i\}_{i=1}^n$ may also have coinciding eigenvalues. The relation $\lambda^i = \lambda^j$, $i \neq j$, defines c_j as a function of c_i:

$$c_j = h_j^{-1}(h_i(c_i)) \overset{\text{def}}{=} g_{i,j}(c_i).$$

From the properties of the adsorption functions it follows that

$$\frac{dg_{i,j}(c_i)}{dc_i} > 0.$$

It may also occur that several members of the family $\{\lambda^i\}_{i=1}^n$ coincide. For such waves we pick one index i, among those that coincide, and call it the leading index of the wave, and then define the index set

$$J = \{j: 1 \leqq j \leqq n, j \neq i, \lambda^i = \lambda^j\}. \tag{16}$$

If $J = \emptyset$ then λ^i is called a single eigenvalue. The rarefaction waves are smooth solutions of the Riemann problem that lie on an integral curve of a right eigenvector of A. For each eigenvalue λ there is a corresponding right eigenvector r. A rarefaction curve is defined to be the integral curve of r directed towards increasing values of λ.

For $\lambda = \lambda^s$ the corresponding eigenvector is $r = (1, 0, \cdots, 0)^T \in \mathbb{R}^{n+1}$, so the integral curves are given by $c = \textit{constant}$. For constant c the system (13) reduces to the standard Buckley–Leverett equation

$$s_t + f(s)_x = 0 \tag{17}$$

where $f(s) = f(s, c)$. Hence, the rarefaction curves for $\lambda = \lambda^s$ correspond to the rarefaction curves of (17). These curves are referred to as s-rarefactions.

The integral curves of the right eigenvector corresponding to the eigenvalue λ^i satisfy (cf. [13]) the differential equation

$$
\begin{aligned}
&\left(f_i + \sum_{j \in J} f_j \frac{dg_{i,j}}{dc_i}\right)\frac{dc_i}{ds} = \lambda^i - \lambda^s, \\
&c_j = g_{i,j}(c_i), \qquad j \in J, \\
&c_k = \textit{constant} \quad \text{for } k \neq i \text{ and } k \notin J
\end{aligned} \tag{18}
$$

where J is the index set given by (16). A c-rarefaction curve is a solution of (18) directed towards increasing values of c_i (cf. [13]). Since $dg_{i,j}/dc_i > 0$, a c-rarefaction is nondecreasing in all c-components. The form of the (s, c_i)-projection of a c-rarefaction curve is depicted in Fig. 4.

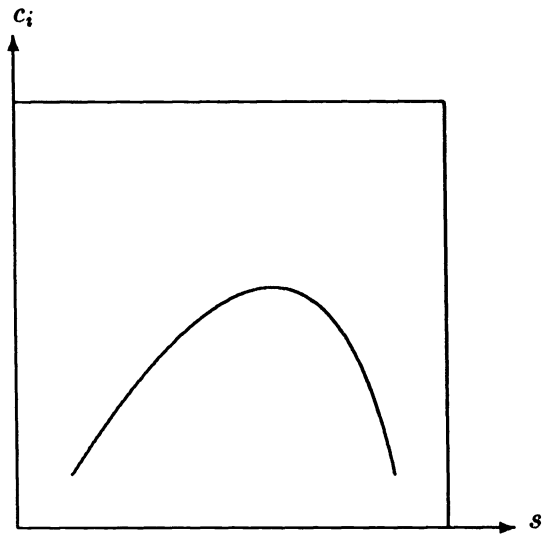


FIG. 4. *The projection of a c-rarefaction curve into the* (s, c_i)*-space.*

4. Shock waves. The Riemann problem (13), (14) can have shock solutions, i.e., weak solutions of the form

$$
u(x,t) = \begin{cases} u^L, & x/t < \sigma, \\ u^R, & x/t > \sigma, \end{cases} \tag{19}
$$

where σ is the shock speed and $u = (s, c) = (s, c_1, \cdots, c_n)$. Weak solutions of the form (19) must satisfy the Rankine-Hugoniot condition:

$$
\begin{aligned}
&f(s^R, c^R) - f(s^L, c^L) = \sigma(s^R - s^L), \\
&c_i^R f(s^R, c^R) - c_i^L f(s^L, c^L) = \sigma(s^R c_i^R + a_i(c_i^R) - s^L c_i^L - a_i(c_i^L))
\end{aligned} \tag{20}
$$

for $i = 1, \cdots, n$ and where $c^L = (c_1^L, \cdots, c_n^L)$, $c^R = (c_1^R, \cdots, c_n^R)$.

Suppose $c^L = c^R$; then (20) reduces to

$$
f(s^R) - f(s^L) = \sigma(s^R - s^L) \tag{21}
$$

where $f(s) = f(s, c^L) = f(s, c^R)$. Equation (21) is the Rankine-Hugoniot condition for the Buckley-Leverett equation (17). Not all shocks satisfying (21) are allowed; the

shocks must in addition satisfy an entropy condition, which is imposed in order to obtain the physically relevant weak solutions. In the case $c^L = c^R$, the entropy condition is well known:

$$(f(s)-f(s^L)-\sigma(s-s^L))\,\text{sign}\,(s-s^L)\geqq 0 \tag{22}$$

for all s between s^L and s^R (cf. [6]). Shocks satisfying (21) and (22) (for $c^R = c^L$) are called s-shocks.

Suppose that $c^L \neq c^R$. Define the functions h_i^L by

$$h_i^L(c_i)=\begin{cases}\dfrac{a_i(c_i)-a_i(c_i^L)}{c_i-c_i^L}, & c_i \neq c_i^L,\\ h_i(c_i), & c_i = c_i^L\end{cases} \tag{23}$$

for $i=1,\cdots,n$. Suppose that $c_i^L \neq c_i^R$ in (19) for some fixed index i, and define the index set

$$J=\{j\colon 1\leqq j\leqq n, j\neq i,\ c_j^L \neq c_j^R\}. \tag{24}$$

Again the i-index is called the leading index of the wave. The Rankine-Hugoniot condition (20) can be written (cf. [13]):

$$\frac{f(s^R,c^R)}{s^R+h}=\frac{f(s^L,c^L)}{s^L+h}=\sigma \tag{25}$$

where

$$h \overset{\text{def}}{=} h_i^L(c_i^R) \quad (=h_j^L(c_j^R) \quad j\in J).$$

The entropy condition for these shocks (cf. [13]) are

$$\begin{aligned}&c_i^L > c_i^R, \quad c_j^L > c_j^R \quad \forall j\in J,\\ &\lambda^s(u^R)<\sigma \quad \text{or} \quad \lambda^s(u^L),\lambda^s(u^R)\geqq\sigma.\end{aligned} \tag{26}$$

The shocks satisfying (25) and (26) are referred to as c-shocks. Observe that a c-shock is nonincreasing in all c-components.

The two cases $s=0$ and $s=1$ must be treated separately. When $s=0$ we must have $c=0$, since c is the concentration of the polymer in the aqueous phase. Then clearly $s=0$ implies $c=0$. If $s=1$ and $c_i^L<c_i^R$, the rarefaction curves are given by

$$\begin{aligned}&s=1,\\ &c_j=g_{i,j}(c_i), \qquad j\in J,\\ &c_k=\textit{constant} \quad \text{for } k\neq i \text{ and } k\notin J\end{aligned}$$

for increasing values of c_i. Similarly, if $c_i^L > c_i^R$, then

$$\begin{aligned}&s=1,\\ &c_k=\begin{cases}c_k^L, & x/t<\sigma,\\ c_k^R, & x/t>\sigma,\end{cases} \quad k=i \text{ or } k\in J,\\ &c_k=\textit{constant} \quad \text{for } k\neq i \text{ and } k\notin J,\end{aligned}$$

where $\sigma=1/(1+h_i^L(c_i^R))$, is a c-shock.

5. The Riemann solver. In this section we will give an algorithm to compute the solution of the Riemann problem (13), (14). The algorithm is based on a constructive

proof of existence and uniqueness of the entropy solution of the Riemann problem given in [13]. Here we will be concerned with the computational aspects of the problem based on the mathematical proofs given there. In principle, we are able to compute the exact solution of the Riemann problem, but since the algorithm involves integration of ordinary differential equations and solution of nonlinear algebraic equations, some numerical errors are introduced. These errors, for sufficiently accurate numerical methods, are negligible, and we refer to the solution as the exact solution.

5.1. An overview of the algorithm. The solution of the Riemann problem consists of a sequence of elementary waves connecting the left state $u^L=(s^L, c^L)$ to the right state $u^R=(s^R, c^R)$. An algorithm to compute this solution is a systematical description of how to piece the elementary waves together. From § 1 we recall that the notation $u^1 \xrightarrow{w} u^2$ means that the state u^1 can be connected to the state u^2 by an elementary wave w. The elementary wave w is either an s-wave or a c-wave. Sometimes the c-waves will be specified to be either a c-shock, cS, or a c-rarefaction, cR. The s-waves are standard Buckley–Leverett waves, so it will not be specified whether it is an s-shock, s-rarefaction, or a composition of s-waves.

We introduce an auxiliary $n\times n$ system of hyperbolic conservation laws

$$(c_i+a_i(c_i))_t+(c_i)_x=0, \qquad i=1,\cdots,n. \tag{27}$$

This model, associated with the two-phase problem (13), describes one-phase flow of the n components. Carrying out the time differentiation in (27), the system can be rewritten in the form:

$$(c_i)_t+\frac{1}{1+h_i(c_i)}(c_i)_x=0, \qquad i=1,\cdots,n. \tag{28}$$

The system is completely decoupled in n-scalar equations, but since it may occur that $h_i(c_i)=h_j(c_j)$ for $i\neq j$, the system is nonstrictly hyperbolic.

Consider the ith equation in (28) with Riemann initial data,

$$c_i(x,0)=\begin{cases} c_i^L, & x<0,\\ c_i^R, & x>0.\end{cases}$$

Suppose $c_i^L<c_i^R$; then the solution of this problem is

$$c_i(x,t)=\begin{cases} c_i^L, & \dfrac{x}{t}<\dfrac{1}{1+h_i(c_i^L)},\\[2ex] c_i\left(\dfrac{x}{t}\right), & \dfrac{x}{t}=\dfrac{1}{1+h_i(c_i(x/t))},\\[2ex] c_i^R, & \dfrac{x}{t}>\dfrac{1}{1+h_i(c_i^R)}\end{cases}$$

which is a rarefaction wave from c_i^L to c_i^R. If $c_i^L>c_i^R$, then the solution is a shock given by

$$c_i(x,t)=\begin{cases} c_i^L, & \dfrac{x}{t}<\dfrac{1}{1+h_i^L(c_i^R)},\\[2ex] c_i^R, & \dfrac{x}{t}>\dfrac{1}{1+h_i^L(c_i^R)}.\end{cases}$$

Let $(s^L, c^L)\xrightarrow{c}(s^R, c^R)$ be a c-wave of the system (13) with leading index i (cf. (16), (24)). Define the projection $P(c)$ of the c-wave into the c-space by imposing the

wave speeds

$$\begin{cases} \dfrac{1}{1+h_i(c_i)}, & c_i^L < c_i^R \quad \text{(rarefaction)}, \\ \dfrac{1}{1+h_i^L(c_i^R)}, & c_i^L > c_i^R \quad \text{(shock)}. \end{cases} \tag{29}$$

Then the wave $c^L \xrightarrow{P(c)} c^R$ is an elementary wave of (28). The following lemma, proved in [13], gives an important relation between the c-waves of (13) and the projected c-waves; the relation is referred to as the *projection principle.*

LEMMA 1. *Compatibility of the waves*

$$(s^L, c^L) \xrightarrow{\bar{c}} (s^1, c) \xrightarrow{s} (s^2, c) \xrightarrow{\bar{c}} (s^R, c^R)$$

of the system (13) *implies compatibility of the waves*

$$c^L \xrightarrow{P(\bar{c})} c \xrightarrow{P(\bar{c})} c^R$$

for the system (27).

It follows from this lemma that if the solution of the Riemann problem for (13) with left state (s^L, c^L) and right state (s^R, c^R) is given, then the projection of the solution into c-space, imposing the wave speeds (29), is the solution of the associated Riemann problem for (27) with left state c^L and right state c^R. As a consequence, the structure of the c-waves of the desired Riemann solution for (13) can be determined by solving the associated Riemann problem for (27) (cf. Fig. 5).

The algorithm for solving the Riemann problem (13), (14) is divided into three parts. First, the c-wave structure is determined by solving the associated Riemann problem for the one-phase model (27). This part of the algorithm depends only on the values of the Riemann data c^L and c^R, and is independent of the values s^L and

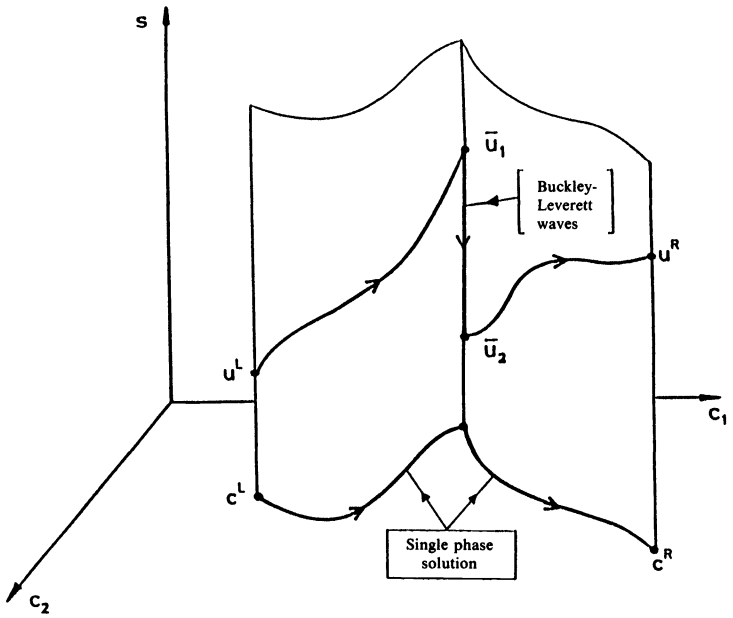


FIG. 5. *The projection principle in the case* $n = 2$.

s^R. Second, a forward sweep is performed. The computation in the forward sweep depends on the Riemann solution of the associated one-phase model and on the given value of s^L. Essentially, we there determine all possible Riemann solutions with s^L, c^L and c^R given, and where $s^R \in \mathbf{I}$ is arbitrary. Finally, in the backward sweep we calculate the desired Riemann solution from the given value of s^R.

5.2. The wave structure of the associated one-phase problem. In order to solve the Riemann problem (13), (14), we first determine the structure of the associated Riemann problem for the system (27). Since (27) consists of n completely decoupled scalar equations, the wave structure is easily found by viewing the wave-cone of each component in the $x-t$ space (cf. Fig. 6). Since the system is nonstrictly hyperbolic, the wave cone of one wave may overlap the cone of another. When cones are overlapping, we will have a wave where more than one c-component changes.

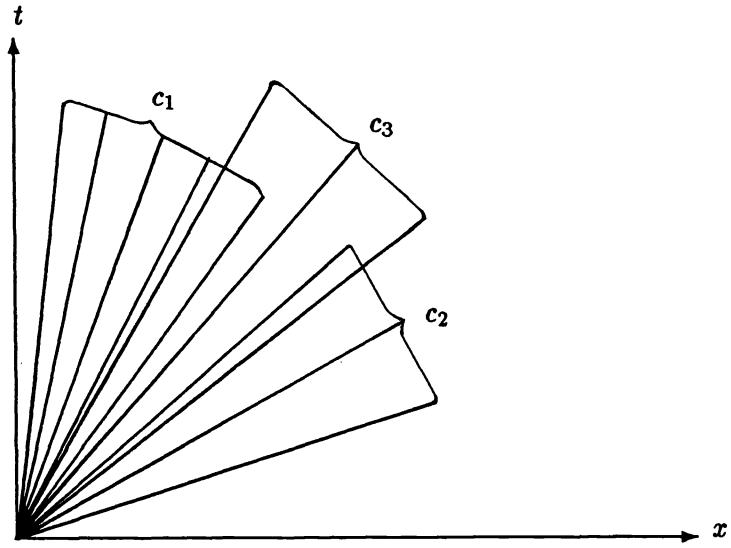


FIG. 6. *The wave cones of the associated one-phase problem in the case* $n = 3$.

Suppose the solution of the Riemann problem for (27) is a composition of k elementary waves such that

$$c^L = c^1 \xrightarrow{w_1} c^2 \xrightarrow{w_2} c^3 \cdots \xrightarrow{w_k} c^{k+1} = c^R. \tag{30}$$

Here $c^\ell = (c_1^\ell, c_2^\ell, \cdots, c_n^\ell)$ is the ℓth intermediate state, $\ell = 1, \cdots, k+1$. c_j^ℓ gives the value of the j-component in the ℓth intermediate state. The determination of the wave structure for (27) consists of computing the intermediate states c^ℓ and storing the following information of each c-wave w_ℓ: (a) the leading index i of the wave (cf. (16) and (24)); (b) the index set J giving the index of each component involved in the wave; and (c) an indication of whether the wave is a shock or a rarefaction.

We do not give any details of this part of the algorithm, since it is a straightforward searching and bookkeeping problem.

5.3. The state space. When the associated one-phase problem is solved, the state space $\mathbf{I}^{n+1}$ may be given a two-dimensional interpretation.

Assume that the ℓth wave in (30) is a rarefaction. Let i be the leading index of the wave and let J be the set of indices involved in the wave (cf. (16)). Then we locally

introduce a two-dimensional state space for this wave with an s- and a c_i-axis. The c_i-component will increase from c_i^ℓ to $c_i^{\ell+1}$ in this wave; the behaviour of the other c-components are given by (18). If the ℓth wave is a shock, then we locally introduce a two-dimensional state space for this wave with an s-axis and a c_i-axis turned upside down. The c_i-component will decrease from c_i^ℓ to $c_i^{\ell+1}$ in this wave. Stacking the strips defined by the local state spaces on each other, a global two-dimensional state space is obtained (cf. Fig. 7). If the ℓth wave in (30) is a rarefaction, then the ℓth strip is called a rarefaction strip; this is sometimes abbreviated "ℓ rarefaction," correspondingly for shocks.

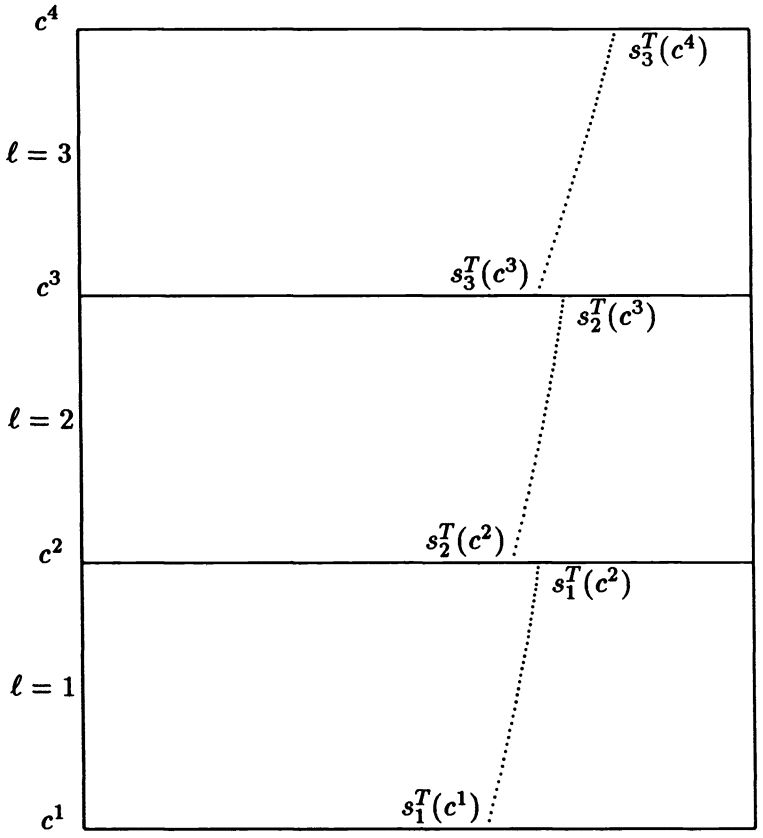


FIG. 7. *The state space for* $k = 3$ *waves.*

The wave speed for a c-wave in the ℓth strip is defined by

$$\sigma_\ell(s, c) = \begin{cases} \dfrac{f(s, c)}{s + h_i(c_i)}, & \ell\text{-rarefaction}, \\ \dfrac{f(s, c)}{s + h_i^L(c_i^R)}, & \ell\text{-shock}. \end{cases} \tag{31}$$

For this strip a transition curve T_ℓ is defined by the relation

$$\sigma_\ell(s, c) = f_s(s, c) \tag{32}$$

for $s > 0$. We assume that for each c in the ℓth strip there exists a unique $s = s_\ell^T(c)$, $0 < s_\ell^T(c) \leqq 1$ such that (32) holds. Observe that in the rarefaction case, the value $s_\ell^T(c)$

corresponds to the point where $\lambda^s = \lambda^i$. On the line $c = c^\ell$, which separates the $(\ell-1)$-strip and the ℓ-strip, there are two possible values of s_ℓ^T: $s_{\ell-1}^T(c^\ell)$, $s_\ell^T(c^\ell)$ corresponding to the $(\ell-1)$-strip and the ℓ-strip definition of the wave speed σ. The two values satisfy the relation $s_{\ell-1}^T(c^\ell) \geqq s_\ell^T(c^\ell)$ (cf. [13] and Fig. 7).

For each state (s, c) in the ℓth strip, we define a critical value $s_\ell^K = s_\ell^K(s, c)$ by the relation

$$\sigma_\ell(s_\ell^K, c) = \sigma_\ell(s, c). \tag{33}$$

If $s \leqq s_\ell^T(c)$, then s_ℓ^K is either the unique value $\geqq s_\ell^T(c)$ such that (33) holds, or if no such $s_\ell^K \leqq 1$ exist, we define $s_\ell^K = +\infty$. If $s > s_\ell^T$, then s_ℓ^K is defined to be the unique value $s_\ell^K(s, c) \leqq s_\ell^T(c)$ such that (33) holds. For a geometrical interpretation of the critical value see Fig. 8.

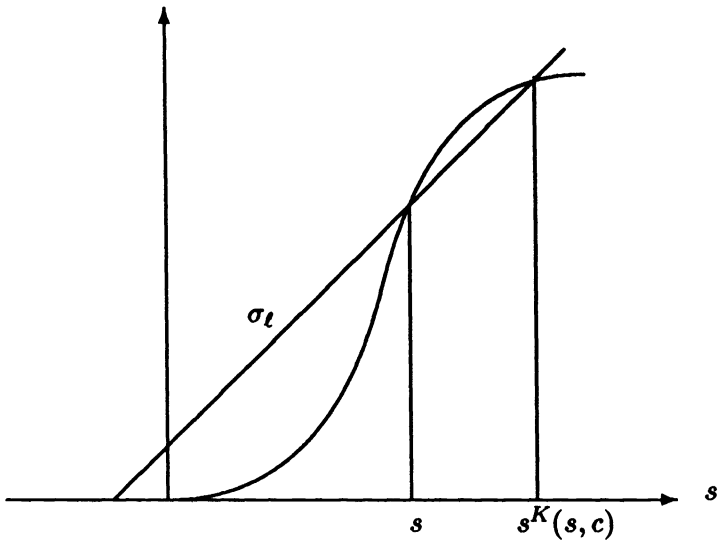


FIG. 8. *The critical value.*

5.4. The forward sweep. Assume the solution of the Riemann problem for (27) is given by (30). To find the solution of the Riemann problem (13), (14), a solution path from (s^L, c^L) to (s^R, c^R) has to be constructed. The next lemma establishes existence and uniqueness of the solution path for the Riemann problem (13), (14).

LEMMA 2. *Assume the solution of the Riemann problem of the associated one-phase problem* (27) *is given by* (30), *then for arbitrary left state* s^L *and right state* s^R *the Riemann problem* (13), (14) *has a unique solution. Furthermore, for any given* s^L *there exist unique values* $\hat{s}$ *and* $\hat{s}^K$, *where*

$$\hat{s}^K = s_k^K(\hat{s}, c^{k+1}) \quad \text{and} \quad \hat{s} \leqq s_k^T(c^{k+1}) \leqq \hat{s}^K,$$

such that the Riemann problem solution terminates with a c-wave if and only if

$$s^R = \hat{s} \quad \text{or} \quad s^R > \hat{s}^K.$$

It should be emphasized that the values $\hat{s}$, $\hat{s}^K$ are determined independently of s^R. By applying the lemma above to a Riemann problem with $u^L = (s^L, c^L)$ and $u^R = (s^R, c^\ell)$ the following corollary is obtained.

COROLLARY 1. *Assume the solution of the Riemann problem of the associated one-phase problem* (27) *with left state* c^L *and right state* c^R *is given by* (30). *Then the*

Riemann problem for (13) *with*

$$u^L = (s^L, c^L) \quad \textit{and} \quad u^R = (s^R, c^\ell) \tag{34}$$

for a fixed ℓ, $2 \leqq \ell \leqq k+1$, *has a unique solution. Furthermore, for any given* s^L *there exist unique values* $\hat{s}_\ell$ *and* $\hat{s}_\ell^K$, *where*

$$\hat{s}_\ell^K = s_{\ell-1}^K(\hat{s}_\ell, c^\ell) \quad \textit{and} \quad \hat{s}_\ell \leqq s_{\ell-1}^T(c^\ell) \leqq \hat{s}_\ell^K$$

such that the Riemann problem solution of (13), (34) *terminates with a c-wave if and only if*

$$s^R = \hat{s}_\ell \quad \textit{or} \quad s^R > \hat{s}_\ell^K.$$

From this corollary it follows that given s^L and the solution of the associated Riemann problem for (27) with left state c^L and right state c^R, there exists a unique sequence of points $\hat{s}_\ell$ independently of the value of s^R which determine for what s-values it is possible to enter an intermediate c-state. This enables us to split the algorithm in a forward sweep where the values $\hat{s}_\ell$, $\hat{s}_\ell^K$ are computed starting at u^L, and a backward sweep where these values are used to compute the solution path starting at u^R.

In the next section we will show how to compute the solution path of the Riemann problem (13), (14) knowing the values $\hat{s}_\ell$, $\hat{s}_\ell^K$, $\ell = 1, \cdots, k+1$. In the rest of this section, it will be discussed how to compute these values; this computation is referred to as the forward sweep. We will restrict ourselves to giving an algorithm for the forward sweep here; the justification of the construction is given in [13].

The values $\hat{s}_\ell$ are computed recursively. Let $\hat{s}_1 = s^L$. Assume the values $\hat{s}_j$, $j = 1, \cdots, \ell$, are known and that the leading index in the ℓth strip is given by i, and that the indices involved in the wave are given by J. Then the computation of $\hat{s}_{\ell+1}$ is divided into four cases:

Case I: ℓ-rarefaction and $\hat{s}_\ell < s_\ell^T(c^\ell)$,
Case II: ℓ-rarefaction and $\hat{s}_\ell \geqq s_\ell^T(c^\ell)$,
Case III: ℓ-shock and $\hat{s}_\ell \leqq s_\ell^T(c^\ell)$,
Case IV: ℓ-shock and $\hat{s}_\ell > s_\ell^T(c^\ell)$.

Recall that the phrase "ℓ-rarefaction" means that the ℓth wave in the solution of (27) represented by (30) is a rarefaction and that "ℓ-shock" means that the ℓth wave is a shock.

Case I. Compute the c-rarefaction curve $c(s)$ given by (18), where i and J are known from the solution of the associated one-phase problem, and where the initial data are $s = \hat{s}_\ell$ and $c(\hat{s}_\ell) = c^\ell$. If there is an $\bar{s} \in [\hat{s}_\ell, 1]$ such that $c(\bar{s}) = c^{\ell+1}$ then let $\hat{s}_{\ell+1} = \bar{s}$, if not let $\hat{s}_{\ell+1} = s_\ell^T(c^{\ell+1})$.

Case II. Let $\hat{s}_{\ell+1} = s_\ell^T(c^{\ell+1})$.

Case III. Let $\hat{s}_{\ell+1} \leqq s_\ell^T(c^{\ell+1})$ be the value such that

$$(\hat{s}_\ell, c^\ell) \xrightarrow{cS} (\hat{s}_{\ell+1}, c^{\ell+1}),$$

where cS means that the wave is a c-shock. The value $\hat{s}_{\ell+1}$ is uniquely determined by the Rankine–Hugoniot condition (25) and the condition $\hat{s}_{\ell+1} \leqq s_\ell^T(c^{\ell+1})$.

Case IV. Let $\hat{s}_{\ell+1} \leqq s_\ell^T(c^{\ell+1})$ be the value such that

$$(s_\ell^T(c^\ell), c^\ell) \xrightarrow{cS} (\hat{s}_{\ell+1}, c^{\ell+1}).$$

Again the value $\hat{s}_{\ell+1}$ is uniquely determined by the Rankine–Hugoniot condition (25) and the condition $\hat{s}_{\ell+1} \leqq s_\ell^T(c^{\ell+1})$.

The four cases are illustrated in Figs. 9–12.

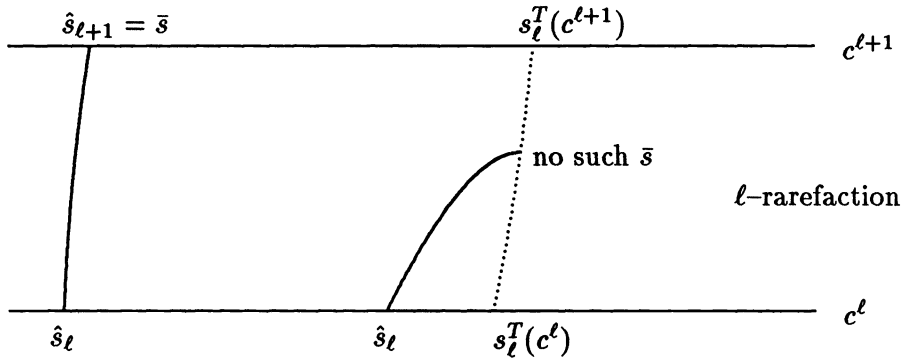


FIG. 9. *Case* I.

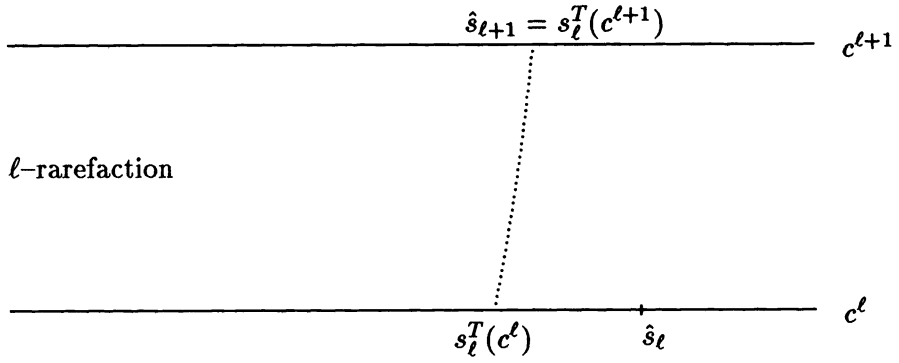


FIG. 10. *Case* II.

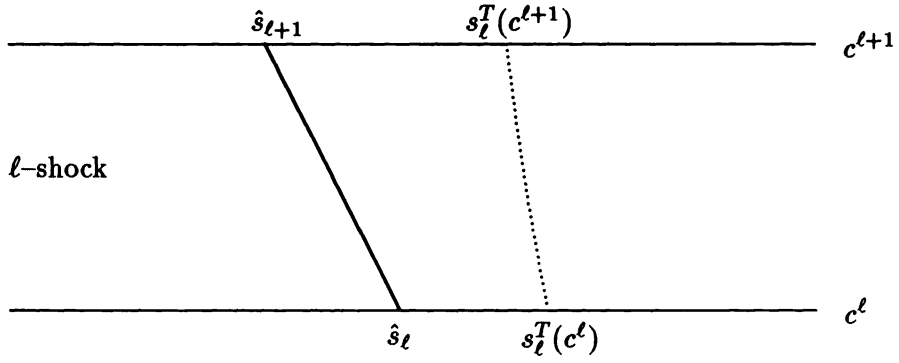


FIG. 11. *Case* III.

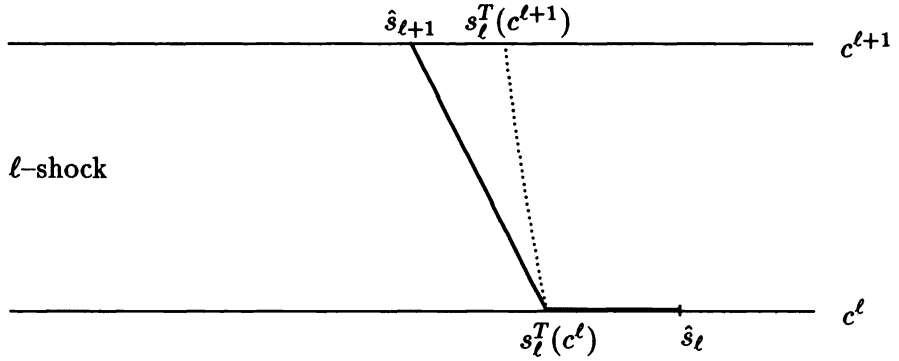


FIG. 12. *Case* IV.

When $\hat{s}_\ell$, $\ell = 1, \cdots, k+1$ are computed, the corresponding critical values are given by $\hat{s}_\ell^K = s_{\ell-1}^K(\hat{s}_\ell, c^\ell)$, $\ell = 2, \cdots, k+1$, and we define $\hat{s}_1^K = \infty$. From the construction it follows that

$$\hat{s}_\ell \leqq s_{\ell-1}^T(c^\ell) \leqq \hat{s}_\ell^K, \qquad \ell = 2, \cdots, k+1.$$

Observe that in Case I an ordinary differential equation must be integrated. In the implementation the fourth-order Runge-Kutta method was used. The curve $c(s)$ is stored since it might be a part of the solution path. In Case III and in Case IV a nonlinear algebraic equation must be solved; this is done by a version of Newton's method. Both the integration of the ordinary differential equation and the solution of the algebraic equation introduce some numerical errors. These errors can be made as small as we want just by using a small enough step length in the Runge-Kutta procedure and by performing enough Newton iterations.

5.5. The backward sweep. In this section we will sketch an algorithm to compute the solution of the Riemann problem (13), (14) based on the sequence of points $(\hat{s}_\ell, \hat{s}_\ell^K)$, $\ell = 1, \cdots, k+1$. That is, we want to find a path in the state space connecting $u^L = (s^L, c^L)$ and $u^R = (s^R, c^R)$. Throughout this section we assume that the solution of the associated one-phase problem is given by (30) and that the sequence $(\hat{s}_\ell, \hat{s}_\ell^K)$, $\ell = 1, \cdots, k+1$, is computed.

Let s^ℓ, $\ell = 2, \cdots, k+1$, be the s-values for which the solution path enters the values $c = c^\ell$ from below in the state space. Consistent with this definition, we can define $s^1 = s^L$. From Corollary 1 a characterization of these points follows directly.

COROLLARY 2. *The values* $(s^\ell)_{\ell=1}^{k+1}$ *satisfy*

$$s^\ell = \hat{s}_\ell \quad \text{or} \quad s^\ell > \hat{s}_\ell^K.$$

The computation of the solution path starts at the right state $u^R = (s^R, c^R)$ and goes backwards ending up at the left state $u^L = (s^L, c^L)$. We refer to this part of the algorithm as the backward sweep.

To start the computation, we define

$$s^{k+1} = \begin{cases} s^R, & s^R > \hat{s}_{k+1}^K, \\ \hat{s}_{k+1}, & s^R \leqq \hat{s}_{k+1}^K. \end{cases}$$

The last wave in the solution path is

$$(s^{k+1}, c^{k+1}) \xrightarrow{s} (s^R, c^{k+1}),$$

which is empty if and only if $s^{k+1} = s^R$. Suppose the solution path from $(s^{\ell+1}, c^{\ell+1})$ for $\ell \leqq k$ to u^R is computed. The induction step in the algorithm is then to find the value s^ℓ, and to extend the solution path from $(s^{\ell+1}, c^{\ell+1})$ backward to (s^ℓ, c^ℓ). Knowing s^ℓ, the remaining problem is essentially a two-dimensional Riemann problem with left state (s^ℓ, c^ℓ) and right state $(s^{\ell+1}, c^{\ell+1})$. This kind of problem was solved in [12].

Assume that i is the leading index in the ℓth strip of the state space and that the indices involved in this strip are given by the index set J.

Rarefaction. Assume first that the ℓth strip is a rarefaction strip. Then there are two possible cases:

Case I: $s^{\ell+1} = \hat{s}_{\ell+1} < \hat{s}^K_{\ell+1}$,

Case II: $s^{\ell+1} > \hat{s}^K_{\ell+1}$ or $s^{\ell+1} = \hat{s}_{\ell+1} = \hat{s}^K_{\ell+1}$.

Here Case I is particularly simple.

Case I. By the construction of $\hat{s}_{\ell+1}$, there is a c-rarefaction curve connecting $(\hat{s}_\ell, c^\ell)$ and $(\hat{s}_{\ell+1}, c^{\ell+1})$. Let $s^\ell = \hat{s}_\ell$ and extend the solution path by

$$(s^\ell, c^\ell) \xrightarrow{cR} (s^{\ell+1}, c^{\ell+1})$$

where cR means that the wave is a c-rarefaction. This solution is illustrated in Fig. 13.

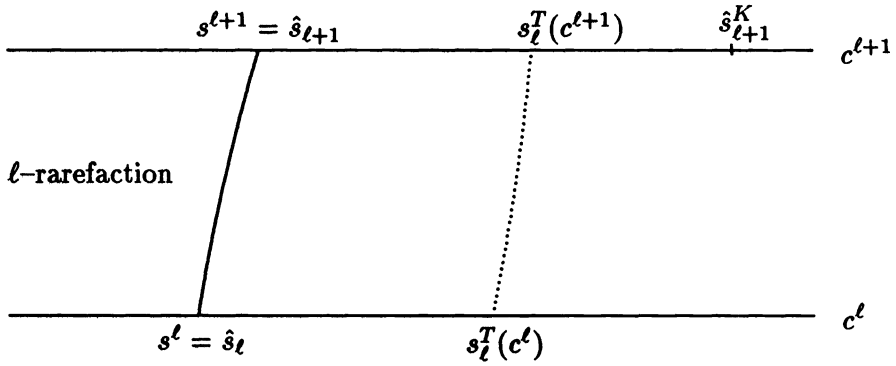


FIG. 13. *Case* I, *rarefaction.*

In Case II we start by computing the c-rarefaction curve $c(s)$ given by (18) with initial data $s = s^{\ell+1}$ and $c(s^{\ell+1}) = c^{\ell+1}$, for increasing values of s. Let $\bar{s} \in [s^{\ell+1}, 1]$ be the value such that $c(\bar{s}) = c^\ell$ (cf. Fig. 14). Then Case II can be divided into two subcases.

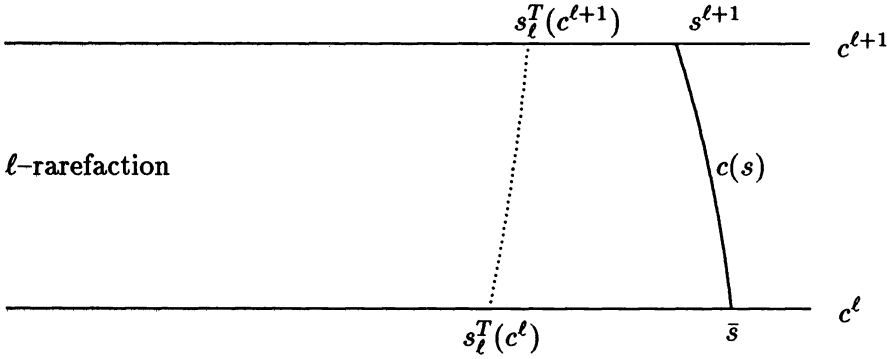


FIG. 14. *Definition of* $\bar{s}$ *for Case* II.

Case II(a). $\bar{s} > \hat{s}_\ell^K$. Let $s^\ell = \bar{s}$ and extend the solution by

$$(s_\ell, c^\ell) \xrightarrow{cR} (s^{\ell+1}, c^{\ell+1}).$$

The next subcase is the most complicated case; the solution in the ℓ-strip can be a composition of three elementary waves.

Case II(b). $\bar{s} \leqq \hat{s}_\ell^K$. Let $s^\ell = \hat{s}_\ell$. If $\bar{s} \geqq s_\ell^K(s^\ell, c^\ell)$ or if $s^\ell \geqq s_\ell^T(c^\ell)$ then the solution path is extended by

$$(s^\ell, c^\ell) \xrightarrow{s} (\bar{s}, c^\ell) \xrightarrow{cR} (s^{\ell+1}, c^{\ell+1})$$

(cf. Fig. 15). If not, then $\bar{s} < s_\ell^K(s^\ell, c^\ell)$, and there exist two s-values s^1, s^2 and a c-value

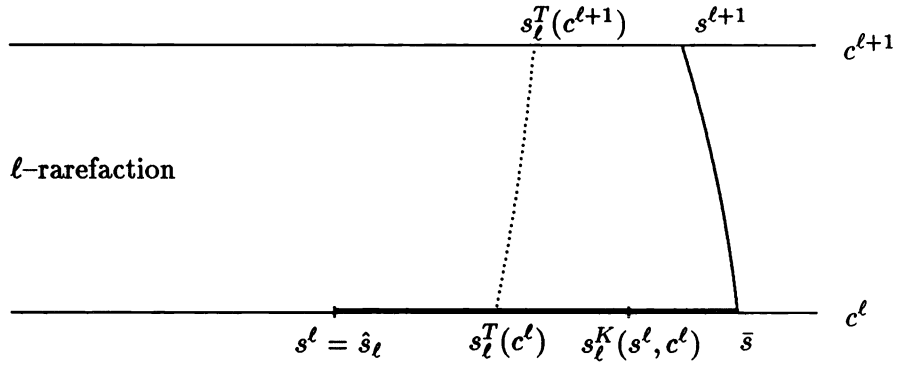


FIG. 15. *Case* II(b), $\bar{s} > s_\ell^K(s^\ell, c^\ell)$.

$\tilde{c}$ such that $(s^1, \tilde{c})$ lies on the c-rarefaction curve going up from (s^ℓ, c^ℓ) and $(s^2, \tilde{c})$ lies on the c-rarefaction curve going down from $(s^{\ell+1}, c^{\ell+1})$ and such that $s^2 = s_\ell^K(s^1, \tilde{c})$ (cf. Fig. 16). The solution path is extended by

$$(s^\ell, c^\ell) \xrightarrow{cR} (s^1, \tilde{c}) \xrightarrow{s} (s^2, \tilde{c}) \xrightarrow{cR} (s^{\ell+1}, c^{\ell+1}).$$

Shock. Assume next that the ℓ-strip is a shock strip. Again there are two cases:
Case I: $s^{\ell+1} = \hat{s}_{\ell+1}$,
Case II: $s^{\ell+1} > \hat{s}_{\ell+1}^K$.

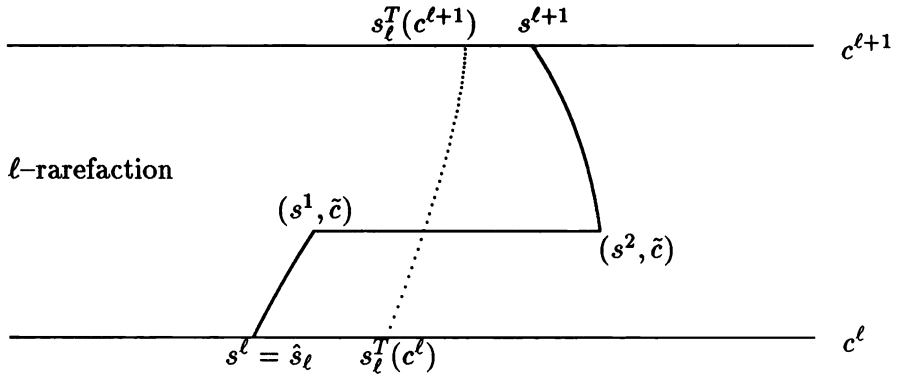


FIG. 16. *Case* II(b), $\bar{s} > s_\ell^K(s^\ell, c^\ell)$ *and* $s^\ell < s_\ell^T(c^\ell)$.

Both cases are divided into two subcases.

Case I(a). $\hat{s}_\ell \leqq s_\ell^T(c^\ell)$. Let $s^\ell = \hat{s}_\ell$ and extend the solution by

$$(s^\ell, c^\ell) \xrightarrow{cS} (s^{\ell+1}, c^{\ell+1})$$

where cS means that the wave is a c-shock. This solution is depicted in Fig. 17.

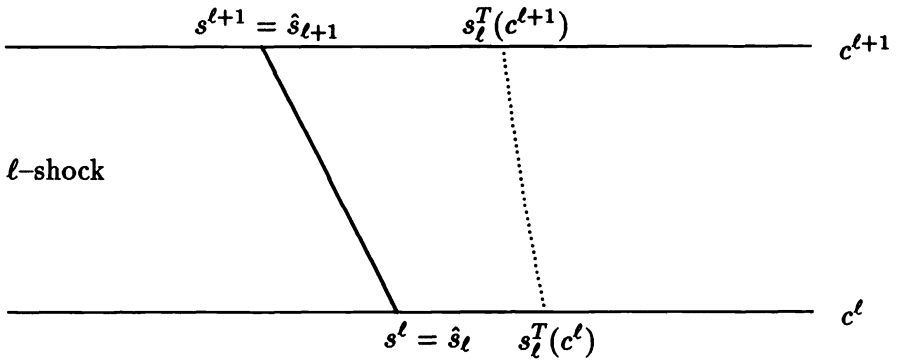


FIG. 17. *Case* I(a).

Case I(b). $\hat{s}_\ell > s_\ell^T(c^\ell)$. Let $s^\ell = \hat{s}_\ell$ and extend the solution by

$$(s^\ell, c^\ell) \xrightarrow{s} (s_\ell^T(c^\ell), c^\ell) \xrightarrow{cS} (s^{\ell+1}, c^{\ell+1})$$

(cf. Fig. 18).

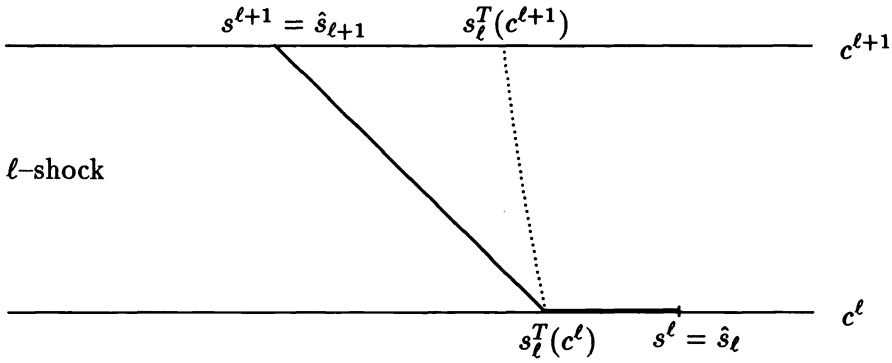


FIG. 18. *Case* I(b).

In Case II we start by computing the value $\bar{s} \in [s^{\ell+1}, 1]$ such that

$$(\bar{s}, c^\ell) \xrightarrow{cS} (s^{\ell+1}, c^{\ell+1})$$

is c-shock satisfying the Rankine-Hugoniot condition (25). Then there are two cases.

Case II(a). $\bar{s} > \hat{s}_\ell^K$. Let $s^\ell = \bar{s}$ and extend the solution by

$$(s^\ell, c^\ell) \xrightarrow{cS} (s^{\ell+1}, c^{\ell+1})$$

(cf. Fig. 19).

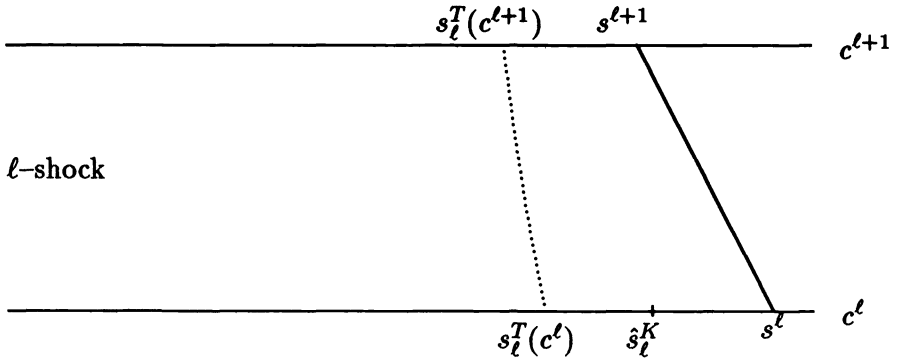


FIG. 19. *Case* II(a).

Case II(b). $\bar{s} \leqq \hat{s}_\ell^K$. Let $s^\ell = \hat{s}_\ell$ and extend the solution by

$$(s^\ell, c^\ell) \xrightarrow{s} (\bar{s}, c^\ell) \xrightarrow{cS} (s^{\ell+1}, c^{\ell+1})$$

(cf. Fig. 20). □

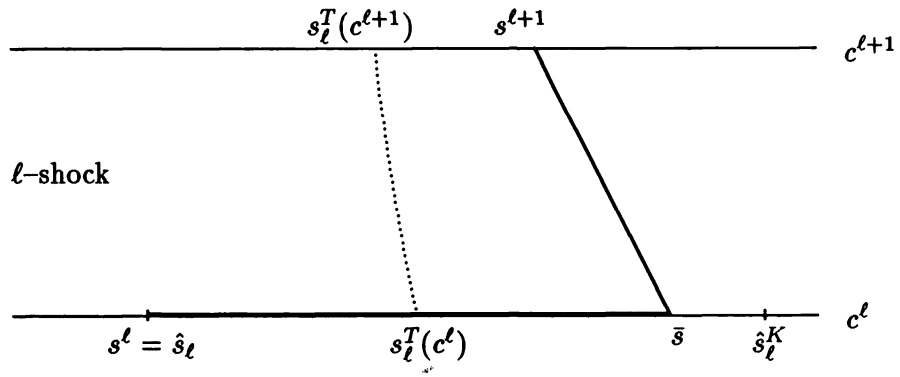


FIG. 20. *Case* II(b).

Finally we remark that since $\hat{s}_1^K$ was defined to be infinity, it follows from the construction that $s^1 = s^L$. Hence, after k induction steps, the Riemann problem (13), (14) is solved.

6. Examples. In this section we will give some examples of solutions given by the Riemann solver described in the previous section. We will start by giving some examples for $n = 4$ c-components. In these examples we have used the fractional flow function

$$f(s, c_1, c_2, c_3, c_4) = \frac{s^2}{s^2 + (1-s)^2(\frac{1}{2} + \sum_{i=1}^4 c_i)} \tag{35}$$

and the adsorption functions

$$a_i(c_i) = \frac{c_i}{5(i + c_i)}, \qquad i = 1, \cdots, 4. \tag{36}$$

We refer to [12] and [11] for examples with $n = 1$.

Example 1. Let $s^L = c_i^L = 1$, $s^R = c_i^R = 0$, $i = 1, \cdots, 4$. This Riemann problem models a situation where the oil saturation initially is 100 percent, and water containing

polymer is injected to displace the oil. The solution of the problem as a function of x for $t = 1$ is given in Fig. 21 for the water saturation, and in Fig. 22 for the c-components.

In Fig. 23 the solution path in the state space is graphed. The solution starts in the lower right corner at "L" and ends up in the upper left corner at "R." The solution consists of an s-rarefaction wave, four c-shocks and an s-shock separated by some constant states. The letter "S" and the number "1" on the left side of the first strip indicate that this is a c-shock where only c-component number 1 is involved. When the letter is "R," it denotes a c-rarefaction strip. The $\hat{s}_\ell$-points (cf. § 5.4) are labelled $\hat{}$ in the figure, and so are the $\hat{s}_\ell^K$-points. Recall that the $\hat{s}_\ell$-points always are located to the left of the transition curve, labelled "T," and that the $\hat{s}_\ell^K$-points are located to the right of the transition curve.

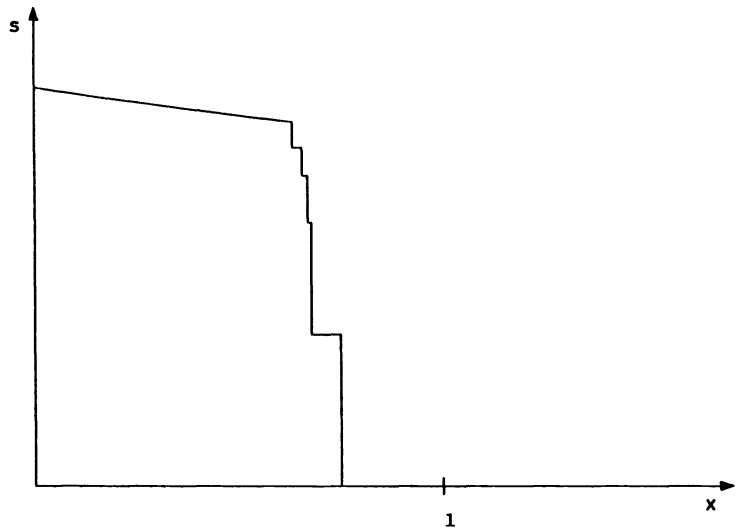


FIG. 21. *The s-component of the solution in Example* 1.

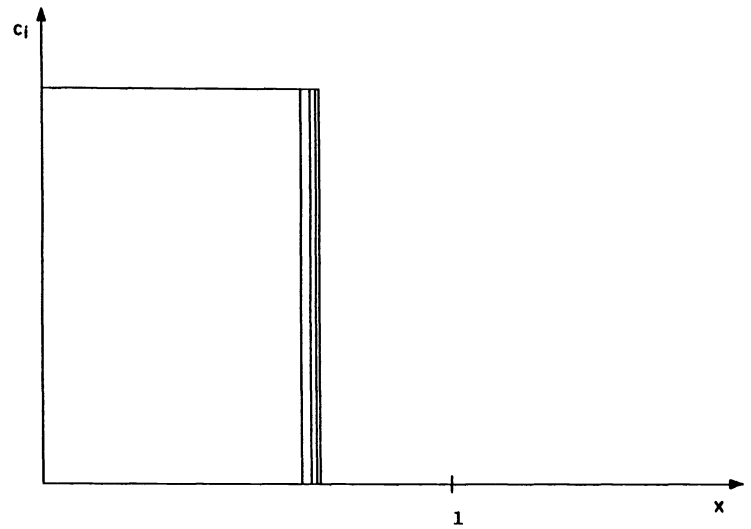


FIG. 22. *The c_i-components of the solution in Example* 1.

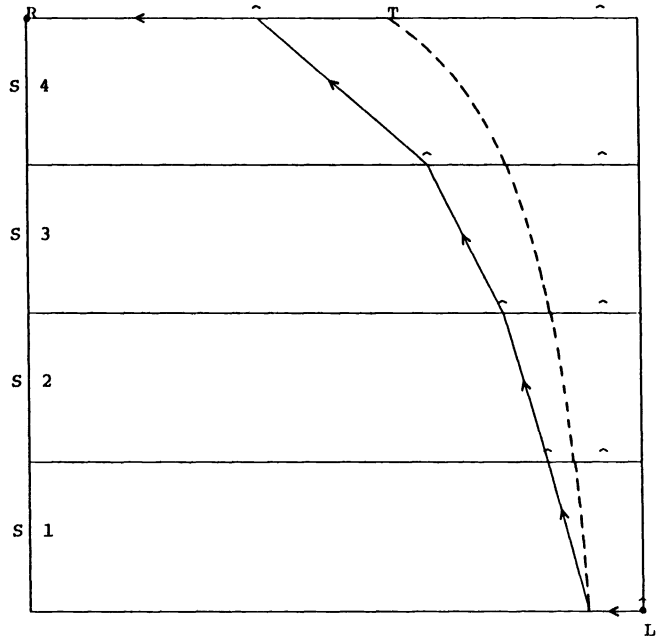


FIG. 23. *The solution path in the state space for Example* 1.

Example 2. Let $s^L = 0.9$, $c_i^L = 1$, $c_i^R = 0$, $i = 1, \cdots, 4$. By varying the value of s^R, we will illustrate that the solution does not depend continuously on the initial data in the L^∞-norm. First let $s^R = 0.96$, the solution path in the state space is given in Fig. 24, and the solution for s as a function of x for $t = 1$ is given in Fig. 26. Observe that

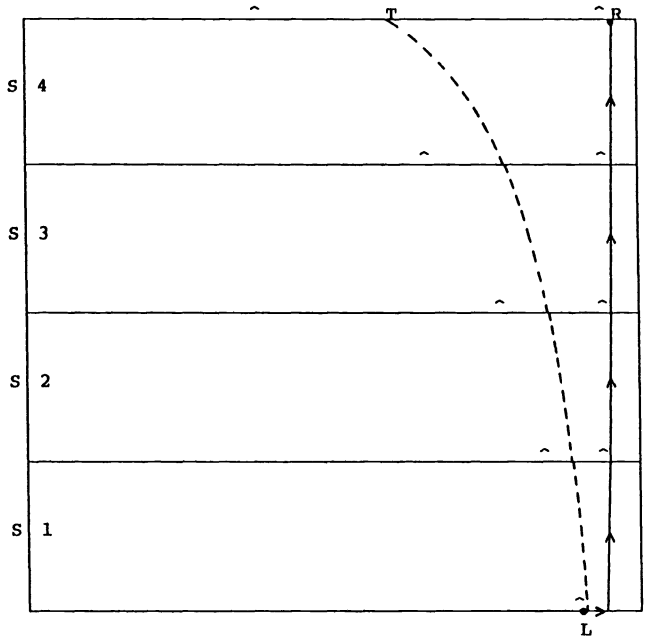


FIG. 24. *The solution path in the state space for Example* 2 *for* $s^R = 0.96$.

the value s^R is greater than $\hat{s}_5^K$, and that almost the entire solution path is located to the right of the transition curve.

Let $s^R = 0.9$, the solution path is given in Fig. 25 and the s-component of the solution is given in Fig. 29. The value s^R is now smaller than $\hat{s}_5^K$ and almost the entire solution path is located to the left of the transition curve.

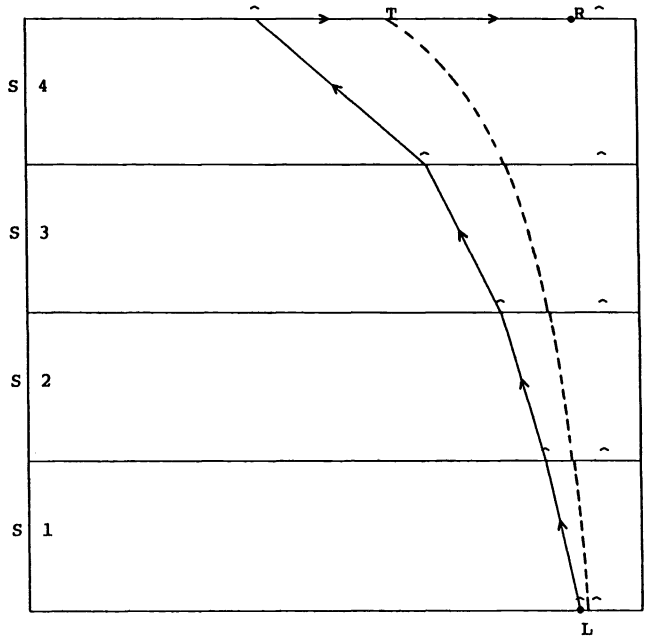


FIG. 25. *The solution path in the state space for Example 2 for* $s^R = 0.9$.

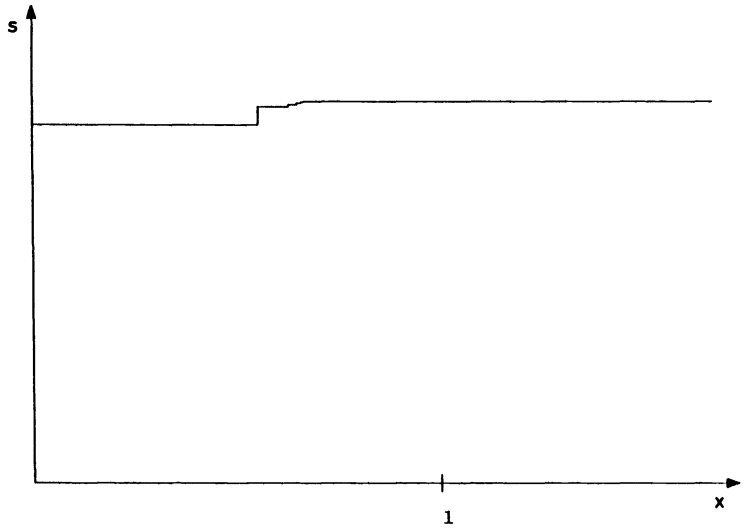


FIG. 26. *The s-component of the solution in Example 2 for* $s^R = 0.96$.

The graphs of $s = s(x, 1)$ for $s^R = 0.94$ and $s^R = 0.92$ are given in Figs. 27 and 28, respectively. By comparing Figs. 26, 27, 28, and 29 we observe that a small change in the value of s^R drastically changes the behaviour of the solution. This example illustrates that the solution does not depend continuously on the initial data in the L^∞-norm. The solution might, however, depend continuously on the data in L^1_{loc}. For a further discussion of this problem we refer to Temple [20], where it is argued that the only L^p-norm for which it is reasonable to obtain continuous dependence results is the L^1-norm.

Example 3. Recall that the solution of a Riemann problem for a scalar conservation law is a monotone function of x/t. This regularity is not carried over to systems as

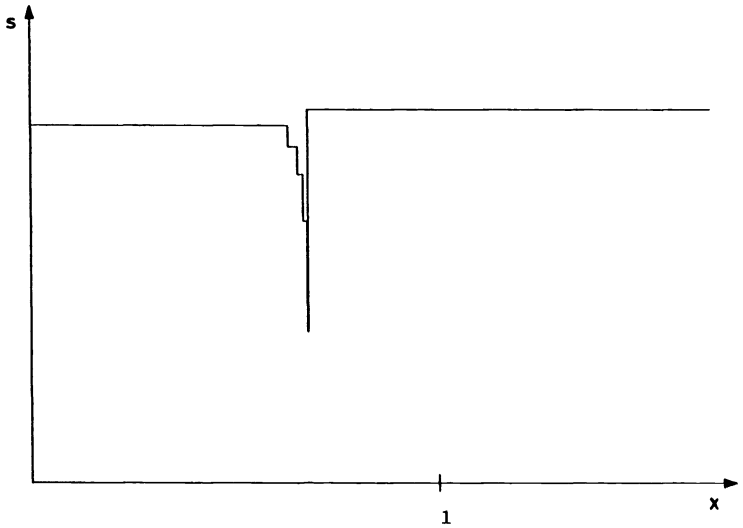


FIG. 27. *The s-component of the solution in Example* 2 *for* $s^R = 0.94$.

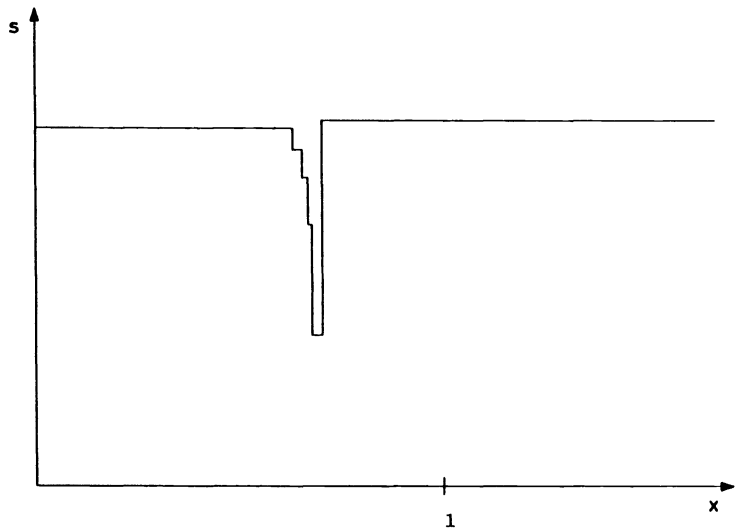


FIG. 28. *The s-component of the solution in Example* 2 *for* $s^R = 0.92$.

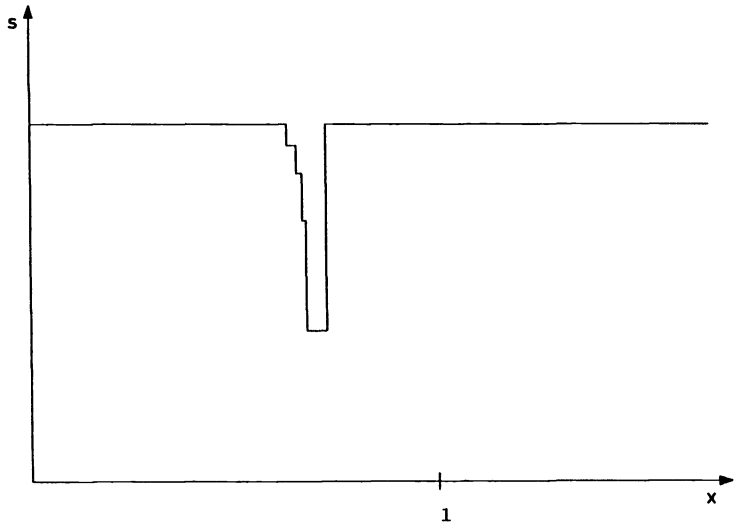


FIG. 29. *The s-component of the solution in Example 2 for* $s^R = 0.9$.

this example will illustrate. Let

$$\begin{aligned} s^L &= 0.9, & s^R &= 0.3, \\ c_1^L &= 1, & c_1^R &= 0, \\ c_2^L &= 0, & c_2^R &= 1, \\ c_3^L &= 1, & c_3^R &= 0, \\ c_4^L &= 0, & c_4^R &= 0.77. \end{aligned}$$

The s-component of the solution of this Riemann problem is given in Fig. 30, and the solution path in the state space is graphed in Fig. 31. This example shows how difficult it is to develop reliable and accurate numerical methods for these equations.

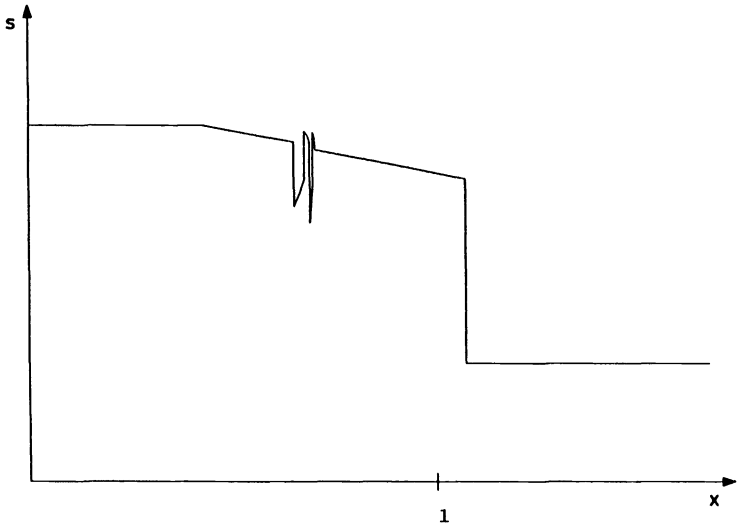


FIG. 30. *The s-component of the solution in Example* 3.

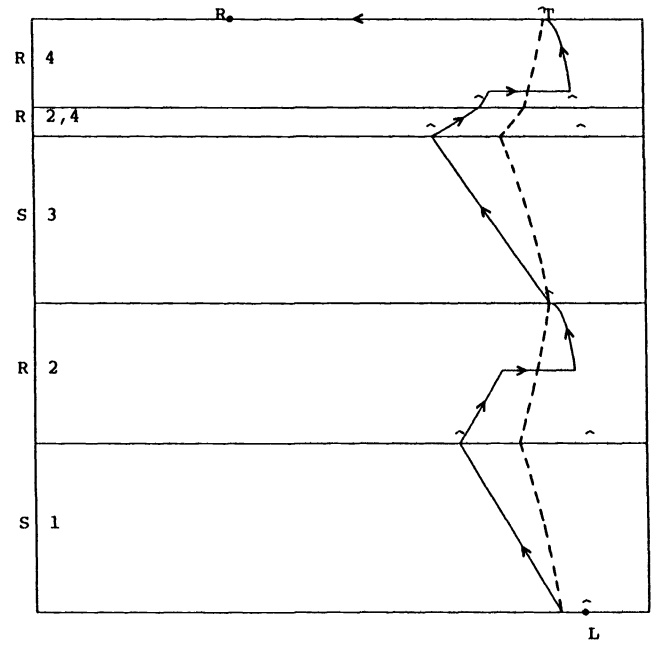


FIG. 31. *The solution path in the state space for Example* 3.

Example 4. As we mentioned in the Introduction, the number of polymer components can be quite large. Actually, the polymer consists of millions of different types of molecules. The polymer should therefore be represented by a molecular weight distribution. Let ν be the molecular weight, which we assume to be scaled such that $\nu \in [0, 1]$, and let the polymer at some point (x, t) be specified by the molecular weight distribution $\Psi = \Psi(\nu)$ (cf. Fig. 32). The concentration of the polymer having molecular weight between ν_0 and ν_1 is then $\int_{\nu_0}^{\nu_1} \Psi(\nu)\, d\nu$ at the point (x, t).

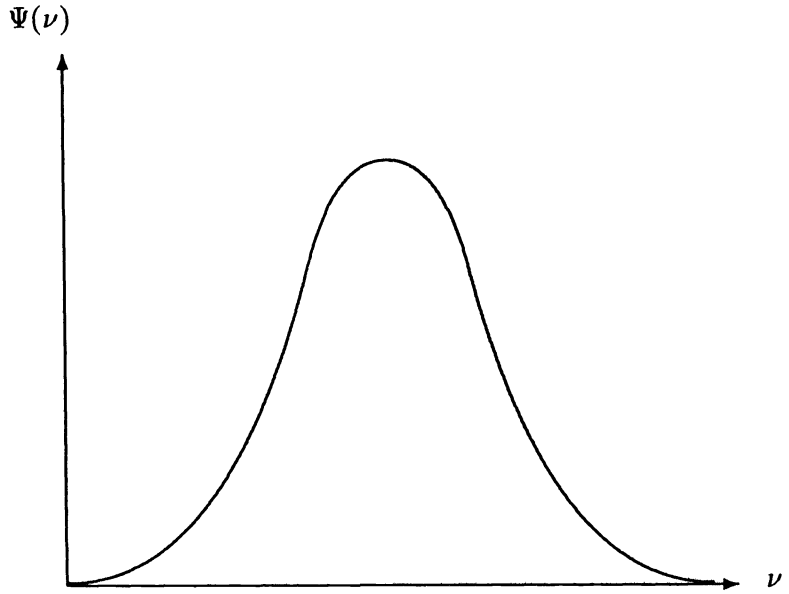


FIG. 32. *An example of a molecular weight distribution* $\Psi = \Psi(\nu)$.

The molecular weight distribution and the saturation of the aqueous phase are assumed to be known for $t=0$, so the problem is to compute them for $t>0$. This problem can be approximated by discretizing the molecular weight function into n components (cf. Fig. 33). These n components are then mathematical components,

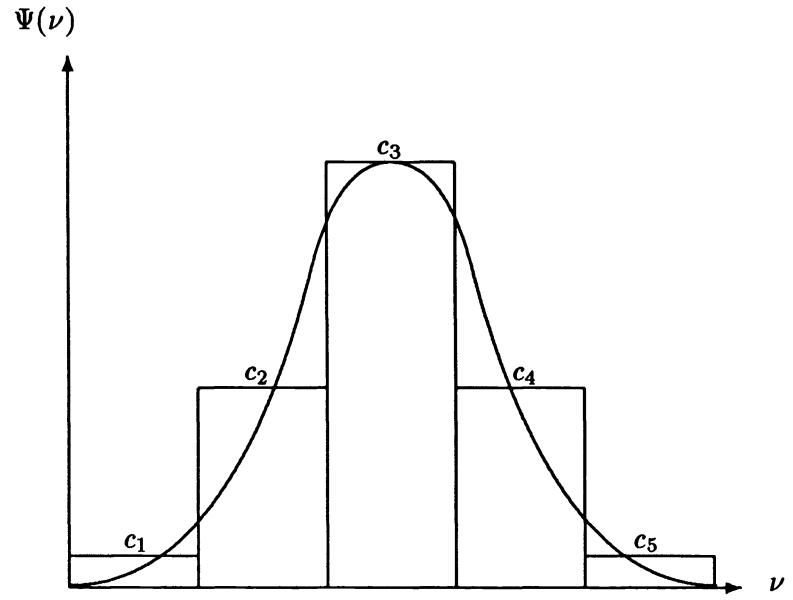


FIG. 33. *The partition of the molecular weight distribution* $\Psi=\Psi(\nu)$ *into* $n=5$ *components.*

rather than physical polymer components. Let $c_i=c_i(x,t)$ denote an approximation to Ψ on the interval $[(i-1)/n, i/n]$, $i=1,\cdots,n$ at the point (x,t). Observe that $c_i(x,t)/n$ is an approximation to the concentration of the polymer having molecular weight in the interval $[(i-1)/n, i/n]$ at the point (x,t), i.e., the c_i components approximate the molecular weight distribution, and not the concentrations. Having done this discretization, we are back to the system (1) of conservation laws.

In this example, we will study the effect of this discretization. Recall that the process often is approximated using $n=1$. We shall see in this example that this is a rather poor approximation.

The Riemann problem is specified by giving s^L, s^R, Ψ^L, and Ψ^R where $\Psi^L(\Psi^R)$ is the initial molecular weight distribution for $x<0$ $(x>0)$. Let

$$s^L=1, \qquad s^R=0,$$
$$\Psi^L(\nu)=16\nu e^{-8\nu^2}, \qquad \Psi^R(\nu)=0.$$

The initial conditions for the components are defined by a midpoint approximation, i.e.,

$$c_i^L=\Psi^L\left(\frac{i-\frac{1}{2}}{n}\right), \quad c_i^R=\Psi^R\left(\frac{i-\frac{1}{2}}{n}\right)=0, \quad i=1,\cdots,n.$$

In this example we used the fractional flow function

$$f(s,c_1,\cdots,c_n)=\frac{s^2}{s^2+(1-s)^2(\frac{1}{2}+1/n\sum_{i=1}^n c_i)} \tag{37}$$

and the adsorption functions

$$a_i(c_i)=10c_i\Big/\left(1+10\left(\frac{n}{n+i-\frac{1}{2}}\right)c_i\right), \qquad i=1,\cdots,n. \tag{38}$$

The s-component of the solution of this Riemann problem for $n = 1, 5, 20, 50, 100$ is shown in Figs. 34, 35, 36, 37, 38, respectively. The solution is composed of an s-rarefaction, n c-shocks, an s-rarefaction, and an s-shock.

We observe that as n increases the c-shocks seem to converge towards a continuous function.

The polymer model (13) is often simplified by choosing $n = 1$ (cf. [12]). This example shows that the solution for $n = 1$ is a rather poor approximation of the solution for $n = 100$. This suggests that the number of c-components should be relatively large.

7. A finite-difference method. The Riemann solver enables us to compare the solution generated by a standard numerical method with the exact solution for Riemann problems. In this section we will study the behaviour of the classical first-order Godunov method.

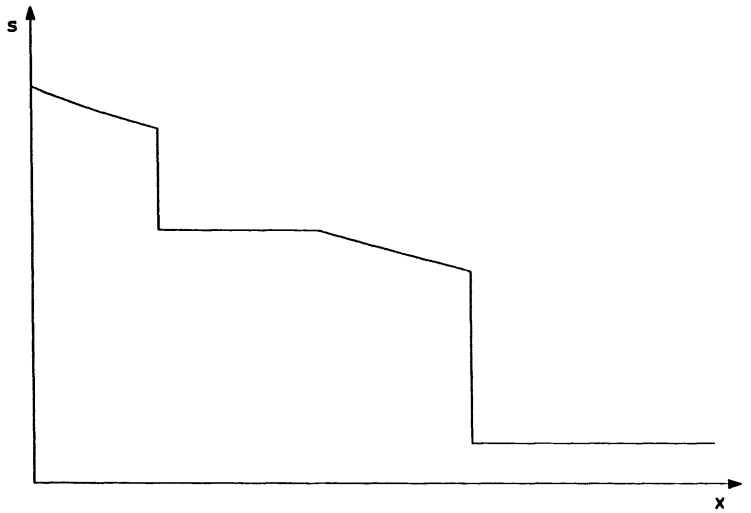


FIG. 34. *The s-component of the solution in Example* 4 *for* $n = 1$.

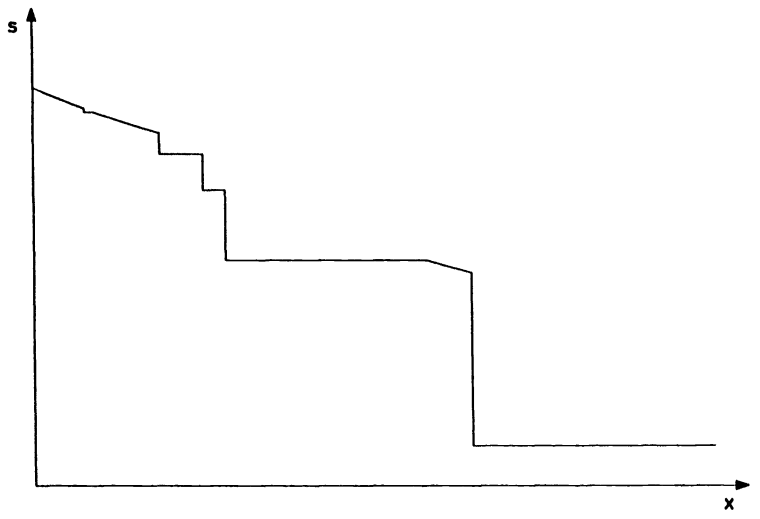


FIG. 35. *The s-component of the solution in Example* 4 *for* $n = 5$.

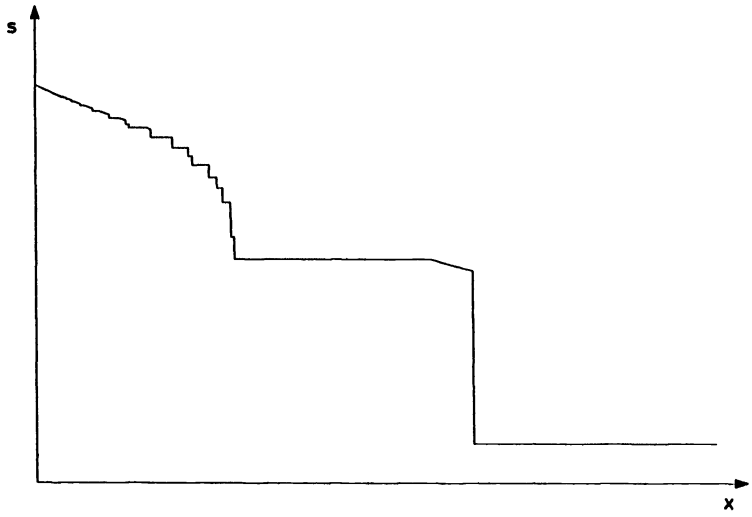


FIG. 36. *The s-component of the solution in Example* 4 *for* $n = 20$.

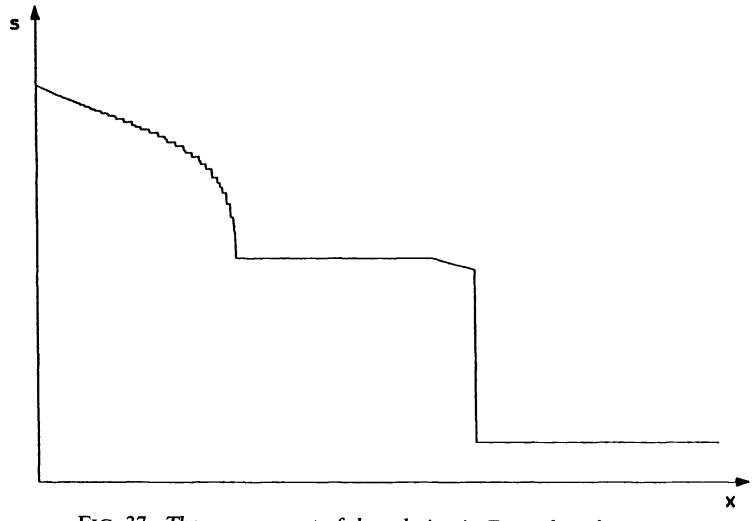


FIG. 37. *The s-component of the solution in Example* 4 *for* $n = 50$.

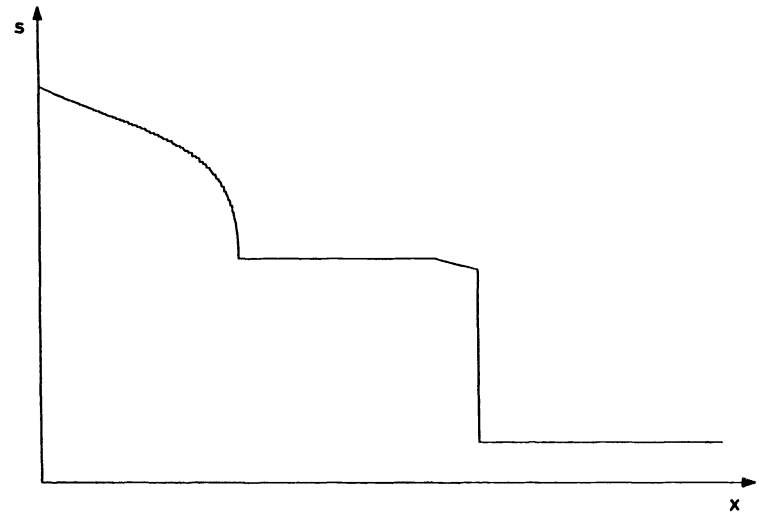


FIG. 38. *The s-component of the solution in Example* 4 *for* $n = 100$.

Define the variable $b_i = sc_i + a_i(c_i)$, $i = 1, \cdots, n$; then the system (13) can be written in the form

$$
\begin{aligned}
&s_t + f(s, c_1, \cdots, c_n)_x = 0, \\
&(b_i)_t + (c_i f(s, c_1, \cdots, c_n))_x = 0, \qquad i = 1, \cdots, n.
\end{aligned} \tag{39}
$$

Denote by s_j^k an approximation to $s(j\Delta x, k\Delta t)$, where Δx and Δt are fixed mesh sizes. Similarly, $c_{i,j}^k$ and $b_{i,j}^k$ denote approximations to $c_i(j\Delta x, k\Delta t)$ and $b_i(j\Delta x, k\Delta t)$, respectively. Furthermore, we define $f_j^k = f(s_j^k, c_{1,j}^k, \cdots, c_{n,j}^k)$. Then the classical Godunov scheme for (39) can be written in the form

$$
s_j^{k+1} = s_j^k - \frac{\Delta t}{\Delta x}(f_j^k - f_{j-1}^k),
$$

$$
b_{i,j}^{k+1} = b_{i,j}^k - \frac{\Delta t}{\Delta x}(c_{i,j}^k f_j^k - c_{i,j-1}^k f_{j-1}^k), \qquad i = 1, \cdots, n,
$$

$c_{i,j}^{k+1}$ solves the equation $s_j^{k+1} c_{i,j}^{k+1} + a_i(c_{i,j}^{k+1}) = b_{i,j}^{k+1}$, $i = 1, \cdots, n$.

Let $n = 4$ and let f and a_i, $i = 1, \cdots, 4$ be as defined in (35) and (36), respectively. Let $s^L = 0.9$, $s^R = 0.92$, $c_i^L = 1$, $c_i^R = 0$, $i = 1, \cdots, 4$. The exact solution of this problem is given in Fig. 28. To study the convergence of the Godunov method for this problem, we compute the numerical solution for some mesh parameters Δx and Δt, which in all cases satisfies the CFL condition, i.e.,

$$
\frac{\Delta t}{\Delta x} \max(\lambda^s, \lambda^1, \cdots, \lambda^n) \leqq 1
$$

for all $(s, c_1, \cdots, c_n) \in \mathbf{I}^{n+1}$. For all the mesh parameters the numerical and the exact solution are graphed for $t = 1$. Only the s and the c_1 component of the solution are shown; the other c-components behave similarly to c_1. In Figs. 39 and 43 the solution for $\Delta x = 0.06$ and $\Delta t = 0.0227$ are shown. We observe that this mesh size is too coarse to reflect the details in the exact solution of the s-component, and that the shock in the c-component is smeared out. When the mesh size is reduced (cf. Figs. 40, 41, 42 for the s-component and Figs. 44, 45, 46 for the c-component), the solution converges to the exact solution, at least in L^1. But we have to use a very fine mesh to discover the qualitative behaviour of the exact solution by studying the solution generated by the Godunov method. The finest grid used here involved nearly 12 million grid points to calculate the solution for $0 \leqq x \leqq 2$, $0 \leqq t \leqq 1$. This implies that for simulations in

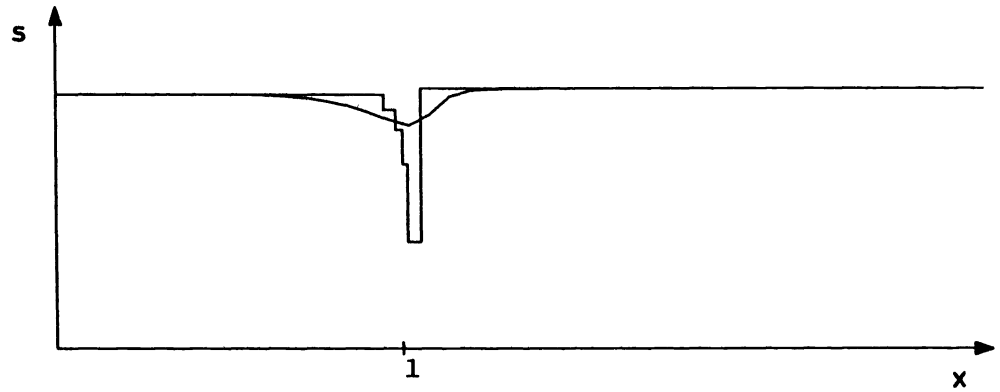


FIG. 39. *The s-component of the solution for* $\Delta x = 0.06$, $\Delta t = 0.0227$.

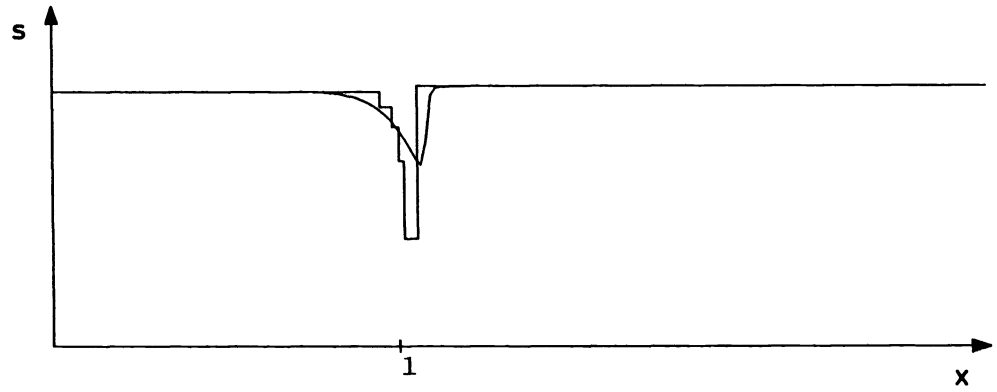


FIG. 40. *The s-component of the solution for* $\Delta x = 0.015$, $\Delta t = 0.0056$.

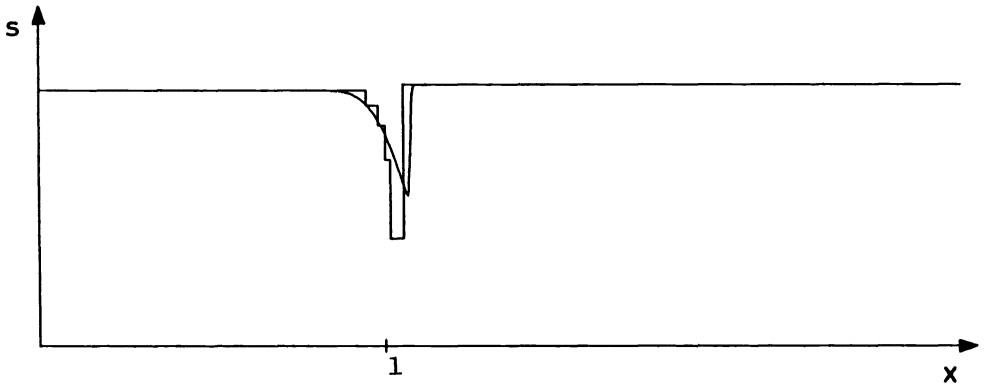


FIG. 41. *The s-component of the solution for* $\Delta x = 0.006$, $\Delta t = 0.0022$.

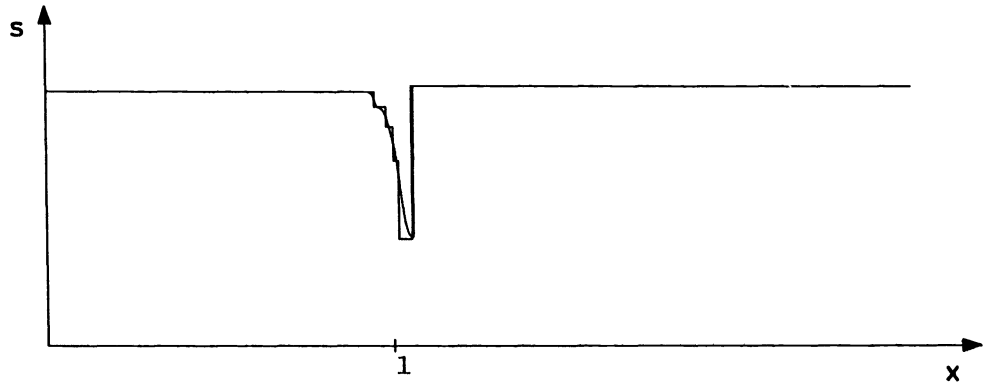


FIG. 42. *The s-component of the solution for* $\Delta x = 0.00067$, $\Delta t = 0.00025$.

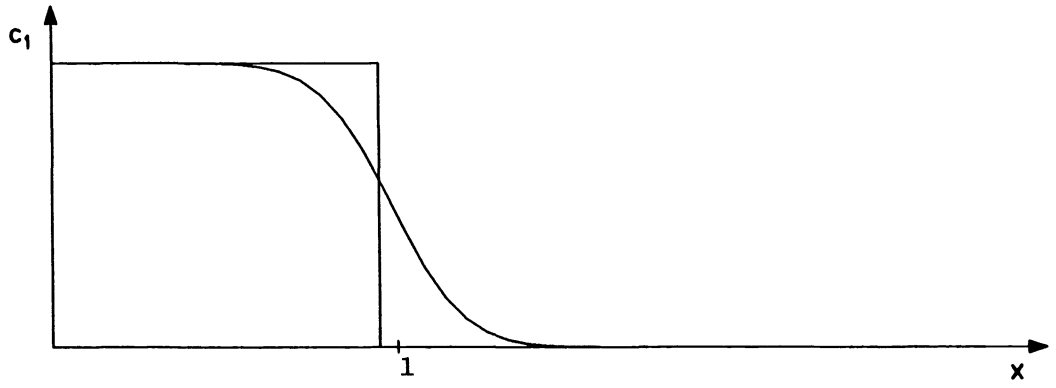


FIG. 43. *The c_1-component of the solution for $\Delta x = 0.06$, $\Delta t = 0.0227$.*

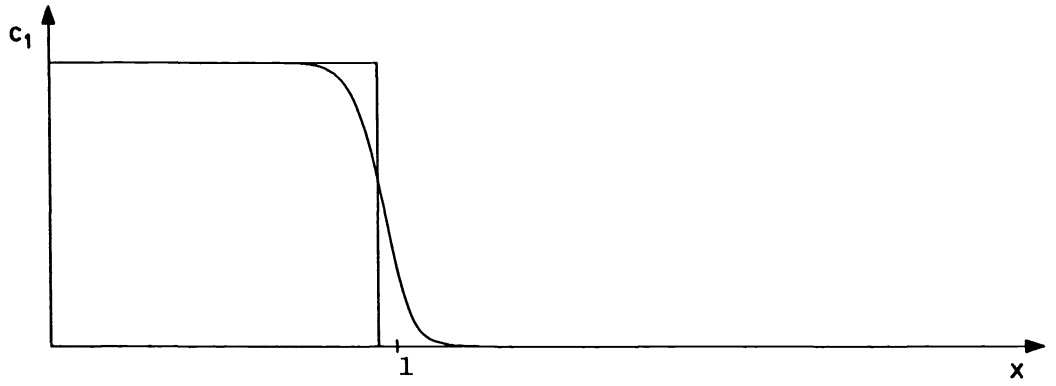


FIG. 44. *The c_1-component of the solution for $\Delta x = 0.015$, $\Delta t = 0.0056$.*

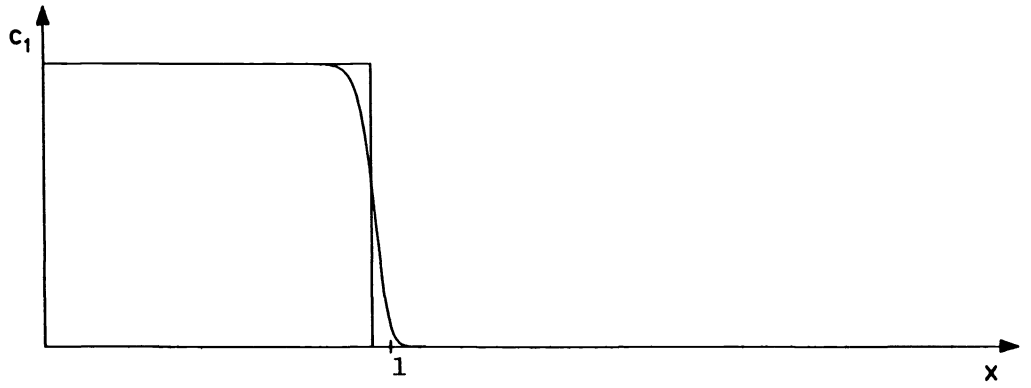


FIG. 45. *The c_1-component of the solution for $\Delta x = 0.006$, $\Delta t = 0.0022$.*

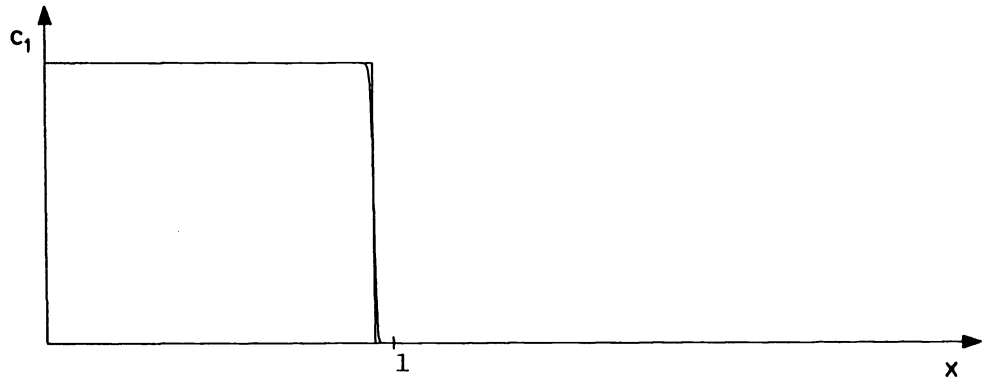


FIG. 46. *The c_1-component of the solution for $\Delta x = 0.00067$, $\Delta t = 0.00025$.*

higher space dimensions we either must use extremely powerful computers or develop reliable higher-order accurate numerical methods.

Finally, we remark that the Godunov method has been applied to many different Riemann problems, including Example 3 in the previous section, and it always seems to converge to the exact solution. For some experiments with $n = 1$ we refer to [11].

REFERENCES

[1] T. BARKVE, *The Riemann problem for a non-strictly hyperbolic system modelling non-iso-thermal, two-phase flow in a porous media*, SIAM J. Appl. Math., 49 (1989), pp. 784–798.

[2] A. I. CHORIN, *Random choice solutions of hyperbolic systems*, J. Comput. Phys., 22 (1976), pp. 517–533.

[3] P. COLELLA, *Glimm's method for gas dynamics*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 76–110.

[4] P. CONCUS AND W. PROSKUROWSKI, *Numerical solutions of a nonlinear hyperbolic equation by the random choice method*, J. Comput. Phys., 30 (1979), pp. 153–166.

[5] 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.

[6] I. GELFAND, *Some Problems in the Theory of Quasilinear Equations*, American Mathematical Society Translation Series, Providence, RI, 1963, pp. 295–381.

[7] J. GLIMM, *Solutions in the large for nonlinear hyperbolic systems*, Comm. Pure Appl. Math., 18 (1965), pp. 697–715.

[8] 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.

[9] E. ISAACSON, *Global solution of a Riemann problem for a non-strictly hyperbolic system of conservation laws arising in enhanced oil recovery*, J. Comp. Phys., to appear.

[10] E. ISAACSON AND B. TEMPLE, *Analysis of a singular hyperbolic system of conservation laws*, J. Differential Equations, 65 (1986), pp. 250–268.

[11] T. JOHANSEN, A. TVEITO, AND R. WINTHER, *Numerical and analytical solutions of a model describing polymer flooding*, in Proc. Seminar on Reservoir Description and Simulation with Emphasis on EOR, Norway, 1986.

[12] 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.

[13] ———, *The Riemann problem for multicomponent polymer flooding*, SIAM J. Math. Anal., to appear.

[14] T. JOHANSEN, L. W. LAKE, A. TVEITO, AND R. WINTHER, *On the fractional flow theory for two phase flow processes in enhanced oil recovery*, in preparation.

[15] B. KEYFITZ AND H. KRANZER, *A system of non-strictly hyperbolic conservation laws arising in elasticity theory*, Arch. Rational Mech. Anal., 72 (1980), pp. 219–241.

[16] P. D. LAX, *Hyperbolic systems of conservation laws* II, Comm. Pure Appl. Math., 10 (1957), pp. 537–566.

[17] G. A. POPE, *The application of fractional flow theory to enhanced oil recovery*, Soc. Pet. Engrg. J., 20 (1980), pp. 191–205.

[18] G. A. POPE, L. W. LAKE, AND F. HELFFERICH, *Cation exchange and chemical flooding: Part 1—basic theory without dispersion*, Soc. Pet. Engrg. J., 18 (1978), pp. 418–434.

[19] J. SMOLLER, *Shock Waves and Reaction-Diffusion Equations*, Springer-Verlag, Berlin, New York, 1982.

[20] B. TEMPLE, *Stability and decay in systems of conservation laws*, in Proc. Nonlinear Hyperbolic Problems, St. Etienne, France, C. Carasso, P. A. Raviart, D. Serre, eds., Springer-Verlag, Berlin, New York, 1986.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 880-912, September 1989
© 1989 Society for Industrial and Applied Mathematics
003

A NEW CLASS OF PARALLEL ALGORITHMS FOR SOLVING SYSTEMS OF LINEAR EQUATIONS*

K. JAINANDUNSING†‡ AND E. F. DEPRETTERE†

Abstract. In this paper a class of novel feed-forward direct methods is presented for solving nonsingular systems of linear equations. The computational complexity of these methods is in the order of an LU, QR, or LL^t matrix factorization. This is also true for the complexity of their systolic implementations. Unlike the direct methods of factorization followed by backsubstitution, the systolic implementations of the novel methods do not suffer from the backsubstitution bottleneck. A numerically stable and robust method, which uses only Givens rotations as elementary operations, is included in the class.

Key words. systems of linear equations, nonsingular, backsubstitution, feed forward, systolic, Givens rotations

AMS(MOS) subject classifications. 15, 15A06, 15A09, 65, 65F05, 65F30, 68, 68Q35, 68U99, 94, 94C99

1. Introduction.

Solving (large) nonsingular systems of linear equations $A\mathbf{x}=\mathbf{b}$, is a key problem in many (real time) engineering applications. Examples are, among others, real time speech coding [Jain 1986], image processing [Dudg 1984], stochastic modeling [Orfa 1985], modeling with finite elements [Bath 1976], and computer-aided realistic three-dimensional image synthesis [Bu 1987]. When the problem at hand is large, we must obtain its solution within a reasonably short period of time. Among the concurrent multiprocessor computers the contemporary, relatively less expensive systolic/wavefront arrays [Kung 1979], [Gent 1981], [Kung 1984] are promising parallel architectures, which can tackle large problems efficiently in terms of time.

In case the coefficient matrix is dense, we usually rely upon so-called direct methods to compute the solution of $A\mathbf{x}=\mathbf{b}$. These direct methods can be divided in two classes:

(1) direct methods of factorization of the coefficient matrix, followed by backsubstitution;

(2) feed-forward direct methods.

We give a quick summary of the methods in both classes.

1.1. Direct methods with backsubstitution.

Let us denote by X either the lower triangular factor in an LU factorization, the orthogonal factor in a QR factorization, or the lower triangular factor in an LL^t factorization (superscript t denotes matrix transposition). Moreover, let R_X denote the corresponding upper triangular factor in each of these factorizations. Then we have $A=XR_X$ and the solution is computed in two steps: (1) $\mathbf{y}=X^{-1}\mathbf{b}$; (2) $\mathbf{x}=R_X^{-1}\mathbf{y}$. In the last step $\mathbf{x}$ is computed via backsubstitution. Systolic arrays for LU and QR factorizations of matrices have been presented by different authors [Gent 1981], [Leis 1981], [Ahme 1982]. Systolic arrays for the backsubstitution were presented in [Leis 1981] and [Gent 1981]. In Fig. 1(a) a systolic array is shown for the QR factorization of a 4×4 matrix $A=[a_{ij}]=QR$, with $R=[r_{ij}]$. At the same time, this array computes $\mathbf{y}=Q^t\mathbf{b}$, as requested in step (1), with $\mathbf{y}=[y_1y_2y_3y_4]^t$ and $\mathbf{b}=[b_1b_2b_3b_4]^t$.

* Received by the editors October 22, 1987; accepted for publication (in revised form) August 19, 1988. This work was partly supported by the Dutch National Applied Science Foundation under grant STW DEL 44.0643.

† Department of Electrical Engineering, Delft University of Technology, Delft, the Netherlands.

‡ Present address, United Nations, UNDTCD/CPS, Room DC1-1522, P.O. Box 20, New York, New York 10163-1522.

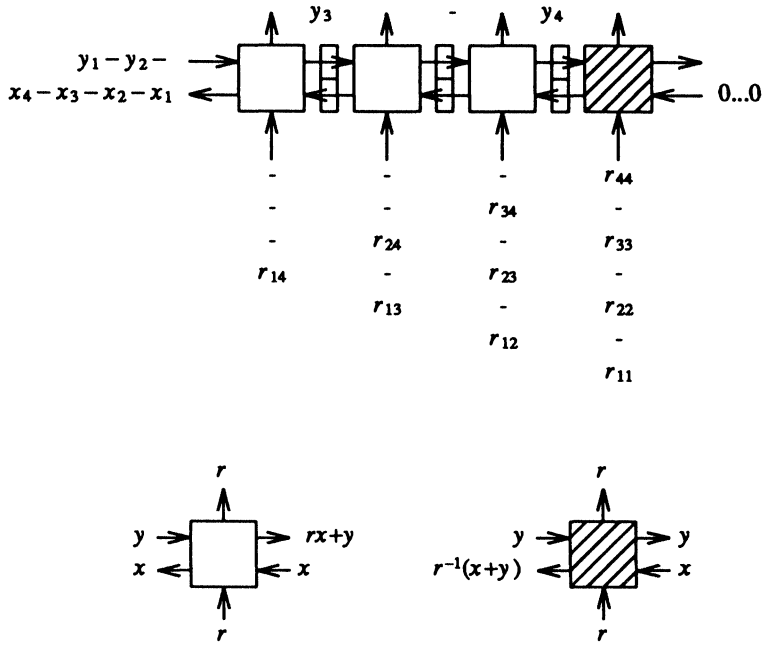


FIG. 1. (a) *Systolic array for the QR factorization.*

Each processor element performs a Givens rotation

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix},$$

with appropriate angle α, on its input data. The angle is stored locally at a processor element and the computations for the upper triangular factor R propagate upwards in the array. Since the processor elements in the array operate synchronously, the input data at the inputs at the left of the top-left two processor elements must be delayed one and two time steps, respectively. This delay is denoted by "-" and "--", respectively. Processor elements are assumed to be delay free. That is, computations are performed instantly. An interconnection with a delay is denoted by an arrow through a rectangle. If the rectangle is not numbered, it denotes a single unit of delay. Otherwise, the number denotes the number of delay units associated with the interconnection. The computed entries of the upper triangular factor R and the vector $\mathbf{y}$ must be stored and reorganized for backsubstitution. This is shown in Fig. 1(b), which also depicts the linear systolic array for the backsubstitution. The entries y_i of the vector $\mathbf{y}$ are propagating to the right, while the computed entries x_i of the solution vector $\mathbf{x}$ propagate to the left.

1.2. Feed-forward direct methods. An example of a feed-forward direct method is due to Faddeev [Fadd 1959]. It avoids backsubstitution and obtains the solution of $A\mathbf{x} = \mathbf{b}$ from an LU factorization only. We summarize Faddeev's method in terms of the following proposition.

PROPOSITION 1. *Let $A \in \mathbf{R}^{N\times N}$ be a nonsingular matrix and $\mathbf{b} \in \mathbf{R}^N$. Let $\mathbf{L}^{2N\times 2N}$ be the inverse of the lower triangular factor of the LU factorization of the augmented matrix $[A^t | -I_N]^t$ and $[R^t | O]^t$ is the upper triangular factor, with $R \in \mathbf{R}^{N\times N}$ an upper triangular matrix (I_N denotes the $N \times N$ identity matrix; $\mathbf{R}^N$ denotes the vector space of $N \times 1$ real*

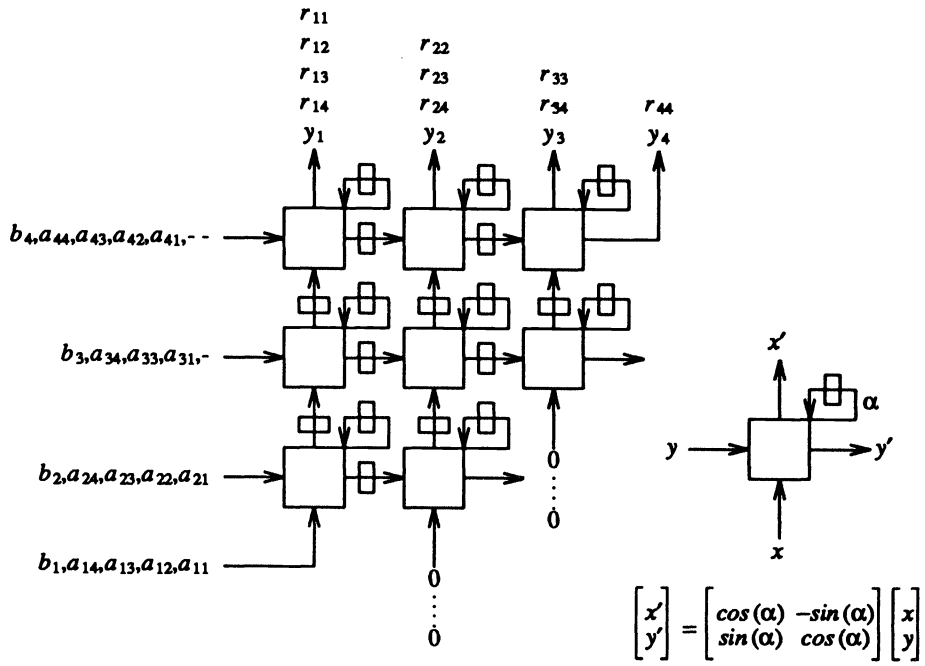


FIG. 1. (b) *Systolic array for the backsubstitution.*

vectors; *and* $\mathbf{R}^{N\times M}$ *denotes the vector space of* $N\times M$ *real matrices*). *Then, the solution* $\mathbf{x}$ *of the system of linear equations* $A\mathbf{x}=\mathbf{b}$ *obeys the following equation*:

$$\mathbf{L}\begin{bmatrix} A & \mathbf{b} \\ -I_N & 0 \end{bmatrix} = \begin{bmatrix} R & * \\ 0 & \mathbf{x} \end{bmatrix}.$$

Proof. The proof is simply constructed by writing the $N\times 2N$ matrix $\mathbf{L}$ as

$$\mathbf{L}=\begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{bmatrix}, \quad \text{with } L_{11}, L_{12}, L_{21}, L_{22}\in\mathbf{R}^{N\times N}$$

and observing that $L_{22}=I_N$. □

Faddeev's method is not only capable of solving $A\mathbf{x}=\mathbf{b}$, but it can be extended to solve for multiple right-hand sides as well. Moreover, it can be extended to compute the Schur complement of a matrix, matrix-matrix, matrix-vector product/accumulations, and innerproduct/accumulations. This is easily understood by applying Faddeev's method to a block matrix:

$$\mathbf{L}\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} R & * \\ O & E \end{bmatrix},$$

where $\mathbf{L}$ is appropriate. It is straightforwardly shown that $E=(D-CA^{-1}B)$. Choosing the dimensions and values of the matrices A, B, C, and D appropriately results in one of the above-mentioned operations. A systolic array for solving $A\mathbf{x}=\mathbf{b}$ with Faddeev's method is shown in Fig. 2, for an example where $A\in\mathbf{R}^{3\times 3}$. Each processor element computes a coefficient α which is the negative ratio of the pivot element and the element to be eliminated in the LU factorization of the matrix $[A^t|-I_N]^t$. Once this

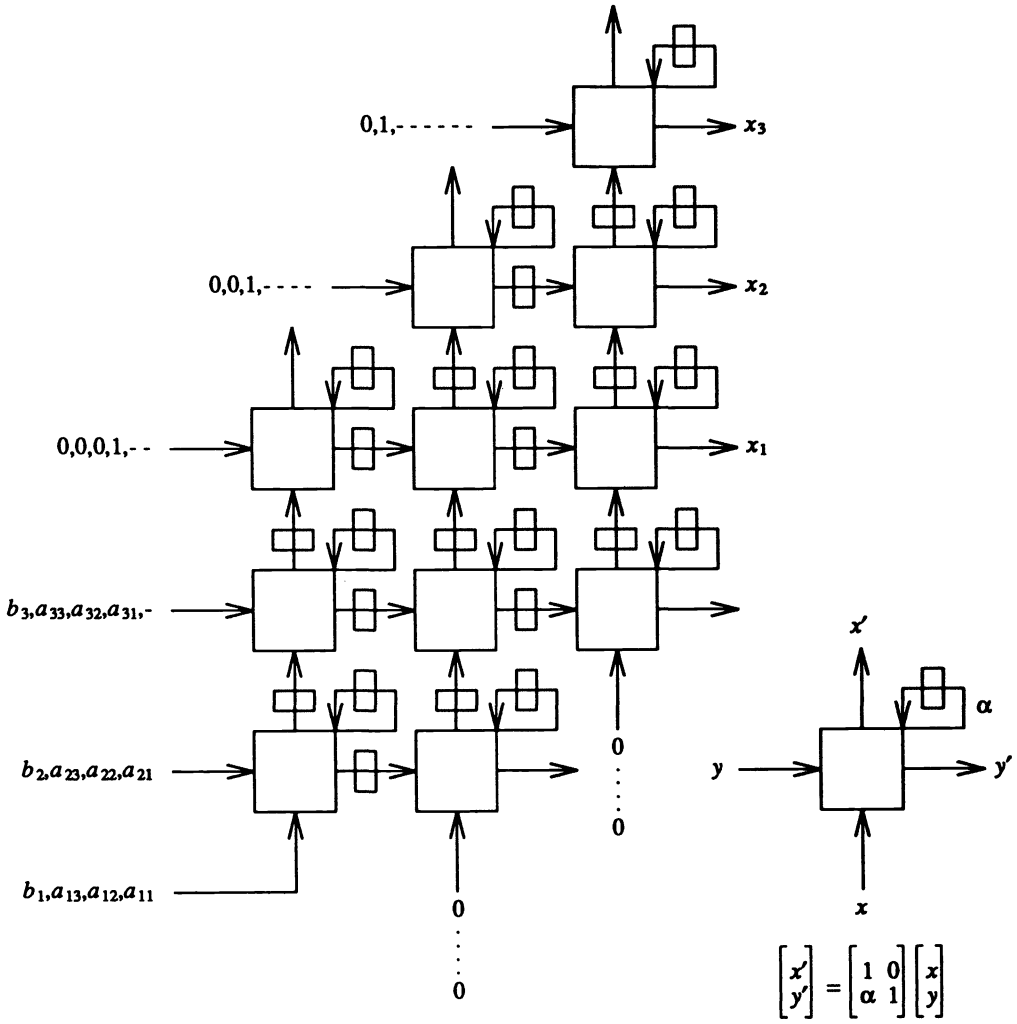


FIG. 2. *The Faddeev array for solving systems of linear equations.*

coefficient is computed, the processor element applies the linear rotation

$$\begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix}$$

to its input data.

Comparing the combination of the factorization and backsubstitution array to the array for Faddeev's method, the simplicity of the last one is striking. But, more important, as shall be illustrated next by a calculation of the number of time steps needed to compute the solution of $A\mathbf{x} = \mathbf{b}$ by the arrays for both methods, the backsubstitution is a bottleneck in a systolic implementation. From a generalization of Fig. 1(a) it follows that it takes $3N-2$ time steps to compute the upper triangular factor of an $N \times N$ matrix and the transformation of the right-hand side vector. And from a generalization of Fig. 1(b) it follows that the backsubstitution requires $4N-3$ time steps to be completed on the linear array ($N-1$ time steps to load the array with the entries of the transformed right-hand side vector $\mathbf{y}$; $2N-1$ time steps to compute all

entries of the solution vector $\mathbf{x}$; and $N-1$ time steps to shift the last computed entry of the solution vector out of the array). Due to lack of parallelism or pipelining between the computations of the QR factorization and the backsubstitution, the total number of time steps to compute a solution is the addition of the number of time steps for each of them. Hence, the total number of time steps is $7N-5$ in this case. On the other hand, from a generalization of Faddeev's array, we find that only $3N$ time steps are required to compute the solution. This is roughly equal to the number of time steps required for the QR factorization. And Faddeev's array computes the solution roughly twice as fast as the combination of the QR factorization and the backsubstitution array. However, Faddeev's array is only useful for solving positive definite systems of linear equations, because (partial) pivoting, which is not straightforwardly done on such arrays due to lack of global communication paths, is not needed in such cases.[1]

The simplicity and speed of the systolic array for Faddeev's feed-forward direct method and the limited use of this array was the motivation to seek for other feed-forward direct methods that have better numerical properties without pivoting and with systolic implementations of comparable simplicity and speed. The outline of the paper is as follows. In § 2 we show how the backsubstitution can be rephrased in terms of an updating or downdating of a Cholesky factorization, or in terms of an LU factorization. This will be the key to a whole class of novel feed-forward direct methods for solving nonsingular systems of linear equations. In § 3 it is explained how an LU, QR, or LL^t factorization of the coefficient matrix is combined with the method in § 2. This leads to the class of novel feed-forward direct methods. This class contains a feed-forward direct method which uses only Givens rotations as elemantary operations. In § 4 a numerical analysis is given of the novel feed-forward direct methods of § 3. In § 5 it is shown that one of the new methods is in fact a semidirect method in the sense of the conjugate gradients method [Golu 1983]. In § 6 it is explained how the methods of § 3 are extended to compute expressions of the kind $B^tA^{-t}C^t+D^t$, where the matrices A, B, C, and D are of proper dimensions and value. This extension is similar to the extension of Faddeev's feed-forward direct method [Fadd 1959].

2. Feed-forward computation of $\mathbf{x}=A^{-1}\mathbf{b}$. In this section we assume that the lower triangular factor L and the inverse X^{-1} of the remainder X in the factorization $A=LX$ are given. With this assumption it is shown how the backsubstitution is rephrased in terms of an updating or downdating of a Cholesky factorization, or in terms of an LU factorization. Let $A\in\mathbf{R}^{N\times N}$ be a nonsingular matrix and let $\mathbf{b}\in\mathbf{R}^N$ be a column vector. The solution of the equation

$$A\mathbf{x}=\mathbf{b} \tag{1}$$

may be expressed in terms of a lower triangular factorization

$$A=LX \tag{2a}$$

where L is a lower triangular matrix and the matrix X is one of the following:

(1) $X=U$ is upper triangular (LU factorization of A);

(2) $X=Q$ is orthogonal ($QQ^t=Q^tQ=I$) (LQ factorization of A);

[1] In [Royce 1988] it was shown that we can devise LU factorization algorithms with partial pivoting, which can be mapped to systolic arrays. However, the sacrifice being made in these algorithms is that the rows of the upper triangular factor are not naturally ordered at the outputs of the array. Thus, the output at which an entry of the solution vector appears will depend on the coefficient matrix A. This property implies additional overhead for locating the entries of the solution vector at the output of the array in practical implementations.

(3) $X = L^t$ is the upper triangular Cholesky factor of the matrix A, which is then symmetric positive definite (LL^t factorization of A).

Hence, denoting

(2b) $$L\mathbf{y} = \mathbf{b},$$

(1) becomes

(2c) $$X\mathbf{x} = \mathbf{y}.$$

Equations (2b) and (2c) can be rewritten as follows:

(3a) $$[\mathbf{y}^t \quad 1]\begin{bmatrix} L^t \\ -\mathbf{b}^t \end{bmatrix} = 0,$$

(3b) $$[\mathbf{y}^t \quad 1]\begin{bmatrix} X^{-t} \\ 0 \end{bmatrix} = \mathbf{x}^t.$$

The computation of $\mathbf{x}$ in (3b) is based on the following observation. The matrix $[L|-\mathbf{b}]^t$ in (3a) is a rectangular $(N+1)\times N$ matrix, so that it can be reduced to an upper triangular form, with zero last row, by applying a proper sequence of rotations to the matrix. Then, since the last row, say $\mathbf{r}$, of the product of embedded plane rotations satisfies

$$\mathbf{r}\begin{bmatrix} L^t \\ -\mathbf{b}^t \end{bmatrix} = 0$$

it must be proportional to $[\mathbf{y}^t \quad 1]$. Therefore, when this row is substituted for $[\mathbf{y}^t \quad 1]$ in (3b), the result will be a vector proportional to the solution vector $\mathbf{x}^t$.

Thus, let the sequence of rotations applied to, $[L|-\mathbf{b}]^t$, be denoted by the $(N+1)\times(N+1)$ matrix $\Theta(m)$, with $m \in \{0, -1\}$

(4a) $$\Theta(m) = \prod_{i=1}^{\tilde{N}} \Theta_i(m) = \Theta_N(m)\Theta_{N-1}(m)\cdots\Theta_1(m),$$

with

(4b) $$\Theta_i(m) = \begin{bmatrix} I_{i-1} & & & \\ & \cos(m^{1/2}\alpha_i) & & -m^{1/2}\sin(m^{1/2}\alpha_i) \\ & & I_{N-i} & \\ & m^{-1/2}\sin(m^{1/2}\alpha_i) & & \cos(m^{1/2}\alpha_i) \end{bmatrix}.$$

In case $m = 0$, $m = 1$ and $m = -1$, (4b) is an embedding of a 2×2 linear rotation

$$\begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix},$$

Givens rotation

$$\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

and a hyperbolic rotation

$$\begin{bmatrix} \cosh(\alpha) & \sinh(\alpha) \\ \sinh(\alpha) & \cosh(\alpha) \end{bmatrix},$$

respectively; the parameterization in m is due to [Walt 1971]. $\Theta(m)$ is such that

(5) $$\Theta(m)\begin{bmatrix} L^t \\ -\mathbf{b}^t \end{bmatrix} = \begin{bmatrix} R(m) \\ 0^t \end{bmatrix},$$

where $R(m)$ is upper triangular. That is, the angles α_i are chosen such that an upper triangularization of the matrix $[L|-\mathbf{b}]^t$ is obtained in the recursion:

$$\begin{bmatrix} L_i^t \\ -\mathbf{b}_i^t \end{bmatrix} = \prod_{j=1}^{i} \Theta_j(m) \begin{bmatrix} L^t \\ \mathbf{b}^t \end{bmatrix}, \qquad i = 1, 2, \cdots, N, \tag{6a}$$

where

$$\mathbf{b}_i^t = [0 \underset{i}{\cdots} 0 \; b_1^{(i)} \cdots b_{N-i}^{(i)}]. \tag{6b}$$

The matrix $\Theta(m)$ is $\Sigma(m)$-orthogonal, i.e.,

$$\Theta^t(m)\Sigma(m)\Theta(m) = \Sigma(m), \tag{7}$$

with

$$\Sigma(m) = \begin{bmatrix} I_N & \\ & m \end{bmatrix}. \tag{8}$$

For $m = \pm 1$, we also have that

$$\Theta(m)\Sigma(m)\Theta^t(m) = \Sigma(m). \tag{9}$$

As yet we have to prove that the angles α_i exist for given lower triangular matrix $L = [l_{ij}]$ and vector $\mathbf{b} = [b_i]$ in either of the cases $m \in \{0, 1, -1\}$. For $m = 0$, $\alpha_i = -b_1^{(i)}/l_{ii}$ (allowing pivoting in case of zero pivots). Using linear rotations, $m = 0$, amounts to an LU factorization in (5), with $U = [R^t(0)|0]^t$. For $m = 1$, $\tan(\alpha_i) = -b_1^{(i)}/l_{ii}$ always exists, since the function $\tan(\alpha_i)$ has image $\mathbf{R}$. Using circular (Givens) rotations, $m = 1$, amounts to a QR factorization in (5), with $R = [R^t(1)|0]^t$.

The use of hyperbolic rotations ($m = -1$) is restricted to special cases. Namely, to those cases where the elements of the matrix of the matrix L and the vector $\mathbf{b}$ are such that $|\tanh(\alpha_i)| = |b_1^{(i)}/l_{ii}| < 1$. Only then do the hyperbolic rotations exist. The following proposition states for which case $\Theta(-1)$ in (5) exists.

PROPOSITION 2. *Let $L \in \mathbf{R}^{N \times N}$ be a lower triangular matrix and $\mathbf{b} \in \mathbf{R}^N$ a vector, such that*

$$LL^t - \mathbf{b}\mathbf{b}^t > 0. \tag{10}$$

Then there exists a matrix $\Theta(-1)$ as defined in (4a) *and* (4b) *for $m = -1$, such that*

$$\Theta(-1)\begin{bmatrix} L^t \\ -\mathbf{b}^t \end{bmatrix} = \begin{bmatrix} R(-1) \\ 0^t \end{bmatrix}, \tag{11}$$

where $R(-1)$ is upper triangular.

Proof. For a proof we refer to Appendix A. □

The proportionality constant between the last row of the matrix $\Theta(m)$ and $[\mathbf{y}^t \;\; 1]$ can be found from the properties of the matrix $\Theta(m)$, as follows. Write for the $(N+1) \times (N+1)$ matrix $\Theta(m)$:

$$\Theta(m) = \begin{bmatrix} \Theta_{11}(m) & \theta_{12}(m) \\ \theta_{21}^t(m) & k(m) \end{bmatrix}, \qquad \Theta_{11}(m) \in \mathbf{R}^{N \times N}, \tag{12}$$

where $k(m)$ is the proportionality constant. For $m = 0$ it follows from (4a) and (4b) that $k(0) = 1$. Hence, $\theta_{21}(0) = \mathbf{y}$ and, consequently

$$\Theta(0)\begin{bmatrix} L^t & X^{-t} \\ -\mathbf{b}^t & 0^t \end{bmatrix} = \begin{bmatrix} R(0) & * \\ 0^t & \mathbf{x}^t \end{bmatrix}. \tag{13}$$

This equation expresses an LU factorization of the matrix $[L|-\mathbf{b}]^t$. For $m=1$ or $m=-1$ we substitute (12) in (9). This gives

$$mk(m)+\theta_{21}^t(m)\theta_{21}(m)=m.$$

Hence, with $\mathbf{y}=k^{-1}(m)\theta_{21}(m)$ it follows that

$$k(m)=(1+m\mathbf{y}^t\mathbf{y})^{-1/2}. \tag{14}$$

Hence

$$\Theta(m)\begin{bmatrix} L^t & X^{-t} & 0 \\ -\mathbf{b}^t & 0^t & 1 \end{bmatrix}=\begin{bmatrix} R(m) & * & * \\ 0^t & k(m)\mathbf{x}^t & k(m) \end{bmatrix}, \tag{15}$$

for $m=1, -1$. Using the $\Sigma(m)$-orthogonality of $\Theta(m)$, we get from (15) $LL^t+m\mathbf{b}\mathbf{b}^t=R^t(m)R(m)$, so that (15) expresses an updating and a downdating of a Cholesky factorization for $m=1$ and $m=-1$, respectively.

A numerically important property of the case $m=-1$, for $LL^t-\mathbf{b}\mathbf{b}^t>0$, is stated in the following theorem.

THEOREM 1. *Let $L\in\mathbf{R}^{N\times N}$ be a lower triangular matrix and $\mathbf{b}\in\mathbf{R}^N$, such that the matrix*

$$\begin{bmatrix} 1 & -\mathbf{b}^t \\ -\mathbf{b} & LL^t \end{bmatrix},$$

is positive definite with all its main diagonal entries equal to 1.[2] *For example, $LL^t-\mathbf{b}\mathbf{b}^t>0$ and $LL^t>0$. Then, the magnitude of the entries of the matrices L_i and the vectors $\mathbf{b}_i$ in the recursion* (6a)-(6b) *is bounded by* 1, *for $m=-1$.*

Proof. The proof is deferred to § 3.3. □

3. The novel class of algorithms. In this section it is shown how the results in (13) and (15) are combined with the lower triangular factorization of the matrix A, to obtain feed-forward methods for solving $A\mathbf{x}=\mathbf{b}$.

3.1 *LU* factorization. Let $A=LU$ be an LU factorization of the matrix A. Then an LU factorization of the augmented matrix $[A^t|I_N]$ will give

$$\bar{U}[A^t|I_N]=[L^t|U^{-t}], \tag{16}$$

where $\bar{U}=U^{-t}$. L^t and U^{-t} are the matrix factors appearing in (3a) and (3b). For example, $X^{-t}=U^{-t}$ in (3b). Next, we show how either of the possible factorizations in (13) and (15) are combined with the LU factorization in (16), such that a feed-forward method is obtained for solving $A\mathbf{x}=\mathbf{b}$. Put

$$U_{ij}=\begin{bmatrix} I_{j-1} & & & & \\ & 1 & & & \\ & & I_{i-j} & & \\ & \alpha_{ij} & & 1 & \\ & & & & I_{N-i} \end{bmatrix}, \qquad i,j=1,\cdots,N-1. \tag{17}$$

The entries of the first column of the strictly lower triangular part of the matrix $[A^t|I_N]$ are eliminated by premultiplication with $\prod_{i=1}^{N-1}U_{i1}$, where the α_{i1} are appropriate. Hence, the first row of the resulting matrix will be the first row of the matrix $[L^t|U^{-t}]$. Next the first rotation $\Theta_1(m)$, in the product $\Theta(m)$ in (4a) is applied to this row and

[2] This is obtained by a trivial normalization.

the vector $[-\mathbf{b}^t|0^t]$, such that the first element of $-\mathbf{b}^t$ is eliminated. Second, we eliminate the elements of the second column of the strictly lower triangular part of the matrix $(\prod_{i=1}^{\tilde{N}-1} U_{i1})[A^t|I_N]$, by premultiplication with $\prod_{i=2}^{\tilde{N}-1} U_{i2}$, with appropriate α_{i2}. Thus, we obtain the second row of the matrix $[L^t|U^{-t}]$. Next, the second rotation $\Theta_2(m)$ in the product $\Theta(m)$ in (4a) is applied to this row and the vector that resulted from the rotation $\Theta_1(m)$ on $[-\mathbf{b}^t|0^t]$, such that the second element of this vector is eliminated. The above procedure is repeated until all the elements of the vector $-\mathbf{b}^t$ are eliminated against the pivots of the matrix L^t.

Thus, we get the following feed-forward computational scheme for $A\mathbf{x}=\mathbf{b}$:

$$\Theta_N(m)(\Theta_{N-1}(m)U_{N-1,N-1})\cdots\left(\Theta_2(m)\prod_{i=2}^{\tilde{N}-1}U_{i2}\right)\left(\Theta_1(m)\prod_{i=1}^{\tilde{N}-1}U_{i1}\right)\begin{bmatrix}A^t & I_N & 0\\ -\mathbf{b}^t & 0^t & 1\end{bmatrix}$$

$$(18)\qquad =\begin{bmatrix}R(m) & * & *\\ 0^t & k(m)\mathbf{x}^t & k(m)\end{bmatrix}$$

($m\in\{0, 1, -1\}$). Note that allowing m to equal -1 is constrained to the condition that $LL^t-\mathbf{b}\mathbf{b}^t>0$, where L is the lower triangular factor of the LU factorization of the matrix A.

Observe that (18) expresses an LU factorization of the matrix $[A|-\mathbf{b}]^t$ in case we use linear rotations for the $\Theta_i(m)$ ($m=0$), namely,

$$(19)\ \Theta_N(0)(\Theta_{N-1}(0)U_{N-1,N-1})\cdots\left(\Theta_2(0)\prod_{i=2}^{\tilde{N}-1}U_{i2}\right)\left(\Theta_1(0)\prod_{i=1}^{\tilde{N}-1}U_{i1}\right)\begin{bmatrix}A^t\\ -\mathbf{b}^t\end{bmatrix}=\begin{bmatrix}R(0)\\ 0^t\end{bmatrix}.$$

Since the LU factorization of a matrix is unique, it follows that different orderings of linear rotations can be applied to obtain the result in (18). For instance, the following order of linear rotations (indicated by combinations (i,j) of row i and row j) gives the same result as in (18):

$(N, N+1), (N-1, N), \cdots, (1, 2)$ followed by $(N, N+1), (N-1, N), \cdots, (2, 3)$, etc.

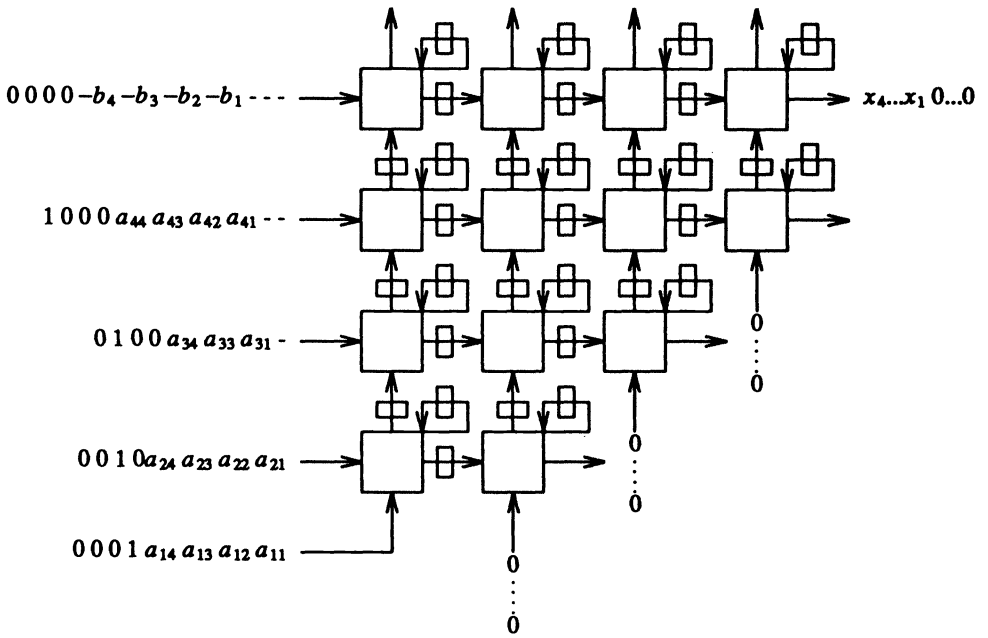


FIG. 3. *Systolic array for the feed-forward LU method.*

A possible systolic processor array which uses the scheme of linear rotations in (19) is shown in Fig. 3 for $N=4$. This array is identical to the one in Fig. 1(a), except that the processor elements here perform linear rotations and there is an extra row of linear rotors for the elimination of the elements of the vector $\mathbf{b}=[b_1 b_2 b_3 b_4]^t$.

3.2. *QR* factorization. Let $A=LQ$ be the LQ factorization of the matrix A. Then, a QR factorization of the matrix $[A^t | I_N]$ will give

$$\bar{Q}[A^t | I_N] = [L^t | Q], \tag{20}$$

where $\bar{Q}=Q$. L^t and Q are the matrix factors appearing in (3a) and (3b), i.e., $X^{-t}=Q$ in (3b). The factorizations in (13) and (15) are combined with the factorization in (20) in the same way as was explained for the LU factorization of $[A^t | I_N]$, but using embedded plane circular (Givens) rotations for the QR factorization of $[A^t | I_N]$. That is, by putting

$$Q_{ij} = \begin{bmatrix} I_{j-1} & & & & \\ & \cos(\alpha_{ij}) & & -\sin(\alpha_{ij}) & \\ & & I_{i-j} & & \\ & \sin(\alpha_{ij}) & & \cos(\alpha_{ij}) & \\ & & & & I_{N-i} \end{bmatrix}, \tag{21}$$

we get the following feed-forward computational scheme for $A\mathbf{x}=\mathbf{b}$:

$$\begin{aligned} &\Theta_N(m)(\Theta_{N-1}(m)Q_{N-1,N-1})\cdots\left(\Theta_2(m)\prod_{i=2}^{\tilde{N}-1}Q_{i2}\right)\left(\Theta_1(m)\prod_{i=1}^{\tilde{N}-1}Q_{i1}\right)\begin{bmatrix} A^t & I_N & 0 \\ -\mathbf{b}^t & 0^t & 1 \end{bmatrix} \\ &\quad=\begin{bmatrix} R(m) & * & * \\ 0^t & k(m)\mathbf{x}^t & k(m) \end{bmatrix} \end{aligned} \tag{22}$$

($m\in\{0, 1, -1\}$). Note that allowing $m=-1$ is constrained to the condition that $LL^t-\mathbf{b}\mathbf{b}^t>0$, where L is the lower triangular factor of the LQ factorization of the matrix A.

Observe that (22) expresses a QR factorization of the matrix $[A|-\mathbf{b}]^t$ in case we use linear rotations for the $\Theta_i(m)$ ($m=1$), i.e.,

$$\Theta_N(1)(\Theta_{N-1}(1)Q_{N-1,N-1})\cdots\left(\Theta_2(1)\prod_{i=2}^{\tilde{N}-1}Q_{i2}\right)\left(\Theta_1(1)\prod_{i=1}^{\tilde{N}-1}Q_{i1}\right)\begin{bmatrix} A^t \\ -\mathbf{b}^t \end{bmatrix}=\begin{bmatrix} R(1) \\ 0^t \end{bmatrix}. \tag{23}$$

And, since the QR factorization of a matrix is unique, it follows that different orderings of circular rotations may be found by which the result in (23) is obtained. For instance, the following order of circular rotations (indicated by combinations (i, j) of row i and row j) gives the same result as in (22): $(N, N+1), (N-1, N), \cdots, (1, 2)$ followed by $(N, N+1), (N-1, N), \cdots, (2, 3)$, etc. A regular, systolic processor array that uses the scheme of rotations in (23) is identical to the one in Fig. 3, except that the processor elements perform Givens rotations.

3.3. Schur–Cholesky factorization. If the coefficient matrix A is a symmetric positive definite matrix, a fast Cholesky factorization of the matrix is possible with the so-called *generalized Schur algorithm* [Depr 1982], [Delo 1984]. This algorithm is a generalization of the Schur algorithm for the Cholesky factorization of a symmetric and positive definite Toeplitz or close to Toeplitz matrix [Dewi 1978], [Lev 1984]. First we shall present the generalized Schur algorithm and its properties, after which it is shown how to combine this algorithm with the factorizations in (13) and (15), in order to obtain feed-forward algorithms.

3.3.1. The generalized Schur algorithm.

THEOREM 2. *Let $A=[a_{ij}]\in\mathbf{R}^{N\times N}$ be a symmetric positive definite matrix, normalized such that we can write*

$$A=R_L+I_N+R_L^t, \tag{24}$$

where $R_L=[a_{ij}]$, $i>j$, is the strictly lower triangular part of A. Put

$$U=R_L+I_N, \tag{25a}$$

$$Y=R_L. \tag{25b}$$

Then there exists a matrix product $\Phi\in\mathbf{R}^{2N\times 2N}$:

$$\Phi=\prod_{j=1}^{\tilde{N}-1}\prod_{i=j}^{\tilde{N}-1}\Phi_{ij}, \tag{26a}$$

of embedded plane hyperbolic rotations

$$\Phi_{ij}=\begin{bmatrix} I_i & & & & \\ & \cosh(\alpha_{ij}) & & \sinh(\alpha_{ij}) & \\ & & I_{N-j-1} & & \\ & \sinh(\alpha_{ij}) & & \cosh(\alpha_{ij}) & \\ & & & & I_{N+j-i-1}\end{bmatrix}, \qquad i,j=1,\cdots,N-1 \tag{26b}$$

such that the Cholesky factor L^t of the matrix A is given from

$$\begin{bmatrix}L^t\\ O\end{bmatrix}=\Phi\begin{bmatrix}U^t\\ Y^t\end{bmatrix}. \tag{27}$$

Proof. We refer to Appendix B for the proof. □

The hyperbolic rotations $\Phi_{kk}, \Phi_{k+1,k},\cdots,\Phi_{N-1,k}$ in the planes $(k+1, N+1)$, $(k+2, N+2),\cdots,(N, 2N-k)$, respectively, eliminate the elements on the $(N-k+1)$st diagonal of the matrix

$$\prod_{j=1}^{k-1}\prod_{i=j}^{\tilde{N}-1}\Phi_{ij}\begin{bmatrix}U^t\\ Y^t\end{bmatrix}.$$

The matrix Φ is J-orthogonal, i.e.,

$$\Phi^tJ\Phi=\Phi J\Phi^t=J, \tag{28}$$

with

$$J=I_N\oplus -I_N. \tag{29}$$

(The symbol $\oplus$ denotes orthogonal sum.) The product form in (26a) implies a *diagonal-recursive* form for the computation of the Cholesky factor of the matrix A:

$$\begin{aligned} &U_0=U, \qquad Y_0=Y,\\ &\begin{bmatrix}U_j^t\\ Y_j^t\end{bmatrix}=\Phi_j\begin{bmatrix}U_{j-1}^t\\ Y_{j-1}^t\end{bmatrix}, \qquad j=1,\cdots,N-1,\\ &U_{N-1}=L, \qquad Y_{N-1}=0,\end{aligned} \tag{30}$$

where

$$\Phi_j=\prod_{i=j}^{\tilde{N}-1}\Phi_{ij}. \tag{31}$$

From Theorem 2 the factor L^t is obtained as required in (3a). It remains to show how the factor $X^{-t} = L^{-1}$, for use in (3b), is obtained. For this purpose, we need the following corollary as it follows from Theorem 2.

COROLLARY 1. *The J-orthogonal matrix* Φ *in* (26a) *is of the global form*

$$\Phi = \begin{bmatrix} L^{-1} & 0 \\ 0 & R^{-1} \end{bmatrix} \begin{bmatrix} U & -Y \\ -Y^t & U^t \end{bmatrix}, \tag{32}$$

where

$$A = LL^t = RR^t. \tag{33}$$

(*L is lower triangular and R is upper triangular.*)

Proof. Let Φ be partitioned into four $N \times N$ submatrices

$$\Phi = \begin{bmatrix} \Phi_{11} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix}.$$

Then, it follows from (27) that

$$\Phi_{11} U^t + \Phi_{12} Y^t = L^t \tag{34}$$

and

$$\Phi_{21} U^t + \Phi_{22} Y^t = 0. \tag{35}$$

With

$$S = U^{-1} Y = Y U^{-1} \tag{36}$$

(as follows directly from (25a) and (25b)), equation (35) becomes

$$\Phi_{21} = -\Phi_{22} S^t. \tag{37}$$

Since Φ is J-orthogonal, it follows that

$$\Phi_{12} = \Phi_{11} \Phi_{21}^t \Phi_{22}^{-t} = -\Phi_{11} S. \tag{38}$$

Substituting this in (34) gives

$$\Phi_{11} U^{-1} (UU^t - YY^t) U^{-t} = L^t U^{-t},$$

with

$$UU^t - YY^t = A = LL^t, \tag{39}$$

as follows from (25a) and (25b). Thus,

$$\Phi_{11} = L^{-1} U \tag{40}$$

and, consequently,

$$\Phi_{12} = -L^{-1} Y. \tag{41}$$

From the J-orthogonality of Φ it follows that

$$\Phi_{22} \Phi_{22}^t - \Phi_{21} \Phi_{21}^t = I_N.$$

Using (35) and (34), this becomes

$$(\Phi_{22} U^{-t}) A (\Phi_{22} U^{-t})^t = I_N. \tag{42}$$

But from (26a) and (26b) it follows that Φ_{22} is upper triangular and, hence, $\Phi_{22}U^{-t}$ is upper triangular. From the uniqueness of the *upper-lower* triangular factorization of a square matrix, it follows that

$$\Phi_{22}U^{-t} = R^{-1}. \tag{43}$$ □

From the gobal form of the matrix Φ it immediately follows that the factor L^{-1} is found from

$$\begin{bmatrix} L^{-1} \\ R^{-1} \end{bmatrix} = \Phi \begin{bmatrix} I_N \\ I_N \end{bmatrix}. \tag{44}$$

Remark. Using (25a) and (25b), a simple calculation shows that the upper triangular factor R is obtained from

$$\begin{bmatrix} O \\ R^t \end{bmatrix} = \Phi \begin{bmatrix} Y \\ U \end{bmatrix}.$$

3.3.2. Feed-forward direct methods with Schur–Cholesky factorization. In the preceding section, it was shown how to obtain the Cholesky factor L^t and its inverse L^{-1}, as required in (3a) and (3b). Next it will be shown how either of the possible factorizations in (13) and (15) are combined with the generalized Schur algorithm, such that a feed-forward algorithm is obtained.

The first row of $[U^t | I_N]$ is the first row of the matrix $[L^t | L^{-1}]$. Application of $\Theta_1(m)$ to this row and the vector $[-\mathbf{b}^t | 0^t]$ elimnates the first element of $-\mathbf{b}^t$ for a proper choice of the angle α_1. Next, the rotations $\Phi_{11}, \Phi_{21}, \cdots, \Phi_{N-1,1}$ are applied to the matrix $[U\ Y]^t$. The result being that the Nth lower subdiagonal of this matrix is eliminated and the second row of the matrix

$$\prod_{i=1}^{\tilde{N}-1} \Phi_{i1} \begin{bmatrix} U^t & I_N \\ Y^t & I_N \end{bmatrix}$$

is equal to the second row of $[L^t | L^{-1}]$. Now a rotation $\Theta_2(m)$ is applied to this row and the modification of the vector $[-\mathbf{b}^t | 0^t]$. This procedure is repeated until all elements of the vector $-\mathbf{b}^t$ are eliminated. With the following embedding of the matrices $\Theta_i(m)$ in (4b)

$$[\Theta_i(m)] = \begin{bmatrix} I_{i-1} & & & \\ & \cos(m^{1/2}\alpha_i) & & -m^{1/2}\sin(m^{1/2}\alpha_i) \\ & & I_{2N-i} & \\ & m^{-1/2}\sin(m^{1/2}\alpha_i) & & \cos(m^{1/2}\alpha_i) \end{bmatrix}, \tag{45}$$

we get the following feed-forward computational scheme for $A\mathbf{x} = \mathbf{b}$:

$$[\Theta_N(m)]([\Theta_{N-1}(m)](\Phi_{N-1,N-1} \oplus 1)) \cdots \left([\Theta_2(m)] \prod_{i=2}^{\tilde{N}-1} (\Phi_{i2} \oplus 1)\right) \times \left([\Theta_1(m)] \prod_{i=1}^{\tilde{N}-1} (\Phi_{i1} \oplus 1)\right),$$

$$\begin{bmatrix} U^t & I_N & 0 \\ Y^t & I_N & 0 \\ -\mathbf{b}^t & 0^t & 1 \end{bmatrix} = \begin{bmatrix} R(m) & * & * \\ O & * & * \\ 0^t & k(m)\mathbf{x}^t & k(m) \end{bmatrix} \tag{46}$$

($m \in \{0, 1, -1\}$). Note that, allowing $m = -1$ is constrained to the condition $LL^t - \mathbf{b}\mathbf{b}^t > 0$. In this case (46) expresses the Cholesky factorization of the positive definite matrix

$A-\mathbf{b}\mathbf{b}^t$. Indeed, since the product of hyperbolic rotations in (46) is orthogonal with respect to the signature matrix $\bar{J}=(I_N \oplus -I_{N+1})$, we deduce that $A-\mathbf{b}\mathbf{b}^t = R(-1)^t R(-1)$. And since the Cholesky factorization of a symmetric positive definite matrix is unique, it follows that $R(-1)^t$ is the Cholesky factor of the matrix $A-\mathbf{b}\mathbf{b}^t$. A systolic processor array for the execution of the above scheme of rotations is given in Fig. 4 for $N=4$ and $m=-1$.

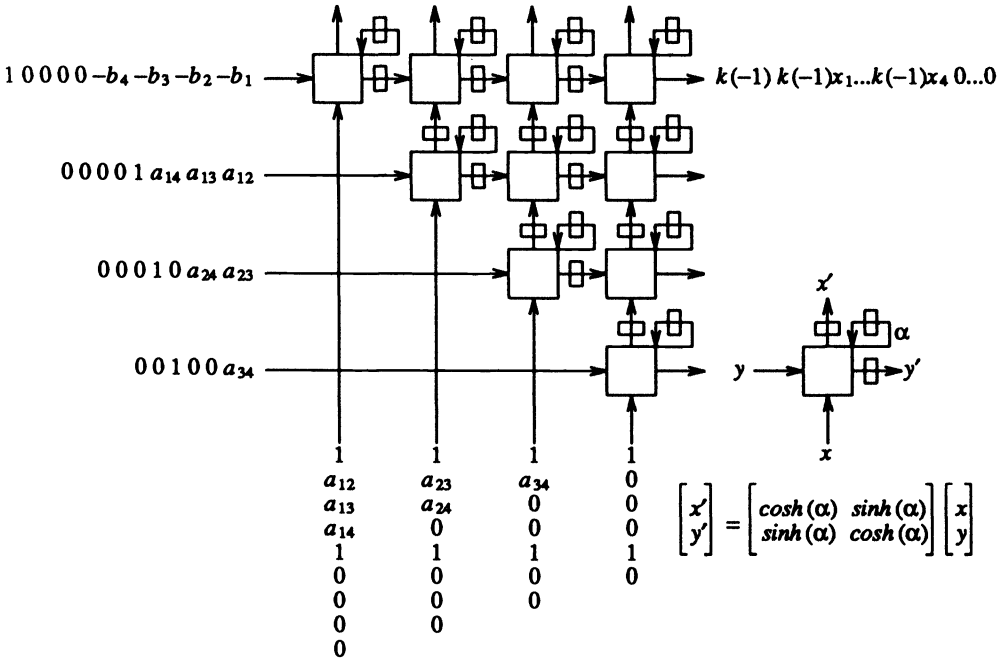


FIG. 4. *Systolic array for the feed-forward Cholesky method.*

The processor elements on the first diagonal of the lower 3×3 triangular part that process the entries of the matrix A, eliminate the Nth lower subdiagonal of the matrix $[U\ Y]^t$. The processor elements on the ith ($i>1$) diagonal of this lower 3×3 triangular part eliminate the $(N-i+1)$st lower subdiagonal of the matrix that results after $i-1$ steps in (46). The processor element at the top of the ith column of this lower 3×3 triangular part produces the $(i+1)$th row of L^t and L^{-1}. These rows are used in the upper row of four rotors, which compute $k(-1)\mathbf{x}^t$ and $k(-1)$.

LEMMA 1. *Let the following be given.* $A\in \mathbf{R}^{N\times N}$ *is a symmetric positive definite matrix with its diagonal entries equal to* 1. $\mathbf{b}\in \mathbf{R}^N$ *is such that* $A-\mathbf{b}\mathbf{b}^t>0$. $B\in \mathbf{R}^{(N+1)\times(N+1)}$ *is the positive definite matrix*

$$B=\begin{bmatrix} 1 & -\mathbf{b}^t \\ -\mathbf{b} & A \end{bmatrix}. \tag{47}$$

U_B^t *is the upper triangular part of the matrix* B

$$U_B^t=\begin{bmatrix} 1 & -\mathbf{b}^t \\ 0 & U^t \end{bmatrix}, \tag{48a}$$

and Y_B^t *is the strictly upper triangular part of the matrix* B

$$Y_B^t=\begin{bmatrix} 0 & -\mathbf{b}^t \\ 0 & Y^t \end{bmatrix}, \tag{48b}$$

where U^t and Y^t are the upper and strictly upper triangular part, respectively, of the matrix A. Let Φ be the sequence of hyperbolic rotations as in (26a)-(26b), *with N substituted by $N+1$, that obtains the Cholesky factor of the matrix B by premultiplication of $[U_B \quad Y_B]^t$ with Φ. Then the solution $\mathbf{x}$ of $A\mathbf{x}=\mathbf{b}$ is obtained from the quation*

$$\Phi \left[\begin{array}{cc|c} 1 & -\mathbf{b}^t & \\ 0 & U^t & I_{N+1} \\ 0 & -\mathbf{b}^t & \\ \hline 0 & Y^t & I_{N+1} \end{array}\right] = \left[\begin{array}{c|cc} R(-1) & * & \\ \hline 0^t & k(-1) & k(-1)\mathbf{x}^t \\ 0 & * & \end{array}\right]. \tag{49}$$

Proof. The proof follows immediately by noticing that the arrangement of the rows in $[U_B \quad Y_B]^t$ and the choice of the sequence of rotations in Φ gives rise to the same angles and accumulation of product terms as in the product sequences in (46). □

3.3.3. Contractivity. The use of hyperbolic rotations deserves special care. When applying a hyperbolic plane rotation to a vector $[x_1 \quad x_2]$

$$[x_1 \quad x_2]\begin{bmatrix} \cosh(\alpha) & \sinh(\alpha) \\ \sinh(\alpha) & \cosh(\alpha) \end{bmatrix} = [y_1 \quad y_2],$$

the length of the vector $[x_1 \quad x_2]$ is preserved when measured with respect to the metric $\Sigma = 1 \oplus -1$, i.e., $x_1^2 - x_2^2 = y_1^2 - y_2^2$. However, the elements y_1 and y_2 may be very large in this equation, even when x_1 and x_2 happen to be small. Such behavior leads to numerical problems. Therefore, it is important that we guarantee in the generalized Schur algorithm that excessive element growth does not occur. This is indeed the case, as stated by the following theorem.

THEOREM 3. *Let $U_0 = U_B$ and $Y_0 = Y_B$ in recursion* (30). *Then, since the absolute value of the elements of U_B and Y_B is bounded by* 1 *in* (48a) *and* (48b), *respectively, the absolute value of the elements of U_j and Y_j in* (30) *is also bounded by* 1.

Proof. We refer to Appendix C for the proof. □

Referring back to Theorem 1 in § 2, we observe that the recursion stated there is contained in the recursion on U_B^t and Y_B^t in (48a) and (48b). Hence, the proof of Theorem 1 follows directly from Lemma 1 and Theorem 3.

3.3.4. Economized Schur–Cholesky factorization. In case the coefficients matrix is a Toeplitz matrix, e.g., $A = [a_{ij} \mid a_{ij} = a_{i+1,j+1}] \in \mathbf{R}^{N \times N}$, or a symmetric positive definite matrix close to Toeplitz, the number of computations involved in the generalized Schur algorithm can be drastically reduced as is shown by the results in [Dewi 1978] and [Lev 1984]. Let the matrix Z be the lower shift matrix

$$Z = \begin{bmatrix} 0 & & & \\ 1 & \ddots & & \\ & \ddots & & \\ & & 1 & 0 \end{bmatrix} \tag{50}$$

and define the so-called box operator on the matrix A as follows [Kail 1979]

$$\square A := A - ZAZ^t. \tag{51}$$

Then $\square A$ can always be decomposed as follows [Kail 1979]:

$$\square A = GSG^t, \tag{52}$$

where the signature matrix S is of the form $S = I_p \oplus -I_q$, with $p+q = r \leqq N$. The matrix $G^t \in \mathbf{R}^{r \times N}$ is such that there exists an S-orthogonal matrix P, such that

$$(GP)^t = \begin{bmatrix} & \delta^t & \\ 0 & & \\ \vdots & & \Gamma^t \\ 0 & & \end{bmatrix}, \tag{53}$$

where $\delta^t = [\delta_1 \cdots \delta_N]$ and $\Gamma = [\gamma_{ij}] \in \mathbf{R}^{(N-1)\times(r-1)}$.

Remark. In case the matrix A is a Toeplitz matrix, we have $S = 1 \oplus -1$ and G^t is the following matrix (assuming that the matrix A is normalized such that $a_{ii} = 1$):

$$G^t = \begin{bmatrix} 1 & a_{12} \cdots a_{1N} \\ 0 & a_{12} \cdots a_{1,N-1} \end{bmatrix}. \tag{54}$$

The rows of the Cholesky factor L^t of the matrix A are computed through elimination of the columns of the matrix Γ^t in $(GP)^t$, with S-orthogonal rotations. The entries of the first column of Γ^t can be eliminated by shifting Γ^t one column to the left and applying $(p-1)$ circular rotations in the planes $(1,2), (1,3), \cdots, (1,p)$, such that the first $p-1$ entries of the first column of Γ^t are eliminated. Next, the remaining entries can be eliminated by applying $(r-p)$ hyperbolic rotations in the planes $(1, p+1)$, $(1, p+2), \cdots, (1, r)$. As can be shown, the first row of the Cholesky factor L^t of the matrix A is equal to the modification of the row δ^t after the sequence of rotations. Next, the new matrix $\bar{\Gamma}^t$ can be shifted one column to the left and again such a sequence of appropriate $(p-1)$ circular and $(r-p)$ hyperbolic rotations can be applied such that the elements in the second column of $\bar{\Gamma}^t$ are eliminated. The second row of the Cholesky factor is now equal to the modification of the row vector δ^t, which results after this second sequence of rotations. The shift-and-rotate procedure is repeated until all columns of Γ^t have been eliminated. The inverse Cholesky factor L^{-1} of the matrix A is found by applying the rotations to the column vector

$$[1 \quad \underbrace{0 \cdots 0}_{r-2} \quad 1]^t.$$

The number of operations is reduced to $O(rN^2)$, compared to the $O(N^3)$ operations of the generalized Schur algorithm.

Combining these Cholesky factorizations with the possible factorizations in (13) and (15) results in efficient solvers as shown in Fig. 5. An example of such a solver can be found in [Jain 1986] for an application in speech coding.

Note that the number of rotors in Fig. 5 (denoted by rectangles) for the elimination of a column of Γ^t is equal to $r-1$, where r may range from 2 to N. The case $r=2$

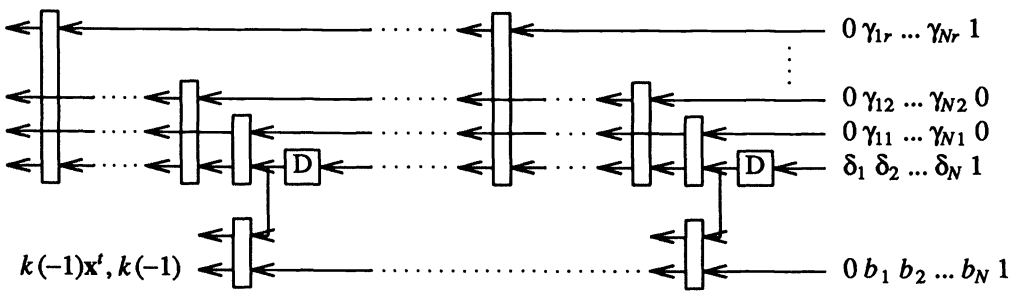


FIG. 5. *Solver with economized Cholesky factorization.*

corresponds to the Toeplitz case and $r = N$ leads to the full complexity of the generalized Schur algorithm. In Fig. 5 the entries of the factors L^t and L^{-1} appear at the taps after the delays marked by "D."

4. Numerical stability of the algorithms. In the numerical stability analysis of the feed-forward direct method of § 3, the central issue is the study of the propagation of roundoff errors due to finite precision. In-depth numerical stability analyses of the LU factorization with linear rotations, the QR factorization with circular rotations, and the LL^t factorization with hyperbolic rotations can be found in, for instance, [Golu 1983], [Gent 1975], and [Bult 1981], respectively. Since the methods of § 3 can all be deduced to either of these three types of factorizations, an in-depth analysis of the numerical properties of the methods shall not be given here. Instead, an intuitive explanation shall be given.

The result of an operation f applied to a vector $\mathbf{u}$ in finite precision arithmetic will be denoted as $fl(f(\mathbf{u}))$. The relation between the finite precision and exact result is the following:

$$fl(f(\mathbf{u})) = (1+\eta)f(\mathbf{u}), \tag{55}$$

where η is bounded by the machine precision. Notice that (55) is a generalization of the case where f denotes either a scalar addition, subtraction, multiplication, or division, as found in [Golu 1983]. Let f_{ij}, $i, j = 1, 2$ be the functions that determine the ij-entries of a plane rotation Θ from a vector $\mathbf{u}$. Let the matrix $\bar{\Theta}$ denote the matrix that results when the entries are computed in finite precision arithmetic. Then, $\bar{\Theta}$ is represented as follows:

$$\bar{\Theta} = \begin{bmatrix} fl(f_{11}(\mathbf{u})) & fl(f_{12}(\mathbf{u})) \\ fl(f_{21}(\mathbf{u})) & fl(f_{22}(\mathbf{u})) \end{bmatrix} = \begin{bmatrix} (1+\eta_{11})f_{11}(\mathbf{u}) & (1+\eta_{12})f_{12}(\mathbf{u}) \\ (1+\eta_{21})f_{21}(\mathbf{u}) & (1+\eta_{22})f_{22}(\mathbf{u}) \end{bmatrix} \tag{56}$$

where the η_{ij}, $i, j = 1, 2$, are bounded by the machine precision. Putting $\eta = \max(|\eta_{11}|, |\eta_{12}|, |\eta_{21}|, |\eta_{22}|)$ we find the following upper bound for the Frobenius norm of the matrix $\bar{\Theta}$:

$$\|\bar{\Theta}\|_F \leqq (1+\eta)\left(\sum_{i=1}^{2}\sum_{j=1}^{2} f_{ij}^2(\mathbf{u})\right)^{1/2} = (1+\eta)\|\Theta\|_F. \tag{57}$$

Now suppose $\bar{\Theta}_k = [\bar{\theta}_{ij}^{(k)}]$, $k = 1, \cdots, n$, are plane rotations computed in finite precision arithmetic. Then applying $\bar{\Theta}_1$ in finite precision arithmetic to a vector $\mathbf{x} = [x_1 \quad x_2]^t$ gives

$$\begin{aligned} fl(\bar{\Theta}_1\mathbf{x}) &= \begin{bmatrix} [\bar{\theta}_{11}^{(1)}x_1(1+\varepsilon_{11}) + \bar{\theta}_{12}^{(1)}x_2(1+\varepsilon_{12})](1+\gamma_1) \\ [\bar{\theta}_{21}^{(1)}x_1(1+\varepsilon_{21}) + \bar{\theta}_{22}^{(1)}x_2(1+\varepsilon_{22})](1+\gamma_2) \end{bmatrix} \\ &= \begin{bmatrix} (1+\gamma_1) & \\ & (1+\gamma_2) \end{bmatrix} \begin{bmatrix} (1+\varepsilon_{11})\bar{\theta}_{11}^{(1)} & (1+\varepsilon_{12})\bar{\theta}_{12}^{(1)} \\ (1+\varepsilon_{21})\bar{\theta}_{21}^{(1)} & (1+\varepsilon_{22})\bar{\theta}_{22}^{(1)} \end{bmatrix} \mathbf{x}. \end{aligned}$$

Putting $\eta_1 = \max(|\varepsilon_{11}|, |\varepsilon_{12}|, |\varepsilon_{21}|, |\varepsilon_{22}|, |\gamma_1|, |\gamma_2|)$ and using the inequalities $\|AB\|_2 \leqq \|A\|_2\|B\|_2$, $\|A\mathbf{x}\|_2 \leqq \|A\|_2\|\mathbf{x}\|_2$, $\|A\|_F \geqq \|A\|_2$, we obtain the following bound for the l_2-norm of $fl(\bar{\Theta}_1\mathbf{x})$:

$$\|fl(\bar{\Theta}_1\mathbf{x})\|_2 \leqq (1+\eta_1)^2 2^{1/2}\|\bar{\Theta}_1\|_F\|\mathbf{x}\|_2.$$

Using the result in (57), with $\xi_1 = \max(\eta, \eta_1)$, we obtain

$$\|fl(\bar{\Theta}_1\mathbf{x})\|_2 \leqq (1+\xi_1)^3 2^{1/2}\|\Theta_1\|_F\|\mathbf{x}\|_2. \tag{58}$$

By induction we can prove that

$$\|fl(\bar{\Theta}_n \cdots fl(\bar{\Theta}_1 \mathbf{x}) \cdots)\|_2 \leqq 2^{n/2}(1+\xi_n)^{3n}\left(\prod_{i=1}^{n} \|\Theta_i\|_F\right)\|\mathbf{x}\|_2, \; n = 1, 2, \cdots$$

$$\xi_n = \max(\eta_n, \xi_{n-1}, \eta), \; \xi_0 = 0.$$

Indeed, assume that the induction hypothesis

$$\|fl(\bar{\Theta}_{n-1} \cdots fl(\bar{\Theta}_1 \mathbf{x}) \cdots)\|_2 \leqq 2^{n-1/2}(1+\xi_{n-1})^{3(n-1)}\left(\prod_{i=1}^{n-1} \|\Theta_i\|_F\right)\|\mathbf{x}\|_2$$

$$\xi_{n-1} = \max(\eta_{n-1}, \xi_{n-2}, \eta)$$

is valid. Then

$$fl(\bar{\Theta}_n \cdots fl(\bar{\Theta}_1 \mathbf{x}) \cdots) = \begin{bmatrix} (1+\gamma_1) & \\ & (1+\gamma_2) \end{bmatrix} \times \begin{bmatrix} (1+\varepsilon_{11})\theta_{11}^{(n)} & (1+\varepsilon_{12})\theta_{12}^{(n)} \\ (1+\varepsilon_{21})\theta_{21}^{(n)} & (1+\varepsilon_{22})\theta_{22}^{(n)} \end{bmatrix} fl(\bar{\Theta}_{n-1} \cdots fl(\bar{\Theta}_1 \mathbf{x}) \cdots).$$

Putting $\eta_n = \max(|\varepsilon_{11}|, |\varepsilon_{12}|, |\varepsilon_{21}|, |\varepsilon_{22}|, |\gamma_1|, |\gamma_2|)$ and using the inequalities, we obtain

$$\begin{aligned} \|fl(\bar{\Theta}_n \cdots fl(\bar{\Theta}_1 \mathbf{x}) \cdots)\|_2 &\leqq 2^{1/2}(1+\eta_n)^2 \|\bar{\Theta}_n\|_F \|fl(\bar{\Theta}_{n-1} \cdots fl(\bar{\Theta}_1 \mathbf{x}) \cdots)\|_2 \\ &\leqq 2^{n/2}(1+\xi_{n-1})^{3(n-1)}(1+\eta_n)^2 \|\bar{\Theta}_n\|_F \left(\prod_{i=1}^{n-1} \|\Theta_i\|_F\right)\|\mathbf{x}\|_2. \end{aligned}$$

Using the result in (57), with $\xi_n = \max(\eta_n, \xi_{n-1}, \eta)$, we obtain

$$\begin{aligned} \|fl(\bar{\Theta}_n \cdots fl(\bar{\Theta}_1 \mathbf{x}) \cdots)\|_2 &\leqq 2^{n/2}(1+\xi_n)^{3n}\left(\prod_{i=1}^{n} \|\Theta_i\|_F\right)\|\mathbf{x}\|_2 \\ &\leqq 2^{n/2}(1+3n\xi_n)\left(\prod_{i=1}^{n} \|\Theta_i\|_F\right)\|\mathbf{x}\|_2 + O(\xi_n^2) \\ &\leqq 2^{n/2}\left(\prod_{i=1}^{n} \|\Theta_i\|_F\right)\|\mathbf{x}\|_2 + e_n. \end{aligned}$$

where e_n is an error term

$$e_n \leqq 2^{n/2} 3n\xi_n \left(\prod_{i=1}^{n} \|\Theta_i\|_F\right)\|\mathbf{x}\|_2 + O(\xi_n^2).$$

And from the inequality $\|A\|_F \leqq N^{1/2}\|A\|_2 = N^{1/2}\lambda(A)$, where $\lambda(A)$ is the spectral radius of the $N \times N$ matrix A, we find

$$e_n \leqq 2^n 3n\xi_n \left(\prod_{i=1}^{n} \lambda(\Theta_i)\right)\|\mathbf{x}\|_2 + O(\xi_n^2). \tag{59}$$

This expression shows that the propagation of the roundoff error is amplified by the spectral radius of the rotations Θ_i. In case the Θ_i are orthogonal, the spectral radii are equal to 1 and there is no amplification of roundoff errors due to the Θ_i. This makes the feed-forward direct method of § 3 with only circular rotations numerically robust.

However, in case the Θ_i are either linear or hyperbolic rotations, the spectral radii are, respectively, $[(1+\frac{1}{2}|\alpha_i|^2)+((1+\frac{1}{2}|\alpha_i|^2)^2-1)^{1/2}]$ and $(1+|\tanh(\alpha_i)|)\times(1-\tanh^2(\alpha_i))^{-1/2}$. In case of linear rotations, the $|\alpha_i|$ become large if the diagonal elements of the matrix A are small, relative to the *off*-diagonal elements. In such cases the spectral radius of Θ_i is approximately equal to $|\alpha_i|^2$. And this already happens for relatively small α_i's, because of the quadratic term $(1+\frac{1}{2}|\alpha_i|^2)^2$. On the other hand, if the matrix $A=[a_{ij}]$ is positive definite, then the $|\alpha_i|<1$ due to the fact that $a_{ii}>|a_{ij}|$, and the amplification of roundoff errors is less.

From § 3 we know that the hyperbolic rotations can only be used when the matrix B in (47) is positive definite and in that case $|\tanh(\alpha_i)|<1$ for all i. From (59) the amplification of the roundoff error due to the hyperbolic rotations is given by

$$\prod_{i=1}^{n}(1+\tanh(\alpha_i))(1-\tanh^2(\alpha_i))^{-1/2}.$$

The magnitude of this product depends on how close the matrix A is to singularity, because the closer to singularity, the closer to 1 the magnitude of $\tanh(\alpha_i)$ will be for some i. This is easily understood from the fact that the determinant of the matrix B is a product of the terms $(1-|\tanh(\alpha_i)|^2)^{1/2}$, as follows from the fact that the diagonal elements of the Cholesky factor L_B of matrix B are products of the factors $(1-\tanh^2(\alpha_i))^{1/2}$ (this is easily derived from the calculation of L_B^{-1}). Hence, the coefficients $\tanh(\alpha_i)$ are a measure of the singularity of the matrix B in (47). Therefore, (59) gives an indication of how far from singularity a matrix can be in finite precision arithmetic, before the factorization will make no sense due to roundoff errors.

5. Semidirect nature of the generalized Schur algorithm. The generalized Schur algorithm of § 3.3 can be shown to be a semidirect method for solving positive definite systems of linear equations. (This was shown in [Neho 1983] for the Levinson algorithm [Orfa 1985].) This means that we may prematurely abort the recursion in (30) and still obtain an approximation for the solution. Let $A\in\mathbf{R}^{N\times N}$ be a symmetric positive definite matrix with its diagonal entries equal to 1. And let $\mathbf{b}\in\mathbf{R}^N$ be such that $A-\mathbf{b}\mathbf{b}^t>0$, and $\mathbf{p}_i$ is the ith row of the inverse of the Cholesky factor L of the matrix A. Then, the computations in (30), with $U_0=U_B$ and $Y_0=Y_B$, can also be stated in terms of the following recursion.

$x_0=0$
for $i=0$ **to** $N-1$ **do**

(60) *compute* $\mathbf{p}_{i+1}$ *such that* $\mathbf{p}_{i+1}^t A\mathbf{p}_j=\delta_{i+1,j}$, $j=1,\cdots,i+1$

(61) $\mathbf{r}_i=A\mathbf{x}_i-\mathbf{b}$,

(62) $\alpha_{i+1}=\mathbf{p}_{i+1}^t\mathbf{r}_i$,

(63) $\mathbf{x}_{i+1}=\mathbf{x}_i+\alpha_{i+1}\mathbf{p}_{i+1}$

This recursion is very similar to the conjugate gradient method [Golu 1983]. The difference being that the vectors $\mathbf{p}_i$ are linear combinations of the Lanczos vectors of the matrix A in case of the conjgate gradient method. Notice that the above recursion can be terminated when $\mathbf{r}_i=0$. This may happen before $i=N-1$, if the solution $\mathbf{x}$ lies in a subspace of the space spanned by the columns of the inverse of the Cholesky factor of A. For instance, if $\mathbf{x}$ lies in the space spanned by $\mathbf{p}_1$, then the residual $\mathbf{r}_i$, $i=1,\cdots,N-1$ is zero and the recursion can stop at $i=1$.

To show that the computations in recursion (60)-(63) are indeed those of recursion (30) with $U_0 = U_B$ and $Y_0 = Y_B$, the following approach is taken. First we show that recursion (61)-(63) can be embedded in recursion (6a)-(6b). And since we can compute the $\mathbf{p}_{i+1}$ by recursion (30), it follows that the computations in (60)-(63) are identical to the computations in recursion (30) when $U_0 = U_B$ and $Y_0 = Y_B$.

Embedding of residual equation (61) in a vector-matrix product gives

$$[1\,|\,\mathbf{x}_i^t]\begin{bmatrix} 1 & -\mathbf{b}^t \\ -\mathbf{b} & A \end{bmatrix} = [(1-\mathbf{x}_i^t\mathbf{b})\,|\,\mathbf{r}_i^t]. \tag{64}$$

Ultimately, we would like to solve the symmetric system of linear equations

$$\begin{aligned}[1\,|\,\mathbf{x}^t]\begin{bmatrix} 1 & -\mathbf{b}^t \\ -\mathbf{b} & A \end{bmatrix} &= [(1-\mathbf{x}^t\mathbf{b})\,|\,0\cdots 0] \\ &= [(1-\|\mathbf{x}\|_A^2)\,|\,0\cdots 0],\end{aligned} \tag{65}$$

where

$$\|\mathbf{x}\|_A^2 = \mathbf{x}^t A\mathbf{x}. \tag{66}$$

Since the augmented matrix in (65) is positive definite, it follows that $(1-\|\mathbf{x}\|_A^2) > 0$. Hence, dividing both sides of (65) by $(1-\|\mathbf{x}\|_A^2)^{1/2}$ is allowed and gives

$$[(1-\|\mathbf{x}\|_A^2)^{-1/2}\,|\,\bar{\mathbf{x}}^t]B = [(1-\|\mathbf{x}\|_A^2)^{1/2}\,|\,0\cdots 0], \tag{67a}$$

with

$$B = \begin{bmatrix} 1 & -\mathbf{b}^t \\ -\mathbf{b} & A \end{bmatrix} = L_B L_B^t \tag{67b}$$

and

$$\bar{\mathbf{x}}^t = (1-\|\mathbf{x}\|_A^2)^{-1/2}\mathbf{x}^t. \tag{67c}$$

Equation (67a) can be solved using a Levinson recursion on B [Orfa 1985] as follows.

Normalize (64) through division by $(1-\mathbf{x}_i^t\mathbf{b})^{1/2}$

$$[(1-\mathbf{x}_i^t\mathbf{b})^{-1/2}\,|\,\bar{\mathbf{x}}_i^t]\begin{bmatrix} 1 & -\mathbf{b}^t \\ -\mathbf{b} & A \end{bmatrix} = [(1-\mathbf{x}_i^t\mathbf{b})^{1/2}\,|\,\bar{\mathbf{r}}_i^t], \tag{68a}$$

where

$$\bar{\mathbf{x}}_i^t = (1-\mathbf{x}_i^t\mathbf{b})^{-1/2}\mathbf{x}_i^t \tag{68b}$$

and

$$\bar{\mathbf{r}}_i^t = (1-\mathbf{x}_i^t\mathbf{b})^{-1/2}\mathbf{r}_i^t. \tag{68c}$$

For the moment assume that

$$[(1-\mathbf{x}_i^t\mathbf{b})^{-1/2}\,|\,\bar{\mathbf{x}}_i^t] = [\mathbf{y}_i^t\,|\,0\underset{N-i}{\cdots}0], \tag{69a}$$

with

$$\mathbf{y}_i^t = [y_{i,0}\; y_{i,1}\cdots y_{i,i}] \tag{69b}$$

and

$$y_{i,0} = (1-\mathbf{x}_i^t\mathbf{b})^{-1/2}. \tag{69c}$$

Also assume that

$$[\mathbf{y}_i^t\,|\,0\underset{N-i}{\cdots}0]B = [(1-\mathbf{x}_i^t\mathbf{b})^{1/2}\,|\,0\underset{i}{\cdots}0\;\; \bar{r}_i(1)\cdots\bar{r}_i(N-i)]. \tag{70}$$

It will follow next, by induction, that both assumptions are true.

Surely (70) holds for $i=0$ (i.e., putting $x_0=0$). Now, let $\bar{\mathbf{p}}_{i+1}$ denote the $(i+1)$st row of L^{-1}, without the zero elements of $\mathbf{p}_{i+1}$. Then $\mathbf{y}_i$ is updated by the rule

$$\mathbf{y}_{i+1}^t=([\mathbf{y}_i^t|0]+\rho_{i+1}[0|\bar{\mathbf{p}}_{i+1}])(1-\rho_{i+1}^2)^{-1/2}. \tag{71}$$

The value of ρ_{i+1} then follows from the residual equation

$$\begin{aligned}&[[\mathbf{y}^t\ 0|0\underbrace{\cdots}_{N-i-1}0]+\rho_{i+1}[0\ \bar{\mathbf{p}}_{i+1}|0\underbrace{\cdots}_{N-i-1}0]]B\\ &=[(1-\mathbf{x}_i^t\mathbf{b})^{1/2}|0\underbrace{\cdots}_{i}0\ \bar{r}_i(1)\cdots\bar{r}_i(N-i)]\\ &\quad+\rho_{i+1}[\gamma_{i+1}\ 0\underbrace{\cdots}_{i}0\ l_{i+1}(1)\cdots l_{i+1}(N-i)]\\ &=[(1-\mathbf{x}_{i+1}^t\mathbf{b})^{1/2}|0\underbrace{\cdots}_{i+1}0\ \bar{r}_{i+1}(1)\cdots\bar{r}_{i+1}(N-i-1)].\end{aligned} \tag{72}$$

The elements $l_{i+1}(k)$ are the nonzero elements of the $(i+1)$st row of L^t. From (72) it then follows that

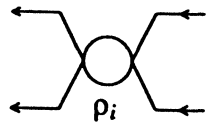
$$\rho_{i+1}=-\frac{\bar{r}_i(1)}{l_{i+1}(1)}. \tag{73}$$

And from the positivity of the matrix B it follows that

$$|\rho_{i+1}|<1. \tag{74}$$

A flow graph representation of the Levinson recursion in (72) is depicted in Fig. 6, for an initial guess of $x_0=0$.

The symbol



denotes the 2×2 hyperbolic rotation with $\rho_i=\tanh(\alpha_i)$ in (4b) ($m=-1$). This sequence hyperbolic rotations is exactly the same as the one in recursion (6a)-(6b). Since (64)

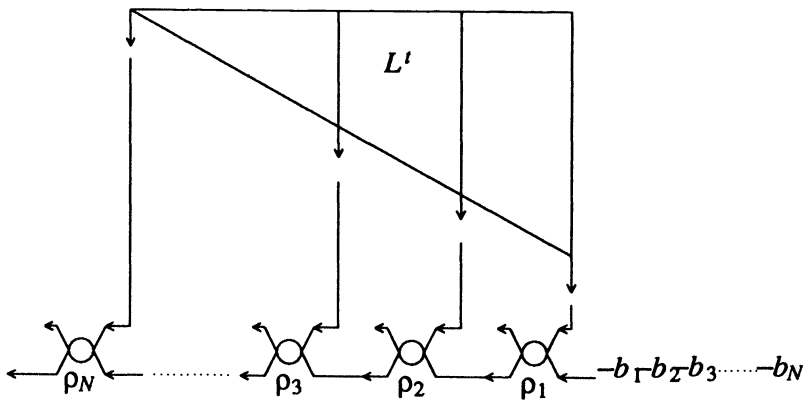


FIG. 6. *Graphical representation of the Levinson recursion in* (72).

is an embedding of the residual equation (61) and (7) is equal to (64), it must be that the update rule for $\mathbf{x}_{i+1}$ in (63) is equivalent to (71), with $\rho_{i+1} = \alpha_{i+1}$,

$$\mathbf{x}_{i+1}^t = [y_{1,t} \cdots y_{i,i} | \underset{N-i}{0 \cdots 0}] + \rho_{i+1}[\bar{\mathbf{p}}_{i+1} | \underset{N-i-1}{0 \cdots 0}], \qquad \mathbf{x}_i^t = [y_{i,1} \cdots y_{i,i} | \underset{N-i}{0 \cdots 0}],$$

$$\mathbf{p}_{i+1}^t = [\bar{\mathbf{p}}_{i+1} | \underset{N-i-1}{0 \cdots 0}] \quad \text{and} \quad \mathbf{r}_i^t = [\underset{i+1}{0 \cdots 0}\ \bar{r}_{i+1}(1) \cdots \bar{r}_{i+1}(N-i-1)].$$

And because recursion (30) combines the computations of recursion (6a)-(6b) and the computations of the rows of L and L^{-1}, for $U_0 = U_B$ and $Y_0 = Y_B$, it follows that the computations in recursion (30) are equivalent to those of recursion (60)-(63). And we conclude that recursion (30) is in fact a semidirect method.

As was noticed before, the recursion in (60)-(63) expresses the conjugate gradient method, when the $\mathbf{p}_i$ are chosen to be linear combinations of the Lanczos vectors of the matrix A. If the Lanczos procedure is prematurely terminated, the resulting tridiagonal matrix T has extreme eigenvalues which approximate the extreme eigenvalues of the matrix A and the obtained solution vector is an approximation to the exact solution vector [Golu 1983].

In contrast, premature termination of the generalized Schur recursion results in a maximum entropy extension approximation of the inverse of the coefficient matrix A (see [Dewi 1981] for the Toeplitz case and [Dewi 1987] for the general case). In fact, the approximative inverse is banded, such that its inverse coincides with the matrix A within the band [Dym 1981]. Hence, early termination of (60)-(63) results in the solution of a system of linear equations, which coincides in the band with the original set of linear equations.

6. Extensions. The algorithms in § 3 can be straightforwardly extended to handle more general matrix arithmetic. Straightforward generalizations can be given for the following computations:

(1) Computing an unknown matrix X from the matrix equation $AX = B$, where $A \in \mathbf{R}^{N \times N}$, $X \in \mathbf{R}^{N \times q}$, and $B \in \mathbf{R}^{N \times q}$;

(2) Computing an inner product/accumulation $\mathbf{b}^t\mathbf{c} + d$;

(3) Computing a vector-matrix product/accumulation $\mathbf{b}^t C^t + \mathbf{d}^t$;

(4) Computing a matrix-matrix product/accumulation $BC^t + D^t$.

We shall consider two cases. The first case presents a generalization based on an LU or LQ factorization of the matrix A. The second case presents a generalization based on an LL^t factorization of the matrix A.

Case 1. Suppose we have a nonsingular matrix $A \in \mathbf{R}^{N \times N}$ and matrices $B \in \mathbf{R}^{N \times q}$, $C \in \mathbf{R}^{r \times N}$, and $D \in \mathbf{R}^{r \times q}$. Let $\mathbf{b}_1, \cdots, \mathbf{b}_q$ and $\mathbf{d}_1, \cdots, \mathbf{d}_q$ denote the columns of the matrices B and D, respectively. Let $\mathbf{m}$ be a vector $\mathbf{m} = [m_1 \cdots m_q]^t$, $m_i \in \{0, 1, -1\}$, $i = 1, \cdots, q$. We show that sequences of rotations as in (18) and (22) can be applied to the matrices

$$W_1 = \begin{bmatrix} A^t & C^t \\ -\mathbf{b}_1^t & \mathbf{d}_1^t \end{bmatrix}, \cdots, W_q = \begin{bmatrix} A^t & C^t \\ -\mathbf{b}_q^t & \mathbf{d}_q^t \end{bmatrix},$$

such that we can combine the results into a matrix $E_{\mathbf{m}}$:

$$E_{\mathbf{m}} = \operatorname{diag}(k_1(m_1) \cdots k_q(m_q))(B^t A^{-t} C^t + D^t). \tag{75}$$

The sequences of rotations in (18) and (22) can be reordered to

$$\Theta(m) \begin{bmatrix} X^{-1} & 0 \\ 0^t & 1 \end{bmatrix},$$

where $\Theta(m) = \prod_{i=1}^{\tilde{N}} \Theta_i(m)$ (see (4a)-(4b)) and

$$\begin{bmatrix} X^{-1} & 0 \\ 0^t & 1 \end{bmatrix} = \prod_{j=1}^{\tilde{N}-1} \prod_{i=j}^{\tilde{N}-1} U_{ij},$$

or

$$\begin{bmatrix} X^{-1} & 0 \\ 0^t & 1 \end{bmatrix} = \prod_{j=1}^{\tilde{N}-1} \prod_{i=j}^{\tilde{N}-1} Q_{ij},$$

respectively. U_{ij} and Q_{ij} are as given in (17) and (21), respectively. Let us denote by $\Theta(m_i)$ the matrix $\Theta(m)$ in the product

$$\Theta(m)\begin{bmatrix} X^{-1} & 0 \\ 0^t & 1 \end{bmatrix},$$

when applied to W_i

$$\Theta(m_i)\begin{bmatrix} X^{-1} & 0 \\ 0^t & 1 \end{bmatrix}\begin{bmatrix} A^t & C^t \\ -\mathbf{b}_i^t & \mathbf{d}_i^t \end{bmatrix} = \begin{bmatrix} R_i & * \\ 0^t & \mathbf{e}_i^t \end{bmatrix},$$

where $[R_i^t | 0]^t$, with $R_i \in \mathbf{R}^{N\times N}$, is the upper triangular factor of $[A | -\mathbf{b}_i]^t$. Writing the $(N+1)\times N$ matrix $\Theta(m_i)$ as

$$\Theta(m_i) = \begin{bmatrix} \Theta_{11}(m_i) & \theta_{12}(m_i) \\ \theta_{21}^t(m_i) & k_i(m_i) \end{bmatrix}, \qquad \Theta_{11}(m_i) \in \mathbf{R}^{N\times N},$$

we find

$$\theta_{21}^t(m_i) = k_i(m_i)\mathbf{b}_i^t A^{-t} X$$

and

$$\mathbf{e}_i^t = \theta_{21}^t(m_i) X^{-1} C^t + k_i(m_i)\mathbf{d}_i^t.$$

Hence, it follows that

$$\mathbf{e}_i^t = k_i(m_i)(\mathbf{b}_i^t A^{-t} C^t + \mathbf{d}_i^t).$$

Thus, for $i = 1, \cdots, q$ we can combine the results $\mathbf{e}_1^t, \cdots, \mathbf{e}_q^t$ as the rows of the matrix $E_{\mathbf{m}}$ in (75).

Case 2. A similar result can be obtained for a *Schur-Cholesky* factorization of the matrix A (in case A is positive definite symmetric). Let the following be given. $A \in \mathbf{R}^{N\times N}$ is a positive definite matrix, $B \in \mathbf{R}^{N\times q}$ is a matrix with columns $\mathbf{b}_1, \cdots, \mathbf{b}_q$, such that $A - \mathbf{b}_i\mathbf{b}_i^t > 0$, $i = 1, \cdots, q$, $\bar{C} \in \mathbf{R}^{r\times 2N}$ is the following augmented matrix: $\bar{C} = [C | C]$ with $C \in \mathbf{R}^{r\times N}$, and $D \in \mathbf{R}^{r\times q}$ is a matrix with columns $\mathbf{d}_1, \cdots, \mathbf{d}_q$. Moreover, let $G = [U | Y]$, where U and Y are the lower and strictly lower triangular part of the matrix A, respectively. We show that sequences of rotations as in (46) can be applied to the matrices

$$W_1 = \begin{bmatrix} G^t & \bar{C}^t \\ -\mathbf{b}_1^t & \mathbf{d}_1^t \end{bmatrix}, \cdots, W_q = \begin{bmatrix} G^t & \bar{C}^t \\ -\mathbf{b}_q^t & \mathbf{d}_q^t \end{bmatrix},$$

such that we can combine the results into a matrix $E_{\mathbf{m}}$ as given in (75).

The sequence of rotations in (46) can be reordered to

$$[\Theta(m)]\begin{bmatrix} \Phi & 0 \\ 0^t & 1 \end{bmatrix},$$

where $[\Theta(m)]=\prod_{i=1}^{\tilde{N}}[\Theta_i(m)]$ (see (45)) and Φ is as given in (26a)-(26b). Let us denote by $[\Theta(m_i)]$ the matrix $[\Theta(m)]$ in the product

$$[\Theta(m)]\begin{bmatrix}\Phi & 0\\ 0^t & 1\end{bmatrix},$$

when applied to W_i

$$[\Theta(m_i)]\begin{bmatrix}\Phi & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}G^t & \bar{C}^t\\ -\mathbf{b}_i^t & \mathbf{d}_i^t\end{bmatrix}=\begin{bmatrix}\bar{R}_i & *\\ 0^t & \mathbf{e}_i^t\end{bmatrix},$$

where $\bar{R}_i=[R_i|O]^t$ with $R_i\in\mathbf{R}^{N\times N}$ being the Cholesky factor of the matrix $A-\mathbf{b}_i\mathbf{b}_i^t$. Writing the $(2N+1)\times(2N+1)$ matrix $[\Theta(m_i)]$ as

$$[\Theta(m_i)]=\begin{bmatrix}\Theta_{11}(m_i) & O & \theta_{12}(m_i)\\ O & I_N & O\\ \theta_{21}^t(m_i) & O & k_i(m_i)\end{bmatrix},\qquad \Theta_{11}(m_i)\in\mathbf{R}^{N\times N},$$

we find

$$\theta_{21}^t(m_i)L^t=k_i(m_i)\mathbf{b}_i^t$$

and because

$$\Phi=\begin{bmatrix}L^{-1} & O\\ O & R^{-1}\end{bmatrix}\begin{bmatrix}U & -Y\\ -Y^t & U^t\end{bmatrix}$$

(see Corollary 1) we get:

$$\mathbf{e}_i^t=\theta_{21}^t(m_i)L^{-1}C^t+k_i(m_i)\mathbf{d}_i^t.$$

Hence, it follows that

$$\mathbf{e}_i^t=k_i(m_i)(\mathbf{b}_i^tA^{-1}C^t+\mathbf{d}_i^t).$$

Thus, for $i=1,\cdots,q$, we can combine the results $\mathbf{e}_1^t,\cdots,\mathbf{e}_q^t$ as the rows of the matrix $E_{\mathbf{m}}$ in (75). Hence, with the appropriate choice of the matrices A, B, C, and D we can realize each of the four above-listed computations.

An example of a systolic implementation for the computation of (75), for $\mathbf{m}=0$, is shown in Fig. 7.

The rows of processing elements in the rectangular part transmit the data coming from the triangular part unchanged, along the vertical interconnections to their upper neighbor. This has been indicated by the arrows across a processing element.

7. Conclusions. Although it seemed there was nothing new left anymore about direct methods for solving nonsingular systems of linear equations, it was surprising to learn that simple and effective feed-forward direct methods can still be invented, as was explained in §§ 2 and 3. The power of these methods is expressed in their high degree of parallelism and pipelining. This allows fast execution on systolic arrays, despite the fact that the amount of input data is nearly doubled due to the need for processing an identity matrix in (18) and (22) and a block identity matrix in (46). Both systolic arrays in Figs. 3 and 4 are roughly two times faster than the combination of the arrays in Fig. 1(a) and (b). This combination requires $7N-5$ time steps to solve $A\mathbf{x}=\mathbf{b}$ ($A\in\mathbf{R}^{N\times N}$) as was explained in the introduction. A generalization of the systolic array in Fig. 3 shows that $A\mathbf{x}=\mathbf{b}$ ($A\in\mathbf{R}^{N\times N}$) is solved in $4N$ time steps on this array. A generalization of the systolic array in Fig. 4 shows that $A\mathbf{x}=\mathbf{b}$ is solved in $3N+1$ time steps. The systolic arrays for the methods of § 3 are simple and with complexity

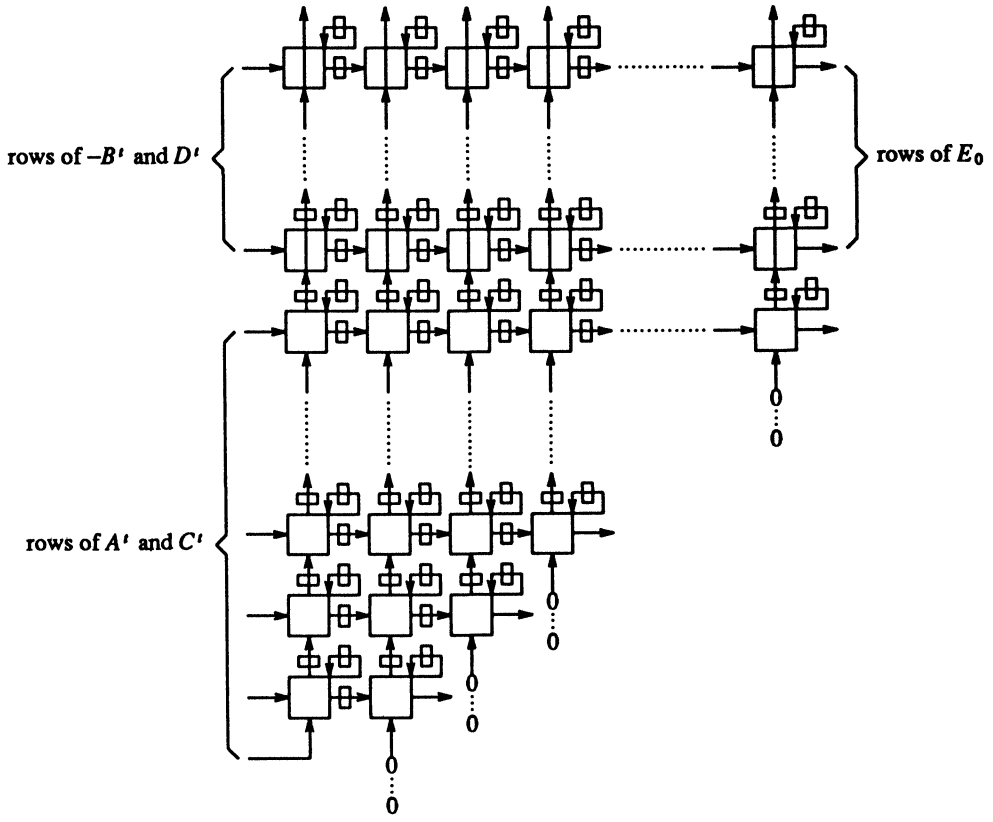


FIG. 7. *Systolic array for expressions* $(B^tA^{-t}C^t + D^t)$.

in the order of the complexity of a *QR* factorization. The feed-forward method which uses only Givens rotations (see (23)) is a numerically stable and robust method for the complete class of nonsingular systems of linear equations. Moreover, the simple extensions given in § 6 show that systolic arrays can be synthesized, which can compute expressions of the form $(B^tA^{-t}C^t + D^t)$. This allows us to solve for multiple right-hand sides and to perform matrix arithmetic.

8. Appendix A.

PROPOSITION 2. *Let* $L \in \mathbf{R}^{N\times N}$ *be a lower triangular matrix and* $\mathbf{b} \in \mathbf{R}^N$ *a vector, such that*

$$LL^t - \mathbf{b}\mathbf{b}^t > 0. \tag{A.1}$$

Then there exists a matrix $\Theta(-1)$ *as defined in* (4a) *and* (4b) *for* $m = -1$, *such that*

$$\Theta(-1)\begin{bmatrix} L^t \\ -\mathbf{b}^t \end{bmatrix} = \begin{bmatrix} R(-1) \\ 0^t \end{bmatrix}, \tag{A.2}$$

where $R(-1)$ *is upper triangular.*

Proof. The proof is by induction. Put $L_0 = L$ and $\mathbf{b}_0 = \mathbf{b} = [b_1^{(0)} \cdots b_N^{(0)}]^t$. Let $\mathbf{s}_0$ be the column vector

$$\mathbf{s}_0 = L_0^{-1}\mathbf{b}_0 = [s_1^{(0)} \cdots s_N^{(0)}]^t. \tag{A.3}$$

From (A.1) we have that

$$L_0 L_0^t - \mathbf{b}_0 \mathbf{b}_0^t = L_0 (I_N - \mathbf{s}_0 \mathbf{s}_0^t) L_0^t > 0. \tag{A.4}$$

Hence,

$$I_N - \mathbf{s}_0 \mathbf{s}_0^t > 0. \tag{A.5}$$

Since $s_1^{(0)} = b_1^{(0)} / l_{11}^{(0)}$, $(I_N - \mathbf{s}_0 \mathbf{s}_0^t)$ is a matrix with its 11-entry equal to $(1 - (b_1^{(0)} / l_{11}^{(0)})^2) > 0$. Hence, it follows that $|b_1^{(0)}| / |l_{11}^{(0)}| < 1$ and there exists a matrix $\Theta_1(-1)$ (see (4b) for $i = 1$), with $\tanh(\alpha_1) = b_1^{(0)} / l_{11}^{(0)}$, such that

$$\begin{bmatrix} L_1^t \\ -\mathbf{b}_1^t \end{bmatrix} = \Theta_1(-1) \begin{bmatrix} L_0^t \\ -\mathbf{b}_0^t \end{bmatrix}, \tag{A.6}$$

where L_1 is a lower triangular matrix and $\mathbf{b}_1 = [0\ b_2^{(1)} \cdots b_N^{(1)}]^t$.

Now, suppose that there exists a matrix $\Theta_{i-1}(-1)$, such that

$$\begin{bmatrix} L_{i-1}^t \\ -\mathbf{b}_{i-1}^t \end{bmatrix} = \Theta_{i-1}(-1) \begin{bmatrix} L_{i-2}^t \\ -\mathbf{b}_{i-2}^t \end{bmatrix}, \tag{A.7}$$

with

$$\mathbf{b}_{i-1}^t = [0 \underset{i-1}{\cdots} 0\ b_i^{(i-1)} \cdots b_N^{(i-1)}] \tag{A.8a}$$

and

$$\mathbf{b}_{i-2}^t = [0 \underset{i-2}{\cdots} 0\ b_{i-1}^{(i-2)} \cdots b_N^{(i-2)}]. \tag{A.8b}$$

We show that there exists a matrix $\Theta_i(-1)$, such that

$$\begin{bmatrix} L_i^t \\ -\mathbf{b}_i^t \end{bmatrix} = \Theta_i(-1) \begin{bmatrix} L_{i-1}^t \\ -\mathbf{b}_{i-1}^t \end{bmatrix},$$

with

$$\mathbf{b}_i^t = [0 \underset{i}{\cdots} 0\ b_{i+1}^{(i)} \cdots b_N^{(i)}].$$

Equation (A.7) can also be written as

$$\begin{bmatrix} L_{i-1}^t \\ -\mathbf{b}_{i-1}^t \end{bmatrix} = \prod_{j=1}^{i-1} \Theta_j(-1) \begin{bmatrix} L_0^t \\ -\mathbf{b}_0^t \end{bmatrix}. \tag{A.9}$$

Since the $\Theta_j(-1)$ are $\Sigma(-1)$-orthogonal, we get

$$L_{i-1} L_{i-1}^t - \mathbf{b}_{i-1} \mathbf{b}_{i-1}^t = L_0 L_0^t - \mathbf{b}_0 \mathbf{b}_0^t > 0. \tag{A.10}$$

Thus

$$I_N - \mathbf{s}_{i-1} \mathbf{s}_{i-1}^t > 0, \tag{A.11}$$

and

$$\mathbf{s}_{i-1} = L_{i-1}^{-1} \mathbf{b}_{i-1} = [0 \underset{i-1}{\cdots} 0\ s_i^{(i-1)} \cdots s_N^{(i-1)})]^t, \tag{A.12}$$

where

$$s_i^{(i-1)} = \frac{b_i^{(i-1)}}{l_{ii}^{(i-1)}}. \tag{A.13}$$

Hence, from (A.11) and (A.13) we see that

$$1 - \left(\frac{b_i^{(i-1)}}{l_{ii}^{(i-1)}}\right)^2 > 0. \tag{A.14}$$

Hence, $|b_i^{(i-1)}|/|l_{ii}^{(i-1)}| < 1$, and there exists a matrix $\Theta_i(-1)$ (see (4b)) with $\tanh(\alpha_i) = b_i^{(i-1)}/l_{ii}^{(i-1)}$, such that

$$\begin{bmatrix} L_i^t \\ -\mathbf{b}_i^t \end{bmatrix} = \Theta_i(m) \begin{bmatrix} L_{i-1}^t \\ -\mathbf{b}_{i-1}^t \end{bmatrix}, \tag{A.15}$$

with L_i lower triangular and $\mathbf{b}_i^t = [0 \cdots 0 \underset{i}{} b_{i+1}^{(i)} \cdots b_N^{(i)}]$. □

9. Appendix B.

THEOREM 2. *Let $A = [a_{ij}] \in \mathbf{R}^{N \times N}$ be a symmetric positive definite matrix, normalized such that we can write*

$$A = R_L + I_N + R_L^t, \tag{B.1a}$$

where $R_L = [a_{ij}]$, $i > j$, is the strictly lower triangular part of A. Put

$$U = R_L + I_N, \tag{B.1b}$$

$$Y = R_L. \tag{B.1c}$$

Then there exists a matrix product $\Phi \in \mathbf{R}^{2N \times 2N}$

$$\Phi = \prod_{j=1}^{\tilde{N}-1} \prod_{i=j}^{\tilde{N}-1} \Phi_{ij}, \tag{B.2a}$$

of embedded plane hyperbolic rotations

$$\Phi_{ij} = \begin{bmatrix} I_i & & & & \\ & \cosh(\alpha_{ij}) & & \sinh(\alpha_{ij}) & \\ & & I_{N-j-1} & & \\ & \sinh(\alpha_{ij}) & & \cosh(\alpha_{ij}) & \\ & & & & I_{N+j-i-1} \end{bmatrix}, \tag{B.2b}$$

such that the Cholesky factor L^t of the matrix A is given from

$$\begin{bmatrix} L^t \\ O \end{bmatrix} = \Phi \begin{bmatrix} U^t \\ Y^t \end{bmatrix}. \tag{B.3}$$

Proof. The proof is by induction. Put $U_0 = U$ and $Y_0 = Y$ and let S_0 be the strictly lower triangular matrix

$$S_0 = U_0^{-1} Y_0. \tag{B.4}$$

Then

$$\begin{aligned} A &= [U_0 \quad Y_0] J [U_0 \quad Y_0]^t = U_0 U_0^t - Y_0 Y_0^t \\ &= U_0 (I_N - S_0 S_0^t) U_0^t > 0. \end{aligned} \tag{B.5}$$

Hence,

$$I_N - S_0 S_0^t > 0. \tag{B.6}$$

The first off-diagonal in the lower triangular part of the matrix S_0 has entries $(y^{(0)}_{i+1,i})/(u^{(0)}_{i+1,i+1})$, $i=1,\cdots,N-1$, where $y^{(0)}_{kl}$ and $u^{(0)}_{kl}$ are the kl-entries of Y_0 and U_0, respectively. And it follows from (B.6) that

$$\frac{|y^{(0)}_{i+1,i}|}{|u^{(0)}_{i+1,i+1}|} < 1, \qquad i=1,\cdots,N-1. \tag{B.7}$$

Thus, there eixst J-orthogonal matrices Φ_{i1} (see (26b) for $j=1$) with $\tanh(\alpha_{i1}) = -(y^{(0)}_{i+1,i})/(u^{(0)}_{i+1,i+1})$, $i=1,\cdots,N-1$, such that

$$\begin{bmatrix} U_1^t \\ Y_1^t \end{bmatrix} = \Phi_1 \begin{bmatrix} U_0^t \\ Y_0^t \end{bmatrix}, \tag{B.8}$$

$$\Phi_1 = \prod_{i=1}^{\tilde{N}-1} \Phi_{i1}, \tag{B.9}$$

where Y_1 is strictly lower triangular with its first off-diagonal zero and U_1 is lower triangular and invertible. From the J-orthogonality of the matrix Φ_1 it follows that $U_1 U_1^t - Y_1 Y_1^t = U_0 U_0^t - Y_0 Y_0^t$.

Now, suppose there exists a matrix Φ_{k-1}

$$\Phi_{k-1} = \prod_{i=k-1}^{\tilde{N}-1} \Phi_{i,k-1}, \tag{B.10}$$

(the matrices $\Phi_{i,k-1}$ are as given in (26b) for $j=k-1$) such that

$$\begin{bmatrix} U_{k-1}^t \\ Y_{k-1}^t \end{bmatrix} = \Phi_{k-1} \begin{bmatrix} U_{k-2}^t \\ Y_{k-2}^t \end{bmatrix}, \tag{B.11}$$

with U_{k-1} and U_{k-2} lower triangular, invertible and Y_{k-1} and Y_{k-2} strictly lower triangular, with the first k and $k-1$ off-diagonals zero, respectively. We shall show next that there exists a matrix Φ_k

$$\Phi_k = \prod_{i=k}^{\tilde{N}-1} \Phi_{ik},$$

(the matrices Φ_{ik} as given in (26b) for $j=k$) such that

$$\begin{bmatrix} U_k^t \\ Y_k^t \end{bmatrix} = \Phi_k \begin{bmatrix} U_{k-1}^t \\ Y_{k-1}^t \end{bmatrix},$$

with U_k lower triangular, invertible and Y_k strictly lower triangular, with the first $k+1$ off-diagonals zero.

From (B.10) we can write (B.11) as

$$\begin{bmatrix} U_{k-1}^t \\ Y_{k-1}^t \end{bmatrix} = \prod_{j=1}^{\tilde{k}-1} \prod_{i=j}^{\tilde{N}-1} \Phi_{ij} \begin{bmatrix} U_0^t \\ Y_0^t \end{bmatrix}. \tag{B.12}$$

Putting

$$S_{k-1} = U_{k-1}^{-1} Y_{k-1}, \tag{B.13}$$

and, since the Φ_{ij} are J-orthogonal, it follows from (B.12) that

$$I_N - S_{k-1}S_{k-1}^t > 0. \tag{B.14}$$

The matrix S_{k-1} is strictly lower triangular, with the first $k-1$ off-diagonals zero and entries

$$\frac{y_{i+1,i}^{(k-1)}}{u_{i+1,i+1}^{(k-1)}}, \qquad i = k-1, \cdots, N-1 \tag{B.15}$$

on the kth off-diagonal. The ii-entry of the product $S_{k-1}S_{k-1}^t$ is equal to the sum of squares of the elements of the ith row of the matrix S_{k-1}. Hence, it follows from (B.14) that

$$\frac{|y_{i+1,i}^{(k-1)}|}{|u_{i+1,i+1}^{(k-1)}|} < 1, \qquad i = k-1, \cdots, N-1, \tag{B.16}$$

and there exist matrices Φ_{ik} (see (26b) for $j = k$) with $\tanh(\alpha_{ik}) = -(y_{i+1,i}^{(k-1)}/u_{i+1,i+1}^{(k-1)})$, such that

$$\Phi_k = \prod_{i=k}^{N-1} \Phi_{ik}, \tag{B.17}$$

$$\begin{bmatrix} U_k^t \\ Y_k^t \end{bmatrix} = \Phi_k \begin{bmatrix} U_{k-1}^t \\ Y_{k-1}^t \end{bmatrix}, \tag{B.18}$$

with U_k lower triangular, invertible and Y_k strictly lower triangular, with its first k off-diagonals zero.

Next, it remains to show that the matrix L in (B.3) is the Cholesky factor of the matrix A. Since the matrix Φ is J-orthogonal, it follows that

$$LL^t = UU^t - YY^t. \tag{B.19}$$

And, from (B.1b) and (B.1c) it follows that $UU^t - YY^t = A$, so that L must be the Cholesky factor of the matrix A. □

10. Appendix C.

THEOREM 3. *Given an $N \times N$ positive definite matrix $A = I_N + R_L + R_L^t$ (R_L strictly lower triangular), let the matrices U and Y be defined as $U = I_N + R_L$ and $Y = R_L$. Then, in the recursion,*

$$U_0 = U, \qquad Y_0 = Y,$$

$$\begin{bmatrix} U_j^t \\ Y_j^t \end{bmatrix} = \Phi_j \begin{bmatrix} U_{j-1}^t \\ Y_{j-1}^t \end{bmatrix}, \qquad j = 1, \cdots, N-1, \tag{C.1}$$

$$Y_{N-1} = 0,$$

$$\Phi_j = \prod_{i=j}^{N-1} \Phi_{ij}, \tag{C.2}$$

(the matrices Φ_{ij} are as given in (26b)*) the entries in both U_j and Y_j are bounded by* 1 *in absolute value, for $j = 1, \cdots, N-1$.*

Proof. Consider recursion (C.1) applied on the $i \times i$ $(i > 1)$ leading principal submatrix $\underline{A}_i$, of the $N \times N$ matrix A

$$\begin{bmatrix} U^{(i)^t} \\ Y^{(i)^i} \end{bmatrix} = \Phi^{(i)} \begin{bmatrix} U^t \\ Y^t \end{bmatrix} \tag{C.3}$$

with

$$\Phi^{(i)} = \prod_{j=1}^{i-1} \begin{bmatrix} I_{i-1} & & & & \\ & \cosh(\alpha_{ij}) & & \sinh(\alpha_{ij}) & \\ & & I_{N-j-1} & & \\ & \sinh(\alpha_{ij}) & & \cosh(\alpha_{ij}) & \\ & & & & I_{N-i+j} \end{bmatrix} \Phi^{(i-1)}, \qquad \Phi^{(1)} = I_{2N}. \tag{C.4}$$

Put

$$\underline{A}_i = \underline{L}_i \underline{L}_i^t = \underline{R}_i \underline{R}_i^t, \tag{C.5}$$

where $\underline{L}_i$ and $\underline{R}_i$ are lower and upper triangular, respectively. According to (32) we find for $\Phi^{(i)}$

$$\Phi^{(i)} = \begin{bmatrix} \mathbf{L}_i^{-1} & 0 \\ 0 & \mathbf{R}_i^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{U}_i & -\mathbf{Y}_i \\ -\mathbf{Y}_i^t & \mathbf{U}_i^t \end{bmatrix}, \tag{C.6}$$

where

$$\mathbf{L}_i^{-1} = \begin{bmatrix} \underline{L}_i^{-1} & 0 \\ 0 & I_{N-i} \end{bmatrix}, \tag{C.7}$$

$$\mathbf{R}_i^{-1} = \begin{bmatrix} \underline{R}_i^{-1} & 0 \\ 0 & I_{N-i} \end{bmatrix}, \tag{C.8}$$

$$\mathbf{U}_i = \begin{bmatrix} \underline{U}_i & 0 \\ 0 & I_{N-i} \end{bmatrix}, \tag{C.9}$$

$$\mathbf{Y}_i = \begin{bmatrix} \underline{Y}_i & 0 \\ 0 & 0 \end{bmatrix}. \tag{C.10}$$

The matrices $\underline{U}_i$ and $\underline{Y}_i$ are the $i \times i$ leading principal submatrices of U and Y, respectively. The structure of L_i^{-1}, R_i^{-1}, U_i and Y_i is inferred from the fact that the matrix $\Phi^{(i)}$ must leave the $N - i$ last rows of U^t and Y^t unaffected in (C.3), while the $i \times i$ leading principal submatrix of $U^{(i)^t}$ must be equal to $\underline{L}_i^t$. Note that the matrix $\Phi^{(i)}$ is J-orthogonal so that

$$[U \quad Y]\Phi^{(i)^t} J \Phi^{(i)} \begin{bmatrix} I_N \\ I_N \end{bmatrix} = I_N = U^{(i)} \mathbf{L}_i^{-1} - Y^{(i)} \mathbf{R}_i^{-1}. \tag{C.11}$$

Now, because the $i \times i$ leading principal submatrix of $Y^{(i)}$ is zero, it follows from (C.11) and (C.6) that the diagonal elements of $U^{(i)}\mathbf{L}_i^{-1}$ are 1 and, consequently, the $i \times i$ leading principal submatrix of $U^{(i)}$ is the inverse of $\underline{L}_i^{-1}$ as expected.

Substituting (C.6) in (C.3) gives:

$$U\mathbf{U}_i^t - Y\mathbf{Y}_i^t = U^{(i)}\mathbf{L}_i^t; \tag{C.12a}$$

(C.12b) $$\mathbf{Y}\mathbf{U}_i - U\mathbf{Y}_i = Y^{(i)}\mathbf{R}_i^t.$$

Substituting

(C.13a) $$U = Y + I_N$$

and

(C.13b) $$\mathbf{U}_i = \mathbf{Y}_i + I_N$$

in (C.12a) gives

$$\begin{aligned} Y + I_N + \mathbf{Y}_i^t &= (Y + Y^t + I_N) + (\mathbf{Y}_i - Y)^t \\ &= A + (\mathbf{Y}_i - Y)^t = U^{(i)}\mathbf{L}_i^t, \end{aligned}$$

or

(C.14) $$A\mathbf{L}_i^{-t} = U^{(i)} - (\mathbf{Y}_i - Y)^t\mathbf{L}_i^{-t}.$$

From (C.7) and (C.10) it follows that the factor $(\mathbf{Y}_i - Y)^t\mathbf{L}_i^{-t}$ is strictly upper triangular. Hence,

(C.15) $$\mathbf{e}_k^t A\mathbf{L}_i^{-t}\mathbf{e}_l = \mathbf{e}_k^t U^{(i)}\mathbf{e}_l = u_{kl}^{(i)} \quad \text{for } k = 1, \cdots, N \quad \text{and} \quad l = 1, \cdots, k,$$

with $\mathbf{e}_j$ the jth natural base vector in $\mathbf{R}^{N\times N}$. Equation (C.15) gives all entries in the lower triangular part of $U^{(i)}$.

Substituting

(C.16a) $$Y = U - I_N$$

and

(C.16b) $$\mathbf{Y}_i = \mathbf{U}_i - I_N$$

in (C.12b) gives

$$\begin{aligned} U - \mathbf{U}_i &= (U + Y^t) - (\mathbf{U}_i + Y^t) \\ &= A - (\mathbf{U}_i + Y^t) = Y^{(i)}\mathbf{R}_i^t, \end{aligned}$$

or

(C.17) $$A\mathbf{R}_i^{-t} = Y^{(i)} + (\mathbf{U}_i + Y^t)\mathbf{R}_i^{-t}.$$

But, as follows from (C.8) and (C.9), we have

(C.18) $$\mathbf{e}_k^t A\mathbf{R}_i^{-t}\mathbf{e}_l = \mathbf{e}_k Y^{(i)}\mathbf{e}_l = y_{kl}^{(i)} \quad \text{for } k = i+1, \cdots, N \quad \text{and} \quad l = 1, \cdots, k-1,$$

i.e., these are entries in the strictly lower triangular part of $Y^{(i)}$, outside the $i \times i$ leading principal submatrix of $Y^{(i)}$.

Next, we define the inner product:
(1) $\langle \mathbf{v}, \mathbf{w}\rangle_A = \mathbf{v}^t A\mathbf{w}$,
(2) $\langle \mathbf{v}, \mathbf{v}\rangle_A = \|\mathbf{v}\|_A^2 > 0$ (A is positive definite),
(3) $\|\mathbf{v}\|_A = 0$, if and only if $\mathbf{v} = 0$ (A is nonsingular),

for which the Cauchy-Schwarz inequality holds

$$|\langle \mathbf{v}, \mathbf{w}\rangle| \leqq \|\mathbf{v}\|_A \|\mathbf{w}\|_A. \tag{C.19}$$

According to the above definition of the inner product, we find for (C.15) and (C.18) the following Cauchy-Schwarz inequalities:

$$|u_{kl}^{(i)}| = |\langle \mathbf{e}_k, \mathbf{L}_i^{-t}\mathbf{e}_l\rangle_A| \leqq \|\mathbf{e}_k\|_A \|\mathbf{L}_i^{-t}\mathbf{e}_l\|_A, \quad k = 1, \cdots, N; \quad l = 1, \cdots, k, \tag{C.20}$$

$$|y_{kl}^{(i)}| = |\langle \mathbf{e}_k, \mathbf{R}_i^{-t}\mathbf{e}_l\rangle_A| \leqq \|\mathbf{e}_k\|_A \|\mathbf{R}_i^{-t}\mathbf{e}_l\|_A, \quad k = i+1, \cdots, N; \quad l = 1, \cdots, k-1. \tag{C.21}$$

From the fact that the matrix A has only 1's on the main-diagonal, it follows that

$$\|\mathbf{e}_k\|_A = 1. \tag{C.22}$$

For $\|\mathbf{L}_i^{-t}\mathbf{e}_l\|_A$ we find from (C.14)

$$\begin{aligned} \|\mathbf{L}_i^{-t}\mathbf{e}_l\|_A^2 &= \mathbf{e}_l^t \mathbf{L}_i^{-1} A \mathbf{L}_i^{-t}\mathbf{e}_l \\ &= \mathbf{e}_l^t \mathbf{L}_i^{-1} U^{(i)}\mathbf{e}_l - \mathbf{e}_l^t \mathbf{L}_i^{-1}(\mathbf{Y}_i - Y)^t \mathbf{L}_i^{-t}\mathbf{e}_l \\ &= \mathbf{e}_l^t \mathbf{L}_i^{-1} U^{(i)}\mathbf{e}_l, \end{aligned} \tag{C.23}$$

since $\mathbf{L}_i^{-1}(\mathbf{Y}_i - Y)^t \mathbf{L}_i^{-t}$ is strictly upper triangular with zero $i \times i$ leading principal submatrix. But, from (C.11) we know that the $i \times i$ leading principal submatrix of $U^{(i)}$ is equal to $\underline{L}_i$, so that $\mathbf{L}_i^{-1} U^{(i)}$ has all 1's on the main-diagonal. Hence,

$$\|\mathbf{L}_i^{-t}\mathbf{e}_l\|_A = 1. \tag{C.24}$$

From (C.20) it then follows that

$$|u_{kl}^{(i)}| \leqq 1 \quad \text{for } k = 1, \cdots, N \quad \text{and} \quad l = 1, \cdots, k. \tag{C.25}$$

For $\|\mathbf{R}_i^{-t}\mathbf{e}_l\|_A$ in (C.21) we find

$$\|\mathbf{R}_i^{-t}\mathbf{e}_l\|_A^2 = \mathbf{e}_l^t \mathbf{R}_i^{-1} A \mathbf{R}_i^{-t}\mathbf{e}_l. \tag{C.26}$$

But, from the fact that the matrix A has all 1's on its main-diagonal and from (C.8) and (C.5) we deduce that the product $\mathbf{R}_i^{-1} A \mathbf{R}_i^{-t}$ also has all 1's on the main-diagonal. Hence,

$$\|\mathbf{R}_i^{-t}\mathbf{e}_l\|_A = 1. \tag{C.27}$$

From (C.21) it then follows that

$$|y_{kl}^{(i)}| \leqq 1 \quad \text{for } k = i+1, \cdots, N \quad \text{and} \quad l = 1, \cdots, k-1. \tag{C.28}$$

Although the proof, as given above, is based on the leading principal submatrices of U and Y, it is valid for the diagonal-recursive scheme in (C.1). In the leading principal submatrix approach we do not consider the elements below the ith row of U and Y (for $i \times i$ leading principals), unlike in the diagonal-recursive scheme. However, the elements of these rows can be considered by taking successively the $(i+1)\times(i+1)$, $(i+2)\times(i+2), \cdots$ leading principals. In other words, it does not matter if we first process the $i \times i$ leading principal and continue with the $(i+1)\times(i+1), \cdots$ leading principals, or if we adhere to a diagonal-recursive scheme as in (C.1). □

REFERENCES

[Ahme 1982] H. M. AHMED, *Signal processing algorithms and architectures*, Ph.D. dissertation, Stanford University, Stanford, CA, 1982.

[Bath 1976] K. J. BATHE AND E. L. WILSON, *Numerical Methods in Finite Element Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[Bu 1987] J. BU AND E. F. DEPRETTERE, *A* VLSI *algorithm for computing form-factors in radiative transfer computer image synthesis*, in Proceedings of the CG International 1987 Conference, May 1987.

[Bult 1981] A. BULTHEEL, *Error analysis of incoming and outgoing schemes for the trigonometric problem*, in Padé Approximation and its Applications, Springer-Verlag, New York, Berlin, 1981, pp. 100–109.

[Delo 1984] J. M. DELOSME AND I. C. F. IPSEN, *Efficient parallel solution of linear systems with hyperbolic rotations*, Res. Report YALEU/DCS/RR-341, Yale University, New Haven, CT, November 1984.

[Depr 1982] E. F. DEPRETTERE, *Mixed form time-variant lattice recursions*, in Outils et Modèles Mathématiques pour l'Automatique l'Analyse de Systèmes et le Traitment du Signal, I. D. Landau, ed., Paris, 1982, pp. 545–561.

[Dewi 1981] P. DEWILDE AND H. DYM, *Schur recursions, error formulas and convergence of rational estimators for stationary stochastic sequences*, IEEE Trans. Inform. Theory, IT-27 (1981), pp. 446–461.

[Dewi 1987] P. M. DEWILDE AND E. F. DEPRETTERE, *Modeling* VLSI *interconnects as an inverse scattering problem*, in IEEE Proceedings ISCAS, May 1987, pp. 147–153.

[Dewi 1978] P. M. DEWILDE, A. VIERA, AND T. KAILATH, *On a generalized Szegö–Levinson realization algorithm for optimal linear predictors based on a network synthesis approach*, IEEE Trans. Circuits Syst., CAS-25, (1978), pp. 663–675.

[Dudg 1984] D. E. DUDGEON AND R. M. MERSEREAU, *Multidimensional Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[Dym 1981] H. DYM AND I. GOHBERG, *Extensions of band matrices with band inverses*, Linear Algebra Appl., (1981), pp. 1–24.

[Fadd 1959] V. N. FADDEEV, *Computational Methods of Linear Algebra*, Dover, New York, 1959.

[Gent 1981] M. W. GENTLEMAN AND H. T. KUNG, *Matrix triangularization by systolic arrays*, SPIE, Real-Time Signal Processing IV, 298 (1981), pp. 19–26.

[Gent 1975] W. M. GENTLEMAN, *Error analysis of QR decompositions by Givens transformations*, Linear Algebra Appl., 10 (1975), pp. 189–197.

[Golu 1983] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, John Hopkins University Press, Baltimore, MD, 1983.

[Jain 1986] K. JAINANDUNSING AND E. F. DEPRETTERE, *Design and* VLSI *implementation of a concurrent solver for N coupled least-squares fitting problems*, IEEE J. Selected Areas Commun., SAC-4 (1986), pp. 39–48.

[Kail 1979] T. KAILATH, S. Y. KUNG, AND M. MORF, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395–407.

[Kung 1979] H. T. KUNG, *Let's design algorithms for* VLSI *systems*, in Proceedings of the First Caltech Conference on VLSI, 1979, pp. 65–90.

[Kung 1984] S. Y. KUNG, *On supercomputing with systolic/wavefront array processors*, IEEE Proceedings, 72 (1984), pp. 867–884.

[Leis 1981] C. E. LEISERSON, *Area-Efficient* VLSI *computation*, Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, PA, 1981.

[Lev 1984] H. LEV ARI AND T. KAILATH, *Lattice filter parameterization and modeling of nonstationary processes*, IEEE Trans. Inform. Theory, 30 (1984), pp. 2–16.

[Neho 1983] A. NEHORAI AND M. MORF, *A relationship between the Levinson algorithm and the conjugate direction method*, IEEE Trans. Acous. Speech Signal Process., ASSP-31 (1983), pp. 506–508.

[Orfa 1985] S. J. ORFANIDIS, *Optimum Signal Processing*, Macmillan, New York, 1985.

[Royc 1988] V. P. ROYCHOWDHURY AND T. KAILATH, *Regular processor arrays for matrix algorithms with pivoting*, in Proceedings of the International Conference on Systolic Arrays, 1988, pp. 237–246.

[Walt 1971] J. S. WALTER, *A unified algorithm for elementary functions*, in Proceedings Spring Joint Computer Conference 38, AFIPS Press, 1971, p. 397.

PREFACE TO THE SPECIAL SECTION ON ORDINARY DIFFERENTIAL EQUATIONS

The following papers were presented at an international meeting on the numerical solution of ordinary differential equations held at the University of Toronto in June 1988 and sponsored, in part, by the Natural Science and Engineering Research Council of Canada. Speakers at the meeting were given the opportunity to submit papers for review for this special section, which has been edited by W. H. Enright, C. W. Gear, K. R. Jackson, L. R. Petzold, and R. D. Skeel. The following collection represents only a part of the total number of excellent presentations made, since presentations were not always appropriate for archival publication. The numerical solution of ordinary differential equations remains an active area. Contributions covered a wide variety of topics, ranging from the very practical to the more theoretical, including the solution of differential-algebraic equations and the nonlinear stability of existing methods.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 915–936, September 1989
© 1989 Society for Industrial and Applied Mathematics
004

NUMERICAL SOLUTION OF BOUNDARY VALUE PROBLEMS IN DIFFERENTIAL-ALGEBRAIC SYSTEMS*

KENNETH D. CLARK† AND LINDA R. PETZOLD‡

Abstract. This paper extends the theory of shooting and finite-difference methods for linear boundary value problems (BVPs) in ordinary differential equations (ODEs) to BVPs in differential-algebraic equations (DAEs) of the form

$$\begin{aligned} E(t)y'(t) + F(t)y(t) &= f(t), \quad t \in [a,b], \\ B_a y(a) + B_b y(b) &= \beta, \end{aligned}$$

where $E(\cdot)$, $F(\cdot)$, and $f(\cdot)$ are sufficiently smooth and the DAE initial value problem (IVP) is solvable. $E(t)$ may be singular on $[a,b]$ with variable rank, and the DAE may have an index that is larger than one. When $E(t)$ is nonsingular, the singular theory reduces to the standard theory for ODEs. The convergence results for backward differentiation formulas and Runge–Kutta methods for several classes of DAE IVPs are applied to obtain convergence of the corresponding shooting and finite-difference methods for these DAE boundary value problems. These methods can be implemented directly without having to (1) regularize the system to a lower index DAE or ODE or (2) convert the system to a particular canonical structure. Finally, some numerical experiments that illustrate these results are presented.

Key words. differential-algebraic systems, boundary value problems, higher index, shooting methods, finite-difference methods, backward differentiation formulas, implicit Runge–Kutta methods

AMS(MOS) subject classifications. 65L05, 34A08

1. Introduction.

In this paper we extend the theory of shooting and finite-difference methods for linear boundary value problems (BVPs) in ordinary differential equations (ODEs) to BVPs in *differential-algebraic systems* (DAEs) of the form

$$\text{(1.1a)} \qquad \mathcal{L}y(t) \equiv E(t)y'(t) + F(t)y(t) = f(t), \quad t \in [a,b],$$

$$\text{(1.1b)} \qquad \mathcal{B}y(t) \equiv B_a y(a) + B_b y(b) = \beta,$$

where $E(\cdot)$, $F(\cdot)$, and $f(\cdot)$ are sufficiently smooth and the DAE initial value problem (IVP) is solvable. We allow $E(t)$ to be singular on $[a,b]$ with variable rank, and the DAE (1.1a) may have an index that is larger than one. See [14] or [33] for a detailed discussion of the index of a DAE. Intuitively, ODEs have index 0, while the solutions to higher index DAE systems (index > 1) involve derivatives of the coefficients E, F, and the input f. Index one systems contain algebraic variables that are uniquely determined by the state variables (not including derivatives).

IVPs in DAEs have been extensively studied in recent years from both a theoretical and a numerical perspective. These problems arise frequently in applications,

* Received by the editors April 25, 1988; accepted for publication (in revised form) January 27, 1989.

† Mathematical Sciences Division, U.S. Army Research Office, Research Triangle Park, North Carolina 17709. The research of this author was performed while he was a visiting faculty member at Lawrence Livermore National Laboratory, Livermore, California.

‡ Computing & Mathematics Research Division, L-316, Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, California 94550. The work of this author was partially supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, by Lawrence Livermore National Laboratory under contract W-7405-Eng-48.

including circuit and control theory [6], [17], [36]; chemical kinetics [25]; fluid dynamics [33], [38]; and robotics [35]. In some cases, the models lead to the nonlinear semi-explicit formulation

$$y' = f(y,z,t), \tag{1.2a}$$

$$0 = g(y,z,t) \tag{1.2b}$$

and with it the interpretation of (1.1a), (1.2) as constrained ODEs or differential equations on manifolds [42]. However, in many applications the fully-implicit formulation (cf. (1.3)) is more appropriate.

As an extension to the initial value theory, it is natural to consider BVPs in DAEs. DAE BVPs arise in the modeling of semiconductor devices [1]; control theory [4], [17]; detonation modeling [28]; the design of heat exchangers [37]; and in parameter-estimation problems for multibody systems [5]. We believe that as information regarding DAEs and software for these problems becomes more widely disseminated in the scientific and engineering community, the number and variety of applications will increase.

In recent years, several researchers have studied various approaches to the general solution of DAE BVPs. The work of März and Griepentrog [34], [23] focuses on difference and shooting methods for BVPs for nonlinear fully-implicit systems:

$$F(y',y,t) = 0, \tag{1.3a}$$

$$G(y(a),y(b)) = 0 \tag{1.3b}$$

under a *transferability* hypothesis that guarantees that (1.3a) is a regular, index one system and the nullspace of $F_{y'}$ is independent of y', y and has constant dimension. All linear solvable index one systems (1.1a) are transferable, as are semi-explicit systems (1.2) where g_z is bounded and invertible. The numerical approach in [23] requires knowledge of some projector onto $\ker(F_{y'})$ and its derivative at each meshpoint. There are some theoretical results for the subclass of *tractable* index two systems [22], but it is implied that a successful numerical approach involves regularizing the DAE to a nonsingular or index one system and then numerically solving the regularization (cf. [31], [32]).

Ascher [1] gives a convergence result for collocation schemes applied to semi-explicit index one DAEs, where the collocation methods are applied in such a way that the algebraic components of the system are approximated in a piecewise discontinuous space. In Ascher [1], a convergence result is outlined and order conditions are given for Gaussian collocation methods applied directly to fully-implicit index one systems. Hanke [24] describes a least-squares collocation method for linear differential-algebraic equations that is applicable to higher index systems.

Bock, Eich, and Schlöder [5] describe numerical methods based on multiple shooting and collocation for equality—and inequality—constrained DAE BVPs arising from parameter-identification problems for multibody systems. Their approach is restricted primarily to semi-explicit index one systems, and the methods distinguish the algebraic from the differential components in their numerical treatment. This distinction in the method between the algebraic and differential components—a distinction that is inherent in methods proposed for semi-explicit systems by März and Griepentrog; Ascher; and Bock, Eich, and Schlöder—is natural and highly appropriate in the semi-explicit index one case, but for the fully-implicit case it is unclear how to accomplish the distinction in general without the expensive computation of projectors at each meshpoint.

This paper serves several purposes. First, we show that under an appropriate formulation, the theory of shooting and finite-difference methods for linear systems (e.g., simple and parallel shooting with partially or completely separated boundary conditions, one-step difference schemes with extrapolation) for ODE BVPs can be formally extended to DAE BVPs, using the characterization of the solution manifold given in [11]. We note that we have not addressed here the issue of conditioning for the DAE BVP and for the numerical methods. For a discussion of these issues for the DAE BVP, see Lentini and März [29], [30]; and for ODE BVPs and numerical methods for ODE BVPs, see Ascher, Mattheij, and Russell [3]. Ascher [1] addresses these issues for a limited class of numerical methods for DAE BVPs. We make no assumptions on the index of the DAE (1.1a), except what is required for convergence of the corresponding methods applied to related DAE IVPs. Thus for many of the methods, the results apply to the solution of higher index systems. The results and details of the theory are straightforward extensions of the results in [26], [27] and reduce to the ODE case when $E(t)$ is nonsingular; thus it is possible to treat ODE and DAE BVPs within the same theoretical framework. For the purposes of clarity and consistency, we will adopt the notation and presentation in these papers to the greatest possible extent. As in the ODE case, the shooting theory provides a necessary theoretical basis for the development of more direct techniques, such as finite differences. We discuss shooting methods in § 3 and treat the finite-difference case in § 4. In § 5 we present the results of some numerical experiments that reinforce the theory of the previous sections.

A consequence of this approach is that the initial value methods that exhibit the restricted convergence and stability properties for certain subclasses of numerically solvable DAEs (1.1a), e.g., *backward differentiation formulas* (BDF) ([6], [7], [16]-[21], [33], [41], [43]); *implicit Runge–Kutta methods* (IRK) ([8], [40]); or the *ith order jth block* (i.e., (i,j)-) series methods ([11], [12]) can in principle be used to construct convergent approximations to the BVP (1.1) under similar restrictions. Furthermore, the DAE can be solved directly by these methods without having to convert the system to a canonical structure. In particular, it is unnecessary to transform the DAE BVP to an ODE BVP on a lower-dimensional space or to regularize the DAE. Knowledge of the solution manifold (which may require derivative information) is required only at the initial time point $t_0 = a$, or in the case of parallel shooting at each parallel node τ_j, and not at every numerical meshpoint t_n.

2. Background and terminology. We assume that $E(\cdot)$, $F(\cdot)$, and $f(\cdot)$ are real matrix- and vector-valued functions of $t \in I = [a,b]$, with dimensions $m \times m$ and $m \times 1$, respectively. The space of s-times continuously differentiable functions on I is denoted by $C^s(I)$, or more conveniently C^s, with the range (e.g., matrix- or vector-valued) understood from the context. Throughout this paper we assume that E, F are at least C^{2m}, while f is at least C^m, although in many cases it suffices to have E, F, $f \in C^\sigma$, where σ is the global index (see the discussion below). As in [11], we adopt the following definition of solvability for (1.1a).

DEFINITION 2.1. The system (1.1a) is *solvable* on I if and only if

(1) for all f there exists a C^1 solution y;

(2) all solutions corresponding to f are defined and at least C^1 on the entire interval I and are uniquely determined by their values $y(t)$ for each $t \in I$;

(3) all solutions of the homogeneous system $\mathcal{L}y = 0$ are at least C^{2m+1}; and

(4) if $f \in C^s$ for $m \le s \le 2m$, then any corresponding solution y is C^{s-m+1}.

Conditions (1) and (2) of the definition constitute the standard definition of solvability (cf. [14], [16]) and imply that solutions are pointwise linearly independent

and do not exhibit finite escape on I. The vector y_0 is a *consistent initial condition* for (1.1a) if there exists a solution y such that $y(a) = y_0$. Let $M_f(a)$ (or simply M_f) denote the set of consistent initial conditions for (1.1a) at $t = a$. When $E(t)$ is nonsingular $M_f = \Re^m$, while if $E(t)$ is singular M_f forms a linear manifold properly contained in $\Re^m$ (a proper subspace if $f \equiv 0$).

If $\mathcal{L}y = 0$ has the unique solution $y \equiv 0$ (equivalently, $\mathcal{L}y = f$ has a unique solution y_p), we say (1.1a) is *degenerate*; otherwise, it is nondegenerate. If (1.1a) is nondegenerate, there exists an integer $r > 0$ and linearly independent functions $\phi_1, \cdots, \phi_r$ such that $(\phi_1(t), \cdots, \phi_r(t))$ forms a basis for the solution manifold $M_0(t)$ for each $t \in I$. In particular, $r = \dim M_0$ is the *rank* or *dimension* of the solution manifold for (1.1a) and every solution y can be written

$$y(t) = y_p(t) + \sum_{i=1}^{r} \xi_i \phi_i(t). \tag{2.1a}$$

If (1.1a) is degenerate then $r = 0$, while if $E(t)$ is nonsingular then $r = m$. As in the case for ODEs, (2.1a) will be the starting point for the development of a theory for shooting methods applied to (1.1).

The addition of conditions (3) and (4) to the standard definition of solvability guarantees the characterization of the solution manifold at $t = \tau$, $\tau \in [a, b]$ as a linear system involving E, F, f, and their derivatives up to order m evaluated at $t = \tau$. Without going into lengthy detail, we briefly describe its derivation and refer the reader to [11] or [12] for a more detailed discussion. Let $j \geq 1$ be an integer and define $\mathbf{E}_j(\tau)$, $\mathbf{F}_j(\tau)$, $\mathbf{y}_j(\tau)$, and $\mathbf{f}_j(\tau)$ by

$$\mathbf{E}_j(\tau) = \begin{pmatrix} E_0 & & & & \\ E_1 + F_0 & 2E_0 & & & \\ E_2 + F_1 & 2E_1 + F_0 & 3E_0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ E_{j-1} + F_{j-2} & 2E_{j-2} + F_{j-3} & \cdots & \cdots & jE_0 \end{pmatrix}, \tag{2.1b}$$

$$\mathbf{F}_j(\tau) = (F_0^T, F_1^T, \cdots, F_{j-1}^T)^T, \qquad \mathbf{f}_j(\tau) = (f_0^T, f_1^T, \cdots, f_{j-1}^T)^T, \tag{2.1c}$$

$$\mathbf{y}_j(\tau) = (y_1^T, y_2^T, \cdots, y_j^T)^T, \tag{2.1d}$$

where $c_i = c^{(i)}(\tau)/i!$ for $c = E$, F, y, and f. Then (1.1a) implies

$$\mathbf{E}_j(\tau)\mathbf{y}_j(\tau) = \mathbf{f}_j(\tau) - \mathbf{F}_j(\tau)y_0. \tag{2.2}$$

Note that (2.2) is an $(mj) \times (mj)$ system. The matrix $\mathbf{E}_j$ is 1-*full* if (2.2) uniquely determines y_1. If (1.1a) is solvable on I, then $\mathbf{E}_j$ is 1-full with constant rank for $j = m+1$ [11]. It follows that $M_f(\tau)$ is the set of all y_0 such that (2.2) is a consistent linear system, i.e.,

$$M_f(\tau) = \{y_0 | \mathbf{f}_j(\tau) - \mathbf{F}_j(\tau)y_0 \quad \in \mathrm{im}(\mathbf{E}_j(\tau))\}, \tag{2.3}$$

where $\mathrm{im}(\cdot)$ denotes the range space. Let $\eta = \dim \ker(\mathbf{E}_j(\tau))$. Solvability implies η is constant on I. Let $W(\tau)$ be a smooth $(mj) \times \eta$ matrix-valued function whose columns

form a basis for $\ker(\mathbf{E}_j(\tau)^T)$. Then $Q = W(W^TW)^{-1}W^T$ is an orthogonal projector onto $\ker(\mathbf{E}_j^T)$ and $I - Q$ is an orthogonal projector onto $\mathrm{im}(\mathbf{E}_j) = \ker(\mathbf{E}_j^T)^\perp$. Clearly (2.2) implies

$$W(\tau)^T\mathbf{F}_j(\tau)y_0 = W(\tau)^T\mathbf{f}_j(\tau). \tag{2.4}$$

But (2.4) implies $f_j - \mathbf{F}_j y_0 \in \ker(Q) = \mathrm{im}(I - Q) = \mathrm{im}(\mathbf{E}_j)$, hence (2.2) and (2.4) are equivalent. In § 4, we write the system (2.4) as

$$M_\tau y_0 = g(\tau), \tag{2.5}$$

where M_τ is $\eta \times m$ with full row rank, $g(\tau) = 0$ if $f \equiv 0$ and $r = \dim\ker(M_\tau) = m - \eta$. It follows that $M_0 = \ker(M_a)$, where $M_a = M_\tau$ at $\tau = a$. Although numerically impracticable or expensive in many cases, this tells us that if r is not known a priori, in principle it can be determined from the rank of $\mathbf{E}_j$.

Frequently, it is possible to take $j < m + 1$ in (2.2). If $\sigma + 1$ is the smallest integer such that $\mathbf{E}_{\sigma+1}$ is 1-full and constant rank, then σ is the *global index* of (1.1a) and it suffices to take $j = \sigma + 1$. It is relatively straightforward to show that this definition agrees with the definition of global index given in [21]. (See also [17] for an equivalent definition of global index in a slightly different context.) If for each $\tau \in I$ there exists a scalar $\lambda_\tau \in \Re$ such that $(\lambda_\tau E(\tau) + F(\tau))^{-1}$ exists, then (1.1a) is a regular system and the *local index* of (1.1a) at $t = \tau$ is the index of nilpotency of the matrix $E_\lambda(\tau) = (\lambda_\tau E(\tau) + F(\tau))^{-1}E(\tau)$, denoted $\mathrm{ind}(E(\tau), F(\tau))$. It is well known that the local and global indices for higher index (index > 1) systems may differ when E, F are time varying, although for index one systems they are the same.

The main results of this paper depend only on solvability and (2.4) and therefore are independent of the index. However, since the index one systems are well understood and arise most frequently in applications, we briefly review several facts for this case. In the following proposition let $(\cdot)^\dagger$ denote the *Moore–Penrose inverse*, while $(\cdot)^D$ is the *Drazin inverse* [15].

PROPOSITION 2.1. *If* (1.1a) *is solvable, then* $\mathrm{im}([I - EE^\dagger]F) = \mathrm{im}\,([I - EE^\dagger])$ *for all* $t \in I$. *Consequently, if* rank $(E(t)) = r$, *then* rank $([I - E(t)E(t)^\dagger]F(t)) = m - r$ *and* dim ker $([I - E(t)E(t)^\dagger]F(t)) = r$.

Proof. Solvability of (1.1a) implies for every solution $y(t)$,

$$[I - EE^\dagger]Fy = [I - EE^\dagger]f \quad \text{for all } t \in [a, b], \tag{2.6}$$

since $[I - EE^\dagger]$ is the orthogonal projector onto $\ker(E^T) = \mathrm{im}(E)^\perp$. Clearly, $\mathrm{im}([I - EE^\dagger]) \supseteq \mathrm{im}([I - EE^\dagger]F)$ but if the inequality is strict, there exists smooth $f(t)$ such that (2.6) is not satisfied for some $t^* \in [a, b]$. This contradicts solvability, hence $\mathrm{im}([I - EE^\dagger]) \equiv \mathrm{im}([I - EE^\dagger]F)$. □

Proposition 2.1 follows directly from the proof of Theorem 2.2 in [16] and is independent of the index of (1.1a). In the special case $\mathrm{ind}(E, F) = 1$, the system (1.1a) is solvable if and only if $\mathrm{rank}(E(t))$ is constant on $[a, b]$. Equivalently, there exist invertible $P(t)$, $Q(t)$ as smooth as $E(t)$ and $F(t)$ such that

$$P(t)E(t)Q(t) = \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix}, \qquad P(t)F(t)Q(t) = \begin{pmatrix} C(t) & 0 \\ 0 & I \end{pmatrix}, \tag{2.7}$$

where the identity block in PEQ has size $\mathrm{rank}(E(t))$. Thus the dimension r of the solution manifold for solvable index one systems is $\mathrm{rank}(E(t))$, while if the index is

greater than one, $r < \text{rank}(E(t))$. In particular, from Proposition 2.1 we have the following.

COROLLARY 2.1. *If* (1.1a) *is solvable index* 1, *the* $m - r$ *linearly independent relations in* (2.6), *evaluated at* $t = \tau$, *completely specify the set of consistent initial conditions for* (1.1a) *starting at* $t = \tau$.

Of course, the additional consistency requirements for higher index systems are obtained from differentiations of (1.1a), as in (2.4). Note that if only one consistent initial condition y_p^0 is needed and $F(\tau)v = f(\tau)$ is a consistent linear system, then we may choose $y_p^0 = v$. Also, compare (2.6) with the equivalent characterization (2.4) with $j = 2$, or the system

$$(I - E_\lambda^D E_\lambda)y = (I - E_\lambda^D E_\lambda)F_\lambda f_\lambda, \quad \text{evaluated at } t = \tau, \tag{2.8}$$

where $q_\lambda = (\lambda E + F)^{-1}q$ for $q = E, F, f$, which is derived in [14]. Using the approach in [23], one can also show that $M_f(\tau)$ is the set of all x such that

$$x = (P - Q[E + FQ]^{-1}FP)x_0 + Q[E + FQ]^{-1}f, \quad x_0 \text{ arbitrary} \tag{2.9}$$

evaluated at $t = \tau$, where Q is an arbitrary projector onto $\ker(E)$ and $P = I - Q$. Which characterization is appropriate depends on the circumstances, but for our purposes it will be convenient to use (2.5), since it is independent of the index.

3. Shooting theory for linear DAEs. Assume that $r = m - \dim(\ker(E_j))$ has been determined by a careful rank determination of E_j using, for example, a singular value decomposition [22] or Gauss elimination with pivoting and scaling if the linearly independent rows of E_j are not nearly linearly dependent in the numerical sense. In many cases of interest (e.g., constant coefficients or the structural forms discussed in [13], [16], and [17]), $r = \text{core-rank}(E_\lambda) = \text{rank}(E_\lambda^D E_\lambda)$. The core-rank of a matrix A is the size of the block corresponding to the nonzero eigenvalues in the Jordan form for A. In order for the boundary conditions (1.1b) to uniquely determine solutions for all β, it is necessary that $B_a, B_b \in \Re^{r\times m}$ with $\text{rank}[B_a, B_b] = r$. Thus a correct formulation for the BVP is

$$\mathcal{L}y(t) \equiv E(t)y'(t) + F(t)y(t) = f(t), \quad t \in I, \tag{3.1a}$$

$$\mathcal{B}y(t) \equiv B_a y(a) + B_b y(b) = \beta, \quad B_a, B_b \in \Re^{r\times m}, \quad \beta \in \Re^r. \tag{3.1b}$$

DEFINITION 3.1. The BVP (3.1) is *solvable* if and only if (3.1a) is a solvable DAE and for every $\beta \in \Re^r$ there exists a unique solution y to (3.1).

Let $y_p^0 \in M_f(a)$ and assume $(\phi_i^0)_1^r$ is any basis for $M_0 = \ker(M_a)$, where M_a is given in (2.5). Correspondingly, let $y_p(t)$ and the fundamental matrix $Y(t) = [\phi_1(t), \cdots, \phi_r(t)]$ be the solutions to the $r + 1$ IVPs

$$\mathcal{L}y_p(t) = f(t), \quad y_p(a) = y_p^0 \in M_f(a), \tag{3.2a}$$

$$\mathcal{L}Y(t) = 0, \quad Y(a) = Y_0 = [\phi_1^0, \cdots, \phi_r^0]. \tag{3.2b}$$

Note that $Y(t)$ has full column rank for all $t \in I$, since (3.1a) is solvable. Using the representation (2.1a) and imposing the boundary conditions (3.1b), we find that y is a solution of (3.1) if and only if the vector $\xi = (\xi_1, \cdots, \xi_r)^T$ satisfies

$$[B_a Y_0 + B_b Y(b)]\xi = \beta - (B_a y_p^0 + B_b y_p(b)). \tag{3.3}$$

As in the ODE case, the $r \times r$ matrix

$$S = B_a Y_0 + B_b Y(b) \tag{3.4}$$

is the *shooting matrix* for (3.1) and is unique up to a change of basis for M_0, i.e., if $\tilde{S}$ is any shooting matrix for (3.1), there exists a constant invertible $r \times r$ matrix Q such that $\tilde{S} = SQ$. Thus we have Theorem 3.1.

THEOREM 3.1. *The* BVP (3.1) *is solvable if and only if S is invertible. The desired solution is given by* (2.1a), *where y_p^0, Y satisfy* (3.2) *and ξ is the solution to* (3.3).

Therefore, *simple shooting* consists of solving the $r+1$ IVPs in (3.2) over $[a,b]$, forming the shooting matrix (3.3), solving the linear system (3.4) for ξ, and finally solving the IVP

$$\mathcal{L}y(t) = f(t), \qquad y(a) = y_p^0 + Y_0 \xi \tag{3.5}$$

for the unique solution y to (3.1). At the end of this section we discuss numerical implementations of shooting procedures. Unfortunately, if the differential part of (3.1a) exhibits stiffness, the shooting matrix may be ill conditioned. One way to inhibit the effects of exponential growth of solutions on the conditioning of the shooting equation is to consider separate shooting problems on smaller subintervals of $[a,b]$ and join the resulting solutions by imposing continuity. This technique of *parallel* or *multiple* shooting is discussed momentarily. As we have already noted, the details are essentially the same as for ODEs, with the exception that we must incorporate information about the solution manifold. To this end the following intuitive result is very useful.

PROPOSITION 3.1. *Suppose* (3.1) *is solvable with solution manifold determined by* (2.5). *Let the $m \times 2m$ matrix X and $m \times (r+m)$ matrix Z be defined by*

$$X = \begin{pmatrix} B_a & B_b \\ M_a & 0 \end{pmatrix}, \qquad Z = \begin{pmatrix} S & B_a \\ 0 & M_a \end{pmatrix}.$$

Then rank(X) = rank$(Z) = m$.

Proof. Clearly rank $(Z) = m$, since S is nonsingular and M_a has full row rank $m - r$. It suffices to show that $\mathrm{im}(X) \supseteq \mathrm{im}(Z)$. Suppose $z = (z_1^T, z_2^T)^T \in \mathrm{im}(Z)$. Then there exists $v = (v_1^T, v_2^T)^T$ such that

$$\begin{aligned} z_1 &= Sv_1 + B_a v_2 = [B_a Y_0 + B_b Y(b)]v_1 + B_a v_2 \\ &= B_a(Y_0 v_1 + v_2) + B_b Y(b) v_1, \\ z_2 &= M_a v_2. \end{aligned}$$

Let $u_1 = Y_0 v_1 + v_2$, $u_2 = Y(b)v_1$, $u = (u_1^T, u_2^T)^T$. Then $z = Xu$, since $M_a Y_0 = 0$. Therefore $\mathrm{im}(X) \supseteq \mathrm{im}(Z)$ and more specifically $\mathrm{im}(X) = \mathrm{im}(Z)$, implying $\mathrm{rank}(X) = m$. □

Partially separated boundary conditions. If either B_a or B_b are rank deficient, the number of IVPs to be solved in (3.2) can be reduced to $q+1$, where $q = \min(\mathrm{rank}(B_a), \mathrm{rank}(B_b))$. In this case the boundary conditions are partially separated. Without loss of generality, assume $\mathrm{rank}(B_b) = q < r$. There exists a nonsingular matrix R such that premultiplying (3.1b) by R yields

$$C_a y(a) = \beta_a, \tag{3.6a}$$

$$C_{ba} y(a) + C_b y(b) = \beta_b, \tag{3.6b}$$

where $\beta_a \in \Re^{r-q}$, $\beta_b \in \Re^q$, C_a is $(r-q)\times m$ with full row rank $(r-q)$ and C_b is $q \times m$ with full row rank q. If $C_{ba} = 0$, the boundary conditions are completely separated.

From Proposition 3.1, the matrix $[C_a^T, M_a^T]^T$ has full row rank $m-q$. Let D_a be a $q \times m$ matrix such that $U = [C_a^T, M_a^T, D_a^T]^T$ is invertible. We impose the left-hand boundary conditions (3.6a) on the particular solution y_p. Thus, suppose y_p satisfies

$$\mathcal{L}y_p(t) = f(t), \qquad Uy_p(a) = [\beta_a^T, g(a)^T, \gamma^T]^T \tag{3.7}$$

for an arbitrary vector $\gamma \in \Re^q$ (e.g., take $\gamma = 0$). Partition U^{-1} as

$$U^{-1} = [U_1, U_2, U_a], \quad U_a \in \Re^{m\times q},$$

and let $V(t)$ satisfy

$$\mathcal{L}V(t) = 0, \qquad V(a) = U_a. \tag{3.8}$$

Note that a unique solution to (3.8) exists, since (3.1a) is solvable and $M_aU_a = 0$ by definition of U_a. Now let

$$y(t) = y_p(t) + V(t)\mu, \quad \mu \in \Re^q, \tag{3.9}$$

and impose the q remaining boundary conditions (3.6b) to get a linear system for the parameters μ

$$[C_{ba}U_a + C_bV(b)]\mu = \beta_b - (C_{ba}y_p(a) + C_by_p(b)). \tag{3.10}$$

To see that $[C_{ba}U_a + C_bV(b)]$ is invertible, note that RS is invertible, where

$$RS = \begin{pmatrix} C_ay_0 \\ C_{ba}Y_0 + C_bY(b) \end{pmatrix}$$

where Y_0 is any basis for $\ker(M_a)$ and $Y(t)$ is the corresponding homogeneous solution. In particular, we can let Y_0 have the form $[Z(a), U_a]$, hence $Y(b) = [Z(b), V(b)]$. By definition of U_a, $C_aU_a = 0$, which implies

$$RS = \begin{pmatrix} C_aZ(a) & 0 \\ C_{ba}Z(a) + C_bZ(b) & C_{ba}U_a + C_bV(b) \end{pmatrix}.$$

Thus $(C_{ba}U_a + C_bV(b))$ is invertible.

Parallel Shooting. Suppose (3.1) is solvable, and let the nodes $(\tau_j)_0^J$ define a partition of $[a,b]$

$$\tau_0 = a < \tau_1 < \cdots < \tau_J = b.$$

On each subinterval $[\tau_{j-1}, \tau_j]$, $j = 1, \cdots, J$, the BVP solution $y(t)$ can be represented as

$$y(t) = y_j(t) = v_j(t) + V_j(t)\xi_j, \quad t \in [\tau_{j-1}, \tau_j], \tag{3.11}$$

where v_j is any particular solution to

$$\mathcal{L}v_j(t) = f(t), \qquad v_j(\tau_{j-1}) = v_j^0 \in M_f(\tau_{j-1}), \tag{3.12}$$

and V_j is a fundamental solution to

$$\mathcal{L}V_j(t) = 0, \qquad V_j(\tau_{j-1}) = V_j^0, \qquad \mathrm{im}(V_j^0) = M_0(\tau_{j-1}). \tag{3.13}$$

Continuity of y on $[a, b]$ and the boundary conditions (3.1b) yield the following equations for the ξ_j's

$$\begin{pmatrix} B_aV_1^0 & & & & B_bV_J(b) \\ -V_1(\tau_1) & V_2^0 & & & \\ & -V_2(\tau_2) & V_3^0 & & \\ & & \ddots & \ddots & \\ & & & -V_{J-1}(\tau_{J-1}) & V_J^0 \end{pmatrix} \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \vdots \\ \xi_J \end{pmatrix} = \begin{pmatrix} \beta - [B_av_1^0 + B_bv_J(b)] \\ v_1(\tau_1) - v_2^0 \\ v_2(\tau_2) - v_3^0 \\ \vdots \\ v_{J-1}(\tau_{J-1}) - v_J^0 \end{pmatrix}$$

or

$$A\underline{\xi} = \underline{\beta}, \tag{3.14}$$

where A is $[(J-1)m + r] \times [rJ]$. Since (3.1) is solvable and $v_j(\tau_j)$ and $V_j(\tau_j)$ are uniquely determined by v_j^0 and V_j^0, respectively, we must have $\underline{\beta} \in \mathrm{im}(A)$ and the solution $\underline{\xi}$ is unique. Thus A has full column rank rJ. The exact solution is given by

$$\underline{\xi} = A^\dagger \underline{\beta}. \tag{3.15}$$

From a numerical standpoint, the fact that A has full column rank is important. If $v_j(\tau_j)$ and $V_j(\tau_j)$ are numerically computed to $O(h^k)$ accuracy, then (3.14) is actually

$$A_h u = \underline{\beta}_h, \tag{3.16}$$

which may be inconsistent. However, $\underline{\beta}_h = \underline{\beta} + O(h^k)$ and $A_h = A + O(h^k)$, which implies $A_h^\dagger = A^\dagger + O(h^k)$ for $h > 0$ sufficiently small [14]. Therefore the least-squares solution $\underline{\xi}_h = A_h^\dagger \underline{\beta}_h$ satisfies

$$\begin{aligned} \underline{\xi}_h - \underline{\xi} &= A_h^\dagger \underline{\beta}_h - A^\dagger \underline{\beta} \\ &= A^\dagger O(h^k) + O(h^k)\underline{\beta} + O(h^{2k}) = O(h^k). \end{aligned} \tag{3.17}$$

The numerical least-squares problem has been extensively investigated [22], and there currently exist well-developed and tested codes for its solution. Generally, the algorithms involve stable implementations of the singular value decomposition (SVD) or some variant of the QR factorization. Thus it is reasonable to expect (3.17) to remain valid under perturbations due to roundoff error if h is not too small. However, for higher index problems, effects due to roundoff become more pronounced as $h \to 0^+$ for certain classes of numerical methods (e.g., BDF [39]).

The variations of parallel shooting based on the choice of parameters (v_j^0), (V_j^0) are restricted by the consistency requirements (2.5). Thus, for example, standard parallel shooting ($v_j^0 = 0$, $V_j^0 = I$) is generally not a valid approach for DAEs. However, at

least one important variation, the *stabilized march technique* [27], is legitimate in that the selections for v_j^0, V_j^0 automatically satisfy the constraint equations. We briefly sketch the details.

Suppose the boundary conditions are partially separated as in (3.6b). Represent $y_j(t)$ as in (3.9). Let $V_1^0 = U_a(m \times q)$, $v_1^0 = y_p(a)$, as in (3.7), (3.8). Inductively define V_{j+1}^0, for $j = 1, \cdots, J$ to be the columns of $V_j(\tau_j)$ orthogonalized, i.e.,

$$V_{j+1}^0 = V_j(\tau_j)P_j, \tag{3.18}$$

where P_j is a $q \times q$ nonsingular upper triangular matrix. But $\mathrm{im}(V_j(\tau_j)) = \ker(M_{\tau_j})$ so that

$$M_{\tau_j} V_{j+1}^0 = 0,$$

that is, V_{j+1}^0 is consistent for the homogeneous problem at $t_0 = \tau_j$. Defining v_{j+1}^0 as the projected component of $v_j(\tau_j)$ in $[\mathrm{im}(V_{j+1}^0)]^{\perp} = \mathrm{im}(M_{\tau_j}^T)$,

$$v_{j+1}^0 = [I - V_{j+1}^0(V_{j+1}^0)^T]v_j(\tau_j), \tag{3.19}$$

we have $M_{\tau_j} v_{j+1}^0 = M_{\tau_j} v_j(\tau_j) = g(\tau_j)$ so that v_{j+1}^0 is also consistent. Therefore algebraic manipulation of (3.14) leads to the system

$$\begin{pmatrix} -I & P_1 & & & \\ & -I & P_2 & & \\ & & \ddots & \ddots & \\ & & & -I & P_J \\ C_{ba}U_a & & & & C_b V_{J+1}^0 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_J \\ \mu_{J+1} \end{pmatrix} = \begin{pmatrix} P_1(V_2^0)^T v_1(\tau_1) \\ P_2(V_3^0)^T v_2(\tau_2) \\ \vdots \\ P_J(V_{J+1}^0)^T v_J(\tau_J) \\ \beta_b - [C_{ba}v_1^0 + C_b v_{J+1}^0] \end{pmatrix}, \tag{3.20}$$

which is square and nonsingular of size $(J+1)q$. Note that if $q = r$ (i.e., the boundary conditions are completely mixed), then the stabilized march version of (3.14) will be square and invertible also so that it will not be necessary to solve (3.14) as a linear least-squares problem.

Numerical methods. Let $G = (t_n)_0^N$ define a grid on $[a, b]$,

$$t_0 = a, \qquad t_n = t_{n-1} + h_n \ (1 \le n \le N), \qquad t_N = b,$$

where the stepsizes satisfy the boundedness criterion

$$h = \max(h_n) \le \theta \cdot \min(h_n) \tag{3.21}$$

for some $\theta > 0$, independent of n, h. Here we briefly discuss the use of numerical initial value methods to approximate the BVP solution $y(t)$ on the grid G by employing the shooting strategy previously described.

Suppose P is a globally $O(h^S)$-convergent method for the DAE IVP (3.1a), given $O(h^S)$ accurate starting values. Let $(u_{v,n})_{n=0}^N$ $(v = p,\ 1 \le v \le r)$ denote the numerical approximations for the $r+1$ IVPs in (3.2) generated by P. Then

$$\begin{aligned} &||u_{p,n} - y_p(t_n)|| \le Lh^S, \quad 0 \le n \le N, \\ &||u_{v,n} - \phi_v(t_n)|| \le Lh^S, \quad 0 \le n \le N, \quad 1 \le v \le r \end{aligned} \tag{3.22}$$

for some constant $L > 0$ (independent of n, $h \leq h_0$). Define the matrix $U_n = [u_{1,n}, u_{2,n}, \cdots, u_{r,n}]$ and let S_h be defined by

$$S_h = B_a U_0 + B_b U_N = S + O(h^S), \tag{3.23}$$

since $U_0 = Y(a)$ and $U_N = Y(b) + O(h^S)$. For $h > 0$ sufficiently small, $S_h^{-1} = S^{-1} + O(h^S)$, hence if ξ_h is the solution to the system

$$S_h \xi_h = \beta - [B_a y_p^0 + B_b u_{p,n}] = \beta - [B_a u_{p,0} + B_b u_{p,n}], \tag{3.24}$$

then $\xi_h = \xi + O(h^S)$, where ξ is the exact solution to the shooting equation (3.3). Note that the initial condition $y(a) = u_{p,0} + U_0 \xi_h$ is consistent if $u_{p,0}$ and U_0 are exact. Therefore, if (u_n) is the numerical approximation to the IVP

$$\mathcal{L}y(t) = f(t), \qquad y(a) = u_{p,0} + U_0 \xi_h \tag{3.25}$$

and P is stable under $O(h^S)$ perturbations in the initial starting values, then

$$||u_n - y(t_n)|| = O(h^S), \quad 0 \leq n \leq N, \tag{3.26}$$

where $y(t)$ is the unique solution to the BVP (3.1).

Clearly, any stable method that converges for the DAE IVP can be used to solve the BVP by shooting. But the class of methods that can be used to solve DAE IVPs is limited. Explicit methods may lead to systems of equations that cannot be solved uniquely for solution vectors at each timestep, and otherwise may not be applicable unless the constraint manifold is explicitly known at each timestep. Symmetric schemes have the problem that for fully-implicit index one and related higher index systems, there is a potential instability. This instability can often be corrected for BVPs by locating some of the consistency conditions at the correct boundary [2]. Even for IVPs, methods for DAEs must be very carefully chosen and implemented.

The traditional ODE methods that have been successful, namely BDF and some implicit Runge–Kutta (IRK) methods, converge and are stable for index one systems [21], [40] but do not converge for all higher index systems except in some cases where the system has a special structure ([20], [13], [17]). Even when these methods do work, they usually exhibit numerical behavior that is not characteristic of the same methods applied to nonstiff ODEs, although there are similarities to stiff ODEs. For example, constant stepsize BDF methods exhibit numerical boundary layers of instability and reduced order (or non-) convergence due to inconsistencies in the starting values. That is, the numerical solutions evolve for some fixed number of steps before achieving the order of convergence expected for ODEs (differential order). The instability is either local or transient in nature and is not a serious problem, unless the stepsize is extremely small. Furthermore, even bounded stepsize variation (3.21) will initiate new boundary layers if the index is greater than 2 [20], [39]. For parallel shooting, these considerations imply the existence of boundary layers on each subinterval $[\tau_{j-1}, \tau_j]$. Thus it is important to take h sufficiently small that the entries in the shooting equations (3.14), (3.20) are accurate to the desired order and not taken from the boundary layer, and large enough so that rounding errors do not dominate.

IRK methods are prone to global order reduction unless the method coefficients satisfy order conditions in addition to the differential order conditions [8], [40], [1], [9]. On the other hand, there exist IRKs that do not exhibit the boundary layer. For example, one can construct extrapolation methods based on the implicit Euler

method by taking enough steps at each stepsize so that the boundary layer has already passed. But here the boundary layer is hidden inside the stages. We do not know whether there exists an IRK method where the intermediate stages do not exhibit boundary layers.

More recently, the (i,j) methods based on (2.2) have been developed and analyzed for linear systems in [11], [12] and extended to nonlinear systems in [10]. The (i,j) methods are based on solving (2.2) for y'_n (and possibly higher order derivatives) in terms of y_n, t_n and then integrating for y_{n+1} using any consistent one-step method that is stable for ODEs. In principle, these methods can be used to solve any singular system that is solvable according to Definition 2.1. However, the (i,j) methods are computation intensive, as they require solving a $(mj) \times (mj)$ singular linear system involving derivatives of the coefficients and input at each step t_n. In practical applications, it may not be easy or even possible to obtain the necessary derivatives, especially if the functions are nonlinear.

4. Finite-difference methods. In this section we show that the more direct approach of finite-difference methods can be used to solve the DAE BVP if the IVP can be numerically solved, and the constraint manifold (2.5) is given at $t = a$. We consider difference approximations to (3.1) of the form

$$\mathcal{L}_h u_j = \sum_{k=0}^{N} C_{jk}(h) u_k = F_j(h,f) \quad j = 1,2,\cdots,N, \tag{4.1a}$$

$$\mathcal{B}_h \mathbf{u}_h = B_a u_0 + B_b u_N = \beta, \tag{4.1b}$$

$$M_a u_0 = g(a), \tag{4.1c}$$

where $\boldsymbol{u}_h = \{u_n\}_0^N$ is the approximation to the solution $\{y(t_n)\}_0^N$ and $h = \max(h_n)$ as before. We also assume that (4.1a) satisfies the property that $f \equiv 0$ implies $F_j(h,f) = 0$ for every j, h. In matrix form (4.1) can be written as

$$A_h \boldsymbol{u}_h = \boldsymbol{F}(h,f), \tag{4.2}$$

where

$$A_h = \begin{pmatrix} \begin{pmatrix} B_a \\ M_a \end{pmatrix} & 0 & \cdots & 0 & \begin{pmatrix} B_b \\ 0 \end{pmatrix} \\ C_{1,0} & C_{1,1} & \cdots & C_{1,N-1} & C_{1,N} \\ \vdots & \vdots & & \vdots & \vdots \\ C_{N,0} & C_{N,1} & \cdots & C_{N,N-1} & C_{N,N} \end{pmatrix}, \tag{4.3}$$

$$\begin{aligned} \boldsymbol{u}_h &= (u_0^T, u_1^T, \cdots, u_N^T)^T, \\ \boldsymbol{F}(h,f) &= ([\beta^T, g(a)^T]^T, F_1(h,f)^T, \cdots, F_N(h,f)^T)^T. \end{aligned}$$

Suppose the local truncation error associated with (4.1a,b) is $O(h^S)$. If (3.1a) is an ODE, then (4.1) is stable if and only if the family of matrices $\{A_h^{-1}\}$ is uniformly bounded as $h \to 0^+$. Convergence of the method to $O(h^S)$ accuracy then follows from stability and consistency. Furthermore, convergence is independent of the forcing function $f(t)$ and the boundary value β, since A_h is independent of these parameters. Unfortunately, when $E(t)$ is singular A_h^{-1} will in general contain terms

that are unbounded, as $h \to 0^+$. For example, if (4.1) is a constant stepsize implicit Euler method, then $||A_h^{-1}|| = O(h^{-\nu})$, where $\nu = \text{ind}(E, F)$. On the other hand, it is interesting to note that with a subtle variation the arguments used in the ODE case to relate convergence of (4.1) to the convergence for IVPs can be used when $E(t)$ is singular.

Consider two solvable DAE BVPs $BV^{(v)}$, $v = 0, 1$, with IVP (3.1a); boundary conditions

$$\mathcal{B}^{(v)}y \equiv B_a^{(v)}y(a) + B_b^{(v)}y(b) = \beta; \tag{4.4}$$

and difference matrices $A_h^{(v)}$, respectively. Before proving the main result in this section, we will need the following simple lemma.

LEMMA 4.1. *Let $y^{(v)}$, $v = 0, 1$, denote the solutions to $BV^{(v)}$ with boundary value β. Furthermore, let $Y^{(0)}$ be the fundamental solution matrix $(n \times r)$ to the* BVP

$$\begin{aligned} \mathcal{L}Y^{(0)} &= 0, \\ \mathcal{B}^{(0)}Y^{(0)} &= I, \end{aligned} \tag{4.5}$$

where I is the identity matrix of size $r \times r$. Then $y^{(1)}$ is the unique solution to $BV^{(0)}$, with boundary value

$$\tilde{\beta} = (\mathcal{B}^{(1)}Y^{(0)})^{-1}(\beta - \mathcal{B}^{(1)}y^{(0)} + (\mathcal{B}^{(1)}Y^{(0)})\beta). \tag{4.6}$$

Proof. Express $y^{(1)}$ as

$$y^{(1)}(t) = y^{(0)}(t) + Y^{(0)}(t)\xi,$$

where ξ solves

$$(\mathcal{B}^{(1)}Y^{(0)})\xi = \beta - \mathcal{B}^{(1)}y^{(0)}. \tag{4.7}$$

Now apply the boundary operator $\mathcal{B}^{(0)}$ to $y^{(1)}$ to get

$$\begin{aligned} \mathcal{B}^{(0)}y^{(1)} &= \mathcal{B}^{(0)}y^{(0)} + (\mathcal{B}^{(0)}Y^{(0)})\xi \\ &= \beta + I \cdot (\mathcal{B}^{(1)}Y^{(0)})^{-1}(\beta - \mathcal{B}^{(1)}y^{(0)}) \\ &= (\mathcal{B}^{(1)}Y^{(0)})^{-1}(\beta - \mathcal{B}^{(1)}y^{(0)} + (\mathcal{B}^{(1)}Y^{(0)})\beta). \end{aligned}$$

□

In the remainder of this section we prove the following theorem.

THEOREM 4.1. *Let $BV^{(v)}$, $v = 0, 1$, be solvable* BVP*s of the form* (3.1a), (4.4) *with right-hand side f_v and boundary value β_v, respectively. Consider the proposition $P^{(v)}$: $A_h^{(v)}$ is invertible and* (4.1) *converges to $O(h^S)$ accuracy to the solution of $BV^{(v)}$, independent of f_v and β_v. Then $P^{(0)}$ if and only if $P^{(1)}$.*

Proof. Assume $P^{(0)}$. Applying (4.1) to $BV^{(1)}$ with arbitrary but fixed $f_1 = f$ and $\beta_1 = \beta$ yields the system

$$A_h^{(1)}\boldsymbol{u}_h^{(1)} = \boldsymbol{F}(h, f) \tag{4.8}$$

or equivalently

$$[I + D_h(A_h^{(0)})^{-1}]A_h^{(0)}\boldsymbol{u}_h^{(1)} = \boldsymbol{F}(h, f), \tag{4.9}$$

where $D_h = A_h^{(1)} - A_h^{(0)}$:

$$D_h = \begin{pmatrix} \begin{pmatrix} B_a^{(1)} - B_a^{(0)} \\ 0 \end{pmatrix} & 0 & \cdots & 0 & \begin{pmatrix} B_b^{(1)} - B_b^{(0)} \\ 0 \end{pmatrix} \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}.$$

Thus, to show $A_h^{(1)}$ is invertible it suffices to show the invertibility of $[I + D_h(A_h^{(0)})^{-1}]$.

Partition $(A_h^{(0)})^{-1}$ as $(Z_{jk}^{(0)})$, $0 \le j, k \le N$, where each block $Z_{jk}^{(0)}$ is $m \times m$. Then

$$I + D_h(A_h^{(0)})^{(-1)} = \begin{pmatrix} Q_{h0} & Q_{h1} & \cdots & Q_{hN} \\ 0 & I & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I \end{pmatrix}, \tag{4.10}$$

where

$$\begin{aligned} Q_{h0} &= I + \begin{pmatrix} B_a^{(1)} - B_a^{(0)} \\ 0 \end{pmatrix} Z_{0,0}^{(0)} + \begin{pmatrix} B_b^{(1)} - B_b^{(0)} \\ 0 \end{pmatrix} Z_{N,0}^{(0)} \\ &= \begin{pmatrix} B_a^{(1)} Z_{0,0}^{(0)} + B_b^{(1)} Z_{N,0}^{(0)} \\ (0 \quad I) \end{pmatrix}, \end{aligned} \tag{4.11}$$

since $B_a^{(0)} Z_{0,0}^{(0)} + B_b^{(0)} Z_{N,0}^{(0)} = (I \quad 0)$ by definition of the $Z_{jk}^{(0)}$. Also,

$$\begin{aligned} Q_{hj} &= \begin{pmatrix} B_a^{(1)} - B_a^{(0)} \\ 0 \end{pmatrix} Z_{0,j}^{(0)} + \begin{pmatrix} B_b^{(1)} - B_b^{(0)} \\ 0 \end{pmatrix} Z_{N,j}^{(0)} \\ &= \begin{pmatrix} B_a^{(1)} Z_{0,j}^{(0)} + B_b^{(1)} Z_{N,j}^{(0)} \\ 0 \end{pmatrix}, \quad j = 1, \cdots, N. \end{aligned} \tag{4.12}$$

Partition $Z_{j,0}^{(0)} = (Z_{j,0,1}^{(0)} \mid Z_{j,0,2}^{(0)})$, where $Z_{j,0,1}^{(0)}$ is $m \times r$. Then the column of blocks $\{Z_{j,0,1}^{(0)}\}_0^N$ is the solution to

$$A_h^{(0)} u_h = ((I \ 0)^T, 0, \cdots, 0)^T, \tag{4.13}$$

where I is $r \times r$, $(I \ 0)^T$ is $m \times r$, and the remaining zeros are $m \times r$. That is, $\{Z_{j,0,1}^{(0)}\}_0^N$ is the difference approximation to the homogeneous BVP (4.5). Note that solvability of (4.5) follows from the solvability of $BV^{(0)}$. Furthermore, since $BV^{(1)}$ is solvable, the shooting matrix

$$\mathcal{B}^{(1)} Y^{(0)} = B_a^{(1)} Y^{(0)}(a) + B_b^{(1)} Y^{(0)}(b)$$

is invertible. Now

$$\begin{aligned} B_a^{(1)} Z_{0,0}^{(0)} + B_b^{(1)} Z_{N,0}^{(0)} &= \left(B_a^{(1)} Z_{0,0,1}^{(0)} + B_b^{(1)} Z_{N,0,1}^{(0)} \mid B_a^{(1)} Z_{0,0,2}^{(0)} + B_b^{(1)} Z_{N,0,2}^{(0)} \right) \\ &= (\tilde{Q}_{h,0,1} \mid \tilde{Q}_{h,0,2}). \end{aligned} \tag{4.14}$$

If (4.1) globally converges to $O(h^S)$ accuracy for $BV^{(0)}$ independent of f and β, then it does so for (4.5), implying $Z_{0,0,1}^{(0)} = Y^{(0)}(a) + O(h^S)$ and $Z_{N,0,1}^{(0)} = Y^{(0)}(b) + O(h^S)$.

Hence $\tilde{Q}_{h,0,1} = \mathcal{B}^{(1)}Y^{(0)} + O(h^S)$ is invertible, for h sufficiently small. It follows that $A_h^{(1)}$ is invertible, since the (1,1) block $Q_{h,0}$ in (4.10) is given by

$$Q_{h,0} = \begin{pmatrix} \tilde{Q}_{h,0,1} & \tilde{Q}_{h,0,2} \\ 0 & I \end{pmatrix}. \tag{4.15}$$

Now we show that (4.1) converges for $BV^{(1)}$. From (4.9) we have that $\boldsymbol{u}_h^{(1)}$ is the unique solution to

$$A_h^{(0)}\boldsymbol{u}_h^{(1)} = [I + D_h(A_h^{(0)})^{-1}]^{-1}\boldsymbol{F} = \tilde{\boldsymbol{F}}. \tag{4.16}$$

From the structure of $[I + D_h(A_h^{(0)})^{-1}]$ given by (4.10), (4.11), and (4.15), we have

$$\tilde{\boldsymbol{F}} = ([\tilde{\beta}_h^T, g(a)^T]^T, F_1^T, \cdots, F_N^T)^T, \tag{4.17}$$

where

$$\tilde{\beta}_h = \tilde{Q}_{h,0,1}^{-1}\left(\beta - \tilde{Q}_{h,0,2}g(a) - \sum_{j=1}^{N}(B_a^{(1)}Z_{0,j}^{(0)} + B_b^{(1)}Z_{N,j}^{(0)})F_j\right). \tag{4.18}$$

That is, $\boldsymbol{u}_h^{(1)}$ is the difference approximation to $BV^{(0)}$, with input f and boundary value $\tilde{\beta}_h$. Simplifying (4.18), we obtain

$$\begin{aligned}\tilde{\beta}_h &= \tilde{Q}_{h,0,1}^{-1}\Bigg(\beta - \Bigg(B_a^{(1)}\Bigg(Z_{0,0,2}^{(0)}g(a) + \sum_{j=1}^{N} Z_{0,j}^{(0)}F_j\Bigg) \\ &\qquad + B_b^{(1)}\Bigg(Z_{N,0,2}g(a) + \sum_{j=1}^{N} Z_{N,j}^{(0)}F_j\Bigg)\Bigg)\Bigg) \\ &= \tilde{Q}_{h,0,1}^{-1}(\beta - (aB_a^{(1)}(u_{h,0}^{(0)} - Z_{0,0,1}^{(0)}\beta) + B_b^{(1)}(u_{h,N}^{(0)} - Z_{N,0,1}^{(0)}\beta))).\end{aligned}$$

Thus

$$\tilde{\beta}_h = \tilde{Q}_{h,0,1}^{-1}(\beta - \mathcal{B}^{(1)}\boldsymbol{u}_h^{(0)} + \tilde{Q}_{h,0,1}\beta), \tag{4.19}$$

where $\boldsymbol{u}_h^{(0)}$ solves $A_h^{(0)}\boldsymbol{u}_h^{(0)} = \boldsymbol{F}$, i.e., $\boldsymbol{u}_h^{(0)}$ is the difference approximation to $BV^{(0)}$ with input f and boundary value β. Since (4.1) converges for $BV^{(0)}$ independent of f, β by hypothesis, $\tilde{\beta}_h$ is bounded independent of h for small $h > 0$.

Let $y^{(v)}$, $v = 0, 1$, denote the exact solution to $BV^{(v)}$ with input f and boundary value β. From (4.19), we have

$$\tilde{\beta}_h = \tilde{\beta} + \Delta, \tag{4.20}$$

where

$$\tilde{\beta} = (\mathcal{B}^{(1)}Y^{(0)})^{-1}(\beta - (\mathcal{B}^{(1)}y^{(0)} + \mathcal{B}^{(1)}Y^{(0)})\beta) \tag{4.21}$$

and $||\Delta|| = O(h^S)$. From Lemma 4.1, $y^{(1)}$ is the solution to $BV^{(0)}$, with boundary value $\tilde{\beta}$. We wish to estimate $||u_{h,j}^{(1)} - y^{(1)}(t_j)||$. By the triangle inequality,

$$||u_{h,j}^{(1)} - y^{(1)}(t_j)|| \le ||u_{h,j}^{(1)} - \tilde{y}_h(t_j)|| + ||\tilde{y}_h(t_j) - y^{(1)}(t_j)||, \tag{4.22}$$

where $\tilde{y}_h$ is the exact solution to $BV^{(0)}$ with input f and boundary value $\tilde{\beta}_h$. By the argument immediately following (4.18) and hypothesis $P^{(0)}$, we have that $||u^{(1)}_{h,j} - \tilde{y}_h(t_j)|| = O(h^S)$. Furthermore, $||\tilde{y}_h(t_j) - y^{(1)}(t_j)|| = O(h^S)$ follows by variation of parameters using (4.20) and the uniform boundedness of $||Y^{(0)}(t)x||$ on $[a,b]$, i.e.,

$$\begin{aligned} ||\tilde{y}_h(t_j) - y^{(1)}(t_j)|| &= ||Y^{(0)}(t_j)(\tilde{\beta}_h - \tilde{\beta})|| \\ &\leq ||Y^{(0)}(t_j)||\,||\Delta|| \\ &= O(h^S). \end{aligned}$$

□

We note that (4.1) need not be globally convergent in order for Theorem 4.1 to remain a valid result in the context of DAEs, where boundary layers may exist. We only require $||Z^{(0)}_{0,0,1} - Y^{(0)}(a)|| = O(h^S)$ and N to be sufficiently large such that $||Z^{(0)}_{N,0,1} - Y^{(0)}(b)|| = O(h^S)$. Thus, by associating $BV^{(0)}$ with the IVP

$$\mathcal{L}y = f, \quad y_0 \in M_f(a), \tag{4.23}$$

and taking $B^{(0)}_a$ so that $[B_a^T, M_a^T]^T$ is invertible, we get Corollary 4.1.

COROLLARY 4.1. *Suppose* (4.1) *converges with* $O(h^S)$ *accuracy to a solution of* (4.23) *for* $n \geq J$ *when initial values are consistent to* $O(h^S)$. *Then* (4.1) *converges to a solution of the boundary value problem* $BV^{(1)}$ *to* $O(h^S)$ *accuracy for* $n \geq J$.

5. Numerical experiments. In this section we present the results of some numerical experiments on linear and nonlinear index one and index two DAE BVPs. The experiments confirm the results of the theory and also raise some interesting questions.

The numerical experiments described in this section were all performed using the finite-difference methods formulated as described in § 4. The nonlinear equations at each timestep were solved by Newton iteration. The iteration was terminated when the ℓ_2 norm of the difference between two successive iterates was less than a specified tolerance. An analytic iteration matrix was provided to the code for all of the problems. All of the computations were performed in double precision on an Alliant FX/8 computer.

The first test problem was a linear variable-coefficient index one DAE BVP on [0,1] given by

$$\begin{pmatrix} 1 & -t & t^2 \\ 0 & 1 & -t \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} y_1' \\ y_2' \\ y_3' \end{pmatrix} + \begin{pmatrix} 1 & -(t+1) & (t^2+2t) \\ 0 & -1 & (t-1) \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \sin(t) \end{pmatrix}, \tag{5.1}$$

with boundary conditions

$$\begin{aligned} y_1(0) &= 1, \\ y_2(1) - y_3(1) &= e. \end{aligned}$$

This problem has true solution

$$\begin{aligned} y_1 &= e^{-t} + te^t, \\ y_2 &= e^t + t\sin(t), \\ y_3 &= \sin(t). \end{aligned}$$

This system is related to the linear constant coefficient DAE system

$$\begin{aligned} y_1' &= -y_1, \\ y_2' &= y_2, \\ y_3 &= \sin(t), \end{aligned}$$

with boundary conditions $y_1(0) = 1$, $y_2(1) = e$ by the nonsingular spatially dependent change of variables $y = Q\tilde{y}$, where

$$Q = \begin{pmatrix} 1 & t & 0 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{pmatrix}.$$

This problem was constructed to show the effects, if any, of spatially dependent coupling between the differential and algebraic parts of the system on solution accuracy.

We solved this linear problem using the implicit midpoint method, with the number of steps in [0,1] ranging from 2 to 64. We computed the approximate order by comparing the errors of the solution with a given mesh spacing to the errors obtained by halving the mesh spacing. In our tests, the error behaved consistently as $O(h^2)$. It is to be expected from Corollary 4.2 (together with the results in [1], [9]) that the implicit midpoint method is globally $O(h^2)$ when applied directly to index one DAE IVPs.

Next we solved a simplified model of a steady-state semiconductor device described in [1] with the same method. The system has the form

$$\begin{aligned} 0 &= n - p - C(t), \\ J_n' &= 0, \\ J_p' &= 0, \\ J_n &= (n' - n\psi'), \\ J_p &= -(p' + p\psi'), \end{aligned} \tag{5.2}$$

and the boundary conditions are given by

$$\begin{aligned} n &= 1/2(C + \sqrt{C^2 + 4\delta^4}), \qquad t = +1, -1, \\ p &= 1/2(-C + \sqrt{C^2 + 4\delta^4}), \qquad t = +1, -1, \\ \psi &= \psi_{bi}(t) + \frac{1}{2}(t+1)V, \qquad t = +1, -1, \end{aligned}$$

where $\psi_{bi}(t) = \ln n(t) - \ln \delta^2$. For the free parameters and functions, we have chosen $\delta = 10^{-4}$, $V = 1$, and $C(t) = 1/2 + (\tan^{-1}(\lambda t))/\pi$, where λ is a parameter that determines the steepness of $C(t)$. $C(t)$ is constructed to be an approximation to a square wave for sufficiently large λ.

We rewrote the problem in a form that was easier to understand by performing a nonsingular constant change of variables $J_+ = J_n + J_p$, $J_- = J_n - J_p$, $N_+ = n + p$, $N_- = n - p$ to arrive at the system

$$\begin{aligned} 0 &= N_- - C(t), \\ J_+' &= 0, \\ J_-' &= 0, \\ J_+ &= (N_-' - N_+\psi'), \\ J_- &= (N_+' - N_-\psi'), \end{aligned} \tag{5.3}$$

with boundary conditions

$$\begin{aligned} N_+ &= \sqrt{C^2 + 4\delta^4}, \qquad t = +1, -1, \\ \psi &= \psi_{bi} + \frac{1}{2}(t+1)V, \qquad t = +1, -1. \end{aligned}$$

This is the proper number of boundary conditions for this problem, which is index one. Note that in the original formulation of (5.2), two of the boundary conditions are redundant. The numerical solution of (5.3) will be the same as that for (5.2), apart from errors due to roundoff, because the problems are related by a constant nonsingular change of variables.

We can substitute $N_- = C(t)$ into the system (5.3), to obtain the related ODE system

$$\begin{aligned} J_+' &= 0, \\ J_-' &= 0, \\ \psi' &= (C'(t) - J_+)/N_+, \\ N_+' &= J_- + C(t)(C'(t) - J_+)/N_+, \end{aligned} \tag{5.4}$$

which we will refer to later.

We solved (5.3) for $\lambda = 20$ with a fixed stepsize and the number of steps ranging from 2 to 2056. We found that for large stepsizes (fewer than approximately 64 steps) and for an odd number of nodes in the interval, there is a problem with oscillations in the solution for $t > 0$. After the stepsize is decreased sufficiently so that the solution is resolved, there is no longer a problem with oscillations, and the error behaves as $O(h^2)$. For an even number of nodes, there is no problem with oscillations, and the error behaves as $O(h^2)$. We observed similar behavior for larger values of λ, where the oscillation disappears when there are a sufficient number of meshpoints to accurately resolve the solution.

There is an explanation for this oscillating behavior, which occurs only for an odd number of nodes. In this test problem, the most rapid change in the solution occurs at $t = 0$. For an even number of meshpoints, $t = 0$ is in the center of a mesh interval, whereas for an odd number of meshpoints there is a meshpoint at $t = 0$. It is easy to see that for the midpoint method applied to an algebraic equation whose solution is a step function, the solution is smooth if the step occurs at the center of a mesh interval, and it oscillates otherwise. Since $C(t)$ approximates a step function, the behavior we have observed with respect to odd and even numbers of meshpoints is what we would expect.

Of course, in this test problem if we did not discretize the algebraic equation with the implicit midpoint method but instead evaluated N_- always at the meshpoints, then the oscillation would disappear. This would give the same results as solving the related ODE system (5.4) with the implicit midpoint method. However, it is not always so easy to isolate the algebraic variable in applications, so we are interested in seeing the effects of not treating that variable specially.

In comparison with the midpoint method applied to the related ODE, the ODE formulation does not exhibit any oscillations. On the other hand, the DAE formulation is apparently much less sensitive to perturbations in the initial guess for N_+. In our experiments, the ODE solution only converged for initial guesses for N_+ that were

very near the exact solution for that variable, whereas the DAE formulation converged for a much wider range of initial values.

The third problem that we tested was a linear index two system on [0,1] given by

$$\begin{aligned} y_1' - t &= -y_1 + (1-t)y_2, \\ 0 &= \beta y_1 + (-1-\beta t)y_2 + \sin(t), \\ z_1 &= y_1', \\ z_2 &= y_2', \end{aligned} \tag{5.5}$$

with boundary conditions

$$\begin{aligned} y_1(0) &= 1, \\ \beta y_1(1) + (-1-\beta)y_2(1) &= -\sin(1). \end{aligned}$$

This problem is an index two extension of an index one problem proposed by Ascher [2]. The boundary conditions are posed such that the midpoint method is stable for the index one part of the problem. We solved the problem for $\beta = 10$ and for $\beta = 100$. The errors in y_1 and y_2 behaved consistently as $O(h^2)$. However, the midpoint method is not convergent for the index two variables z_1 and z_2, even for IVPs, and for this problem the solutions for these variables exhibited oscillations whose amplitudes did not decrease as $h \to 0$. By modifying the formula slightly and discretizing the subsystem $z = y'$ by

$$z_n = \frac{y_{n+1} - y_{n-1}}{2h},$$

the formula becomes $O(h^2)$ accurate for z, IVPs and BVPs. In our experiments the modified formula was $O(h^2)$ accurate for y and z, and the numerical results exhibited no oscillations. These experiments confirm the results of § 4.

One interesting point on this index two example is that the condition number of the matrix generated by straightforward application of the finite-difference technique is $O(h^{-4})$. We experimented with several different scalings of the DAE system that reduced the condition number to $O(h^{-3})$, but the scaling had almost no effect on the errors that were obtained.

It is possible to transform a semi-explicit index two system

$$\begin{aligned} y' &= f(y,z), \\ 0 &= g(y,z) \end{aligned}$$

to an index one system

$$\begin{aligned} y' &= f(y,w'), \\ 0 &= g(y,w') \end{aligned}$$

coupled with $w' = z$ [19]. Using this transformation and then scaling the resulting matrix reduces the condition number to $O(h^{-1})$. However, in our experiments we found that the errors in z did not change appreciably from the other formulations.

The final problem that we tested was a linear index two system. This problem has the property that the matrix $E(t)$ is not of constant rank. The problem is given

on $[0,1]$ by

$$\begin{pmatrix} (2+t) & 1 & -t \\ -2 & -1 & 0 \\ -t(t+1) & 0 & t(t+1) \end{pmatrix} y' + \begin{pmatrix} (1-t^2) & 2 & (t^2-1) \\ -3 & -1 & 1 \\ (2+t) & -(1+t) & t \end{pmatrix} y = \begin{pmatrix} \sin(t) + e^{-t}t^2 - e^t \\ e^{-t} - \sin(t) \\ e^t(t+1) - e^{-t} \end{pmatrix},$$

with boundary conditions

$$\begin{pmatrix} 3 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 1 \end{pmatrix} y(0) + \begin{pmatrix} -2 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} y(1) = \begin{pmatrix} -(e+\frac{3}{2}) \\ 1 \\ 1 \end{pmatrix}.$$

This problem can be obtained via a nonsingular constant change of variables and a nonsingular time-dependent scaling of the system from a simpler index two system for which the matrix $E(t)$ is also not of constant rank. The results of Clark [18] imply that the implicit Euler method converges with order $O(h)$ for IVPs of this type. Thus the results in § 4 imply that the implicit Euler method, formulated as in § 4, should yield $O(h)$ accuracy for this problem. Our numerical experiments confirm these conclusions. It should be noted that it is possible to use a one-sided difference scheme for this problem because it is not stiff, and also that there are index two problems that are not in semi-explicit form for which the implicit Euler method, as well as more general Runge–Kutta and multistep methods, is not stable [21]. The results of this paper imply that a method is convergent for the BVP if and only if it is convergent for the related IVP. They do not make any statements about which methods are convergent for the IVP.

Acknowledgment. The authors would like to thank Clement Ulrich for his assistance in performing the numerical experiments.

REFERENCES

[1] U. ASCHER, *On numerical differential algebraic problems with application to semiconductor device simulation,* SIAM J. Numer. Anal., 26(1989), pp. 517-538.

[2] ——— *On symmetric schemes and differential-algebraic equations,* SIAM J. Sci. Statist. Comput., this issue, pp. 937–949.

[3] U. ASCHER, R. M. MATTHEIJ, AND R. RUSSELL, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations,* Englewood Cliffs, NJ, Prentice Hall, 1988.

[4] J. BETTS, T. BAUER, W. HUFFMAN, AND K. ZONDERVAN, *Solving the optimal control problem using a nonlinear programming technique, Part* I*: General formulation,* AIAA-84-2037, Proc. 1984 AIAA/ AAS Astrodynamics Conference, Seattle, WA, 1984.

[5] H. G. BOCK, E. EICH, AND J. P. SCHLÖDER, *Numerical Solution of Constrained Least Squares Boundary Value Problems in Differential-Algebraic Equations,* Universität Heidelberg, Preprint No. 440, December 1987.

[6] K. BRENAN, *Numerical simulation of trajectory prescribed path control problems,* IEEE Trans. Automat. Control, AC-31, (1986), pp. 266-269.

[7] K. BRENAN AND B. ENGQUIST, *Backward differentiation approximations of nonlinear differential-algebraic systems,* Aerospace Corp. Report ATR-85(9990)-5, 1985.

[8] K. E. BRENAN AND L. R. PETZOLD, *The numerical solution of higher index differential-algebraic equations by implicit Runge–Kutta methods,* UCRL-95905, Lawrence Livermore National Laboratory, 1986; SIAM J. Numer. Anal., 26(1989), pp. 981-1001.

[9] K. BURRAGE AND L. PETZOLD, *On order reduction for Runge–Kutta methods applied to differential-algebraic systems and to stiff systems of* ODEs, UCRL-98046, Lawrence Livermore National Laboratory, Livermore, CA, 1988.

[10] S. L. CAMPBELL, *A computational method for general higher index nonlinear singular systems of differential equations,* CSRC Tech. Report 063087-01, Center for Research in Scientific Computation, North Carolina State University, Raleigh, NC, 1987.

[11] ——— *A general form for solvable linear time varying singular systems of differential equations,* SIAM J. Math Anal., 18 (1987), pp. 1101-1115.

[12] ——— *The numerical solution of higher index, linear time varying singular systems of differential equations,* SIAM J. Sci. Statist. Comput., 6(1985), pp. 334-348.

[13] ——— *One canonical form for higher index, linear time varying singular systems,* Circuits Systems Signal Process., 2(1983), pp. 311-326.

[14] ——— *Singular Systems of Differential Equations* II, Pitman, Marshfield, MA, 1982.

[15] S. L. CAMPBELL AND C. D. MEYER, JR., *Generalized Inverses of Linear Transformations,* Pitman, Marshfield, MA, 1979.

[16] K. CLARK, *The numerical solution of some higher index time varying semistate systems by difference methods,* Circuits Systems and Signal Process., 6(1987), pp. 261-275.

[17] ——— *A structural form for higher index semistate equations* I: *Theory and applications to circuit and control theory,* Linear Algebra Appl., 98 (1988), pp. 169-197.

[18] ——— *On structure and the numerical solution of singular systems,* in Recent Advances in Singular Systems, Proc. International Conference on Singular Systems, Atlanta, GA, 1987, pp. 41-45.

[19] C. W. GEAR, *Differential-algebraic equation index transformations,* SIAM J. Sci. Statist. Comput., 9(1988), pp. 39-47.

[20] C. W. GEAR, G. K. GUPTA, AND B. LEIMKUHLER, *Automatic integration of Euler–Lagrange equations with constraints,* J. Comput. Appl. Math., 12&13(1985), pp. 77-90.

[21] C. W. GEAR AND L. R. PETZOLD, ODE *methods for the solution of differential/algebraic systems,* SIAM J. Numer. Anal., 21(1984), pp. 716-728.

[22] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations,* The John Hopkins University Press, Baltimore, MD, 1983.

[23] E. GRIEPENTROG AND R. MÄRZ, *Differential-Algebraic Equations and Their Numerical Treatment,* Teubner-Texte zur Mathematik, Band 88, Leipzig, GDR, 1986.

[24] M. HANKE, *On a least-squares collocation method for linear differential-algebraic equations,* Numer. Math., 54 (1988), pp. 79-90.

[25] R. J. KEE, L. R. PETZOLD, M. D. SMOOKE, AND J. F. GRCAR, *Implicit methods in combustion and chemical kinetics modeling,* in Multiple Time Scales, J. Brackbill and B. Cohen, eds., Academic Press, New York, 1985, pp. 113-144.

[26] H. B. KELLER, *Approximation methods for nonlinear problems with application to two-point boundary value problems,* Math. Comp., 29(1975), pp. 464-474.

[27] ——— *Numerical Solution of Two-Point Boundary Value Problems,* Regional Conference Series in Applied Mathematics, Vol. 24, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1976.

[28] I. J. KIRBY AND G. A. LEIPER, *A small divergent detonation theory for intermolecular explosives,* Proc. 8th International Symposium on Detonation, Albuquerque, NM, (1985), pp. 760-768.

[29] M. LENTINI AND R. MÄRZ, *The condition of boundary value problems in transferable differential-algebraic equations,* Humboldt–Universität, Sekt. Math., Berlin, GDR, 1987.

[30] ——— *Conditioning and dichotomy in differential algebraic equations,* Humboldt–Universität, Sekt. Math., Berlin, GDR, 1988.

[31] P. LÖTSTEDT, *Discretization of singular perturbation problems by* BDF *methods,* Report No. 99, Dept. Computer Science, Uppsala University, Uppsala, Sweden, 1985.

[32] ——— *On the relationship between singular perturbation problems and differential-algebraic equations,* Report No. 100, Dept. Computer Science, Uppsala University, Uppsala, Sweden 1985.

[33] P. LÖTSTEDT AND L. R. PETZOLD, *Numerical solution of nonlinear differential equations with algebraic constraints: Convergence results for the backward differentiation formulas,* Math. Comp., 46(1986), pp. 491-516.

[34] R. MÄRZ, *On difference and shooting methods for boundary value problems in differential/algebraic equations,* Preprint No. 24, Humboldt-Universitat Sekt. Math. Berlin, GDR, 1982.

[35] N. H. MCCLAMROCH, *Singular systems of differential equations as dynamic models for constrained robot systems,* Tech. Report RSD-TR-2-86, Center for Research on Integrated Manufacturing, College of Engineering, Univ. Michigan, Ann Arbor, MI, 1986.

[36] R. W. NEWCOMB, *The semistate description of nonlinear time variable circuits,* IEEE Trans. Circ. Sys., CAS-28(1981), pp. 62-71.
[37] B. OWREN, personal communication, 1987.
[38] J. F. PAINTER, *Solving the Navier–Stokes equations using* LSODI *and the method of lines,* Report UCID-19262, Lawrence Livermore National Laboratory, Livermore, CA, 1981.
[39] L. R. PETZOLD, *Differential-algebraic equations are not* ODEs, SIAM J. Sci. Statist. Comput., 3(1982), pp. 367-384.
[40] ——— *Order results for implicit Runge–Kutta methods applied to differential/algebraic systems,* SIAM J. Numer. Anal., 23(1986), pp. 837-852.
[41] L. R. PETZOLD AND P. LÖTSTEDT, *Numerical solution of nonlinear differential equations with algebraic constraints* II*: Practical implications,* SIAM J. Sci. Statist. Comput., 7(1986), pp. 720-733.
[42] W. RHEINBOLDT, *Differential-algebraic systems as differential equations on manifolds,* Math. Comp., 4(1984), pp. 473-482.
[43] R. SINCOVEC, A. M. ERISMAN, E. L. YIP, AND M. A. EPTON, *Solvability of large scale descriptor systems,* Final Report, Boeing Computer Services Co., Tukwila, WA, 1979.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 937-949, September 1989
© 1989 Society for Industrial and Applied Mathematics
005

ON SYMMETRIC SCHEMES AND DIFFERENTIAL-ALGEBRAIC EQUATIONS*

URI ASCHER†

Abstract. An example is given that demonstrates a potential risk in using symmetric difference schemes for initial value differential-algebraic equations (DAEs) or for very stiff ordinary differential equations (ODEs). The basic difficulty is that the stability of the scheme is controlled by the stability of an auxiliary (ghost) ODE problem that is not necessarily stable even when the given problem is.

The stability of symmetric schemes is better understood in the context of boundary value problems. In this context, such schemes are more naturally applied as well. For initial value problems, better alternatives may exist. A computational algorithm is proposed for boundary value index-1 DAEs.

Key words. symmetric difference schemes, differential-algebraic equations, stability constant

AMS(MOS) subject classification. 65L10

1. Introduction. The possibility of using a symmetric difference scheme, like the midpoint scheme (collocation at one Gauss point), for solving differential-algebraic equations (DAEs), has been considered recently in the literature. This appears to be particularly attractive for fully implicit index-1 boundary value problems (BVPs). However, as we demonstrate in § 2, a careless use of such schemes, even for initial value problems (IVPs), can be dangerous.

We consider the linear DAE

$$E(t)\mathbf{x}' = A(t)\mathbf{x} + \mathbf{q}(t), \qquad 0 < t < 1, \tag{1}$$

where $E(t)$ is a singular matrix with constant rank. For a nonlinear DAE

$$\boldsymbol{\phi}(t, \mathbf{x}, \mathbf{x}') = 0, \qquad 0 < t < 1,$$

we have in mind a quasilinearization method (see, e.g., [AMR, § 2.3.4]) that yields at each iteration a linear DAE like (1) with

$$E(t) = \frac{\partial \boldsymbol{\phi}(t, \hat{\mathbf{x}}(t), \hat{\mathbf{x}}'(t))}{\partial \mathbf{x}'},$$

$\hat{\mathbf{x}}(t)$ being the current iterate.

Following the example, we will analyze the situation in §§ 3 and 4. Conclusions are offered in § 5. These suggest that for IVPs, better alternatives than using symmetric schemes may exist. But for BVPs, pursuing symmetric schemes is more worthwhile, at least for DAEs with index 1 (see [As2], [C1Pe] and references therein for some applications). We propose an algorithm for this class of problems.

2. Example. Consider the IVP (1) with

$$E = \begin{pmatrix} 1 & -t \\ 0 & 0 \end{pmatrix}, \quad A = \begin{pmatrix} -1 & 1+t \\ \beta & -1-\beta t \end{pmatrix}, \quad \mathbf{q} = \begin{pmatrix} 0 \\ \sin t \end{pmatrix}, \tag{2}$$

β a parameter, and $x_1(0) = 1$.

* Received by the editors on July 11, 1988; accepted for publication (in revised form) October 5, 1988. This research was partly supported by Natural Sciences and Engineering Research Council operating grant 4306.

† Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1W5.

This problem is an extension of the example of Petzold [Pe] who considered it for the parameter value $\beta = 0$. The transformation of variables

$$\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} x_2 \\ x_1 - tx_2 \end{pmatrix} \equiv T^{-1}x, \qquad T^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & -t \end{pmatrix} \tag{3}$$

gives the equations

$$0 = -y + \beta z + \sin t, \tag{4a}$$

$$z' = -z \tag{4b}$$

with $z(0) = 1$. The unique solution is therefore

$$z = e^{-t}, \quad y = \beta e^{-t} + \sin t, \quad x_2 = y, \quad x_1 = t \sin t + (1 + \beta t) e^{-t}. \tag{5}$$

To apply the midpoint scheme

$$E(t_{i+1/2}) \frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{h_i} = A(t_{i+1/2}) \frac{\mathbf{x}_i + \mathbf{x}_{i+1}}{2} + \mathbf{q}(t_{i+1/2}), \qquad 1 \leqq i \leqq N, \tag{6}$$

on a mesh

$$\begin{aligned} &\pi: \quad 0 = t_1 < t_2 < \cdots < t_N < t_{N+1} = 1, \\ &h_i := t_{i+1} - t_i, \quad h := \max_{1 \leqq i \leqq N} h_i, \quad t_{i+1/2} := t_i + \tfrac{1}{2} h_i, \end{aligned} \tag{7}$$

we need another side value (in addition to the given one on x_1) that we take as the exact initial value

$$x_2(0) = \beta.$$

In Tables 1(a)-1(f) and 2(a)-2(f) we list the errors and computed rates of convergence at $t = 1$ when uniform stepsizes $h = 1/N$ are used for various values of β. The error magnitudes in $x_j(1)$ are listed under err j, the convergence rates under rate j, $j = 1, 2$. For each value of β for which solutions are computed we list in a separate table results where the coarsest mesh is with $h = 0.2/|\beta|$ ($h = 0.2$ for $\beta = 0$ as well), and then refining by halving the stepsize a number of times. While all meshes used are uniform, there appears to be no reason to take nonuniform meshes here. The computations were performed on a SUN 3 running a UNIX f77 compiler in double precision.

For Tables 1(a)-1(f) we use the midpoint scheme (6).

Tables 1(a) and 1(b) indicate good results for small values of β. But Tables 1(c)-1(e) show an exponential increase in the error size as β is increased. The computed rates of convergence, especially in Table 1(e), clearly show that the difficulty is *not* in just roundoff error accumulation. Table 1(f) shows that the difficulty is not an approximation question either, as it does not occur for $\beta < 0$.

To emphasize the point and clarify the difficulty further, we list in Tables 2(a)-2(f) results of comparable runs, made with the backward Euler scheme

$$E(t_{i+1}) \frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{h_i} = A(t_{i+1})\mathbf{x}_{i+1} + \mathbf{q}(t_{i+1}), \tag{8}$$

instead of the midpoint scheme.

The results in Tables 2(a)-2(f) demonstrate the first-order convergence rate of the backward Euler scheme. (An exception is in Table 2(a), where $x_2(1)$ is reproduced

TABLE 1(a)
$\beta = 0$, $x_1(1) = 0.121+01$, $x_2(1) = 0.841+00$.

h	err 1	err 2	rate 1	rate 2
0.200+00	0.212−02	0.422−02		
0.100+00	0.524−03	0.105−02	0.202+01	0.200+01
0.500−01	0.131−03	0.263−03	0.200+01	0.200+01
0.250−01	0.326−04	0.657−04	0.200+01	0.200+01
0.125−01	0.816−05	0.164−04	0.200+01	0.200+01

TABLE 1(b)
$\beta = 1$, $x_1(1) = 0.158+01$, $x_2(1) = 0.121+01$.

h	err 1	err 2	rate 1	rate 2
0.200+00	0.115−02	0.380−02		
0.100+00	0.143−03	0.798−03	0.302+01	0.225+01
0.500−01	0.416−04	0.205−03	0.178+01	0.196+01
0.250−01	0.108−04	0.517−04	0.195+01	0.199+01
0.125−01	0.272−05	0.129−04	0.199+01	0.200+01

TABLE 1(c)
$\beta = 10$, $x_1(1) = 0.489+01$, $x_2(1) = 0.452+01$.

h	err 1	err 2	rate 1	rate 2
0.200−01	0.202+03	0.202+03		
0.100−01	0.498+02	0.498+02	0.202+01	0.202+01
0.500−02	0.124+02	0.124+02	0.201+01	0.201+01
0.250−02	0.310+01	0.310+01	0.200+01	0.200+01
0.125−02	0.774+00	0.774+00	0.200+01	0.200+01

because the backward Euler scheme reproduces the algebraic equations (4a) at mesh-points, and when $\beta = 0$ this determines $y = x_2$ exactly.) Thus, it is not surprising to see that the second-order midpoint scheme performs much more accurately for $\beta = 0, 1, -100$. However, for larger and positive values of β the backward Euler scheme obviously does not share whatever it is that is bothering the midpoint scheme, and so produces better results for the range of h listed.

The demonstrated difficulty is not restricted to DAEs: If we replace (4a) by the ODE

$$\varepsilon y' = -y + \beta z + \sin t \tag{9a}$$

$0 < \varepsilon \ll 1$, and then transform to $\mathbf{x}$, yielding a replacement of $E(t)$ in (2) by

$$E(t;\varepsilon) = \begin{pmatrix} 1 & -t \\ 0 & \varepsilon \end{pmatrix}, \tag{9b}$$

then a very stiff ODE is obtained. The IVP for this ODE exhibits a similar phenomenon to that depicted in Tables 1(a)-1(f), as long as $\varepsilon \ll h$. This is so even though the scheme used is not only A-stable, but also D-stable [Ve] and algebraically stable [BuBu].

3. Analysis. It is important to identify the source of the difficulty demonstrated above. In recent discussions emphasis has been placed on roundoff error accumulation: März [Ma] has observed the merely marginal stability of symmetric schemes that allows for a linear roundoff error accumulation; Ascher and Weiss [AsWe1], [AsWe2],

TABLE 1(d)
$\beta = 50$, $x_1(1) = 0.196+02$, $x_2(1) = 0.192+02$.

h	err 1	err 2	rate 1	rate 2
0.400−02	0.594+20	0.594+20		
0.200−02	0.132+20	0.132+20	0.217+01	0.217+01
0.100−02	0.321+19	0.321+19	0.204+01	0.204+01
0.500−03	0.796+18	0.796+18	0.201+01	0.201+01
0.250−03	0.199+18	0.199+18	0.200+01	0.200+01

TABLE 1(e)
$\beta = 100$, $x_1(1) = 0.380+02$, $x_2(1) = 0.376+02$.

h	err 1	err 2	rate 1	rate 2
0.200−02	0.368+42	0.368+42		
0.100−02	0.721+41	0.721+41	0.235+01	0.235+01
0.500−03	0.170+41	0.170+41	0.209+01	0.209+01
0.250−03	0.418+40	0.418+40	0.202+01	0.202+01
0.125−03	0.104+40	0.104+40	0.201+01	0.201+01
0.625−04	0.260+39	0.260+39	0.200+01	0.200+01
0.313−04	0.650+38	0.650+38	0.200+01	0.200+01
0.156−04	0.162+38	0.162+38	0.200+01	0.200+01
0.781−05	0.406+37	0.406+37	0.200+01	0.200+01
0.391−05	0.102+37	0.102+37	0.200+01	0.200+01

TABLE 1(f)
$\beta = -100$, $x_1(1) = -0.356+02$, $x_2(1) = -0.359+02$.

h	err 1	err 2	rate 1	rate 2
0.200−02	0.283−04	0.916−05		
0.100−02	0.703−05	0.233−05	0.201+01	0.198+01
0.500−03	0.175−05	0.584−06	0.200+01	0.199+01
0.250−03	0.438−06	0.146−06	0.200+01	0.200+01
0.125−03	0.110−06	0.358−07	0.199+01	0.203+01

[AsWe3], [We], [As1] have analyzed related stiff problems and have computed many solutions that did not display any difficulty with roundoff error; and Burrage and Petzold [BuPe] recently also have observed the same lack of pronounced roundoff error effect. Indeed, asymptotically the roundoff error accumulation here is similar in order to that obtained when discretizing directly with a uniform step size a second-order ODE, and the latter rarely (although not never) causes difficulties in practice. Roundoff error accumulation is *not* a cause for concern in the above example either, as the computed rates of convergence clearly indicate (i.e., it is dominated by the discretization error).

The computational difficulty in this example arises because the *stability constant* of the discretization method becomes exponentially large in β. This has been analyzed by Ascher [As2, § 3.1], where it is shown that for a general DAE (1) of index 1 as $h \to 0$ the discretization tends to approximate an auxiliary or "ghost" differential problem. The stability constant of the discretization method therefore depends on the stability (or conditioning—see [AMR, § 3.2]) constant of this ghost problem, which is not necessarily of moderate size.

TABLE 2(a)
$\beta = 0$, $x_1(1) = 0.121+01$, $x_2(1) = 0.841+00$.

h	err 1	err 2	rate 1	rate 2
0.200+00	0.131+00	0.111−15		
0.100+00	0.671−01	0.111−15	0.961+00	
0.500−01	0.340−01	0.111−15	0.979+00	
0.250−01	0.171−01	0.222−15	0.989+00	
0.125−01	0.860−02	0.888−15	0.995+00	

TABLE 2(b)
$\beta = 1$, $x_1(1) = 0.158+01$, $x_2(1) = 0.121+01$.

h	err 1	err 2	rate 1	rate 2
0.200+00	0.135+00	0.673−01		
0.100+00	0.659−01	0.329−01	0.103+01	0.103+01
0.500−01	0.326−01	0.163−01	0.101+01	0.101+01
0.250−01	0.162−01	0.811−02	0.101+01	0.101+01
0.125−01	0.809−02	0.405−02	0.100+01	0.100+01

TABLE 2(c)
$\beta = 10$, $x_1(1) = 0.489+01$, $x_2(1) = 0.452+01$.

h	err 1	err 2	rate 1	rate 2
0.200−01	0.723+00	0.657+00		
0.100−01	0.345+00	0.313+00	0.107+01	0.107+01
0.500−02	0.168+00	0.153+00	0.103+01	0.103+01
0.250−02	0.831−01	0.756−01	0.102+01	0.102+01
0.125−02	0.413−01	0.376−01	0.101+01	0.101+01

In detail, given the DAE (1) and assuming that $E(t)$ can be written as

$$E = S\begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix} T^{-1}, \tag{10}$$

where for each t, $S(t)$, and $T(t)$ are smooth and nonsingular, consider the *semiexplicit form*

$$0 = U^{11}(t)\mathbf{y} + U^{12}(t)\mathbf{z} + \mathbf{g}^1(t), \tag{11a}$$

$$\mathbf{z}' = U^{21}(t)\mathbf{y} + U^{22}(t)\mathbf{z} + \mathbf{g}^2(t) \tag{11b}$$

obtained from (1) through the transformation of variables

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{z} \end{pmatrix} = \hat{\mathbf{x}} = T^{-1}\mathbf{x}, \tag{12}$$

with

$$\begin{pmatrix} U^{11} & U^{12} \\ U^{21} & U^{22} \end{pmatrix} = U = S^{-1}AT - \begin{pmatrix} 0 & 0 \\ 0 & I \end{pmatrix} T^{-1}T', \qquad 0 \leqq t \leqq 1. \tag{13}$$

TABLE 2(d)
$\beta = 50$, $x_1(1) = 0.196+02$, $x_2(1) = 0.192+02$.

h	err 1	err 2	rate 1	rate 2
0.400−02	0.399+01	0.391+01		
0.200−02	0.190+01	0.186+01	0.107+01	0.107+01
0.100−02	0.926+00	0.907+00	0.104+01	0.104+01
0.500−03	0.457+00	0.448+00	0.102+01	0.102+01
0.250−03	0.227+00	0.223+00	0.101+01	0.101+01

TABLE 2(e)
$\beta = 100$, $x_1(1) = 0.380+02$, $x_2(1) = 0.376+02$.

h	err 1	err 2	rate 1	rate 2
0.200−02	0.806+01	0.798+01		
0.100−02	0.383+01	0.379+01	0.107+01	0.107+01
0.500−03	0.187+01	0.185+01	0.104+01	0.104+01
0.250−03	0.923+00	0.914+00	0.102+01	0.102+01
0.125−03	0.459+00	0.454+00	0.101+01	0.101+01
0.625−04	0.229+00	0.226+00	0.100+01	0.100+01
0.313−04	0.114+00	0.113+00	0.100+01	0.100+01
0.156−04	0.570−01	0.564−01	0.100+01	0.100+01
0.781−05	0.285−01	0.282−01	0.100+01	0.100+01
0.391−05	0.142−01	0.141−01	0.100+01	0.100+01

TABLE 2(f)
$\beta = -100$, $x_1(1) = -0.356+02$, $x_2(1) = -0.359+02$.

h	err 1	err 2	rate 1	rate 2
0.200−02	0.673+01	0.679+01		
0.100−02	0.353+01	0.357+01	0.930+00	0.930+00
0.500−03	0.181+01	0.183+01	0.964+00	0.964+00
0.250−03	0.916+00	0.925+00	0.982+00	0.982+00
0.125−03	0.461+00	0.466+00	0.991+00	0.991+00

In (11a) there are n_y equations and in (11b) there are n_z equations, $n_y + n_z = n$. For the example in § 2, a semiexplicit form is (4), with $n = 2$, $n_y = n_z = 1$,

$$S = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad U = \begin{pmatrix} -1 & \beta \\ 0 & -1 \end{pmatrix}, \quad T = \begin{pmatrix} t & 1 \\ 1 & 0 \end{pmatrix}. \tag{14}$$

The general linear DAE (1) has (global) index 1 if $U^{11}(t)$ is nonsingular for all t, as we shall assume. Then

$$\mathbf{y}(t) = -[U^{11}(t)]^{-1}[U^{12}(t)\mathbf{z}(t) + \mathbf{g}^1(t)], \qquad 0 \leqq t \leqq 1. \tag{15}$$

An IVP for (11), or (1), is therefore completely specified by specifying $\mathbf{z}(0)$.

First, consider a midpoint scheme for (11), with $\mathbf{y}_\pi$, $\mathbf{z}_\pi$ denoting the approximation for $\mathbf{y}$, $\mathbf{z}$, respectively. The correct scheme for this simple case is obtained by recognizing that $\mathbf{y}$ is less smooth than $\mathbf{z}$, being defined similarly to $\mathbf{z}'$. Thus we let $\mathbf{y}_\pi$ be a piecewise constant function

$$\mathbf{y}_\pi(t) = \mathbf{y}_{i+1/2}, \qquad t_i \leqq t < t_{i+1}$$

and $\mathbf{z}_\pi$ be a continuous piecewise linear function

$$\mathbf{z}_\pi(t) = \mathbf{z}_i + \frac{\mathbf{z}_{i+1} - \mathbf{z}_i}{h_i}(t - t_i), \qquad t_i \leqq t \leqq t_{i+1}$$

(so $\mathbf{z}'_\pi$ is in the same space as $\mathbf{y}_\pi$), obtaining

$$0 = U^{11}(t_{i+1/2})\mathbf{y}_{i+1/2} + U^{12}(t_{i+1/2})\frac{\mathbf{z}_i + \mathbf{z}_{i+1}}{2} + \mathbf{g}^1(t_{i+1/2}), \tag{16a}$$

$$\frac{\mathbf{z}_{i+1} - \mathbf{z}_i}{h_i} = U^{21}(t_{i+1/2})\mathbf{y}_{i+1/2} + U^{22}(t_{i+1/2})\frac{\mathbf{z}_i + \mathbf{z}_{i+1}}{2} + \mathbf{g}^2(t_{i+1/2}). \tag{16b}$$

The usual collocation theory then applies with a slight twist (see [As2, § 2]).

Unfortunately, in (1) we cannot distinguish between $\mathbf{y}$ and $\mathbf{z}$ without an explicit transformation, so we next consider the midpoint scheme for (11a), (11b) in case that $\mathbf{y}_\pi$ is from the same approximation space as $\mathbf{z}_\pi$,

$$0 = U^{11}(t_{i+1/2})\frac{\mathbf{y}_i + \mathbf{y}_{i+1}}{2} + U^{12}(t_{i+1/2})\frac{\mathbf{z}_i + \mathbf{z}_{i+1}}{2} + \mathbf{g}^1(t_{i+1/2}), \tag{17a}$$

$$\frac{\mathbf{z}_{i+1} - \mathbf{z}_i}{h_i} = U^{21}(t_{i+1/2})\frac{\mathbf{y}_i + \mathbf{y}_{i+1}}{2} + U^{22}(t_{i+1/2})\frac{\mathbf{z}_i + \mathbf{z}_{u+1}}{2} + \mathbf{g}^2(t_{i+1/2}). \tag{17b}$$

In contrast to (16), we now must specify side conditions on $\mathbf{y}_\pi$, in addition to those that are imposed on $\mathbf{z}_\pi$. We assume for now that $\mathbf{y}_1$ is specified such that (15) holds at $t = 0$.

An analysis for (17) was carried out in [We], [AsWe2], and we only mention essential points here. Thus, we can eliminate $(\mathbf{y}_i + \mathbf{y}_{i+1})/2$ from the first equation and substitute into the second, obtaining an ordinary midpoint scheme for $\mathbf{z}$. Stability and second-order convergence for $\mathbf{z}_i$ then follow, as usual for an ODE. To obtain results for $\mathbf{y}_i$ as well, discussion reduces to the initial value problem

$$\mathbf{y}_{i+1} = -\mathbf{y}_i + 2\mathbf{f}(t_{i+1/2}), \quad \mathbf{y}_1 \text{ given}, \tag{18a}$$

as an approximation to the problem

$$\mathbf{y} = \mathbf{f}(t). \tag{18b}$$

The solution of the recursion is

$$\mathbf{y}_{i+1} = (-1)^i\mathbf{y}_1 + 2\sum_{j=1}^{i}(-1)^{i-j}\mathbf{f}(t_{j+1/2}). \tag{18c}$$

In this we see the unfortunate properties of the scheme, namely, that no error is damped. Hence the error is not localized. There is also a linear growth of roundoff error that usually is only of theoretical concern. Still, if $\mathbf{y}_1 = \mathbf{f}(0)$ (corresponding to setting $\mathbf{y}(0)$ explicitly via (15) in terms of $\mathbf{z}(0)$) and $\mathbf{f}$ is smooth then the error is $O(h)$, and it is $O(h^2)$ if the mesh satisfies

$$h_{i+1} = h_i(1 + O(h_i)) \quad \text{for all } i \text{ odd or for all } i \text{ even}. \tag{19}$$

Now we may consider the general case, by observing how the discretization approximates the decoupling transformation. With the midpoint scheme (6), using (10) at $t_{i+1/2}$ and multiplying through by $S^{-1}(t_{i+1/2})$, we obtain for

$$\begin{pmatrix}\mathbf{y}_i\\ \mathbf{z}_i\end{pmatrix} \equiv \hat{\mathbf{x}}_i := T^{-1}(t_i)\mathbf{x}_i$$

the system

$$0 = U^{11}(t_{i+1/2})\frac{\mathbf{y}_i+\mathbf{y}_{i+1}}{2} + U^{12}(t_{i+1/2})\frac{\mathbf{z}_i+\mathbf{z}_{i+1}}{2} + \mathbf{g}^1(t_{i+1/2})$$
$$+\frac{h_i}{4}P_y(UT^{-1}T')(t_{i+1/2})(\hat{\mathbf{x}}_{i+1}-\hat{\mathbf{x}}_i) + O(h_i^2)\hat{\mathbf{x}}_i + O(h_i^2)\hat{\mathbf{x}}_{i+1},$$

(20a)

$$\frac{\mathbf{z}_{i+1}-\mathbf{z}_i}{h_i} = U^{21}(t_{i+1/2})\frac{\mathbf{y}_i+\mathbf{y}_{i+1}}{2} + U^{22}(t_{i+1/2})\frac{\mathbf{z}_i+\mathbf{z}_{i+1}}{2} + \mathbf{g}^2(t_{i+1/2})$$
$$+ O(h_i)\hat{\mathbf{x}}_i + O(h_i)\hat{\mathbf{x}}_{i+1},$$

where

(20b) $$P_y := (I_{n_y} \quad 0) \in \mathbf{R}^{n_y \times n}.$$

Comparing this to (17) and considering the marginal stability implied from (18c), attention reduces to the IVP for

(21) $$0 = \frac{\mathbf{y}_i+\mathbf{y}_{i+1}}{2} + \frac{h_i}{4}M(t_{i+1/2})(\mathbf{y}_{i+1}-\mathbf{y}_i) + \mathbf{f}_{i+1/2}, \qquad 1 \leqq i \leqq N$$

where

(22) $$M := [U^{11}]^{-1}P_yUT^{-1}T'P_y^T = [U^{11}]^{-1}(S^{-1}AT')^{11} \in \mathbf{R}^{n_y \times n_y}$$

(if $U^{12}=0$ then simply $M=(T^{-1}T')^{11}$), and $\mathbf{f}_{i+1/2}$ is an inhomogeneity. For

(23a) $$\hat{\mathbf{y}}_i := (-1)^i\mathbf{y}_i$$

(a trick due to Kreiss [Kr]; see [As1] and references therein), (21) gives

(23b) $$0 = \frac{\hat{\mathbf{y}}_{i+1}-\hat{\mathbf{y}}_i}{h_i} + M(t_{i+1/2})\frac{\hat{\mathbf{y}}_{i+1}+\hat{\mathbf{y}}_i}{2} + (-1)^{i+1}2h_i^{-1}\mathbf{f}_{i+1/2}, \qquad 1 \leqq i \leqq N.$$

The homogeneous part of (23b) is just a midpoint discretization for the ODE

(23c) $$\hat{\mathbf{y}}' = -M(t)\hat{\mathbf{y}}.$$

If the IVP for (23c) has a stability constant K, then there is a constant $\tilde{K}$ of size comparable to K such that for h small enough,

(24) $$\max_i |\mathbf{y}_i| \leqq \tilde{K}\left(|\mathbf{y}_1| + 2\left|\sum_{j=1}^{N}(-1)^j\mathbf{f}_{j+1/2}\right|\right).$$

Subsequently, the results for (17) may be retrieved here, too: In $\mathbf{f}_{j+1/2}$ we have $O(h_j)$ terms in $\mathbf{z}$ that sum up to a bounded quantity; terms such as $F(t_{j+1/2})\,(\mathbf{z}_j+\mathbf{z}_{j+1})/2$ that also sum up to a bounded quantity because of the sign alternation and the smoothness of $F(t)$; and $O(h_j^2)$ terms in $\mathbf{y}$ that are handled by a contraction argument. For a k-stage Gauss collection (the 1-stage scheme is just the midpoint one (6)), we obtain convergence with possible order reduction,

(25) $$\max_i |\mathbf{y}_i - \mathbf{y}(t_i)| \leqq \text{const.}\, h^{k+q}$$

where $q=1$ if k is odd and the mesh satisfies (19), $q=0$ otherwise. *But the constant* const. *in* (25) *depends on* K, *and not just on the stability constant of the original* IVP *for* (1) (!).

4. Example explained and other symmetric schemes. For the Example in § 2 we now consider the ghost IVP (23). A short calculation gives

$$M(t) \equiv -\beta.$$

Equation (23c) here is therefore the scalar, constant coefficient ODE

$$\hat{y}' = \beta \hat{y}, \qquad 0 < t < 1$$

and the stability constant of the IVP for (23) is

$$K = \max(1, e^{\beta}).$$

The *discretization* error with a uniform mesh is $\sim Kh^2$. If $\beta \leqq 0$ then $K = 1$ and good results are obtained. But if, e.g., $\beta = 100$ then we have to reckon with a stability constant of $\approx e^{100}$ for the numerical method. Note that the problem itself is well conditioned (its stability constant grows linearly in $|\beta|$, and the underlying ODE is not even stiff), as can be seen from (4). The poor approximation effect (exponential in β) is caused by the symmetric discretization scheme.

As noted before, a similar phenomenon occurs for very stiff ODEs. Curiously, related effects have been considered more often in the BVP literature (see references in [As1]). In our DAE context, if (23c) is subject to boundary conditions, then in general it may not have a solution at all, in which case (25) does not hold for any constant const. No such danger arises for the linear IVP, and this perhaps has caused unawareness to the stability question hitherto.

As mentioned above, collocation schemes at Gaussian points all have similar stability properties. Other symmetric schemes (or any other Runge-Kutta scheme with a damping factor that is not strictly less than 1) cannot be expected to do better in general. In fact, collocation at Gaussian points is in some sense the most stable among symmetric schemes for very stiff ODEs (see [AsBa]).

In particular, suppose we wish to extend the usual trapezoidal scheme for (1). If we consider the scheme

$$\frac{E(t_{i+1})\mathbf{x}_{i+1} - E(t_i)\mathbf{x}_i}{h_i} = \frac{A(t_i)\mathbf{x}_i + A(t_{i+1})\mathbf{x}_{i+1}}{2} + \frac{\mathbf{q}(t_{i+1}) + \mathbf{q}(t_i)}{2}, \qquad 1 \leqq i \leqq N, \tag{26}$$

then it is clear that the expression on the left-hand side approximates $(E(t)\mathbf{x}(t))'$ instead of $E(t)\mathbf{x}'(t)$, so the approximation is meaningless if $E'(t)\mathbf{x}(t)$ is not very small in magnitude. Writing (1) as

$$(E(t)\mathbf{x})' = (A(t) + E'(t))\mathbf{x} + \mathbf{q}(t), \qquad 0 < t < 1,$$

and applying a discretization like (26) to this form, we obtain a correct trapezoidal scheme, and this can be generalized to higher-order Lobatto schemes. However, terms such as $E'(t_i)\mathbf{x}_i$ need to be further approximated if practical use is contemplated.

One (second-order) possibility gives

$$\frac{1}{2}(E(t_{i+1}) + E(t_i))\frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{h_i} = \frac{A(t_i)\mathbf{x}_i + A(t_{i+1})\mathbf{x}_{i+1}}{2} + \frac{\mathbf{q}(t_{i+1}) + \mathbf{q}(t_i)}{2}, \qquad 1 \leqq i \leqq N. \tag{27a}$$

Another idea could be a hybrid scheme between the trapezoidal and the midpoint

schemes,

$$E(t_{i+1/2})\frac{\mathbf{x}_{i+1}-\mathbf{x}_i}{h_i}=\frac{A(t_i)\mathbf{x}_i+A(t_{i+1})\mathbf{x}_{i+1}}{2}+\frac{\mathbf{q}(t_{i+1})+\mathbf{q}(t_i)}{2}, \qquad 1\leqq i\leqq N. \tag{27b}$$

These are symmetric schemes whose generalization to higher order (such as the Gaussian collocation schemes for midpoint, or the Lobatto collocation schemes for trapezoidal) is less obvious. Yet, they may look attractive at a first glance, because as it turns out they work very well for the Example of § 2. For instance, with $\beta = 100$ and $h = 0.001$, the errors are err 1 = 0.310 − 05, err 2 = 0.307 − 05 with a second-order convergence rate (cf. second rows of Tables 1(e) and 2(e)).

However, this improvement in the computed results is not general: For these schemes, too, the stability constant is controlled by a ghost IVP, that may or may not be stable even when the original IVP is. The ghost ODE is simply a different one than for the midpoint scheme. Instead of (22), (23c), it can be shown that we now get

$$\hat{\mathbf{y}}' = -\tilde{M}(t)\hat{\mathbf{y}}, \tag{28a}$$

with

$$\tilde{M} := -(S'^{-1}AT)^{11}[U^{11}]^{-1}. \tag{28b}$$

(Note that if $(S^{-1}AT)^{21}=0$ then $\tilde{M}=(S^{-1}S')^{11}$.) For the previous example we get $\tilde{M}=0$ because $S'=0$, so no stability problems arise. But other examples can be constructed.

Another example. Consider (1) with

$$E(t)=\begin{pmatrix}0 & \alpha t\\ 0 & 1\end{pmatrix}, \qquad A(t)=\begin{pmatrix}\alpha t+1 & 0\\ 1 & 1\end{pmatrix}. \tag{29a}$$

We can choose

$$T=I, \qquad S=\begin{pmatrix}1 & \alpha t\\ 0 & 1\end{pmatrix}, \tag{29b}$$

(it is easy to show that the considerations given here are independent of the particular choice of S and T satisfying (10)) and this time the midpoint scheme performs well because $T'=0$, so $M=0$ regardless of α. On the other hand, a short calculation gives

$$\tilde{M}=\alpha. \tag{29c}$$

The underlying ODE is

$$x_2'=(\alpha t+1)x_2+q_2, \tag{29d}$$

so for $\alpha \ll -1$ the given IVP is stable, but the ghost IVP for (28a), (29c) is not. Qualitatively similar results to those displayed in Tables 1(a)-1(f) are now obtained for any of the schemes (27). □

5. Using symmetric schemes. In the context of IVPs, not only are symmetric schemes dangerous, but there are also good alternatives such as backward differentiation formulae (BDF). Thus, there appears to be little incentive to use them. Still, if such use is contemplated then the previous discussion suggests how to go about it: Finding $T(0)$ (e.g., using an RQ-factorization), we identify $\mathbf{z}(0)$ that should be specified

as part of the given DAE problem, and using (15) we may then specify $\mathbf{y}(0)$ as well. (This is generally necessary for other one-step schemes, too.) Finding also $T'(0)$ (using numerical differentiation) and the eigenvalues of $M(0)$, we may hope for accurate approximations to be obtained by collocation at Gaussian points using reasonable stepsizes *if* none of these eigenvalues has a large negative real part.

Symmetric schemes become more attractive in the context of BVPs, where a well-conditioned problem may have both fast decreasing and fast increasing modes, so BDF schemes become dangerous to use because they do not preserve the dichotomy—see, e.g., [AMR, Chaps. 3, 10].

The BVP context appears to be more suitable for symmetric schemes also for a different reason. Given that the stability of the scheme is controlled by a ghost problem we can *choose* boundary conditions on $\mathbf{y}$ such that the ghost problem becomes a well-conditioned BVP, provided that the ghost ODE has a dichotomy. For IVPs this means possibly solving a BVP instead, and this again is not competitive. But if the original problem is a BVP then we can provide an efficient, usually stable algorithm using a symmetric scheme for a fully implicit index-1 DAE. We concentrate on the midpoint scheme as an instance.

Thus, consider the DAE (1) subject to consistent BC

$$B_0\mathbf{x}(0)+B_1\mathbf{x}(1)=\beta. \tag{30}$$

ALGORITHM (linear BVPs for index-1 DAEs).

Step 1. Find T and (an approximation for) T' at the two interval ends.

Using T we may now isolate the solution components $\mathbf{y}$ and $\mathbf{z}$, and form U^{11}, U^{12}, $\mathbf{g}^1$ at each interval end.

Step 2. Using the relation (15), obtain from (30) a set of n_z BC on $\mathbf{z}$ alone.

If there are more than n_z BC in (30) then they may be projected appropriately, as described in [As2].

Step 3. Form the matrices $M(0)$ and $M(1)$, and analyze their eigenvalues using the QR algorithm. Let

$$V(t)=Q^T(t)M(t)Q(t) \quad \text{at } t=0,1,$$

where Q is orthogonal and V is upper triangular with the eigenvalues arranged in increasing order of real parts, from large negative to large positive. Let n_0^- (n_0^+) be the number of eigenvalues of $M(0)$ that have a large negative (positive) real part, and similarly define n_1^-, n_1^+ for $M(1)$.

If $n_0^-+n_1^+>n_y$ or $n_0^++n_1^->n_y$ *then exit*: another method (e.g., transforming explicitly to (11) everywhere first) should be used.

Otherwise set $k:=\max(n_0^-,n_1^-)$; set the last n_y-k components of $Q^T\mathbf{y}$ at $t=0$ according to (15), and set the first k components of $Q^T\mathbf{y}$ at $t=1$ according to (15). Transform back to obtain BC on $\mathbf{x}(0)$ and $\mathbf{x}(1)$.

This completes specification of the BC for the application of the midpoint scheme.

Step 4. Solve the discretized equations (6) with the obtained BC.

This algorithm gives a general solution method that often works very well. The utility of Step 3 depends on an assumption that, in case that $M(t)$ has eigenvalues with large real part (the only case where this matters), its variation in t is slow compared to the size of such eigenvalues. Then the eigenvalue analysis approximates Lyapunov's equation well. (The large size eigenvalues approximate the large size kinematic eigenvalues well, and $Q^T\mathbf{y}$ are decoupled variables.) This step includes a check for a

dichotomy of the ghost ODE, with k being the number of increasing modes (that render an IVP for the ghost ODE unstable). Note that except at the endpoints we never find $T(t)$ or form $U(t)$. For the example of § 2, if $\beta \leqq 1$ then $k = 0$ and we have an IVP as before. But with $\beta \gg 1$ we get $k = 1$, so the side condition that Step 3 above dictates is at $t = 1$, and it reads

$$-\beta x_1(1) + (1+\beta)x_2(1) = \sin 1.$$

In addition, the original condition $x_1(0) = 1$ remains unchanged through Step 2. Solving the discretization equations (6) under these boundary conditions is a stable process, and the obtained errors are much smaller than those listed in Tables 2(a)-2(e) for (8).

It should be realized that this algorithm does not always produce moderate stability constants. In fact, a priori there is no guarantee that the ghost ODE has a dichotomy. If it does not, then playing with boundary conditions will not be of much help.

A third example. Consider the IVP (1) with

$$E = \begin{pmatrix} 0 & 0 \\ -1 & \beta t \end{pmatrix}, \quad A = \begin{pmatrix} \frac{1}{2} - t & \beta t(t-\frac{1}{2}) - 1 \\ 1 & -\beta(1+t) \end{pmatrix}, \quad \mathbf{q} = \begin{pmatrix} \sin t \\ 0 \end{pmatrix},$$

β a parameter, and $x_1(0) = -1$.

Here we can let $S = I$,

$$T^{-1} = \begin{pmatrix} t-\frac{1}{2} & 1-\beta t(t-\frac{1}{2}) \\ -1 & \beta t \end{pmatrix}.$$

Then for $\binom{y}{z} = T^{-1}\mathbf{x}$ we have

$$y = \sin t, \qquad z' = -z,$$

i.e., $U = -I$. The initial condition gives $z(0) = 1$, yielding $z = e^{-t}$. This problem has index 1 and solution

$$x_1 = (\beta t(t-\tfrac{1}{2}) - 1)e^{-t} + \beta t \sin t, \qquad x_2 = \sin t + (t-\tfrac{1}{2})e^{-t}.$$

(Note that, while the particular semiexplicit form above is very simple, the conditioning of the problem does grow linearly with $|\beta|$, and also $\|T^{-1}T'\| \sim |\beta|$.)

For the ghost ODE (23c) we obtain

$$M = (T^{-1}T')^{11} = \beta(t-\tfrac{1}{2})$$

so it has no dichotomy. Computing solutions with the midpoint scheme, we get for $|\beta|$ large (≈ 1000) poor results as in Table 1(e), regardless of whether we plant the side condition on y at 0 or at 1.

Note that for $\beta \gg 1$ we have $n_0^- = n_1^+ = 1$, and for $\beta \ll -1$ we have $n_0^+ = n_1^- = 1$. In both cases the check in Step 3 of the algorithm discovers the potential trouble. □

While we expect our algorithm to handle most problems well, it is certainly possible that its Step 3 would not discover a lack of dichotomy in a given problem. In such a case, the algorithm may lead to a computed solution with large errors. Therefore, an error estimate should be computed along with the solution by one of the usual techniques. If the error is estimated to be too large, and the stepsize needed to meet a given tolerance is deemed too small, then we may switch to a different solution method.

REFERENCES

[As1] U. ASCHER, *On some difference schemes for singular singularly perturbed boundary value problems*, Numer. Math., 46 (1986), pp. 1-30.

[As2] ———, *On numerical differential algebraic problems with application to semiconductor device simulation*, SIAM J. Numer. Anal., 26 (1989), pp. 517-538.

[AsBa] U. ASCHER AND G. BADER, *Stability of collocation at Gaussian points*, SIAM J. Numer. Anal., 23 (1986), pp. 412-422.

[AMR] U. ASCHER, R. MATTHEIJ, AND R. D. RUSSELL, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[AsWe1] U. ASCHER AND R. WEISS, *Collocation for singular perturbation problems* I: *First-order systems with constant coefficients*, SIAM J. Numer. Anal., 20 (1983) pp. 537-557.

[AsWe2] ———, *Collocation for singular perturbation problems* II: *Linear first order systems without turning points*, Math. Comp., 43 (1984), pp. 157-187.

[AsWe3] ———, *Collocation for singular perturbation problems* III: *Nonlinear problems without turning points*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 811-829.

[BuBu] K. BURRAGE AND J. C. BUTCHER, *Stability criteria for implicit Runge-Kutta methods*, SIAM J. Numer. Anal., 16 (1979), pp. 46-57.

[BuPe] K. BURRAGE AND L. PETZOLD, *On order reduction for Runge-Kutta methods applied to differential/algebraic systems and to stiff systems of* ODEs, UCRL-98046, Lawrence Livermore Laboratories, Livermore, CA, 1988.

[ClPe] K. CLARK AND L. PETZOLD, *Numerical solution of boundary value problems in differential/algebraic equations*, UCRL-98449, Lawrence Livermore Laboratories, Livermore, CA, 1988.

[Kr] H. O. KREISS, *Centered differences approximation to singular systems of* ODEs, Symposia Mathematica x, Istituto Nazionale di Alta Matematica, 1972.

[Ma] R. MÄRZ, *On difference and shooting methods for boundary value problems in differential-algebraic equations*, Z. Angew. Math. Mech., 64 (1984), pp. 463-473.

[Pe] L. R. PETZOLD, *Order results for implicit Runge-Kutta methods applied to differential/algebraic systems*, SIAM J. Numer. Anal., 23 (1986), pp. 837-852.

[Ve] M. VAN VELDHUISEN, *D-stability*, SIAM J. Numer. Anal., 18 (1981), pp. 45-64.

[We] R. WEISS, *An analysis of the box and trapezoidal schemes for linear singularly perturbed boundary value problems*, Math. Comp., 42 (1984) pp. 41-68.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 950-963, September 1989
© 1989 Society for Industrial and Applied Mathematics
006

ASYMPTOTIC EXPANSIONS OF THE GLOBAL DISCRETIZATION ERROR FOR STIFF PROBLEMS*

W. AUZINGER† AND R. FRANK†

Abstract. The existence of asymptotic expansions of the global discretization error for a general class of nonlinear stiff differential equations

$$y'(t) = A(t)y(t) + \varphi(t, y(t)),$$

where $A(t)$ has a "stiff spectrum" characterized by a small parameter ε and where $\varphi(t, y)$ is smooth, is discussed. The following methods are considered: implicit Euler, implicit midpoint, and trapezoidal rules. In strongly stiff situations (ε significantly smaller than the stepsize h) the implicit Euler scheme admits a full asymptotic expansion; the same is true for the midpoint rule and for the trapezoidal rule under certain coupling conditions. In those strongly stiff cases where a full expansion does not exist for the midpoint or trapezoidal rule, the remainder term is of a reduced order but shows a regular, oscillating behavior that is described in detail. In mildly stiff situations, order reductions of the remainder term inevitably occur in any case after the start or after the change of stepsize but—as can be shown by discrete singular perturbation techniques—these order reductions are rapidly damped out as the integration proceeds. Results are illustrated by various numerical examples; in particular, numerical experience with extrapolation and defect correction is reported.

Key words. stiff systems, asymptotic error expansions

AMS(MOS) subject classification. 65L05

1. Survey. Asymptotic expansions of the global discretization error in powers of the stepsize h are well known for many types of problems and discretization methods (cf., for instance, [8], [12]). If, for instance, the implicit Euler scheme:

$$\frac{1}{h}(\zeta_\nu - \zeta_{\nu-1}) = f(t_\nu, \zeta_\nu), \qquad \zeta_0 = z_0, \tag{1.1}$$

or the implicit midpoint rule:

$$\frac{1}{h}(\zeta_\nu - \zeta_{\nu-1}) = f\left(\frac{1}{2}(t_{\nu-1} + t_\nu), \frac{1}{2}(\zeta_{\nu-1} + \zeta_\nu)\right), \qquad \zeta_0 = z_0, \tag{1.2}$$

or the implicit trapezoidal rule:

$$\frac{1}{h}(\zeta_\nu - \zeta_{\nu-1}) = \frac{1}{2}(f(t_{\nu-1}, \zeta_{\nu-1}) + f(t_\nu, \zeta_\nu)), \qquad \zeta_0 = z_0 \tag{1.3}$$

are applied to an initial value problem of ordinary differential equations,

$$y'(t) = f(t, y(t)), \qquad y(0) = z_0 \tag{1.4}$$

with the exact solution $z(t)$, then—under suitable smoothness assumptions with respect to (1.4)—the global discretization error $\zeta_\nu - z(t_\nu)$ at $t_\nu = \nu h$ admits an expansion in powers of the stepsize h. For the implicit Euler scheme,

$$\zeta_\nu - z(t_\nu) = he_1(t_\nu) + h^2 e_2(t_\nu) + \cdots + h^q e_q(t_\nu) + R_\nu, \tag{1.5}$$

for the midpoint and trapezoidal rules,

$$\zeta_\nu - z(t_\nu) = h^2 e_2(t_\nu) + h^4 e_4(t_\nu) + \cdots + h^{2q} e_{2q}(t_\nu) + R_\nu, \tag{1.6}$$

* Received by the editors July 11, 1988; accepted for publication (in revised form) October 13, 1988.
† Institute for Applied and Numerical Mathematics, Technical University of Vienna, Wiedner Hauptstrasse 6-10, A-1040 Vienna, Austria.

where the functions $e_i(t)$ are solutions of certain linearized initial value problems—the so-called "variational equations." The expansions (1.5), (1.6) are asymptotically correct, i.e., there holds

$$\|R_\nu\| \leqq C(t_\nu; L, \cdots)h^r \tag{1.7}$$

with $r=q+1$ for (1.5) and $r=2q+2$ for (1.6), where $C(\cdots)$ depends (smoothly) on certain parameters that characterize the given problem, namely, on the Lipschitz bound L of the right-hand side f of (1.4), on certain bounds for higher derivatives of f, etc. For nonstiff problems with smooth data functions these problem-characterizing parameters are of moderate size, and therefore (1.5) or (1.6) together with (1.7) describe the structure of the global error in a satisfactory way. The implicit methods (1.1), (1.2) and (1.3), however, are designed to solve (nonsmooth) *stiff* initial value problems with large Lipschitz constants $L \gg 0$, for which the quantity $C(t; L, \cdots)$ in (1.7) is very large; therefore, the estimate (1.7) is worthless in the stiff case.

Recently, asymptotic error expansions for stiff problems have been investigated by several authors (cf. [1], [2], [3], [9], [13]). The crucial question is whether also in the stiff case these expansions are asymptotically correct, i.e., whether there holds

$$\|R_\nu\| \leqq C(t_\nu; m, \cdots)h^r \tag{1.8}$$

with a moderate quantity C depending (not on the Lipschitz constant L but) on the one-sided Lipschitz constant m and on certain other parameters that remain moderate for the stiff problems under consideration. As our analysis in [1]–[3] shows, (1.8) is usually not valid for stiff problems. In these papers we

- Characterize stiff cases where (1.8) turns out to be valid;
- Present a structural analysis of the "remainder term" R_ν for a general class of stiff problems where (1.8) does not hold and R_ν is therefore an essential, nonnegligible part of the global error.

In the present paper we give a survey of our results from [1]–[3].

For the methods under consideration, the variational equations defining the $e_i(t)$ and the respective difference equations defining the remainder term R_ν read as follows (as can be shown by simple Taylor expansions and equating coefficients of powers of h).

Implicit Euler scheme

$$\begin{aligned}
e_1'(t) &= f_y(t, z(t))e_1(t) + \tfrac{1}{2}z''(t) \\
e_2'(t) &= f_y(t, z(t))e_2(t) - \tfrac{1}{6}z'''(t) + \tfrac{1}{2}e_1''(t) + \tfrac{1}{2}f_{yy}(t, z(t))e_1^2(t) \\
e_3'(t) &= f_y(t, z(t))e_3(t) + \tfrac{1}{24}z^{IV}(t) - \tfrac{1}{6}e_1'''(t) + \tfrac{1}{2}e_2''(t) \\
&\quad + f_{yy}(t, z(t))e_1(t)e_2(t) + \tfrac{1}{6}f_{yyy}(t, z(t))e_1^3(t) \\
&\ \ \vdots
\end{aligned} \tag{1.9}$$

$$\frac{1}{h}(R_\nu - R_{\nu-1}) = J_\nu(R_\nu) \cdot R_\nu + b_\nu - c_\nu. \tag{1.10}$$

Here

$$J_\nu(R) := \int_0^1 f_y(t_\nu, z(t_\nu) + he_1(t_\nu) + \cdots + h^q e_q(t_\nu) + \sigma R)\, d\sigma; \tag{1.11}$$

b_ν denotes the collection of all terms of the Taylor expansion of $f(t_\nu, z(t_\nu)+he_1(t_\nu)+\cdots+h^q e_q(t_\nu))$ about $(t_\nu, z(t_\nu))$ that are $O(h^{q+1})$ or of a higher order, i.e.,

$$b_\nu := h^{q+1} f_{yy}(t_\nu, z(t_\nu)) e_1(t_\nu) e_q(t_\nu) + \cdots, \tag{1.12}$$

and

$$c_\nu := -\frac{1}{h} I_{\nu,0} - I_{\nu,1} - \cdots - h^{q-1} I_{\nu,q} \tag{1.13}$$

where the $I_{\nu,j}$ are remainder terms of certain Taylor expansions of the $e_j(t)$:[1]

$$I_{\nu,j} := (-1)^{q+2-j} \frac{h^{q+2-j}}{(q+1-j)!} \int_0^1 e_j^{(q+2-j)}(t_\nu - \sigma h)(1-\sigma)^{q+1-j}\, d\sigma, \qquad j = 0, \cdots, q. \tag{1.14}$$

Implicit midpoint rule

$$\begin{aligned} e_2'(t) &= f_y(t, z(t)) e_2(t) - \tfrac{1}{24} z'''(t) + \tfrac{1}{8} f_y(t, z(t)) z''(t) \\ e_4'(t) &= f_y(t, z(t)) e_4(t) - \tfrac{1}{1920} z^{V}(t) + \tfrac{1}{384} f_y(t, z(t)) z^{IV}(t) - \tfrac{1}{24} e_2'''(t) \\ &\quad + \tfrac{1}{8} f_y(t, z(t)) e_2''(t) + \tfrac{1}{2} f_{yy}(t, z(t))(\tfrac{1}{8} z''(t) + e_2(t))^2 \\ &\vdots \end{aligned} \tag{1.15}$$

$$\frac{1}{h}(R_\nu - R_{\nu-1}) = \hat{J}_\nu\left(\frac{1}{2}(R_{\nu-1}+R_\nu)\right) \cdot \frac{1}{2}(R_{\nu-1}+R_\nu) + f_y(\hat{t}_\nu, z(\hat{t}_\nu)) a_\nu + b_\nu - c_\nu. \tag{1.16}$$

Here, $\hat{t}_\nu := \frac{1}{2}(t_{\nu-1}+t_\nu)$; the nonlinear function $\hat{J}_\nu$ is defined in a way similar to J_ν in (1.11) (i.e., $\hat{J}_\nu(R)$ is a certain integral average of $f_y(\hat{t}_\nu, y)$ depending on R). a_ν and c_ν are defined as

$$\begin{aligned} a_\nu &:= \frac{1}{2}(I^-_{\nu,0} + I^+_{\nu,0}) + \frac{h^2}{2}(I^-_{\nu,2} + I^+_{\nu,2}) + \cdots + \frac{h^{2q}}{2}(I^-_{\nu,2q} + I^+_{\nu,2q}), \\ c_\nu &:= \frac{1}{h}(I^+_{\nu,0} - I^-_{\nu,0}) + h(I^+_{\nu,2} - I^-_{\nu,2}) + \cdots + h^{2q-1}(I^+_{\nu,2q} - I^-_{\nu,2q}) \end{aligned} \tag{1.17}$$

where the $I^\pm_{\nu,j}$ are the remainder terms of certain Taylor expansions of the $e_j(t)$ about $\hat{t}_\nu$ (similar to (1.14)). Also b_ν is defined similarly as for the Euler scheme. (For details, see [2].)

Implicit trapezoidal rule

$$\begin{aligned} e_2'(t) &= f_y(t, z(t)) e_2(t) + \tfrac{1}{12} z'''(t) \\ e_4'(t) &= f_y(t, z(t)) e_4(t) - \tfrac{1}{120} z^{V}(t) + \tfrac{1}{12} e_2'''(t) + \tfrac{1}{2} f_{yy}(t, z(t)) e_2^2(t) \\ &\vdots \end{aligned} \tag{1.18}$$

$$\frac{1}{h}(R_\nu - R_{\nu-1}) = \frac{1}{2}(J_{\nu-1}(R_{\nu-1}) \cdot R_{\nu-1} + J_\nu(R_\nu) \cdot R_\nu) + b_\nu - c_\nu + d_\nu. \tag{1.19}$$

Again, $J_{\nu-1}$ and J_ν denote certain integral averages of the Jacobian f_y. b_ν and c_ν are defined in a similar way as for the midpoint rule; also d_ν is defined as the collection of certain Taylor remainder terms involving derivatives of the $e_i(t)$. (For details, see [2].)

[1] For convenience here we use the denotation $e_0(t) := z(t)$ (the true solution of (1.4)).

The starting values $e_i(0)$ and R_0 for the above equations have to be chosen such that (1.5) or (1.6), respectively, is satisfied for $t=0$. Due to $\zeta_0=z(0)$, the "natural" choice is

$$e_i(0)=0 \quad \text{for all } i, \qquad R_0=0. \tag{1.20}$$

By construction, the inhomogeneities of the difference equations defining R_ν (cf. (1.10), (1.16), (1.19)) contain a factor h^r (where r is the desired order of the remainder term; cf. (1.8)) and depend on certain derivatives of $z(t)$ and of the $e_i(t)$. For nonstiff problems with smooth data functions, all occurring deriatives $z^{(l)}(t)$, $e_i^{(l)}(t)$ are of moderate size and therefore a conventional stability estimate, based on the (conventional) Lipschitz constant L for f, leads immediately to the desired estimate (1.7).

In the stiff case, where $L \gg 0$, B-stability[2] (based on the one-sided Lipschitz constant m for f) might be used to estimate the remainder term R_ν. For the Euler scheme, for instance, R_ν can be estimated by

$$\|R_\nu\| \leqq \begin{cases} \|R_0\| + t_\nu \max\limits_{\mu=1(1)\nu} \|b_\mu - c_\mu\|, & m=0, \\ \left(\dfrac{1}{1-hm}\right)^\nu \|R_0\| + \left(\left(\dfrac{1}{1-hm}\right)^\nu - 1\right)\Big/ m \max\limits_{\mu=1(1)\nu} \|b_\mu - c_\mu\|, & \end{cases} \tag{1.21}$$

$$m \neq 0, \quad hm \leqq h_0 m < 1.$$

The question is whether such a B-estimate is sufficient to show (1.8). The essential difficulty arising here is the following: The variational equations are of course stiff, and it must therefore be expected that the choice (1.20) for the starting values $e_i(0)$ does not define smooth solutions $e_i(t)$. As a consequence, the inhomogeneities of the remainder equations (1.10), (1.16), (1.19) (depending on derivatives of the $e_i(t)$) must be expected to be *large* despite the factor h^r, and therefore the desired estimate (1.8) cannot be concluded. To overcome this difficulty it is natural to work with *smooth* solutions $e_i(t)$ of the variational equations,[3] such that the inhomogeneity of the respective remainder equation is at the desired $O(h^r)$-level. (This idea goes back to Dahlquist and Lindberg [5].) But the requirement that the $e_i(t)$ be smooth usually contradicts the choice (1.20), i.e., the $e_i(0)$ cannot be chosen as zero. But then,

$$R_0 = -\sum_i h^i e_i(0) + \zeta_0 - z(0) = -\sum_i h^i e_i(0) \tag{1.22}$$

is only $O(h)$ (Euler) or $O(h^2)$ (midpoint and trapezoidal rules). So again a B-estimate like (1.21) does not help. A more sophisticated analysis is necessary to understand the structure of the global error in the stiff case.

The class of stiff problems considered in the following is

$$y'(t) = A(t)y(t) + \varphi(t, y(t)) \tag{1.23}$$

where

$$A(t) = T(t)\Lambda(t)T^{-1}(t), \qquad \Lambda(t) = \begin{pmatrix} c_1(t) & 0 \\ 0 & -c_2(t)/\varepsilon \end{pmatrix} \tag{1.24}$$

with smooth functions $T(t)$, $T^{-1}(t)$, $c_1(t)$ and $c_2(t)$. We assume

$$\operatorname{Re}(c_2(t)) \geqq \kappa > 0 \tag{1.25}$$

---

[2] The trapezoidal rule is not B-stable in the general sense; nevertheless, estimates for R_ν can be derived under certain assumptions (cf. [2]).

[3] In a transient phase, where the true solution $z(t)$ of the original problem (1.4) is not smooth, the variational equations do not admit smooth solutions $e_i(t)$. Therefore a different argumentation is necessary to understand the effects of a transient phase. The reader interested in this point is referred to [1]-[3].

where κ is significantly away from zero independently of ε. $\varepsilon > 0$ is a small positive parameter characterizing the stiffness of the problem. We are considering the two-dimensional case, i.e., $c_1(t)$ and $c_2(t)$ are scalar functions; but this is not a crucial restriction. All the material could easily be rewritten for problems, which are more general in the sense that $c_1(t)$ and $c_2(t)$ are matrix functions of certain dimensions. The function $\varphi(t, y)$ is allowed to be $O(1/\varepsilon)$ (note, however, that some of our results are only valid for $\varphi(t, y) = O(\varepsilon^0)$). Furthermore, we assume that all occurring derivatives $\varphi_y, \varphi_{yy}, \cdots$ exist and are of moderate size (no assumptions about the t-derivatives $\varphi_t, \varphi_{tt}, \varphi_{ty}, \cdots$ are required).

Our theory is based on the following:

- A singular perturbation analysis (cf. [10], [11]) of the variational equations and discrete singular perturbation techniques for the analysis of the (discrete) remainder term R_ν.
- Inductive arguments with respect to several subintervals (with constant stepsizes) of the whole integration interval to cover arbitrary, nonequidistant grids.

As already mentioned, "full" asymptotically correct expansions usually do not exist in the stiff case. Our results concerning the structure of the global error are in the following sense weaker:

- The solutions $e_i(t)$ of the variational equations, which describe the smooth terms of the global error, are not always h-independent but for t fixed the value $e_i(t) = e_i(t; h)$ is a function in h (or h^2). This is of course unusual but practically sufficient if the degree of smoothness of this dependence on h (or h^2) is well understood. This can be quantitatively studied on the basis of our results.
- The "remainder term" R_ν contains nonsmooth terms that are not always at the desired $O(h^r)$-level. We describe the behavior of these nonsmooth terms by discrete singular perturbation techniques. In some cases these terms are rapidly damped out as the integration proceeds; in other cases they are oscillating with a smoothly varying amplitude (cf. § 2 for details).

In some situations the nonsmooth terms appearing in R_ν contain the additional factor ε. If ε is sufficiently small compared to h ("strongly stiff case") the order of magnitude of these terms is therefore actually $O(h^r)$, so the expansion is asymptotically correct in this case (see § 2, Theorems 1, 4, 8, and 9). If ε is not small enough compared to h—"mildly stiff case"—this additional factor ε is not sufficient to ensure $R_\nu = O(h^r)$. Therefore, in mildly stiff situations, R_ν is always a dominant, nonnegligible part of the global error. But, typically, R_ν is damped out as the integration proceeds and is therefore only dominant after the start or after a change of stepsize.

2. Results. In this section we present a menagerie of our results from [1]-[3] (without proofs). We note that all these results can be applied inductively (induction with respect to several subintervals with constant stepsizes h) and are therefore valid for nonequidistant grids.[4] The original formulation of the various theorems in [1]-[3] involves inductive assumptions concerning the starting values $e_i(0)$ and R_0 at the beginning of such a subinterval. In the following survey we do not state these inductive assumptions explicitly; the interested reader is referred to the original papers.

All theorems are valid for the class of stiff problems (1.23)-(1.25) with a smooth solution $z(t)$. Unless otherwise stated, $\varphi(t, y) = O(1/\varepsilon)$ is admitted. Throughout we tacitly assume that a certain number of derivatives $\varphi_y, \varphi_{yy}, \cdots$ (namely, those which play a role in the proof of the respective theorems) are of moderate size. Some of the results depend on the *type of coupling* between the stiff and the nonstiff components,

[4] Also the effects of a transient phase are discussed in [1]-[3].

which can be characterized by the off-diagonal elements of the matrix

$$-T^{-1}(t)T'(t) =: D(t) = \begin{pmatrix} d_{11}(t) & d_{12}(t) \\ d_{21}(t) & d_{22}(t) \end{pmatrix}. \tag{2.1}$$

The coupling from the stiff to the nonstiff component is called *weak* if $d_{12}(t) = O(\varepsilon)$. Conversely, the coupling from the nonstiff to the stiff component is called weak if $d_{21}(t) = O(\varepsilon)$. (See [2], [14] for a more detailed discussion of coupling properties.)

Implicit Euler scheme

THEOREM 1 (The Strongly Stiff Case). *Assume that*

$$\varepsilon \leqq Ch^q \tag{2.2}$$

with some moderate constant C. Then the global discretization error of the implicit Euler scheme admits a full asymptotic expansion

$$\zeta_\nu - z(t_\nu) = he_1(t_\nu) + \cdots + h^q e_q(t_\nu) + R_\nu \tag{2.3}$$

with smooth, h-independent functions $e_i(t)$ that are solutions of the variational equations (1.9). *Furthermore,*

$$R_\nu = O(h^{q+1}) \tag{2.4}$$

at all gridpoints $t_\nu = \nu h$.

THEOREM 2 (The Mildly Stiff Case, (i)). *Assume that*

$$\varepsilon \leqq Ch \tag{2.5}$$

with some moderate constant C. Then the global discretization error of the implicit Euler scheme admits an asymptotic expansion

$$\zeta_\nu - z(t_\nu) = he_1(t_\nu; \chi) + \cdots + h^q e_q(t_\nu; \chi) + R_\nu \tag{2.6}$$

with smooth functions $e_i(t; \chi)$ that are solutions of the variational equations (1.9) *and that depend smoothly on the parameter*

$$\chi := \frac{\varepsilon}{h}. \tag{2.7}$$

Furthermore,

$$R_\nu = T(t_\nu)\begin{pmatrix} h^3\xi_\nu^{(3)} + \cdots + h^q\xi_\nu^{(q)} \\ h^2\eta_\nu^{(2)} + h^3\eta_\nu^{(3)} + \cdots + h^q\eta_\nu^{(q)} \end{pmatrix} + O(h^{q+1}). \tag{2.8}$$

The terms $\xi_\nu^{(l)}$ and $\eta_\nu^{(l)}$ are rapidly decaying and bounded by

$$\chi C(\chi) p(\nu) \left(1 + \frac{\kappa}{\chi}\right)^{-\nu}. \tag{2.9}$$

Here $C(\chi)$ depends smoothly on χ, and $p(\nu)$ is some polynomial of low degree in ν.

Remark. In certain cases it can be shown that the $e_i(t)$ are actually independent of χ, i.e., independent of h. This is the case for $q \leqq 4$, and under the assumption that each of the subintervals with constant stepsize is sufficiently long such that at the endpoint of such a subinterval R_ν is damped out to the $O(h^{q+1})$-level. (Concerning the induction with respect to several subintervals, see the remark at the beginning of this section.)

THEOREM 3 (The Mildly Stiff Case, (ii)). *Assume that*

$$h \leqq C\varepsilon \tag{2.10}$$

with some moderate constant C. Then the global discretization error of the implicit Euler scheme admits an asymptotic expansion

$$\zeta_\nu - z(t_\nu) = he_1(t_\nu; \chi) + \cdots + h^q e_q(t_\nu; \chi) + R_\nu \tag{2.11}$$

with smooth functions $e_i(t; \chi)$ *that are solutions of the variational equations* (1.9) *and that depend smoothly on the parameter*

$$\chi := \frac{h}{\varepsilon}. \tag{2.12}$$

Furthermore,

$$R_\nu = T(t_\nu)\begin{pmatrix} \varepsilon^3 \xi_\nu^{(3)} + \cdots + \varepsilon^q \xi_\nu^{(q)} + x_\nu^{(rem)} \\ \varepsilon^2 \eta_\nu^{(2)} + \varepsilon^3 \eta_\nu^{(3)} + \cdots + \varepsilon^q \eta_\nu^{(q)} + y_\nu^{(rem)} \end{pmatrix}. \tag{2.13}$$

The terms $\xi_\nu^{(l)}$ *and* $\eta_\nu^{(l)}$ *are rapidly decaying and bounded by*

$$\chi C(\chi) p\left(\frac{t_\nu}{\varepsilon}\right)(1+\kappa\chi)^{-\nu}. \tag{2.14}$$

Here $C(\chi)$ *depends smoothly on* χ, *and* $p(t_\nu/\varepsilon)$ *is some polynomial of low degree in* t_ν/ε. *Moreover,* $x_\nu^{(rem)}$ *and* $y_\nu^{(rem)}$ *are bounded by*

$$\varepsilon^{q+1} C(\chi) p\left(\frac{t_\nu}{\varepsilon}\right)(1+\kappa\chi)^{-\nu} + O(h^{q+1}). \tag{2.15}$$

By Theorems 1-3 we have covered the whole "ε-h-plane." At first sight we might think that $h \leqq C\varepsilon$ (in particular, $h \to 0$) is a "classical situation" that can be treated by conventional arguments. This is, however, not the case: As can be seen from (2.15), the remainder term R_ν does not satisfy (not even for arbitrarily small h) an estimate $\|R_\nu\| \leqq Ch^{q+1}$ with a *moderate* constant C. The "initial layer terms" in (2.15) are always present and are not $O(h^{q+1})$. However, these terms decay very rapidly; for very small h, $(1+\kappa\chi)^{-\nu}$ behaves like $e^{-t_\nu/\varepsilon}$. (See also the discussion in [2], Remark 2.1.)

Related results have been obtained by Hairer and Lubich in [9] for a slightly different class of stiff problems.

Implicit midpoint rule

THEOREM 4 (The Strongly Stiff Case). *Assume that*

$$\varepsilon \leqq Ch^{2q} \tag{2.16}$$

with some moderate constant C. Assume further that

$$d_{21}(t) = O(\varepsilon) \tag{2.17}$$

(*weak coupling from the nonstiff to the stiff component*) *and*

$$\varphi(t, y) = O(\varepsilon^0). \tag{2.18}$$

Then the global discretization error of the implicit midpoint rule admits a full asymptotic expansion

$$\zeta_\nu - z(t_\nu) = h^2 e_2(t_\nu) + h^4 e_4(t_\nu) + \cdots + h^{2q} e_{2q}(t_\nu) + R_\nu \tag{2.19}$$

with smooth, h-independent functions $e_i(t)$ *that are solutions of the variational equations* (1.15). *Furthermore,*

$$R_\nu = O(h^{2q+2}) \tag{2.20}$$

at all gridpoints $t_\nu = \nu h$.

THEOREM 5. *Assume that*

$$\varepsilon \leqq Ch^2 \tag{2.21}$$

with some moderate constant C. Then the global discretization error of the implicit midpoint rule admits an asymptotic expansion

$$\zeta_\nu - z(t_\nu) = h^2 e_2(t_\nu) + h^4 e_4(t_\nu; \chi) + R_\nu \tag{2.22}$$

where the h-independent function $e_2(t)$ and the function $e_4(t; \chi)$ that depends smoothly on the parameter

$$\chi := \frac{\varepsilon}{h^2}, \tag{2.23}$$

are smooth solutions of the variational equations (1.15). *Furthermore,*

$$R_\nu = T(t_\nu)\begin{pmatrix} h^4\alpha_4(t_\nu; \chi)(-1)^\nu \\ h^2\beta_2(t_\nu; \chi)(-1)^\nu + h^4\beta_4(t_\nu; \chi)(-1)^\nu \end{pmatrix} + O(h^6). \tag{2.24}$$

$\alpha_4(t; \chi)$, $\beta_2(t; \chi)$, and $\beta_4(t; \chi)$ are smooth functions in t that depend smoothly on the parameter χ. $\beta_2(t; \chi)$ is a solution of

$$\beta_2'(t; \chi) = \left(d_{22}(t) - \frac{4\chi}{c_2(t)}\right)\beta_2(t; \chi). \tag{2.25}$$

The equations defining $\alpha_4(t; \chi)$ and $\beta_4(t; \chi)$ are given in the Proof of Theorem 3.1 [3].

Theorem 5 applies to strongly stiff situations in those cases where the assumptions (2.17), (2.18) of Theorem 4 are not satisfied, showing that R_ν is oscillating with a smoothly varying amplitude. But the assumption $\varepsilon \leqq Ch^2$ is rather weak and therefore also certain mildly stiff situations are covered. In the nonlinear case and for expansions longer than (2.22), R_ν also contains smooth, nonoscillating terms (cf. [3]).

THEOREM 6. *Assume that*

$$\varepsilon \leqq Ch \tag{2.26}$$

with some moderate constant C. Then the global discretization error of the implicit midpoint rule admits an asymptotic expansion

$$\zeta_\nu - z(t_\nu) = h^2 e_2(t_\nu; \chi^2) + R_\nu; \tag{2.27}$$

here the function $e_2(t; \chi^2)$ that depends smoothly on the parameter χ^2 where

$$\chi := \frac{\varepsilon}{h} \tag{2.28}$$

is a smooth solution of the first variational equation in (1.15). *Furthermore,*

$$R_\nu = T(t_\nu)\begin{pmatrix} h^2\varepsilon\check{\alpha}_2(t_\nu; \chi^2)\Pi_\nu \\ h^2[\beta_2(t_\nu; \chi^2) + \varepsilon\check{\beta}_2(t_\nu; \chi^2)]\Pi_\nu \end{pmatrix} + O(h^4). \tag{2.29}$$

Here

$$\Pi_\nu := \prod_{l=1}^{\nu} \left(\frac{1 - c_2(\hat{t}_l)h/2\varepsilon}{1 + c_2(\hat{t}_l)h/2\varepsilon}\right) \tag{2.30}$$

and $\check{\alpha}_2(t; \chi^2)$, $\beta_2(t; \chi^2)$, and $\check{\beta}_2(t; \chi^2)$ are smooth functions in t that depend smoothly on the parameter χ^2. $\beta_2(t; \chi^2)$ is a solution of

$$\beta_2'(t; \chi^2) = d_{22}(t)\beta_2(t; \chi^2). \tag{2.31}$$

The equations defining $\check{\alpha}_2(t; \chi^2)$ and $\check{\beta}_2(t; \chi^2)$ are given in the Proof of Theorem 4.1 [3].

Theorem 6 shows that the behavior of the nonsmooth terms of the global error is characterized by the factor Π_ν. If even $\varepsilon \leqq Ch^2$ holds, Π_ν is oscillating with a smoothly varying amplitude (this conforms with the assertion of Theorem 5). For $Ch^2 \leqq \varepsilon \leqq (\kappa/2)h$, Π_ν is still oscillating but with a nonsmooth, significantly *decreasing* amplitude. This damping property becomes stronger as ε/h increases. We note further that for fixed $t = t_\nu = \nu h$, Π_ν is also "even," i.e., Π_ν can be expanded in powers of h^2 and the coefficients of this expansion depend smoothly on χ^2 (cf. [3, § 4].).

In the case of longer expansions than covered by Theorem 6 not only terms with the factor Π_ν appear but also higher powers Π_ν^k, $k > 1$ (due to nonlinear effects).

In the case $h \leqq C\varepsilon$ the error structure of the implicit midpoint rule can be described in a similar way as in Theorem 3 for the Euler scheme (i.e., R_ν is rapidly damped out to the $O(h^{2q+2})$-level).

Implicit trapezoidal rule

THEOREM 7 (The Strongly Stiff Case, $q = 2$). *Assume that*

$$\varepsilon \leqq Ch^2 \tag{2.32}$$

with some moderate constant C. Then the global discretization error of the implicit trapezoidal rule admits a full asymptotic expansion

$$\zeta_\nu - z(t_\nu) = h^2 e_2(t_\nu) + R_\nu \tag{2.33}$$

with a smooth, h-independent function $e_2(t)$ that is a solution of the first variational equation in (1.18). *Furthermore,*

$$R_\nu = O(h^4) \tag{2.34}$$

at all gridpoints $t_\nu = \nu h$.

THEOREM 8 (The Strongly Stiff Case, q arbitrary, (i)). *Assume that*

$$\varepsilon \leqq Ch^{2q-2} \tag{2.35}$$

with some moderate constant C. Assume further that

$$d_{21}(t) = O(\varepsilon) \tag{2.36}$$

(*weak coupling from the nonstiff to the stiff component*) *and*

$$\varphi(t, y) = O(\varepsilon^0). \tag{2.37}$$

Then the global discretization error of the implicit trapezoidal rule admits a full asymptotic expansion

$$\zeta_\nu - z(t_\nu) = h^2 e_2(t_\nu) + h^4 e_4(t_\nu) + \cdots + h^{2q} e_{2q}(t_\nu) + R_\nu \tag{2.38}$$

with smooth, h-independent functions $e_i(t)$ that are solutions of the variational equations (1.18). *Furthermore,*

$$R_\nu = O(h^{2q+2}) \tag{2.39}$$

at all gridpoints $t_\nu = \nu h$.

THEOREM 9 (The Strongly Stiff Case, q arbitrary, (ii)). *Assume that*

$$\varepsilon \leqq Ch^{2q} \tag{2.40}$$

with some moderate constant C. Assume further that

$$d_{12}(t) = O(\varepsilon) \tag{2.41}$$

(weak coupling from the stiff to the nonstiff component). Then the global discretization error of the implicit trapezoidal rule admits a full asymptotic expansion

$$\zeta_\nu - z(t_\nu) = h^2 e_2(t_\nu) + h^4 e_4(t_\nu) + \cdots + h^{2q} e_{2q}(t_\nu) + R_\nu \tag{2.42}$$

with smooth, h-independent functions $e_i(t)$ that are solutions of the variational equations (1.18). *Furthermore,*

$$R_\nu = O(h^{2q+2}) \tag{2.43}$$

at all gridpoints $t_\nu = \nu h$.

Theorems 7-9 show that asymptotically correct expansions exist under weaker assumptions than for the midpoint rule. In particular, Theorem 7 (h^4-expansion for $\varepsilon \leqq Ch^2$) is valid for "general" $\varphi(t, y) = O(1/\varepsilon)$ and irrespective of any coupling condition; also in Theorem 9, $\varphi(t, y) = O(1/\varepsilon)$ is admitted. So we might draw the conclusion that the trapezoidal rule is more "robust." However, the practical applicability of the trapezoidal rule is limited to some extent: It is well known that the trapezoidal rule is (A-stable but) not B-stable. (Only for problems with a weak coupling from the stiff to the nonstiff components do we have nonlinear stability; cf. Theorem 3.4 of [2].) This lack of stability entails an increased sensitivity with respect to roundoff errors. (See also Dahlquist and Lindberg [5].)

The structure of the global error of the trapezoidal rule in the general case (not covered by Theorems 7-9) can be described in a similar way as in Theorems 5 and 6 above for the midpoint rule. (Note, however, that due to Theorem 7 the nonsmooth terms are not as dominant as for the midpoint rule.)

3. Applications. A look at Theorems 1-9 shows that in some stiff situations "pure" asymptotic expansions exist with a remainder term R_ν at the desired $O(h^r)$-level, whereas, in general the structure of the global error is less regular: In R_ν there occur nonsmooth (decaying and/or oscillating) terms in a dominant way; moreover, the functions $e_i(t)$ are usually not independent of h. However, these results about the structure of the global error are useful for various applications.

In those cases where the global error admits an asymptotically correct expansion with smooth, h-independent functions $e_i(t)$, this immediately serves as the theoretical justification of acceleration techniques such as extrapolation or defect correction. The existence of such an expansion implies that for fixed t the global error behaves essentially as does a polynomial in h (or h^2). But to ensure that extrapolation works successfully this is not necessary; any smooth function in h (or h^2) will do as well. For the midpoint rule, for instance (cf. Theorem 5), the dependence of the e_i and of the α_i and β_i (describing the amplitude of the oscillating remainder term) is nonpolynomial in h^2 (via the parameter $\chi := \varepsilon/h^2$). Theorem 5, which describes this dependence explicitly, provides the information concerning to what extent the global error is a *smooth* function in h^2 (with moderate derivatives with respect to h^2). This enables us to judge the efficiency of extrapolation for the midpoint rule. For defect correction based on the midpoint rule and on polynomials interpolating at even gridpoints, the dependence on h^2 is irrelevant because there is only one fixed grid involved. Here the degree of smoothness with respect to t is essential, in particular that of the "amplitude functions" α_i and β_i. In a similar way, acceleration techniques based on one of the other methods can be studied on the basis of our results.

A detailed discussion of all these questions and various numerical examples, including extrapolation and defect correction, can be found in [4] (cf. also [6], [7]). (Extrapolation for stiff problems is also discussed in [9].) In the following we present

some of our numerical results[5] for h^2-extrapolation applied to a linear, nonautonomous test problem of the type (1.23)-(1.25) with $A(t) = T(t)\Lambda(t)T^{-1}(t)$,

$$\Lambda(t) = \begin{pmatrix} e^{-t/10} & 0 \\ 0 & -e^{-t/5}/\varepsilon \end{pmatrix}, \qquad T(t) = \begin{pmatrix} e^{t/10} & 1+\varepsilon t \\ \sin t & 2+\varepsilon t \end{pmatrix} \tag{3.1}$$

and

$$\varphi(t, y) \equiv \varphi(t) = g'(t) - A(t)g(t), \qquad g(t) = \begin{pmatrix} e^{t/10}\, e^{-t} \\ (\sin t + \varepsilon \cos t)\, e^{-t} \end{pmatrix}. \tag{3.2}$$

For the initial value $z_0 = g(0)$ the solution is $z(t) \equiv g(t)$. Note that $g(t)$ is chosen such that $\varphi(t) = O(1)$ (but not $O(1/\varepsilon)$). A simple check shows that for this problem the coupling from the stiff to the nonstiff component is weak, i.e., $d_{12}(t) = O(\varepsilon)$. Therefore the trapezoidal rule behaves stable for this example (cf. the remark at the end of § 2). In [4] we shall also present examples where $d_{12}(t) = O(\varepsilon)$ does not hold, to illustrate the influence of roundoff error in a situation where the trapezoidal rule is not stable.

Tables 1(a), 1(b) and 2(a), 2(b) display the L_2-norm of the global error of the extrapolated trapezoidal rule at $t = 1$ for $\varepsilon = 10^{-10}$ and 10^{-5}, respectively. Table 1 illustrates a typical strongly stiff situation; due to $d_{12}(t) = O(\varepsilon)$ the existence of an asymptotically correct expansion of the required order follows from Theorem 9. Indeed, all extrapolation steps shown in Tables 1(a) and 1(b) perform successfully.

The situation shown in Tables 2(a) and 2(b) may be called mildly stiff. More precisely, condition (2.40) is violated for $\varepsilon = 10^{-5}$, the stepsizes from Tables 2(a) and 2(b) and for $q \geqq 2$. Thus Theorem 9 can only be applied for $q = 1$, which guarantees the optimal performance of one extrapolation step.[6] To judge the efficiency of further

TABLE 1(a)
Implicit trapezoidal rule, $\varepsilon = 10^{-10}$.

Global error	ITR	1st EX	2nd EX	3rd EX	4th EX
$h = \frac{1}{4}$	2.38E−02				
$h = \frac{1}{8}$	5.95E−03	1.60E−05			
$h = \frac{1}{16}$	1.49E−03	1.00E−06	1.71E−09		
$h = \frac{1}{32}$	3.71E−04	6.26E−08	2.84E−11	1.63E−12	
$h = \frac{1}{64}$	9.29E−05	3.92E−09	4.49E−13	6.31E−15	6.42E−17

TABLE 1(b)

Observed order	ITR	1st EX	2nd EX	3rd EX
$h = \frac{1}{4}$				
$h = \frac{1}{8}$	2.00			
$h = \frac{1}{16}$	2.00	4.00		
$h = \frac{1}{32}$	2.00	4.00	5.92	
$h = \frac{1}{64}$	2.00	4.00	5.98	8.01

[5] The computations were performed on a CDC Cyber 180 in double precision (mantissa length ≈ 29 decimal digits); roundoff error is not visible.

[6] The full theoretical justification of one extrapolation step for $\varepsilon \leqq Ch^2$ is also given by Theorem 7—not only for the present example but for any problem of the type (1.23)-(1.25). (Theorem 7 requires no coupling conditions; $\varphi(t, y) = O(1/\varepsilon)$ is admitted.)

TABLE 2(a)
Implicit trapezoidal rule, $\varepsilon = 10^{-5}$.

Global error	ITR	1st EX	2nd EX	3rd EX	4th EX
$h=\frac{1}{4}$	2.38E−02				
$h=\frac{1}{8}$	5.95E−03	1.60E−05			
$h=\frac{1}{16}$	1.49E−03	1.00E−06	1.80E−09		
$h=\frac{1}{32}$	3.71E−04	6.27E−08	1.24E−10	9.93E−11	
$h=\frac{1}{64}$	9.29E−05	4.00E−09	8.99E−11	8.94E−11	8.94E−11

TABLE 2(b)

Observed order	ITR	1st EX	2nd EX	3rd EX
$h=\frac{1}{4}$				
$h=\frac{1}{8}$	2.00			
$h=\frac{1}{16}$	2.00	4.00		
$h=\frac{1}{32}$	2.00	4.00	3.86	
$h=\frac{1}{64}$	2.00	3.97	0.46	0.15

extrapolation steps, Theorem 9 is too weak but the error structure of the trapezoidal rule in the mildly stiff case has to be understood. (Such an analysis can be done in a way very similar to the midpoint rule in Theorems 5 and 6.)

Tables 3(a), 3(b) and 4(a), 4(b) display the L_2-norm of the global error of the extrapolated midpoint rule at $t=1$ for $\varepsilon = 10^{-10}$ and 10^{-5}, respectively. Tables 3(a) and 3(b) illustrate a strongly stiff situation. But Theorem 4 does not apply here because condition (2.17) is violated. However, the first (and to some extent also the second)

TABLE 3(a)
Implicit midpoint rule, $\varepsilon = 10^{-10}$.

Global error	IMR	1st EX	2nd EX	3rd EX	4th EX
$h=\frac{1}{4}$	2.14E−02				
$h=\frac{1}{8}$	5.53E−03	2.30E−04			
$h=\frac{1}{16}$	1.39E−03	1.45E−05	1.45E−07		
$h=\frac{1}{32}$	3.48E−04	9.10E−07	3.41E−09	1.17E−09	
$h=\frac{1}{64}$	8.72E−05	5.68E−08	1.56E−10	2.06E−10	2.11E−10

TABLE 3(b)

Observed order	IMR	1st EX	2nd EX	3rd EX
$h=\frac{1}{4}$				
$h=\frac{1}{8}$	1.95			
$h=\frac{1}{16}$	1.99	3.99		
$h=\frac{1}{32}$	2.00	4.00	5.41	
$h=\frac{1}{64}$	2.00	4.00	4.45	2.50

TABLE 4(a)
Implicit midpoint rule, $\varepsilon = 10^{-5}$.

Global error	IMR	1st EX	2nd EX	3rd EX	4th EX
$h=\frac{1}{4}$	3.50E−02				
$h=\frac{1}{8}$	8.74E−03	1.06E−05			
$h=\frac{1}{16}$	2.19E−03	1.07E−05	1.19E−05		
$h=\frac{1}{32}$	5.56E−04	1.19E−05	1.20E−05	1.20E−05	
$h=\frac{1}{64}$	1.46E−04	1.11E−05	1.10E−05	1.10E−05	1.10E−05

TABLE 4(b)

Observed order	IMR	1st EX	2nd EX	3rd EX
$h=\frac{1}{4}$				
$h=\frac{1}{8}$	2.00			
$h=\frac{1}{16}$	2.00	−0.02		
$h=\frac{1}{32}$	1.98	−0.15	−0.02	
$h=\frac{1}{64}$	1.93	0.11	0.13	0.13

extrapolation step performs successfully. This can be studied theoretically on the basis of Theorem 5; the crucial point is the degree of smoothness of the global error (represented by (2.22)-(2.24)) with respect to h^2 for fixed t, which depends on the actual ratio of ε and h^2 (which is very small in Table 3). (For a more detailed discussion we refer the reader to [4].)

Tables 4(a) and 4(b) display a mildly stiff situation; the ratio ε/h^2 is small but not very small. A study of the smoothness of the global error with respect to h^2 on the basis of Theorem 5 shows that even the first extrapolation step cannot work successfully.

In mildly stiff situations an asymptotically correct expansion of the global error usually does not exist because of the presence of nonsmooth terms that are not at the "remainder term level" $O(h^r)$. However, it is typical that these nonsmooth terms are rapidly damped out as the integration proceeds (cf. Theorems 2, 3, 6). A successful implementation of extrapolation should take account of these damping properties: The usual "local connection strategy" (where the integration is continued starting from the most accurate extrapolated value) should be replaced by a "global connection strategy," where the basic approximations at the different grid levels are continued on

TABLE 5(a)
Implicit midpoint rule, $\varepsilon = 10^{-3}$, *error at* $t = 1$.

Local connection	IMR	1st EX	2nd EX	3rd EX	4th EX
$h=\frac{1}{100}$	9.43E−06				
$h=\frac{1}{200}$	3.69E−06	1.78E−06			
$h=\frac{1}{400}$	9.36E−07	1.76E−08	1.00E−07		
$h=\frac{1}{800}$	2.34E−07	1.74E−13	1.18E−09	3.92E−10	
$h=\frac{1}{1600}$	5.85E−08	8.13E−15	2.89E−15	1.87E−11	1.72E−11

TABLE 5(b)
Implicit midpoint rule, $\varepsilon = 10^{-3}$, error at $t = 1$.

Global connection	IMR	1st EX	2nd EX	3rd EX	4th EX
$h = \frac{1}{100}$	8.08E−05				
$h = \frac{1}{200}$	2.02E−05	5.77E−11			
$h = \frac{1}{400}$	5.05E−06	3.61E−12	7.68E−15		
$h = \frac{1}{800}$	1.26E−06	2.26E−13	1.85E−17	1.36E−16	
$h = \frac{1}{1600}$	3.16E−07	1.41E−14	3.85E−19	1.14E−19	4.21E−19

their own. (Only after a change of stepsize, where the reappearance of nonsmooth error terms is unavoidable, local connection is appropriate.)

Our final example illustrates this situation. Tables 5(a) and 5(b) display the global error of the extrapolated midpoint rule for $t = 1$ after 50 extrapolations at the grid points $t_i = i/50$, $i = 1, \cdots, 50$. The global version performs significantly better.

REFERENCES

[1] W. AUZINGER, R. FRANK, AND F. MACSEK, *Asymptotic error expansions for stiff equations: The implicit Euler scheme*, Report Nr. 72/87, Institut für Angewandte und Numerische Mathematik, TU Wien, Austria, 1987; SIAM J. Numer. Anal., 27 (1990), to appear.

[2] W. AUZINGER AND R. FRANK, *Asymptotic error expansions for stiff equations: An analysis for the implicit midpoint and trapezoidal rules in the strongly stiff case*, Report Nr. 73/88, Institut für Angewandte und Numerische Mathematik, TU Wien, Austria, 1988; Numer. Math., submitted.

[3] ———, *Asymptotic error expansions for stiff equations: The implicit midpoint rule*, Report Nr. 77/88, Institut für Angewandte und Numerische Mathematik, TU Wien, Austria, 1988; Numer. Math., submitted.

[4] W. AUZINGER, R. FRANK, AND G. KIRLINGER, *Asymptotic error expansions for stiff equations: Applications*, Report No. 78/89, Institut für Angewandte und Numerische Mathematik, TU Wien, Austria, 1989.

[5] G. DAHLQUIST AND B. LINDBERG, *On some implicit one-step methods for stiff differential equations*, Report TRITA-NA-7302, Department of Computer Sciences, Royal Institute of Technology, Stockholm, Sweden, 1973.

[6] R. FRANK, J. HERTLING, AND H. LEHNER, *B-Convergence properties of defect correction methods* I, Numer. Math., 49 (1986), pp. 139-162.

[7] ———, *B-Convergence properties of defect correction methods* II, Numer. Math., 49 (1986), pp. 163-188.

[8] W. B. GRAGG, *Repeated extrapolation to the limit in the numerical solution of ordinary differential equations*, Ph.D. thesis, University of California, Los Angeles, CA, 1963.

[9] E. HAIRER AND CH. LUBICH, *Extrapolation at stiff differential equations*, Numer. Math., 52 (1988), pp. 377-400.

[10] R. E. O'MALLEY, JR., *Introduction to Singular Perturbations*, Academic Press, New York, London, Toronto, Sydney, San Francisco, 1974.

[11] D. R. SMITH, *Singular Perturbation Theory*, Cambridge University Press, Cambridge, London, New York, 1985.

[12] H. J. STETTER, *Asymptotic expansions for the error of discretization algorithms for non-linear functional equations*, Numer. Math., 7 (1965), pp. 18-31.

[13] M. VAN VELDHUIZEN, *Asymptotic expansions of the global error for the implicit midpoint rule (stiff case)*, Computing, 33 (1984), pp. 185-192.

[14] ———, *D-stability*, SIAM J. Numer. Anal., 18 (1981), pp. 45-64.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 964-976, September 1989
© 1989 Society for Industrial and Applied Mathematics
007

DEFECT ESTIMATION IN ADAMS PECE CODES*

DESMOND J. HIGHAM†

Abstract. Many modern codes for solving the nonstiff initial value problem $y'(x)-f(x, y(x))=0$, $y(a)$ given, $a \leqq x \leqq b$, produce, in addition to a discretised solution, a function $p(x)$ that approximates $y(x)$ over $[a, b]$. The associated defect $\delta(x) := p'(x)-f(x, p(x))$ is a natural measure of the error. In this paper the problem of reliably estimating the defect in Adams PECE methods is considered. Attention is focused on the widely used Shampine-Gordon variable order, variable step code fitted with a continuously differentiable interpolant $p(x)$ due to Watts and Shampine [*SIAM J. Sci. Statist. Comput.*, 7 (1986), pp. 334-345]. It is shown that over each step an asymptotically correct estimate of the defect can be obtained by sampling at a single, suitably chosen point. It is also shown that a valid "free" estimate can be formed without recourse to sampling. Numerical results are given to support the theory.

Key words. Adams PECE method, interpolant, defect

AMS(MOS) subject classification. 65L05

C.R. classification. G.1.7

1. Introduction.

We consider the numerical solution of the nonstiff initial value problem

$$y'(x)=f(x, y(x)), \quad y(a)=y_a, \quad a \leqq x \leqq b, \tag{1.1}$$
$$f: \mathbf{R} \times \mathbf{R}^N \to \mathbf{R}^N.$$

In addition to a discretised solution, many modern codes produce a function $p(x)$ that approximates $y(x)$ over $[a, b]$. The associated defect,

$$\delta(x) := p'(x)-f(x, p(x)),$$

is a natural measure of the error, and an estimate of the defect can provide valuable information about the accuracy of the solution. Indeed it has recently been suggested that a defect estimate may be used as an alternative to the traditional local error estimate in the stepsize control mechanism [1], [2]. In this paper we are concerned with the problem of estimating the defect when a variable order, variable step Adams PECE method is used. We restrict our attention to the latest DEPAC code [8], [12], but our results can be adapted for other implementations.

In the next section we make use of some analysis from Stetter [9] to obtain an asymptotic expansion for the defect over a step from x_n to x_n+h_{n+1}. We find that, asymptotically, the shape of the defect is determined only by the current order and the local stepsize pattern. This means that a valid estimate of $\delta(x)$ over $[x_n, x_n+h_{n+1}]$ can be formed by sampling at a single point. In particular a technique for approximating $\max_{x \in [x_n, x_n+h_{n+1}]} \|\delta(x)\|$ is given that is a more sophisticated version of that in [7]. Practical details are discussed further in § 3. In § 4 we show that an estimate of the leading term in the expansion of the defect can be formed from quantities available in the code. This gives a "free" estimate of the defect—no extra function evaluations are needed. The final section describes the results of some numerical experiments that test the two approaches.

* Received by the editors July 11, 1988; accepted for publication (in revised form) October 26, 1988. This work was supported by a Science and Engineering Research Council Research Studentship.

† Department of Mathematics, University of Manchester, Manchester M13 9PL, United Kingdom. Present address, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, M5S1A4.

2. The defect. We begin this section with a brief description of the variable order, variable step Adams PECE method implemented, and extensively documented, by Shampine and Gordon [7]. An updated version of the code forms part of the DEPAC ODE solving package [8]; see also [12].

The basic aim is to produce discrete solution vectors $y_n \approx y(x_n)$ by stepping along a mesh $a = x_0 < x_1 < \cdots$. Given $y_n \approx y(x_n)$ and the past f values

$$f_{n+1-j} = f(x_{n+1-j}, y_{n+1-j}), \qquad j = 1, \cdots, k,$$

a kth order step ($1 \leqq k \leqq 12$) is taken as follows. First a predicted approximation to $y(x_{n+1})$ is formed according to the kth order Adams-Bashforth formula,

$$p_{n+1} = y_n + \int_{x_n}^{x_{n+1}} P_{k,n}(t)\, dt,$$

where $P_{k,n}(t)$ denotes the unique interpolating polynomial of degree $\leqq k-1$ that satisfies

$$P_{k,n}(x_{n+1-j}) = f_{n+1-j}, \qquad j = 1, \cdots, k.$$

The new approximation y_{n+1} is then formed using the $(k+1)$st order Adams-Moulton formula,

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} P^*_{k+1,n}(t)\, dt. \tag{2.1}$$

Here $P^*_{k+1,n}(t)$ denotes the polynomial of degree $\leqq k$ satisfying

$$P^*_{k+1,n}(x_{n+1-j}) = f_{n+1-j}, \qquad j = 1, \cdots, k$$

and

$$P^*_{k+1,n}(x_{n+1}) = f^p_{n+1} := f(x_{n+1}, p_{n+1}).$$

The term PECE is used to indicate the four stages – predict the value p_{n+1}, evaluate f^p_{n+1}, correct to give y_{n+1}, and then evaluate f_{n+1}, which is needed for the next step.

From (2.1), a natural way to approximate the solution at a point $x \in (x_n, x_{n+1})$ is to use the function

$$\eta(x) = y_n + \int_{x_n}^{x} P^*_{k+1,n}(t)\, dt \tag{2.2a}$$

that may also be written

$$\eta(x) = y_{n+1} + \int_{x_{n+1}}^{x} P^*_{k+1,n}(t)\, dt. \tag{2.2b}$$

The polynomials $\eta(x)$ from each step can be joined together to give a globally continuous approximation to $y(x)$ over $[a, b]$. However, because $\eta'(x_{n+1})$ matches $f(x_{n+1}, p_{n+1})$ and not $f(x_{n+1}, y_{n+1})$, there is a discontinuity in the first derivative at each meshpoint.

An obvious alternative to $P^*_{k+1,n}(t)$ in (2.2) is the polynomial $P_{k+1,n+1}(t)$ that satisfies

$$P_{k+1,n+1}(x_{n+1-j}) = f_{n+1-j}, \qquad j = 0, \cdots, k.$$

This leads to the approximations

$$S(x) = y_n + \int_{x_n}^{x} P_{k+1,n+1}(t)\, dt \tag{2.3}$$

$$y_I(x) = y_{n+1} + \int_{x_{n+1}}^{x} P_{k+1,n+1}(t)\, dt. \tag{2.4}$$

Note that $S'(x) = y_I'(x)$ and hence $S(x)$ and $y_I(x)$ differ only by a constant term. When we use (2.1) and the Lagrangian form of the interpolating polynomials, it is easy to show that

$$S(x) - y_I(x) = \{f_{n+1} - f_{n+1}^p\} \frac{\int_{x_n}^{x_{n+1}} \prod_{i=1}^{k} (t - x_{n+1-i})\, dt}{\prod_{i=1}^{k} (x_{n+1} - x_{n+1-i})}. \tag{2.5}$$

Since $y_I(x_n) \neq y_n$ and $S(x_{n+1}) \neq y_{n+1}$, the corresponding piecewise polynomials over $[a, b]$ both have jumps at the meshpoints, although the first derivatives are continuous.

The interpolant $y_I(x)$ was implemented in the original Shampine-Gordon code and in early versions of the DEPAC Adams code. However, the latest version of DEPAC uses a higher degree polynomial interpolant due to Watts and Shampine [12] that has better continuity properties. This interpolant, which is a polynomial of degree $\leqq k+2$ over $[x_n, x_{n+1}]$, can be written

$$T(x) = S(x) + [y_{n+1} - S(x_{n+1})] \frac{\Phi(x)}{\Phi(x_{n+1})}$$

where

$$\Phi(x) = \int_{x_n}^{x} \prod_{i=0}^{k} (t - x_{n+1-i})\, dt.$$

It is clear that $T(x)$ satisfies the conditions

$$T(x_n) = y_n, \qquad T(x_{n+1}) = y_{n+1},$$
$$T'(x_n) = f_n, \qquad T'(x_{n+1}) = f_{n+1},$$

and consequently can be used to provide a global approximation to $y(x)$ that is continuously differentiable. We may regard $T(x)$ as a slightly perturbed version of $S(x)$ (or $y_I(x)$) with the small perturbation added in order to force C^1 continuity.

We will denote the defect in $\eta(x)$ by $\delta^{\eta}(x)$, that is,

$$\delta^{\eta}(x) = \eta'(x) - f(x, \eta(x)).$$

Similarly, $\delta^{y_I}(x)$, $\delta^{S}(x)$ and $\delta^{T}(x)$ will denote the defects in $y_I(x)$, $S(x)$, and $T(x)$, respectively. We are mainly concerned with the defect in $T(x)$, since this is the interpolant currently used in DEPAC, but it will prove helpful and enlightening if we also look at $\delta^{\eta}(x)$, $\delta^{y_I}(x)$, and $\delta^{S}(x)$. In performing an asymptotic analysis, we make use of some results of Stetter [9]. Although Stetter's main concern was to assess the quality of the interpolants themselves, his analysis can be adapted in a straightforward manner to give the desired information about the behaviour of the defect.

Consider taking a step of length h_{n+1} at order k from x_n to $x_{n+1} = x_n + h_{n+1}$. The PECE formula uses information from the past k meshpoints $\{x_{n+1-k}, \cdots, x_n\}$. We define the quantities σ_i by

$$x_{n+1-i} = x_n + \sigma_i h_{n+1}. \tag{2.6}$$

For the step-changing strategy used in [7], the $\{\sigma_i\}_{i=0}^{k}$ can be bounded above and below. This is discussed in more detail in the next section. We define the local solution $z(x)$ by

$$z'(x) = f(x, z(x)), \qquad z(x_n) = y_n,$$

and the local error for the step as $y_{n+1} - z(x_{n+1})$. For clarity we will drop the subscript $n+1$ from h_{n+1} in the succeeding analysis. The basic approach in [9] is to assume that

the stepsize and error control mechanism keeps the local error at the same level over $[x_{n+1-k}, x_{n+1}]$. Then, assuming that f is sufficiently smooth, standard interpolation theory is used to show that

$$p_{n+1} - z(x_{n+1}) = O(h^{k+1}) \tag{2.7}$$

and

$$y_{n+1} - z(x_{n+1}) = O(h^{k+2}). \tag{2.8}$$

Furthermore, at a point $x \in [x_n, x_{n+1}]$,

$$\eta(x) - z(x) = O(h^{k+2}) \tag{2.9}$$

and

$$S(x) - z(x) = O(h^{k+2}). \tag{2.10}$$

We can use (2.7)-(2.10) to deduce similar results for $y_I(x)$ and $T(x)$. From (2.5),

$$S(x) - y_I(x) = \{f_{n+1} - f^p_{n+1}\} h \frac{\int_0^1 \prod_{i=1}^k (\alpha - \sigma_i)\, d\alpha}{\prod_{i=1}^k (1 - \sigma_i)}.$$

Employing a Lipschitz condition on f, (2.7) and (2.8), and the fact that $\sigma_1, \sigma_2, \cdots, \sigma_k$ are nonpositive and bounded below it follows that

$$S(x) - y_I(x) = O(h^{k+2}). \tag{2.11}$$

Hence, from (2.10)

$$y_I(x) - z(x) = O(h^{k+2}). \tag{2.12}$$

Also, writing $x = x_n + sh$, we have

$$\begin{aligned} T(x) - S(x) &= [\, y_{n+1} - S(x_{n+1})] \frac{\int_0^s \prod_{i=0}^k (\alpha - \sigma_i)\, d\alpha}{\int_0^1 \prod_{i=0}^k (\alpha - \sigma_i)\, d\alpha} \\ &= O(\| y_{n+1} - S(x_{n+1}) \|) \\ &= O(h^{k+2}), \end{aligned}$$

from (2.8) and (2.10), and hence

$$T(x) - z(x) = O(h^{k+2}).$$

Since we are assuming that f is Lipschitzian, we have, using (2.9) and (2.2a),

$$\begin{aligned} \delta^{\eta}(x) &= \eta'(x) - f(x, \eta(x)) \\ &= \eta'(x) - z'(x) + f(x, z(x)) - f(x, \eta(x)) \\ &= P^*_{k+1,n}(x) - z'(x) + O(h^{k+2}). \end{aligned}$$

Using an expansion of $P^*_{k+1,n}(x) - z'(x)$ from [9] we find that

$$\begin{aligned} \delta^{\eta}(x_n + sh) = -h^{k+1} \Bigg\{ & \pi(s) \frac{z^{(k+2)}(x_{n+1})}{(k+1)!} \\ & + \hat{\pi}(s) \frac{\bar{g}_{n+1} f_y(x_{n+1}, z(x_{n+1})) z^{(k+1)}(x_{n+1})}{\hat{\pi}(1)} \Bigg\} + O(h^{k+2}) \end{aligned} \tag{2.13}$$

where

$$\pi(s) = \prod_{i=0}^k (s - \sigma_i), \quad \hat{\pi}(s) = \prod_{i=1}^k (s - \sigma_i), \quad \bar{g}_{n+1} = \frac{1}{k!} \int_0^1 \hat{\pi}(\alpha)\, d\alpha,$$

and f_y is the Jacobian of f. Similarly,

$$\begin{aligned}\delta^S(x) &= S'(x) - z'(x) + f(x, z(x)) - f(x, S(x)) \\ &= P_{k+1,n+1}(x) - z'(x) + O(h^{k+2}),\end{aligned}$$

and using Stetter's asymptotic result for $P_{k+1,n+1}(x) - z'(x)$ we have

$$\delta^S(x_n + sh) = -h^{k+1}\pi(s)\frac{z^{(k+2)}(x_{n+1})}{(k+1)!} + O(h^{k+2}). \tag{2.14}$$

Asymptotically, the defect in $y_I(x)$ is the same as that in $\delta^S(x)$. This follows from

$$\delta^{y_I}(x) = y_I'(x) - z'(x) + O(h^{k+2})$$

and

$$y_I'(x) = S'(x) = P_{k+1,n+1}(x).$$

Finally, an expression for $\delta^T(x)$ can be deduced from (2.14). We have

$$\begin{aligned}\delta^T(x) &= T'(x) - z'(x) + O(h^{k+2}) \\ &= S'(x) + [y_{n+1} - S(x_{n+1})]\frac{\Phi'(x)}{\Phi(x_{n+1})} - z'(x) + O(h^{k+2}) \\ &= \delta^S(x) + [y_{n+1} - S(x_{n+1})]\frac{\Phi'(x)}{\Phi(x_{n+1})} + O(h^{k+2}).\end{aligned}$$

Using (2.14) this may be written

$$\delta^T(x_n + sh) = -h^{k+1}\pi(s)\left\{\frac{z^{(k+2)}(x_{n+1})}{(k+1)!} - \frac{[y_{n+1} - S(x_{n+1})]}{\Phi(x_{n+1})}\right\} + O(h^{k+2}). \tag{2.15}$$

The key point arising from this analysis is that the defects in $S(x)$, $y_I(x)$, and $T(x)$ are all of the form

$$\delta(x_n + sh) = A(h)\pi(s) + O(h^{k+2}) \tag{2.16}$$

where $A(h)$ is $O(h^{k+1})$ and independent of s. Recalling that $\pi(s) = \prod_{i=0}^{k}(s - \sigma_i)$, we see that, asymptotically, the shape of the defect over $[x_n, x_{n+1}]$ is determined by the local stepsize pattern only. The practical importance of this result is that by sampling the defect at an internal point $x_n + \hat{s}h$, an approximation $\delta(x_n + \hat{s}h)/\pi(\hat{s}) \approx A(h)$ can be formed. Hence an estimate of $\delta(x_n + sh)$ for $s \in [0, 1]$ is available and so any desired measure of the defect can be estimated. In particular, to approximate $\max_{[0,1]} \|\delta(x_n + sh)\|$ for any vector norm $\|\cdot\|$, we simply form $\|\delta(x_n + s^*h)\|$ where s^* in $[0, 1]$ maximises $|\pi(s)|$. This approach is discussed in more detail in the next section. We mention that the idea of estimating the maximum defect in $y_I(x)$ by sampling at a single point was proposed in [7, p. 122]. Shampine and Gordon suggest sampling at the midpoint on heuristic grounds; the defect is zero at the two meshpoints. We will show that whilst this is never a bad choice, it is only optimal when the lowest order $k = 1$ is used for the step.

The expression for $\delta^\eta(x)$ in (2.13) is of a slightly different form to (2.16). The leading term in (2.13) is a problem-dependent linear combination of $\pi(s)$ and $\hat{\pi}(s)$. In this case two samples would be needed to form an estimate of the defect over a step.

To conclude this section we mention some related work by Hanson and Enright [5]. In its simplest form, the error control mechanism used in [7] ensures that on each step

$$\|y_{n+1} - y_{n+1}(k)\|_2 \leqq \mathrm{TOL} \tag{2.17}$$

for some user-supplied parameter TOL. Here $y_{n+1}(k)$ denotes the result of an Adams-Bashforth predictor of order k followed by an Adams-Moulton corrector of order k. Under certain reasonable assumptions, Hanson and Enright show that this strategy indirectly controls the defect in $S(x)$ and $\eta(x)$, in the sense that

$$\|\delta(x)\| \leqq (C + O(h_{\max}))\text{TOL} \quad \text{for all } x \in [a, b]$$

where C is a constant that depends on f, and $h_{\max} = \max_{n \geqq 0} h_{n+1}$. We note that a similar relationship can be shown to hold for the more recent interpolant $T(x)$. This follows because $\delta^T(x)$ is an $O(h^{k+1})$ perturbation of $\delta^S(x)$ and the left-hand side of (2.17) is also $O(h^{k+1})$. The numerical tests of § 5 confirm that the tolerance parameter can be used to control the defect in $T(x)$, albeit in a problem-dependent manner.

3. Location of s^*. In the last section we have shown that an asymptotically valid estimate of $\max_{[0,1]} \|\delta^T(x_n + sh_{n+1})\|$ is given by sampling at $x_n + s^* h_{n+1}$, where s^*, which we will call the optimal sample point, satisfies $|\pi(s^*)| = \max_{[0,1]} |\pi(s)|$. We now examine the practical details of computing s^* and gain some insight into the behaviour of s^* in the course of an integration.

Recall that after a kth order step from x_n to $x_n + h_{n+1}$,

$$\pi(s) = \prod_{i=0}^{k} (s - \sigma_i)$$

where

$$\sigma_i = \frac{x_{n+1-i} - x_n}{h_{n+1}}.$$

Note that $\sigma_k < \sigma_{k-1} < \cdots < \sigma_1 = 0 < \sigma_0 = 1$. By Rolle's Theorem, $\pi'(s)$ has roots in (σ_i, σ_{i-1}), for $i = 1, \cdots, k$, and because $\pi'(s)$ is a polynomial of degree k, each interval must contain exactly one root. Since $\pi(0) = \pi(1) = 0$ the point s^* that we require is the root of $\pi'(s)$ in $(0, 1)$. To compute s^*, given that $\pi'(s)$ and $\pi''(s)$ are inexpensive to evaluate, a straightforward application of Newton's method is sufficient. The fact that s^* is the largest root of a polynomial with purely real roots implies that, providing the initial guess is greater than s^*, the Newton iterates converge monotonically and quadratically to s^* (see [10, p. 272]). In particular, convergence is guaranteed if we start at the point $s^{(0)} = 1$, but later we will show that a better choice is available.

The following theorem gives a useful characterization of s^*.

THEOREM 1. *For $k \geqq 2$, the point s^* defined above is the solution in $(0, 1)$ of*

$$\frac{1 - 2s}{s(s-1)} = \sum_{i=2}^{k} \frac{1}{s - \sigma_i}, \tag{3.1}$$

and for $k = 1$, $s^* = \frac{1}{2}$.

Proof. When $k = 1$, $\pi(s) = (s-1)s$ and the result is trivial. When $k \geqq 2$ we have

$$\pi(s) = (s-1)s(s-\sigma_2)\cdots(s-\sigma_k),$$

so that

$$\begin{aligned} \pi'(s) &= s(s-\sigma_2)\cdots(s-\sigma_k) + (s-1)(s-\sigma_2)\cdots(s-\sigma_k) \\ &\quad + \sum_{i=2}^{k} \frac{(s-1)s(s-\sigma_2)\cdots(s-\sigma_k)}{s-\sigma_i} \end{aligned} \tag{3.2}$$

in $(0, 1)$. Defining

$$g(s) = \frac{\pi'(s)}{(s-\sigma_2)\cdots(s-\sigma_k)}, \tag{3.3}$$

we note that the zeros of $g(s)$ in $(0, 1)$ are precisely the zeros of $\pi'(s)$ in $(0, 1)$. From (3.2) and (3.3),

$$g(s) = 2s - 1 + \sum_{i=2}^{k} \frac{s(s-1)}{s - \sigma_i}$$

and the result follows. □

The left-hand side of (3.1), which is independent of $\{\sigma_i\}_{i=2}^k$, is plotted in Fig. 1. Since the right-hand side is positive in $(0, 1)$, it follows that $s^* > 1/2$ for $k \geqq 2$. To obtain more precise bounds for s^* we must use information about the step-changing mechanism. First we give a corollary to the theorem.

COROLLARY 1. *For $k \geqq 2$, if the two sets $\{\hat{\sigma}_i\}_{i=2}^k$ and $\{\bar{\sigma}_i\}_{i=2}^k$ satisfy*

$$\hat{\sigma}_i \leqq \bar{\sigma}_i < 0, \qquad i = 2, \cdots, k,$$

then the corresponding optimal sample points, $\hat{s}^*$ and $\bar{s}^*$, satisfy $\hat{s}^* \leqq \bar{s}^*$.

Proof. For $s \in (0, 1)$,

$$\sum_{i=2}^{k} \frac{1}{s - \hat{\sigma}_i} \leqq \sum_{i=2}^{k} \frac{1}{s - \bar{\sigma}_i}. \tag{3.4}$$

The intersection of $\sum_{i=2}^k 1/(s - \hat{\sigma}_i)$ and $(1-2s)/s(s-1)$ will therefore be to the left of the intersection of $\sum_{i=2}^k 1/(s - \bar{\sigma}_i)$ and $(1-2s)/s(s-1)$ (see Fig. 1). □

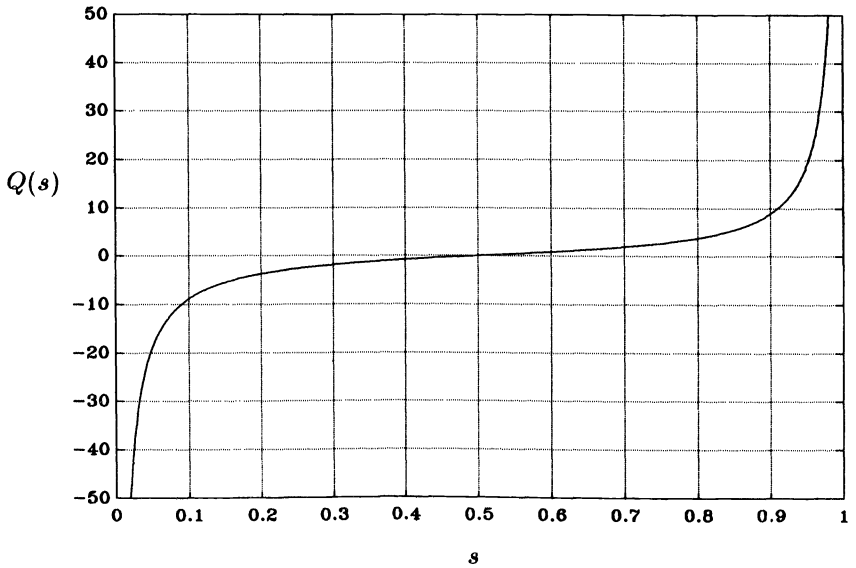


FIG. 1. $Q(s) = (1-2s)/s(s-1)$.

The code we are considering uses a variable order $1 \leqq k \leqq 12$ and a variable stepsize. However, for several reasons (an important one being to ensure numerical stability) the ratio of successive stepsizes is bounded in the following way [7, p. 64]:

$$\frac{1}{2} \leqq \frac{h_{i-1}}{h_i} \leqq 8$$

for all i, except, possibly, those for which the step involving h_i is taken at order one. The right-hand inequality is extremely unlikely to be close to equality in practice [7, p. 117]. From the definition of $\{\sigma_i\}_{i=0}^{k}$ it follows that on a general step of order k we have

$$\sigma_i^l \leqq \sigma_i \leqq \sigma_i^u, \qquad i = 0, \cdots, k$$

where the upper and lower bounds are defined by

$$\sigma_i^u = \frac{-(2^{i-1}-1)}{2^{i-1}} \quad \text{and} \quad \sigma_i^l = \frac{-(8^i-8)}{7}.$$

Denoting the corresponding optimal sample points by s^{*u} and s^{*l}, it follows from the corollary that

$$s^{*l} \leqq s^* \leqq s^{*u}.$$

The values of s^{*l} and s^{*u} for $k = 1, \cdots, 12$ are given in Table 1. Note that since s^{*u} is an upper bound for s^*, it may be used as the initial guess in the Newton iteration that we discussed earlier.

TABLE 1
Values of s^{*l}, s^{*u}, and $s^{*\text{const}}$ at order k.

k	1	2	3	4	5	6	7	8	9	10	11	12
s^{*l}	.500	.515	.516	.517	.517	.517	.517	.517	.517	.517	.517	.517
s^{*u}	.500	.608	.672	.717	.750	.777	.799	.817	.832	.844	.855	.865
$s^{*\text{const}}$	.500	.577	.618	.644	.663	.678	.690	.699	.708	.715	.721	.726

In addition to bounding the extreme values of s^*, it is possible to say something about the typical behaviour of s^*. Although the code allows the stepsize to vary, it is strongly biased against doing so. This reduces the overheads involved in the formation of p_{n+1} and y_{n+1}. Another reason for the bias is that the order changing mechanism is set up so that an increase in order is only considered if the past $k+1$ steplengths are equal. Consequently, the overall step pattern can usually be divided into a small number of groups each with a constant steplength. When $h_{n+2-k} = h_{n+3-k} = \cdots = h_{n+1}$, the sigma values become

$$\sigma_i = \sigma_i^{\text{const}} = 1 - i, \qquad i = 0, \cdots, k,$$

and the corresponding $s^{*\text{const}}$ values appear in Table 1.

4. The free estimate. The cost of taking a step with a discrete ODE method is often measured in terms of the number of f evaluations required. Sampling the defect at one intermediate point clearly increases this quantity by one. In the case of Runge-Kutta-interpolation schemes, where a large number (usually $\geqq 6$) of f evaluations are needed for each step, this increase in cost is relatively small. However, the PECE method considered here uses only two evaluations and so the cost of sampling on each step is more significant. In this section we show that it is possible to form an estimate of $\delta^T(x)$ without using any additional f evaluations. The basic idea is to approximate the leading term in the expansion (2.15) of $\delta^T(x_n + sh)$. In particular, to obtain a valid estimate of $\max_{[0,1]} \|\delta^T(x_n + sh)\|$ it is sufficient to form an $O(h)$ accurate approximation to

$$h^{k+1}\pi(s^*)\left\{\frac{z^{(k+2)}(x_{n+1})}{(k+1)!} - \frac{[y_{n+1} - S(x_{n+1})]}{\Phi(x_{n+1})}\right\}, \tag{4.1}$$

and take its norm. To this end we consider below the computation of $\Phi(x_{n+1})$, $y_{n+1}-S(x_{n+1})$ and an approximation to $z^{(k+2)}(x_{n+1})$. To keep the overall technique as simple as possible we will attempt to make use of quantities that have already been computed during the course of a step.

From Watts and Shampine [12, p. 341], $\Phi(x_{n+1})$ may be written

$$\Phi(x_{n+1}) = -h^2 g_{k+1,2}(1) \prod_{i=1}^{k} (x_{n+1}-x_{n+1-i})$$

where $g_{k+1,2}(1)$ is a certain double integral. The values of $\{x_{n+1}-x_{n+1-i}\}_{i=1}^{k}$ are immediately available, and the $g_{k+1,2}(1)$ term can be readily computed from quantities that are used in the code. In fact $g_{k+1,2}(1)$ is formed in the interpolation routine, SINTRP, where it is called GDI (for more details see [11]).

The vector y_{n+1} is, of course, already available and $S(x_{n+1})$ can be formed in a straightforward way (see [11]). However, the computed difference $y_{n+1}-S(x_{n+1})$ is liable to suffer from catastrophic cancellation at stringent tolerances. To see this, we observe from (2.8) and (2.10) that $y_{n+1}-S(x_{n+1})$ is $O(h^{k+2})$, which is one order higher than the quantity being controlled by TOL in (2.17). Putting $x = x_{n+1}$ in (2.5) we obtain the alternative representation

$$y_{n+1}-S(x_{n+1}) = \{f_{n+1}^{p}-f_{n+1}\} \frac{\int_{x_n}^{x_{n+1}} \prod_{i=1}^{k} (t-x_{n+1-i})\, dt}{\prod_{i=1}^{k} (x_{n+1}-x_{n+1-i})},$$

which may be written

$$y_{n+1}-S(x_{n+1}) = \{f_{n+1}^{p}-f_{n+1}\} hG(k+1)$$

where $G(k+1)$ is computed in the code. The difference $f_{n+1}^{p}-f_{n+1}$ is $O(h^{k+1})$ and so this representation should be less susceptible to cancellation.

To produce an estimate for $z^{(k+2)}(x_{n+1})$ we turn again to Stetter's analysis [9]. Under the assumption that the stepsize and error control mechanism has kept the local error at the same level over $[x_{n+1-k}, x_{n+1}]$, he concludes that

$$f_{n+1-i} = z'(x_{n+1-i}) + O(h^{k+2}), \qquad i = 0, \cdots, k. \tag{4.2}$$

We need to extend the assumption to cover $[x_{n-k}, x_{n+1}]$ so that

$$f_{n-k} = z'(x_{n-k}) + O(h^{k+2}) \tag{4.3}$$

also holds. We use the standard notation $f[\,]$ and $z'[\,]$ for divided differences based on $\{f_{n+1-i}\}_{i=0}^{k+1}$ and $\{z'(x_{n+1-i})\}_{i=0}^{k+1}$, respectively; that is, $f[x_{n-k}] = f_{n-k}$ and

$$f[x_{n-k}, \cdots, x_{n-k+i}] = \frac{f[x_{n-k}, \cdots, x_{n-k+i-1}] - f[x_{n-k+1}, \cdots, x_{n-k+i}]}{x_{n-k}-x_{n-k+i}},$$

$$i = 1, \cdots, k+1.$$

Using classical interpolation theory (see, for example, [7, p. 39]), we have

$$z'[x_{n-k}, \cdots, x_{n+1}] = \frac{z^{(k+2)}(x_{n+1})}{(k+1)!} + O(h),$$

for sufficiently smooth z. Now, from (4.2) and (4.3), it can be proved by induction that

$$f[x_{n-k}, \cdots, x_{n+1}] - z'[x_{n-k}, \cdots, x_{n+1}] = O(h).$$

Hence

$$f[x_{n-k}, \cdots, x_{n+1}] = \frac{z^{(k+2)}(x_{n+1})}{(k+1)!} + O(h),$$

and so it is reasonable to use the divided difference $f[x_{n-k}, \cdots, x_{n+1}]$ to approximate $z^{(k+2)}(x_{n+1})/(k+1)!$ in (4.1). The DEPAC routine requires $f[x_{n-k}, \cdots, x_{n+1}]$ when deciding whether to increase the order. An order increase is only considered when the stepsize is locally constant, that is when $h_{n+1-k} = h_{n+2-k} \cdots = h_{n+1}$, so the divided difference is only available from the code in this situation. However, a scaled version of $f[x_{n+1-k}, \cdots, x_{n+1}]$ is always available and hence, by storing divided difference information from the previous step, the required quantity can be computed at little cost.

Finally we point out that there will be a small number of steps at the start of the integration on which $f[x_{n-k}, \cdots, x_{n+1}]$ is not directly obtainable. On the first step from x_0 to x_1, for which the order k is always equal to one, we require $f[x_{-1}, x_0, x_1]$. This term involves a function evaluation f_{-1} that has not actually be made. Similarly if the order is increased to two for the next step, then at the end of that step we need $f[x_{-1}, x_0, x_1, x_2]$. Generally, f_{-1} remains until we step from x_n to x_{n+1} with order $k \leqq n$. In the numerical testing described in the next section, rather than computing an extra f value to act as f_{-1} we refrained from forming an estimate of the defect until $n \geqq k$.

5. Numerical results. The techniques we have described produce estimates of the defect that are valid asymptotically. In this section we investigate the accuracy of the estimates in practical computations. To do this we used the DEPAC Adams package in "single-step" mode by making repeated calls to the routines STEPS and SINTRP. STEPS is the routine that advances the solution by a single step and, once a step has been taken, SINTRP can be called to evaluate $T(x)$ and $T'(x)$ at some point within the step. After advancing from x_n to $x_n + h$ we formed

$$\mathrm{DEF}_n := \max_{0 \leqq j \leqq 100} \|\delta^T(x_n + jh/100)\|_\infty$$

as a discrete approximation to $\max_{s \in [0,1]} \|\delta^T(x_n + sh)\|_\infty$, along with

$$\mathrm{SAM}_n := \|\delta^T(x_n + s^* h)\|_\infty,$$

and

$$\text{(5.1)} \quad \mathrm{APPROX}_n := \left\| h^{k+1} \pi(s^*) \left[f[x_{n-k}, \cdots, x_{n+1}] + \frac{\{f^p_{n+1} - f_{n+1}\} G(k+1)}{h g_{k+1,2}(1) \prod_{i=1}^{k} (x_{n+1} - x_{n+1-i})} \right] \right\|_\infty,$$

as discussed in §§ 3 and 4. We recorded the quantities

$$\mathrm{DEFMAX} := \max_{n \geqq 0} \{\mathrm{DEF}_n\},$$

$$\mathrm{SAMMAX} := \max_{n \geqq 0} \left\{ \frac{\mathrm{DEF}_n}{\mathrm{SAM}_n} \right\},$$

$$\mathrm{AUNDERMAX} := \max_{n \geqq 0} \left\{ \frac{\mathrm{DEF}_n}{\mathrm{APPROX}_n} \right\},$$

$$\mathrm{AOVERMAX} := \max_{n \geqq 0} \left\{ \frac{\mathrm{APPROX}_n}{\mathrm{DEF}_n} \right\}.$$

Hence SAMMAX measures the worst case of the sampled value underestimating DEF_n. Similarly, AUNDERMAX and AOVERMAX measure the extreme cases of the free approximation under- and overestimating DEF_n. We would like all three values to be as close to one as possible.

We mention that in order to compute $\mathbf{APPROX}_n$ using (5.1) an extra parameter, used to return f_{n+1}^p to the driver program, was added to the call list of STEPS. No other changes were made to STEPS.

The following test problems were used:

(i) The orbit equations [3, Class D]:

$$y_1' = y_3, \qquad y_1(0) = 1 - \varepsilon,$$

$$y_2' = y_4, \qquad y_2(0) = 0,$$

$$y_3' = \frac{-y_1}{(y_1^2 + y_2^2)^{3/2}}, \qquad y_3(0) = 0,$$

$$y_4' = \frac{-y_2}{(y_1^2 + y_2^2)^{3/2}}, \qquad y_4(0) = \left(\frac{1+\varepsilon}{1-\varepsilon}\right)^{1/2},$$

$$0 \leqq x \leqq 20,$$

where values of .1, .5, and .9 were chosen for the eccentricity parameter ε.

(ii) A logistic curve [3, Problem A4]:

$$y' = \frac{y}{4}\left(1 - \frac{y}{20}\right), \quad y(0) = 1, \quad 0 \leqq x \leqq 20.$$

(iii) A problem due to Fehlberg that is used in [4, p. 174]:

$$y_1' = 2xy_1 \log(\max(y_2, 10^{-3})), \qquad y_1(0) = 1,$$

$$y_2' = -2xy_2 \log(\max(y_1, 10^{-3})), \qquad y_2(0) = e,$$

$$0 \leqq x \leqq 5.$$

In each case an absolute error criterion was specified with tolerances $\mathrm{TOL} = 10^{-2}, 10^{-3}, \cdots, 10^{-8}$. Results are presented in Tables 2-6. We see that the SAMMAX values are extremely good with the maximum over all problems and tolerances being 1.34. In fact on the vast majority of steps the ratio $\mathbf{DEF}_n/\mathbf{SAM}_n$ was less than 1.01. The sample

TABLE 2
Orbit problem, $\varepsilon = .1$.

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
DEFMAX	9E−2	1E−2	2E−3	2E−4	2E−5	2E−6	4E−7
SAMMAX	1.01	1.00	1.01	1.01	1.02	1.02	1.01
AUNDERMAX	1.50	2.18	2.36	2.24	4.32	3.28	5.03
AOVERMAX	1.78	2.77	2.48	5.44	3.97	2.81	2.67

TABLE 3
Orbit problem, $\varepsilon = .5$.

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
DEFMAX	2E−1	3E−2	6E−3	5E−4	5E−5	6E−6	4E−7
SAMMAX	1.04	1.02	1.02	1.01	1.03	1.02	1.02
AUNDERMAX	3.47	2.66	8.59	6.41	5.05	8.18	5.75
AOVERMAX	2.64	3.36	3.82	3.95	5.19	4.51	5.63

TABLE 4
Orbit problem, $\varepsilon = .9$.

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
DEFMAX	3E+0	3E−1	9E−2	1E−2	8E−4	7E−5	7E−6
SAMMAX	1.09	1.02	1.02	1.01	1.02	1.02	1.15
AUNDERMAX	5.23	21.33	12.58	19.21	18.38	15.30	8.69
AOVERMAX	6.11	5.46	9.15	7.16	13.03	38.85	10.51

TABLE 5
Logistic curve.

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
DEFMAX	7E−3	5E−4	5E−5	7E−6	6E−7	7E−8	7E−9
SAMMAX	1.01	1.00	1.28	1.01	1.05	1.06	1.34
AUNDERMAX	11.05	1.58	9.62	2.20	5.77	9.85	4.11
AOVERMAX	1.32	7.36	9.29	1.84	1.61	8.72	5.64

TABLE 6
Fehlberg problem.

TOL	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}
DEFMAX	8E−1	1E−1	2E−2	1E−3	2E−4	2E−5	1E−6
SAMMAX	1.09	1.03	1.03	1.11	1.25	1.04	1.21
AUNDERMAX	5.35	6.89	14.07	26.06	11.89	9.32	47.38
AOVERMAX	3.12	2.63	2.44	9.35	10.83	7.79	13.74

point s^* for each step was found to full accuracy (unit roundoff $\approx 2 \times 10^{-16}$) in the manner described in § 3. The average number of Newton iterations required was five and the maximum was six. The free approximation, whilst being less accurate, could usually be relied on to give the correct order of magnitude of the defect over a step. At more stringent tolerances of 10^{-9} and 10^{-10}, however, we found that the AUNDERMAX and AOVERMAX values began to worsen considerably. This was caused by severe cancellation in the formation of $f^P_{n+1} - f_{n+1}$. The problem only arose during the first few steps of an integration; here the code appeared to use conservatively small stepsizes.

Throughout the testing we also sampled the defect at the midpoint of each step, as suggested in [7], and computed

$$\text{MIDPTMAX} = \max_{n \geq 0} \left\{ \frac{\text{DEF}_n}{\|\delta^T(x_n + .5h)\|_\infty} \right\}.$$

We found that although the midpoint sample never gave a very bad estimate of DEF_n, MIDPTMAX was always greater than the corresponding SAMMAX value and had an average of 1.9. Generally, sampling at $x_n + .5h$ was least successful on steps where the order k was high. This is to be expected from Table 1; at higher orders s^* is likely to be further from .5. Finally we mention that DEFMAX is seen to be controlled by TOL as predicted at the end of § 2.

In summary, we have given an asymptotic analysis of the defect associated with some commonly used Adams-interpolation procedures and have presented two techniques for estimating the defect in the DEPAC implementation. A reasonable order-of-magnitude estimate can be formed at little cost, and for the price of one f evaluation per step an extremely accurate alternative is available. As well as being of interest in their own right, these estimates may prove useful in the development of an error-control scheme of the type discussed in [1], [2], [6] where the defect, rather than the local error, is directly controlled.

Acknowledgments. Helpful comments from Nick Higham and George Hall improved this manuscript.

REFERENCES

[1] W. H. ENRIGHT, *A new error-control for initial value solvers*, Numerical Analysis Report No. 122, University of Manchester, UK, 1986.

[2] ———, *Analysis of error control strategies for continuous Runge-Kutta methods*, Tech. Report No. 205/87, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1987.

[3] W. H. ENRIGHT AND J. D. PRYCE, *Two* FORTRAN *packages for assessing initial value methods*, ACM Trans. Math. Software, 13 (1987), pp. 1-27.

[4] E. HAIRER, S. P. NORSETT, AND G. WANNER, *Solving Ordinary Differential Equations* I, Springer-Verlag, Berlin, New York, 1987.

[5] P. M. HANSON AND W. H. ENRIGHT, *Controlling the defect in existing variable-order Adams codes for initial-value problems*, ACM Trans. Math. Software, 9 (1983), pp. 71-97.

[6] D. J. HIGHAM, *Robust defect control with Runge-Kutta schemes*, Numerical Analysis Report No. 150, University of Manchester, UK, 1987.

[7] L. F. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman, San Francisco, CA, 1975.

[8] L. F. SHAMPINE AND H. A. WATTS, DEPAC—*Design of a user oriented package of* ODE *solvers*, Report SAND79-2374, Sandia National Laboratories, Albuquerque, NM, 1979.

[9] H. J. STETTER, *Interpolation and error estimation in Adams* PC-*Codes*, SIAM J. Numer. Anal., 16 (1979), pp. 311-323.

[10] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, Berlin, New York, 1980.

[11] H. A. WATTS, *A smoother interpolant for* DE/STEP, INTRP *and* DEABM: II, Report SAND84-0293, Sandia National Laboratories, Albuquerque, NM, 1984.

[12] H. A. WATTS AND L. F. SHAMPINE, *Smoother interpolants for Adams codes*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 334-345.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 977-989, September 1989
© 1989 Society for Industrial and Applied Mathematics
008

PRACTICAL RUNGE-KUTTA PROCESSES*

J. R. DORMAND† AND P. J. PRINCE†

Abstract. The development of embedded Runge-Kutta and Runge-Kutta-Nystrom formulae subject to various criteria is reviewed. An important criterion concerns the cost of achieving a particular global error in the numerical solution. By consideration of local truncation errors in the two formulae of an embedded pair, it is possible to produce a good process. Another criterion involves the provision of continuous solutions. Such a requirement can be at odds with the previous one of basic cost-effectiveness. However, it seems important to provide dense output without excessive cost in new function evaluations. Special RK/RKN formulae are preferable for practical global error estimation using the Zadunaisky pseudo-problem or related technique of solving for the error estimate. Two-term error estimation can be achieved and the pseudo-problem can be based on dense output values.

Key words. Runge-Kutta, global error, density output

AMS(MOS) subject classification. 65L05

1. Introduction.

We consider the first-order system of nonstiff ordinary differential equations:

$$\mathbf{y}'(x) = \mathbf{f}[x, \mathbf{y}(x)] \quad \text{with } \mathbf{y}(x_0) \text{ known.} \tag{1}$$

The system may be solved using an embedded Runge-Kutta (RK) process with formulae of orders q and p $(q > p)$ of the following form:

$$\hat{\mathbf{y}}_{n+1} = \hat{\mathbf{y}}_n + h_n \hat{\boldsymbol{\Phi}}(x_n, \hat{\mathbf{y}}_n, h_n), \qquad \mathbf{y}_{n+1} = \hat{\mathbf{y}}_n + h_n \boldsymbol{\Phi}(x_n, \hat{\mathbf{y}}_n, h_n) \tag{2}$$

where

$$\hat{\boldsymbol{\Phi}}(x_n, \hat{\mathbf{y}}_n, h_n) = \sum_{i=1}^{s} \hat{b}_i \mathbf{g}_i, \qquad \boldsymbol{\Phi}(x_n, \hat{\mathbf{y}}_n, h_n) = \sum_{i=1}^{s} b_i \mathbf{g}_i,$$

$$\mathbf{g}_i = \mathbf{f}\left(x_n + c_i h_n, \hat{\mathbf{y}}_n + h_n \sum_{j=1}^{i-1} a_{ij} \mathbf{g}_j \right), \qquad i = 1, 2, \cdots, s.$$

The latter summation is taken as zero when $i = 1$, $c_i = \sum_{j=1}^{i-1} a_{ij}$, $x_{n+1} = x_n + h_n$, $h_n = \theta(x_n)h$, $0 < \theta(x) \leqq 1$, $\hat{\mathbf{y}}_0 = \mathbf{y}(x_0)$ and $\hat{\mathbf{y}}_n$ and $\mathbf{y}_n$ denote the approximations to the true solution $\mathbf{y}(x_n)$ from the formulae of orders q and p, respectively, at the meshpoints x_n, $n = 0, 1, \cdots, N$. It should be noted that the embedded process here is applied in local extrapolation, or higher-order, mode [1], i.e., the qth order approximation $\hat{\mathbf{y}}_n$ is used as the initial value for the $(n+1)$th step. Assuming appropriate smoothness of $\mathbf{f}$, the local truncation error, $\hat{\mathbf{t}}_{n+1}$, at x_{n+1} of the RK q process may be written [2]:

$$\begin{aligned} \hat{\mathbf{t}}_{n+1} &= \mathbf{y}(x_n) + h_n \hat{\boldsymbol{\Phi}}[x_n, \mathbf{y}(x_n), h_n] - \mathbf{y}(x_{n+1}), \\ &= \sum_{i=q}^{q+r} h_n^{i+1} \hat{\boldsymbol{\Phi}}_i[x_n, \mathbf{y}(x_n)] + O(h_n^{q+r+2}) \end{aligned}$$

where

$$\hat{\boldsymbol{\Phi}}_i[x, \mathbf{y}(x)] = \sum_{j=1}^{n_{i+1}} \hat{\tau}_j^{(i+1)} \mathbf{F}_j^{(i+1)}[x, \mathbf{y}(x)], \qquad i = 0, 1, \cdots, q+r,$$

* Received by the editors July 11, 1988; accepted for publication (in revised form) January 12, 1989.
† Numerical Computation Research Group, Department of Mathematics and Statistics, Teesside Polytechnic, Middlesbrough, Cleveland TS1 3BA, United Kingdom.

are termed error functions and $\mathbf{F}_j^{(i+1)}, j=1, 2, \cdots, n_{i+1}$, are the elementary differentials of order $i+1$ of $\mathbf{f}$ [3]. A similar expression holds for $\mathbf{t}_{n+1}$, the local truncation error of the RK p. The error coefficients (τ) are functions of the RK parameters (a, b, c) [4] chosen so that

$$\begin{aligned} \hat{\tau}_j^{(i+1)} &= 0, \quad i=0, 1, \cdots, q-1, \\ \tau_j^{(i+1)} &= 0, \quad i=0, 1, \cdots, p-1, \end{aligned} \qquad j=1, 2, \cdots, n_{i+1}.$$

The number of stages, s (the number of times $\mathbf{f}$ is evaluated at each step of the integration), is usually kept as small as possible. Some embedded formulae make use of the FSAL idea [5] in which the last evaluation at any step is the same as the first at the next step. For a formula employed in local extrapolation mode this requires

$$c_s = 1, \quad \hat{b}_s = 0, \quad \text{and} \quad a_{sj} = \hat{b}_j, \quad j=1, 2, \cdots, s-1. \tag{3}$$

The global error in $\hat{\mathbf{y}}$ at $x = x_n$ is defined as $\boldsymbol{\varepsilon}_n = \hat{\mathbf{y}}_n - \mathbf{y}(x_n)$, and it is in this that most users of integration packages are interested. In practice, however, most packages attempt to control the local error or defect [6], with the hope that this has a similar effect on the global error. It is also desirable that a tolerance reduction causes a proportional reduction in the global error.

2. Local error estimation. From the embedded process (2) an estimate $\boldsymbol{\delta}_{n+1} = \mathbf{y}_{n+1} - \hat{\mathbf{y}}_{n+1}$ of the local truncation error $\mathbf{t}_{n+1}$ in the pth order formula can be obtained that may be used to control the stepsize h_n. The formula used by Hull et al. [7],

$$h_{n+1} = 0.9 h_n \{\text{TOL}/\text{EST}\}^{1/(p+1)} \tag{4}$$

where $\text{EST} = \|\boldsymbol{\delta}_{n+1}\|_\infty$ and TOL is the maximum allowable local error, can be used for the control of error per step. An error per unit step test is sometimes used and TOL may be based on absolute error, relative error or mixed error. Shampine [8] gives a good account of the various possibilities. It is easily shown [9] that for the embedded RK$q(p)(q>p)$ defined by (2)

$$\boldsymbol{\delta}_{n+1} = \mathbf{t}_{n+1} - \hat{\mathbf{t}}_{n+1} + O(h_n^{p+q+1}) \quad \text{and so}$$

$$\mathbf{t}_{n+1} - \boldsymbol{\delta}_{n+1} = h_n^{q+1} \sum_{j=1}^{n_{q+1}} \hat{\tau}_j^{(q+1)} \mathbf{F}_j^{(q+1)}[x_n, \mathbf{y}(x_n)] + O(h_n^{q+2}). \tag{5}$$

However, the global error is usually regarded as more important from the practical point of view. For test purposes we normally compute

$$\max \|\boldsymbol{\varepsilon}_n\|_\infty = \mu, \quad \text{say},$$

where all steps and variables are considered. If two RK processes yield the same μ, then the one costing least is deemed more efficient. Alternatively, of two methods costing the same, the one giving best accuracy (smallest μ) is most efficient.

Since it is nearly impossible to achieve either of the above comparisons directly, we have drawn efficiency curves for a range of problems and RK algorithms. A particular problem is solved for a range of tolerances (TOL) with different RK methods; μ against cost is plotted in each case and linear interpolations yield the curves. Although computational cost is easily determined, it is a machine-dependent feature. A more convenient measure of cost is the number of evaluations of $\mathbf{f}$ particularly when comparing different RK methods. It will be clear that these evaluations will dominate the computational cost, particularly when $\mathbf{f}$ is expensive. This measure of cost would not be appropriate if comparing RK methods with linear multistep methods that may

require less function evaluations but may have a higher overall cost [10]. Thus we prefer to plot global error against function evaluations to form efficiency curves for each RK method.

We define the following criteria for a "good" RK embedded pair:

(i) $A^{(q+1)} = \|\hat{\boldsymbol{\tau}}^{(q+1)}\|$ should be as small as possible to reduce the local truncation errors of the higher-order formulae. Practical experience [1], [11], [12] shows that global errors tend to be reduced if this is so.

(ii) $C^{(p+2)} = \|\boldsymbol{\tau}^{(p+2)} - \hat{\boldsymbol{\tau}}^{(p+2)}\| / \|\boldsymbol{\tau}^{(p+1)}\|$ should be made as small as possible subject to (i), thus ensuring dominance of the principal term of $\boldsymbol{\delta}_{n+1}$. The step control formula (4) depends on $\boldsymbol{\delta}_{n+1}$ being $O(h^{p+1})$.

(iii) $B^{(p+2)} = \|\boldsymbol{\tau}^{(p+2)}\| / \|\boldsymbol{\tau}^{(p+1)}\|$ should be small enough to make the two formulae sufficiently distinct. Practical results indicate that the dominance of the principal term of $\mathbf{t}_{n+1}$ leads to well "tuned" formulae in which the global error achieved is not wildly different to the local error tolerance specified.

(iv) The parameters c_i, a_{ij}, $\hat{b}_i$, b_i should not be too large so as to avoid possible rounding effects.

(v) $\tau_j^{(p+1)} \neq 0$, $j = 1, 2, \cdots, n_{p+1}$, thus ensuring that the "near quadrature" problem discussed by Shampine [13] and Prince and Dormand [11] does not arise.

(vi) The regions of absolute stability should be as large as possible subject to (i)-(v).

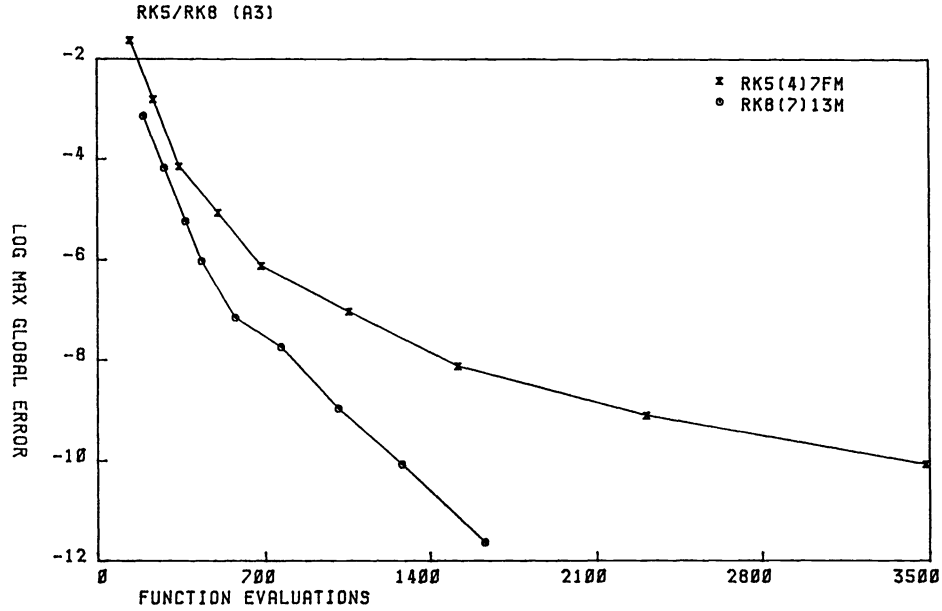
Although a large value of $\tau_j^{(p+1)}$ might imply (ii) and (iii) being satisfied, practical results indicate that a poorly "tuned" formula is obtained. Of course an enlarged value of $\tau_j^{(p+1)}$ is usually associated with a large $\tau_j^{(p+2)}$. It will be apparent that criterion (i) being satisfied tends to reduce the principal terms in (5). If $q = p+2$, then the principal terms in (5) are $O(h^{q+2})$ and B and C are identical. Two particularly efficient embedded pairs developed following the previous criteria are the RK5(4)7FM [9], [1] and the RK8(7)13M of [11].

The efficiency curve comparison is, of course, viable for methods with differing orders and numbers of stages. It is often assumed that a low-order formula should be chosen in preference to a high-order formula when lax tolerances are appropriate. Consequently a user may prefer the RK5(4)7FM to the RK8(7)13M when modest accuracy or only few steps are required. The eighth-order formula is certainly going to be more efficient for a small enough tolerance since its error decreases more rapidly with h than is the case with an RK5. Our tests [11] suggest that the RK8 is better than a fifth-order method almost always (Fig. 1). Numerical tests show that the Hull stepsize formula (4) is not always particularly good for the large stepsizes that arise from an RK8 but even with a high proportion of rejected steps it is very competitive.

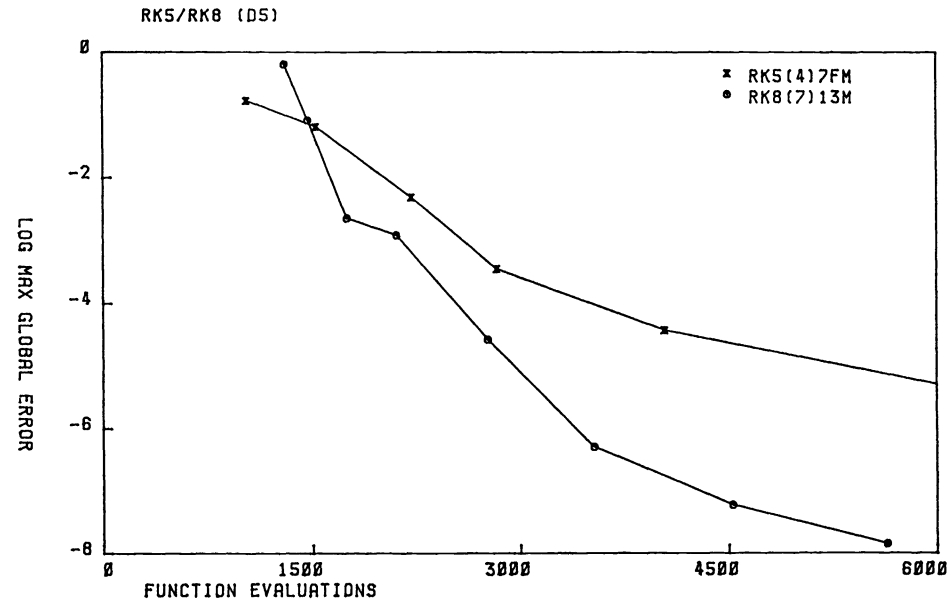
3. Global error estimation. We are concerned here with global error estimation based on the technique employed by Zadunaisky [14], [15] and on the improvement discussed by Skeel [16] and Peterson [17]. The Zadunaisky technique for estimation of $\boldsymbol{\varepsilon}_n$ is based on the construction of a neighbouring problem (NP), with known true solution, which is "close" to the main problem (MP) (1). Consider the form of NP discussed in [2] and [4]:

$$\begin{aligned} &\mathbf{y}_h'(x) = \mathbf{f}_h[\mathbf{y}_h(x)] = \mathbf{f}[\mathbf{y}_h(x)] + \mathbf{d}_h(x), \\ &\mathbf{y}_h(x_0) = \mathbf{y}(x_0) = \mathbf{P}(x_0), \\ &\mathbf{d}_h(x) = \mathbf{P}'(x) - \mathbf{f}[\mathbf{P}(x)], \end{aligned} \tag{6}$$

whose solution is $\mathbf{y}_h(x) = \mathbf{P}(x)$. $\mathbf{P}(x)$ was chosen as a piecewise polynomial function



(a)



(b)

FIG. 1. *Efficiency of* RK5 *and* RK8. (a) *Problem* A3. (b) *Problem* D5.

that interpolated the numerical solution of (1) in subintervals of $[x_0, x_N]$. Each subinterval or block was of size m, i.e., consisted of $m+1$ points, and three cases were discussed: (i) Interpolation of $\mathbf{y}_i$ values; (ii) Interpolation of $\mathbf{f}(\mathbf{y}_i)$ values; and (iii) Hermite interpolation using both the $\mathbf{y}_i$ and $\mathbf{f}(\mathbf{y}_i)$ values. Zadunaisky's basic idea was to integrate (6) with the same RK q formula and same step sequence as for the MP producing numerical solution $\mathbf{y}_{hn}$, approximating $\mathbf{y}_h(x_n)$, and hence to use $\boldsymbol{\varepsilon}_{hn} = \mathbf{y}_{hn} - \mathbf{y}_h(x_n)$ as an estimate of $\boldsymbol{\varepsilon}_n$. Frank [18] has analysed the estimation process for case (i) with constant step and further considerations can be found in [2], [4], [9],

[16], and [17]. Valid asymptotic estimates of $\boldsymbol{\varepsilon}_n$ are obtained if $\mathbf{E}_n = \boldsymbol{\varepsilon}_{hn} - \boldsymbol{\varepsilon}_n$ is $O(h^u)$ where $u > q$. In case (i) the following bound on the defect, $\mathbf{d}_h(x)$, was obtained in [2]:

$$\|\mathbf{d}_h^{(j)}(x)\|_{x\in[x_0,x_N]} \leqq \begin{cases} N_j h^{\min[q,q+r-j,m-j]}, & 0 \leqq j \leqq m-1, \\ N_j, & j \geqq m \end{cases}$$

where $\mathbf{f}$ and $\mathbf{d}_h$ are $(q+r)$th order differentiable and Lipschitz continuous, and this leads to the following bound on $\mathbf{E}_n$:

$$\max_{0\leqq n\leqq N} \|\mathbf{E}_n\| \leqq \begin{cases} Ah^q, & m \leqq q, \\ Ah^{\min[m,q+r,2q]}, & m > q \end{cases} \tag{7}$$

where A is independent of h. The corresponding results for cases (ii) and (iii), respectively, are

$$\|\mathbf{d}_h^{(j)}(x)\|_{x\in[x_0,x_N]} \leqq \begin{cases} N_j h^{\min[q,q+r+1-j,m+1-j]}, & 0 \leqq j \leqq m, \\ N_j, & j \geqq m+1 \end{cases}$$

and

$$\|\mathbf{d}_h^{(j)}(x)\|_{x\in[x_0,x_N]} \leqq \begin{cases} N_j h^{\min[q-j,2m+1-j]}, & 0 \leqq j \leqq 2m, \\ N_j, & j \geqq 2m+1, \end{cases}$$

leading to

$$\max_{0\leqq n\leqq N} \|\mathbf{E}_n\| \leqq \begin{cases} Ah^q, & m \leqq q-1, \\ Ah^{\min[m+1,q+r+1,2q]}, & m \geqq q \end{cases} \tag{8}$$

and

$$\max_{0\leqq n\leqq N} \|\mathbf{E}_n\| \leqq Ah^q \quad \text{irrespective of } m. \tag{9}$$

These bounds on $\|\mathbf{E}_n\|$ indicate that valid asymptotic error estimation is possible if $m > q$ in case (i) and if $m \geqq q$ in case (ii), but that it will not generally be possible in case (iii). Practical results over a wide range of problems have confirmed the validity of these results. Further investigations in [2] and [4] have shown that the order of $\|\mathbf{E}_n\|$ for $q > 1$ was governed by the expression

$$\sum_{i=q+1}^{q+r} h_n^{i-1} \sum_{j=1}^{n_i} \hat{\tau}_j^{(i)} \{\mathbf{F}_{hj}^{(i)}[\mathbf{y}(x_n)] - \mathbf{F}_j^{(i)}[\mathbf{y}(x_n)]\} \tag{10}$$

and depending on m and the resulting order of $\mathbf{d}_h$ and its derivatives, only certain of the $\hat{\tau}_j^{(i)}$ appear in the leading terms of (10). So by choosing a special RK process with appropriate $\hat{\tau}_j^{(i)}$ zero the order of $\mathbf{E}_n$ can be improved over that predicted from (7), (8), and (9). In case (iii) these special processes will allow valid asymptotic estimation. Tests with these particular processes confirm the superiority of Hermite type interpolation and also indicate the preference for two-term estimation where $\|\mathbf{E}_n\|$ is $O(h^{q+2})$. One drawback of this estimation technique occurs when high-order RK processes are used since larger values of m are necessary, which results in high-degree polynomial interpolation and this can be a serious problem particularly in cases (i) and (ii) when variable step sizes are used. Useful estimation with high-order processes has been found only to be practical for stringent tolerances. These findings have been confirmed by Peterson [17] who in addition to considering the Zadunaisky approach using (6) also has investigated the technique of "Solving for the Correction" that has been discussed by Skeel in [16] and is concerned with the differential system

$$\begin{aligned} \boldsymbol{\varepsilon}_h'(x) &= \bar{\mathbf{f}}[\boldsymbol{\varepsilon}_h(x)] = \mathbf{P}'(x) - \mathbf{f}[\mathbf{P}(x) - \boldsymbol{\varepsilon}_h(x)], \\ \boldsymbol{\varepsilon}_h(x_0) &= \boldsymbol{\varepsilon}_{h0} = \mathbf{P}(x_0) - \mathbf{y}(x_0), \end{aligned} \tag{11}$$

whose solution is $\boldsymbol{\varepsilon}_h(x) = \mathbf{P}(x) - \mathbf{y}(x)$. The function $\mathbf{P}(x)$ can be chosen as in the

Zadunaisky cases. The previous system is integrated using an RK formula of order $\bar{q}$ giving numerical solution ε_{hn} that can then be used to estimate ε_n. Usually $\bar{q}=q$ but this formula need not be the same as the main RK q used in the integration of (1). Peterson [17] has analysed the technique for the previous choices of interpolants and has shown that the same orders of estimation are possible as in the Zadunaisky approach. The use of (1) with (11) as opposed to (1) with (6) has the advantages that it costs only $2s$ **f** evaluations per step compared to $3s$ and that it is not necessary to integrate (11) with the same RK process used for (1). So in addition to being cheaper the error correction approach allows greater flexibility with regard to the use of the various RK formulae. Thus we can integrate (1) with *any* RK $q(p)$ process, form $\mathbf{P}(x)$, preferably using the Hermite approach of case (iii), and then integrate (11) with a special RK $\bar{q}$ formula in which the appropriate $\bar{\tau}_j^{(i)}$ are zero. This means that the efficiency of the integration of the MP is not compromised as is the case when using the Zadunaisky approach.

4. Dense output. In recent years much attention [12], [19]-[23] has been given to the development of RK processes that produce numerical solutions for $\mathbf{y}(x)$ at non-meshpoints, a capability that enhances the practicality of any variable steplength integrator. One way of achieving this is to add a third RK formula to the embedded pair (called a *dense* formula) that will be used to integrate (1) from x_n with a step of size, σh_n, where normally $0<\sigma<1$. Following Dormand and Prince [23] the dense formula may be written

$$\mathbf{y}^*_{n+\sigma}=\hat{\mathbf{y}}_n+\sigma h_n\mathbf{\Phi}^*(x_n,\hat{\mathbf{y}}_n,\sigma h_n) \tag{12}$$

where

$$\mathbf{\Phi}^*(x_n,\hat{\mathbf{y}}_n,\sigma h_n)=\sum_{i=1}^{s^*} b_i^*\,\mathbf{g}_i^*,$$

$$\mathbf{g}_i^*=\mathbf{f}\left(x_n+c_i^*\sigma h_n,\hat{\mathbf{y}}_n+\sigma h_n\sum_{j=1}^{i-1}a_{ij}^*\mathbf{g}_j^*\right),\qquad i=1,2,\cdots,s^*.$$

This formula, of order p^*, will produce estimates $\mathbf{y}^*_{n+\sigma}$ of $\mathbf{y}(x_n+\sigma h_n)$, $n=0,1,\cdots,N-1$ and ideally we would like $s^*=s$ and $\mathbf{g}_i^*=\mathbf{g}_i$, $i=1,2,\cdots,s^*$, so that no extra function evaluations are necessary. This may be possible depending on q, p, s, and p^*. To satisfy the RK p^* equations of condition it may be necessary for s^* to be greater than s so that extra Runge-Kutta parameters are available. Common function evaluations occur if

$$\begin{aligned} &c_i^*=c_i/\sigma,\qquad i=1,2,\cdots,s_m,\\ &a_{ij}^*=a_{ij}/\sigma,\qquad i=2,3,\cdots,s_m,\quad j=1,2,\cdots,i-1 \end{aligned} \tag{13}$$

where $s_m=\max(s,s^*)$. It seems natural to require that $\mathbf{y}^*_{n+\sigma}$ and $\hat{\mathbf{y}}_n$ should have the same global accuracy and this requires $\varepsilon^*_{n+\sigma}=\mathbf{y}^*_{n+\sigma}-\mathbf{y}(x_n+\sigma h_n)$ to be $O(h^q)$. As shown in [23] this requires $p^*\geqq q-1$. The embedded pair together with the dense formula constitute an RK triple using s_m stages such that if $s_m>s$ then $\hat{b}_i=b_i=0$, $i=s+1,s+2,\cdots,s_m$. The RK p^* equations of condition now involve the parameter σ, and assuming (13) it is important to keep the c_i and a_{ij} independent of σ to avoid the $\mathbf{g}_i$ being dependent on σ. As mentioned by Shampine [20] continuity of $\mathbf{y}^*_{n+\sigma}$, as a function of σ, is desirable. This has also been discussed by Dormand and Prince [23], [24] who have shown that the function $\mathbf{y}^*_{n+\sigma}$ is C^1 continuous if

$$\begin{aligned} &b_i^*(\sigma=1)=\hat{b}_i,\qquad b_i^*(\sigma=0)=\delta_{i1},\\ &\hat{b}_i+\frac{db_i^*}{d\sigma}(\sigma=1)=\delta_{is} \end{aligned} \tag{14}$$

where

$$\delta_{ij} = \begin{cases} 0, & i \neq j, \\ 1, & i = j, \end{cases} \quad i, j = 1, 2, \cdots, s_m.$$

Two methods of implementing dense output solutions are possible:

(A) If the RK p^* equations can be satisfied for all $\sigma \in [0, 1]$, the $b_i^*(\sigma)$ may be computed for any desired output point and hence (12) may be applied directly.

(B) An interpolating polynomial based on the endpoints and some intermediate points (possibly including derivatives) obtained from (12) can be constructed for any step interval. As shown in [23] the two modes are equivalent if the degree of the interpolating polynomial in mode (B) is at least $q-1$. The RK5(4)7FM [9], [1] has been modified and enhanced with a fourth-order dense output formula in no extra function evaluations [12], [23], and recently we have developed a new RK6(5)9FM process with a 10-stage dense output formula. We have also reconsidered eighth-order processes and have developed a RK8(6)12M process that is competitive when compared to the RK8(7)13M of [11]. As a normal integrator this 8(6) does not use (3), but using this 13th FSAL evaluation and three more we have obtained a seventh-order dense formula.

5. Runge–Kutta–Nystrom processes. The ideas and strategies of the previous sections can also be applied to the special second-order system:

$$\mathbf{y}''(x) = \mathbf{f}[x, \mathbf{y}(x)] \quad \text{with } \mathbf{y}(x_0) \text{ and } \mathbf{y}'(x_0) \text{ known.} \tag{15}$$

The situation is deemed special because $\mathbf{f}$ is assumed independent of $\mathbf{y}'(x)$. Numerical solution of (15) is possible by reducing the system to one of first order, (1) and then applying (2). However, it is generally more efficient [9], [25], [26] to apply directly a Runge-Kutta-Nystrom (RKN) technique that consists of formulae of orders q and p of the following form:

$$\begin{aligned} \hat{\mathbf{y}}_{n+1} &= \hat{\mathbf{y}}_n + h_n \hat{\boldsymbol{\Phi}}(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, h_n), \\ \hat{\mathbf{y}}'_{n+1} &= \hat{\mathbf{y}}'_n + h_n \hat{\boldsymbol{\Phi}}'(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, h_n), \\ \mathbf{y}_{n+1} &= \hat{\mathbf{y}}_n + h_n \boldsymbol{\Phi}(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, h_n), \\ \mathbf{y}'_{n+1} &= \hat{\mathbf{y}}'_n + h_n \boldsymbol{\Phi}'(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, h_n) \end{aligned} \tag{16}$$

where

$$\hat{\boldsymbol{\Phi}}(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, h_n) = \hat{\mathbf{y}}'_n + h_n \sum_{i=1}^{s} \hat{b}_i \mathbf{g}_i,$$

$$\hat{\boldsymbol{\Phi}}'(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, h_n) = \sum_{i=1}^{s} \hat{b}'_i \mathbf{g}_i,$$

$$\boldsymbol{\Phi}(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, h_n) = \hat{\mathbf{y}}'_n + h_n \sum_{i=1}^{s} b_i \mathbf{g}_i,$$

$$\boldsymbol{\Phi}'(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, h_n) = \sum_{i=1}^{s} b'_i \mathbf{g}_i,$$

$$\mathbf{g}_i = \mathbf{f}\left(x_n + c_i h_n, \hat{\mathbf{y}}_n + c_i h_n \hat{\mathbf{y}}'_n + h_n^2 \sum_{j=1}^{i-1} a_{ij} \mathbf{g}_j \right), \qquad i = 1, 2, \cdots, s$$

where

$$\frac{c_i^2}{2}=\sum_{j=1}^{i-1} a_{ij}, \quad \hat{\mathbf{y}}_0=\mathbf{y}(x_0), \quad \hat{\mathbf{y}}'_0=\mathbf{y}'(x_0), \quad \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n,$$

$\mathbf{y}_n$ and $\mathbf{y}'_n$ denote approximations to the true solution $\mathbf{y}(x_n)$ and $\mathbf{y}'(x_n)$ at the point $x=x_n$. The notation is similar to that of § 1, where local extrapolation mode is assumed. Assuming appropriate smoothness of $\mathbf{f}$, the local truncation errors, $\mathbf{t}_{n+1}$ and $\mathbf{t}'_{n+1}$, at x_{n+1} of an RKN process may be written

$$\mathbf{t}_{n+1}=\sum_{i=1}^{q+r} h_n^{i+1} \sum_{j=1}^{n_i} \tau_j^{(i+1)} \mathbf{F}_j^{(i)}[x_n, \mathbf{y}(x_n), \mathbf{y}'(x_n)]+O(h_n^{q+r+2}),$$

and

$$\mathbf{t}'_{n+1}=\sum_{i=0}^{q+r} h_n^{i+1} \sum_{j=1}^{n_{i+1}} \tau_j'^{(i+1)} \mathbf{F}_j^{(i+1)}[x_n, \mathbf{y}(x_n), \mathbf{y}'(x_n)]+O(h_n^{q+r+2})$$

where $\mathbf{F}_j^{(i)}$, $j=1, 2, \cdots, n_i$, are the elementary differentials of $\mathbf{f}$ similar to those of the normal RK process [3]. The error coefficients τ are functions of the RKN parameters, a, b, and c and are chosen so that

$$\begin{aligned} &\hat{\tau}_j^{(i+1)}=0, && i=1, 2, \cdots, q-1, \\ &\tau_j^{(i+1)}=0, && i=1, 2, \cdots, p-1, \end{aligned} \qquad j=1, 2, \cdots, n_i,$$

$$\begin{aligned} &\hat{\tau}_j'^{(i+1)}=0, && i=0, 1, \cdots, q-1, \\ &\tau_j'^{(i+1)}=0, && i=0, 1, \cdots, p-1. \end{aligned} \qquad j=1, 2, \cdots, n_{i+1}.$$

Tables giving the values of n_{i-1} and expressions for the $\tau_j^{(i)}$ can be found in [27]. The global errors in $\hat{\mathbf{y}}$ and $\hat{\mathbf{y}}'$ at $x=x_n$ are defined as $\boldsymbol{\varepsilon}_n=\hat{\mathbf{y}}_n-\mathbf{y}(x_n)$ and $\boldsymbol{\varepsilon}'_n=\hat{\mathbf{y}}'_n-\mathbf{y}'(x_n)$ and it may be shown that provided $\mathbf{t}_{n+1}+\mathbf{t}'_{n+1}$ is $O(h^{q+1})$ and $\boldsymbol{\varepsilon}_0+\boldsymbol{\varepsilon}'_0$ is $O(h^q)$, then $\boldsymbol{\varepsilon}_n+\boldsymbol{\varepsilon}'_n$ is $O(h^q)$. From the embedded process (16) estimates $\boldsymbol{\delta}_{n+1}=\mathbf{y}_{n+1}-\hat{\mathbf{y}}_{n+1}$ and $\boldsymbol{\delta}'_{n+1}=\mathbf{y}'_{n+1}-\hat{\mathbf{y}}'_{n+1}$ of the local truncation errors, $\mathbf{t}_{n+1}$ and $\mathbf{t}'_{n+1}$, respectively, in the pth-order formula can be obtained that may be used to control the step size. As in § 2, (4) may be used to do this where in this case EST can be based on estimates of either $\mathbf{t}_{n+1}$ or $\mathbf{t}'_{n+1}$, or both. We prefer EST $=\max\{\|\boldsymbol{\delta}_{n+1}\|_\infty, \|\boldsymbol{\delta}'_{n+1}\|_\infty\}$. The strategy we adopt for the development of efficient RKN embedded processes is analogous to that of § 2 and leads to the similar criteria for a "good" RKN embedded pair (see [26]-[28]).

Non-meshpoint solutions can be obtained from dense RKN formulae that are used to integrate from x_n with stepsize σh_n. The dense formulae may be written

$$\begin{aligned} \mathbf{y}^*_{n+\sigma}&=\hat{\mathbf{y}}_n+\sigma h_n \boldsymbol{\Phi}^*(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, \sigma h_n), \\ \mathbf{y}'^*_{n+\sigma}&=\hat{\mathbf{y}}'_n+\sigma h_n \boldsymbol{\Phi}'^*(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, \sigma h_n) \end{aligned} \tag{17}$$

where

$$\boldsymbol{\Phi}^*(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, \sigma h_n)=\hat{\mathbf{y}}'_n+\sigma h_n \sum_{i=1}^{s^*} b_i^* \mathbf{g}_i^*,$$

$$\boldsymbol{\Phi}'^*(x_n, \hat{\mathbf{y}}_n, \hat{\mathbf{y}}'_n, \sigma h_n)=\sum_{i=1}^{s^*} b_i'^* \mathbf{g}_i^*,$$

$$\mathbf{g}_i^*=\mathbf{f}\left(x_n+c_i^*\sigma h_n, \hat{\mathbf{y}}_n+c_i^*\sigma h_n \hat{\mathbf{y}}'_n+\sigma^2 h_n^2 \sum_{j=1}^{i-1} a_{ij}^* \mathbf{g}_j^*\right), \qquad i=1, 2, \cdots, s^*.$$

If $s^* = s$ and $\mathbf{g}_i^* = \mathbf{g}_i$, $i = 1, 2, \cdots, s^*$, then no extra function evaluations are necessary to compute approximations for $\mathbf{y}(x_n + \sigma h_n)$ and $\mathbf{y}'(x_n + \sigma h_n)$. This will only be possible for certain values of q, p, and p^*. Other than for low-order RKN processes it will usually be necessary for s^* to be greater than s to satisfy the RKNp^* equations of condition. If the following relations are satisfied, then common function evaluations ($\mathbf{g}_i$) are guaranteed:

$$c_i^* = c_i/\sigma, \qquad i = 1, 2, \cdots, s_m,$$

$$a_{ij}^* = a_{ij}/\sigma^2, \qquad i = 2, 3, \cdots, s_m, \quad j = 1, 2, \cdots, i-1,$$

where $s_m = \max(s, s^*)$. As in the RK case, the most obvious choice for p^* is either $q-1$ or q. The embedded $q(p)$ pair together with the dense formulae constitute an RKN triple using s_m stages such that if $s_m > s$, then $\hat{b}_i = \hat{b}_i' = b_i = b_i' = 0$, $i = s+1, s+2, \cdots, s_m$. Both the b_i^* and $b_i'^*$ will be functions of σ. The desirable continuity conditions on $\mathbf{y}_{n+\sigma}^*$ and $\mathbf{y}_{n+\sigma}'^*$ are given in Table 1.

As in § 4, there are two methods of implementing dense output solutions:

(A) If the RKNp^* equations (for $\mathbf{y}$ and $\mathbf{y}'$) can be satisfied for all $\sigma \in [0, 1]$, the $b_i^*(\sigma)$ and $b_i'^*(\sigma)$ may be computed for any desired output point and hence (17) may be applied directly.

TABLE 1

Continuity	$\mathbf{y}_{n+\sigma}^*$	$\mathbf{y}_{n+\sigma}'^*$
C^0	$b_i^*(1) = \hat{b}_i$	$b_i'^*(1) = \hat{b}_i'$
		$b_i'^*(0) = \delta_{i1}$
C^1	$2\hat{b}_i + \dfrac{db_i^*}{d\sigma}(1) = \hat{b}_i'$	$\hat{b}_i' + \dfrac{db_i'^*}{d\sigma}(1) = \delta_{is}$
	$b_i^*(0) = \delta_{i1}/2$	
C^2	$2\hat{b}_i + 4\dfrac{db_i^*}{d\sigma}(1) + \dfrac{d^2 b_i^*}{d\sigma^2}(1) = \delta_{is}$	

(B) An interpolating polynomial based on values of $\mathbf{y}$, $\mathbf{y}'$, and $\mathbf{f}$ at the meshpoints and some intermediate points (possibly including $\mathbf{f}$ evaluations) obtained from (17) can be constructed for any step interval.

As far as the choice of p^* is concerned, we would argue that the estimates of $\mathbf{y}(x_n + \sigma h_n)$ and $\mathbf{y}'(x_n + \sigma h_n)$ should have the same order global accuracy as $\hat{\mathbf{y}}_n$ and $\hat{\mathbf{y}}_n'$. As shown in [24] this requires $p^* \geqq q-1$ for mode (A) implementation. It is also shown in [24] that mode (B) is equivalent to mode (A) if the degree of the interpolating polynomial is at least q and if the dense formula for $\mathbf{y}$ is locally of order at least q.

Particularly practical processes developed according to the previous criteria are the RKN6(4)6FD of [24], which consists of a 6(4) embedded pair for both $\mathbf{y}$ and $\mathbf{y}'$ together with fifth-order dense formulae for $\mathbf{y}$ and $\mathbf{y}'$ and the RKN12(10)17M of [28], both of which form the basis of the code in [29].

6. Global error estimation using dense output. Part of our recent work has considered the possibility of using dense output points within each integration step to form the polynomial required when using either (6) or (11). These dense points together with the end step values of either or both $\mathbf{y}$ and $\mathbf{f}$ can be used to form the polynomial

$\mathbf{P}(x)$. An immediate advantage of this approach is that the integration can proceed in a single step fashion rather than m steps at a time. The polynomial $\mathbf{P}(x)$ within each step, $[x_n, x_{n+1}]$, is taken to interpolate the points $\mathbf{y}^*_{n+\sigma_k}$, $k=1,2,\cdots,r_1$ and $\mathbf{f}(\mathbf{y}^*_{n+\sigma_k})$, $k=1,2,\cdots,r_2$, where $\sigma_k \in [0,1]$, $\sigma_1=0$, $\sigma_2=1$ and because of (14) and (3)

$$\mathbf{y}^*_{n+\sigma_1}=\hat{\mathbf{y}}_n, \quad \mathbf{y}^*_{n+\sigma_2}=\hat{\mathbf{y}}_{n+1}, \qquad \mathbf{f}(\mathbf{y}^*_{n+\sigma_1})=\mathbf{g}_1, \quad \mathbf{f}(\mathbf{y}^*_{n+\sigma_2})=\mathbf{g}_s.$$

Similar to the analysis of § 3, the following cases are considered: (i) in which $r_2=0$; (ii) in which $r_1=0$; and (iii) in which $r_1\neq 0$ and $r_2\neq 0$. Possibility (ii) is practically less attractive since r_2-2 new function evaluations are necessary. The following results hold in cases (i) or (iii):

$$\|\mathbf{d}_h^{(j)}(x)\|_{x\in[x_0,x_N]} \leqq \begin{cases} N_j h^{\min[q,p^*,m]-j}, & 0\leqq j\leqq m-1, \\ N_j h^{\min[0,q-m+1,p^*-m+1]}, & j\geqq m, \end{cases}$$

and

$$\max_{0\leqq n\leqq N}\|\mathbf{E}_n\| \leqq \begin{cases} Ah^q, & q\geqq m, \\ Ah^{\min[q,p^*]}, & q<m, \end{cases} \tag{18}$$

where $m=r_1+r_2-1$ and it has been assumed that $p^*\geqq q-1$. For case (ii) the corresponding results are

$$\|\mathbf{d}_h^{(j)}(x)\|_{x\in[x_0,x_N]} \leqq \begin{cases} N_j h^{\min[q,p^*+1,m+1]}, & j=0, \\ N_j h^{\min[q,p^*,m]+1-j}, & 1\leqq j\leqq m, \\ N_j h^{\min[0,q-m+1,p^*-m+1]}, & j>m, \end{cases}$$

and

$$\max_{0\leqq n\leqq N}\|\mathbf{E}_n\| \leqq \begin{cases} Ah^q, & q\geqq m, \\ Ah^{\min[q+1,p^*+1]}, & q<m. \end{cases} \tag{19}$$

Consideration of (18) and (19) shows that as for the Hermite case in § 3 it would appear that viable asymptotic global error estimation is not possible in either cases (i) or (iii) but that in case (ii) it is possible if $q<m$ *and* $p^*\geqq q$. Practical tests have confirmed the validity of these results in either situation where (6) or (11) is employed. However, it can be shown that the order of $\mathbf{E}_n$ is still governed by (10), which allows for the construction of special RK processes. Tests with special processes have confirmed this. As in § 3 use of (11) instead of (6) is preferred since it is cheaper and does not compromise the main embedded integration formulae. Use of a special process based on case (iii) with $r_2=2$ appears to be the most attractive from a practical point of view since it requires no extra function evaluations and, assuming the σ independence of the a_{ij} and c_i and that the dense formula satisfies the continuity conditions (14), $\mathbf{P}(x)\equiv\mathbf{y}^*_{n+\sigma}$, for $x\in[x_n, x_{n+1}]$. This, therefore, allows the possiblity of reducing the overheads in the integration of (11) using the RK $\bar{q}$ "estimator" process since $\bar{\mathbf{f}}$ is evaluated at $x_n+\bar{c}_j h_n$, i.e., at $\sigma=\bar{c}_j$, and so $b_i^*(\bar{c}_j)$ and $db_i^*/d\sigma(\bar{c}_j)$ can be predetermined.

7. Dense output test results. In Fig. 2 we show plots of actual global error from the continuous solution of the orbit problems D1 and D5 [7] using the new RK8(6)12M process. The meshpoint global errors are superimposed and suggest that the seventh-order (locally) dense output yields good results, particularly with D5, where the dominance of global errors over local errors is very notable.

Figure 3 shows some efficiency curves relating to dense output global error estimation. Formulae of orders four, five, and six have been allied with special global error estimators to yield two-term estimation. The direct error method (Solving for the

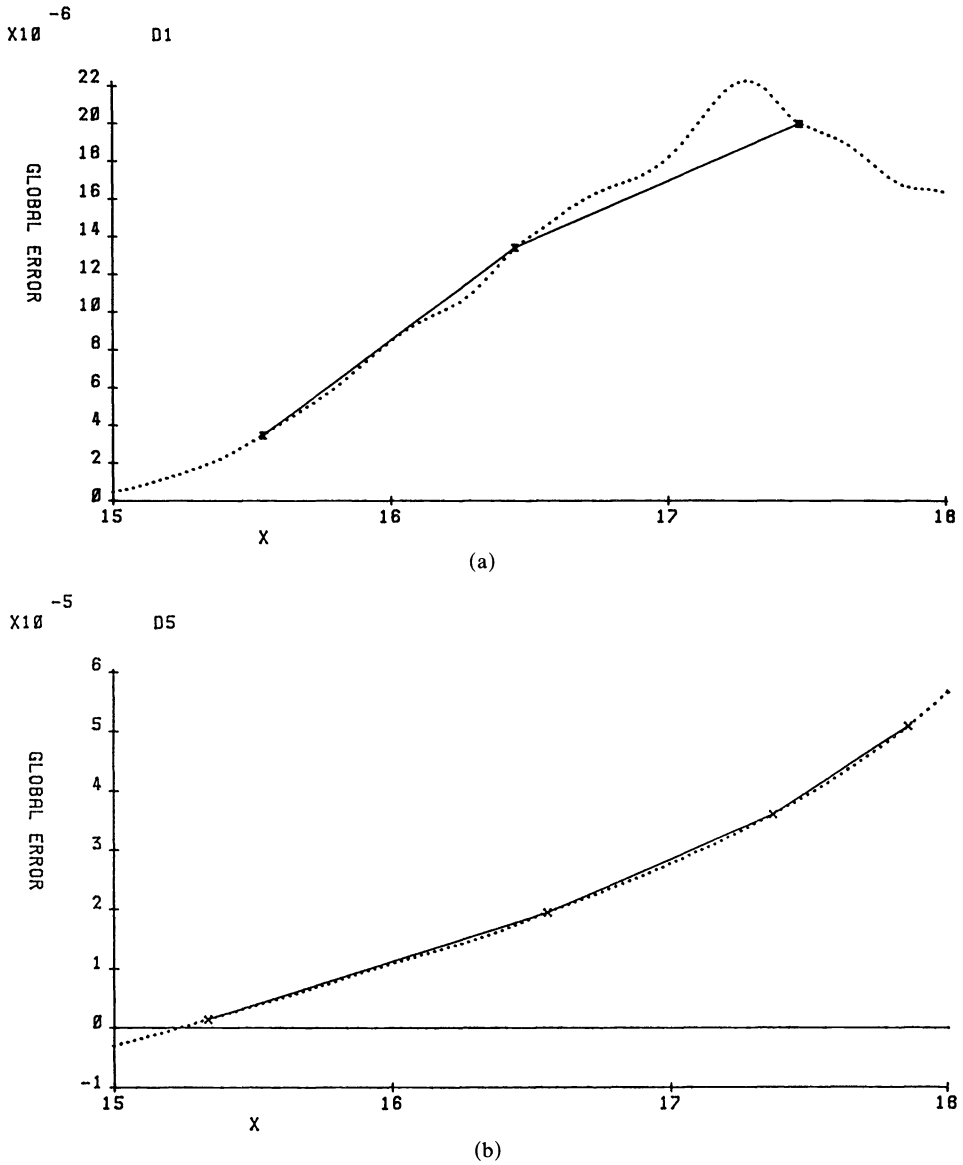


FIG. 2. *Global error of dense solution.* RK8(6)12M *applied to* D1 (a), D5 (b). $x \in [15, 18]$, TOL = 0.00001. · · · · ·, *Dense output (locally seventh order).* ——, *Main solution points (eighth order).*

Correction) has been employed to the problem D5. It is notable that useful global error estimates occur even at comparatively lax tolerances, and it is not unusual for the technique to yield at least four significant figures correct in the error. Unlike the Zadunaisky block method [2] the new method seems to recommend use of the sixth-order formula at all tolerances.

8. Discussion. We have considered above, the development of RK formulae according to various criteria. The most desirable property, from a practical point of view, concerns delivery of a numerical solution of good enough accuracy as cheaply as possible. The provision of dense output is important since it permits solution at a specified point without increasing the number of steps to be computed. The estimation

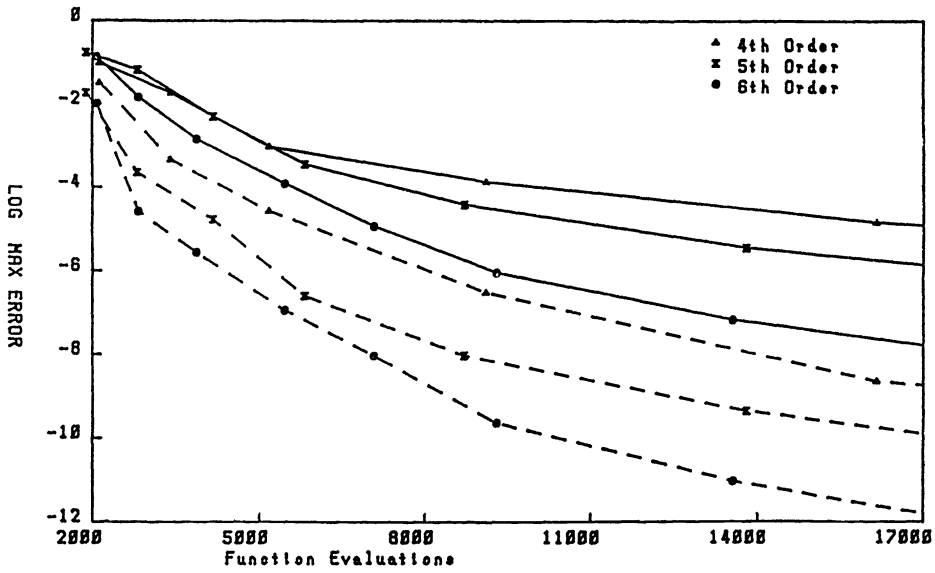


FIG. 3. *Global error efficiency. Problem* D5. o——o, *Global error* ε. o- - -o, *Error of global error estimate* E. *The maximum values of* E *and* ε *over all steps and variables are plotted.*

of global error now seems to be a practical proposition. The extra cost of this, using the "Solving for the Correction" technique, is only about the same as the original solution; this is somewhat cheaper than the step-halving technique used in classical extrapolation. Since two-term estimation can be achieved, the quality of global error estimates is far better than in the Richardson case, and, when the dense output "in-step" interpolant is employed, the computational algorithm is very simple. A general package for solution with global error is presently being considered; this will also aim to estimate global error at dense output solution points.

In the future some of the developments reported above will be extended to higher-order formulae and also directly to second-order equations (RKN).

REFERENCES

[1] J. R. DORMAND AND P. J. PRINCE, *A family of embedded Runge-Kutta formulae*, J. Comput. Appl. Math., 6 (1980), pp. 19-26.

[2] ———, *Global error estimation with Runge-Kutta methods* II, IMA J. Numer. Anal., 5 (1985), pp. 481-497.

[3] J. C. BUTCHER, *Coefficients for the study of Runge-Kutta integration processes*, J. Austral. Math. Soc. Ser., 3 (1963), pp. 185-201.

[4] J. R. DORMAND, R. R. DUCKERS, AND P. J. PRINCE, *Global error estimation with Runge-Kutta methods*, IMA J. Numer. Anal., 4 (1984), pp. 169-184.

[5] J. R. DORMAND AND P. J. PRINCE, *A reconsideration of some embedded Runge-Kutta formulae*, J. Comput. Appl. Math., 15 (1986), pp. 203-211.

[6] W. H. ENRIGHT, *A new error-control for initial value solvers*, Numerical Analysis Report 122, Department of Mathematics, University of Manchester, Manchester, UK, 1986.

[7] T. E. HULL, W. H. ENRIGHT, B. M. FELLEN, AND A. E. SEDGWICK, *Comparing numerical methods of ordinary differential equations*, SIAM J. Numer. Anal., 9 (1972), pp. 603-637.

[8] L. F. SHAMPINE, *Local error control in codes for ordinary differential equations*, Appl. Math. Comput., 3 (1977), pp. 189-210.

[9] P. J. PRINCE, *Runge-Kutta processes and global error estimation*, Ph.D. thesis, CNAA, 1979.

[10] E. HAIRER, S. P. NORSETT, AND G. WANNER, *Solving Ordinary Differential Equations* I, Springer-Verlag, Berlin, New York, 1986.

[11] P. J. PRINCE AND J. R. DORMAND, *High order embedded Runge-Kutta formulae*, J. Comput. Appl. Math., 7 (1981), pp. 67-75.

[12] L. F. SHAMPINE, *Some practical Runge-Kutta formulas*, Math. Comput., 46 (1986), pp. 135-150.

[13] ———, *Quadrature and Runge-Kutta formulas*, Appl. Math. Comput., 2 (1976), pp. 161-171.

[14] P. E. ZADUNAISKY, *A method for the estimation of errors propagated in the numerical solution of a system of ordinary differential equations*, in Proc. International Astronomical Union, 25, Theessoloniki, Academic Press, New York, 1966, pp. 281-287.

[15] ———, *On the estimation of errors propagated in the numerical integration of ordinary differential equations*, Numer. Math., 27 (1976), pp. 21-39.

[16] S. D. SKEEL, *Thirteen ways to estimate global error*, Numer. Math., 48 (1986), pp. 1-20.

[17] P. J. PETERSON, *Global error estimation using defect correction techniques for explicit Runge-Kutta methods*, Tech. Report 192/86, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1986.

[18] R. FRANK, *Schatzungen des globalen diskretisierungsfehlers bei Runge-Kutta methoden*, ISNM, 27 (1975), pp. 45-70.

[19] M. K. HORN, *Fourth- and fifth-order scaled Runge-Kutta algorithms for treating dense output*, SIAM J. Numer. Anal., 20 (1983), pp. 558-568.

[20] L. F. SHAMPINE, *Interpolation for Runge-Kutta methods*, SIAM J. Numer. Anal., 22 (1985), pp. 1014-1027.

[21] W. H. ENRIGHT, K. R. JACKSON, S. P. NORSETT, AND P. G. THOMSEN, *Interpolants for Runge-Kutta formulas*, Tech. Report 180/85, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1985.

[22] I. GLADWELL, L. F. SHAMPINE, L. S. BACA, AND R. W. BRANKIN, *Practical aspects of interpolation in Runge-Kutta codes*, Numerical Analysis Report 102, Department of Mathematics, University of Manchester, Manchester, UK, 1985.

[23] J. R. DORMAND AND P. J. PRINCE, *Runge-Kutta triples*, Comput. Math. Appl., 12A (1986), pp. 1007-1017.

[24] ———, *Runge-Kutta-Nystrom triples*, Comput. Math. Appl., 13 (1987), pp. 937-949.

[25] E. HAIRER, *Méthodes de Nystrom pour l'équation différentiale* $y'' = f(x, y)$, Numer. Math., 27 (1977), pp. 283-300.

[26] J. R. DORMAND AND P. J. PRINCE, *New Runge-Kutta-Nystrom algorithms for simulation in dynamical astronomy*, Celestial Mech., 18 (1978), pp. 223-232.

[27] J. R. DORMAND, M. E. A. EL-MIKKAWY, AND P. J. PRINCE, *Families of Runge-Kutta-Nystrom formulae*, IMA J. Numer. Anal., 7 (1987), pp. 235-250.

[28] ———, *High-order embedded Runge-Kutta-Nystrom formulae*, IMA J. Numer. Anal., 7 (1987), pp. 423-430.

[29] R. W. BRANKIN, J. R. DORMAND, I. GLADWELL, P. J. PRINCE, AND W. L. SEWARD, *A Runge-Kutta-Nystrom code*, ACM ToMS, (1988), pp. 31-40.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 990-999, September 1989
© 1989 Society for Industrial and Applied Mathematics
009

EFFICIENT SPLIT LINEAR MULTISTEP METHODS FOR STIFF ORDINARY DIFFERENTIAL EQUATIONS*

DAVID A. VOSS† AND MARK J. CASPER†

Abstract. A new family of predictor-corrector schemes is designed for the numerical solution of stiff differential systems. Based on split Adams-Moulton formulas through sixth order, members of the new family achieve higher order and possess smaller error constants than corresponding split backward differentiation formulas of the same stepnumber, while maintaining similar stability properties. Some confirmation of this is obtained using a variable step implementation on test problems from the literature.

Key words. stiff ODE, split linear multistep method, $L(\alpha)$-stable

AMS(MOS) subject classification. 65L05

1. Introduction. In this paper we will be concerned with the numerical solution of stiff differential systems of the form

$$y' = f(x, y), \quad y(x_0) = y_0, \quad y \in R^s, \tag{1.1}$$

using new types of split linear multistep methods (SLMMs). Cash [1] has introduced the concept of a SLMM wherein the basic linear multistep method (LMM)

$$\sum_{i=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f_{n+j} \tag{1.2}$$

is rewritten in the split form

$$\sum_{j=0}^{k} \alpha_j y_{n+j} - h \sum_{j=0}^{k-1} \beta_j f_{n+j} = h\theta f(x_{n+k}, \bar{y}_{n+k}) + h(\beta_k - \theta) f(x_{n+k}, y_{n+k}) \tag{1.3}$$

where $\theta \neq 0$, $\theta \neq \beta_k$, and the quantity $\bar{y}_{n+k}$ is obtained using a separate, implicit predictor. By taking the basic LMM to be the widely used class of backward differentiation formulas (BDFs), Cash derives a family of SLMMs of orders up to six possessing stability regions considerably larger than those of the corresponding BDF methods.

As with BDF methods, all of the formulas presented in [1] of stepnumber k possess order k. In [2], Cash has considered finite-difference schemes for parabolic partial differential equations (PDEs) and has mentioned that it would be of interest to see whether it is possible to obtain an L_0-stable SLMM of stepnumber 2 having order three or four. Voss and Casper have addressed this question in [3], and have derived a two-step L_0-stable SLMM of order three and a three-step L_0-stable SLMM of order four. While [3] is concerned with L_0-stable methods for solving semidiscretized parabolic PDEs for which the Jacobian matrix has real eigenvalues, this paper concentrates on stiff systems of ordinary differential equations (ODEs) for which the Jacobian matrix may have eigenvalues anywhere in the negative half of the complex plane. We derive k-step SLMMs of order $k+1$, $k = 1(1)5$, possessing comparable regions of stability to those in [1] by splitting the (basic) Adams-Moulton formulas (AMFs).

* Received by the editors July 26, 1988; accepted for publication (in revised form) December 12, 1988. This research was supported in part by a grant from the University Research Council at Western Illinois University, Macomb, Illinois.

† Department of Mathematics, Western Illinois University, Macomb, Illinois 61455.

2. General formulation. The main idea behind a SLMM is to rewrite the basic LMM in the predictor-corrector form:

Predictor:

$$\bar{\alpha}_k \bar{y}_{n+k} - h\bar{\beta}_k f(x_{n+k}, \bar{y}_{n+k}) = -\sum_{j=0}^{k-1} (\bar{\alpha}_j y_{n+j} - h\bar{\beta}_j f_{n+j}); \tag{2.1}$$

Corrector:

$$\alpha_k y_{n+k} - h(\beta_k - \theta) f(x_{n+k}, y_{n+k}) = -\sum_{j=0}^{k-1} (\alpha_j y_{n+j} - h\beta_j f_{n+j}) + h\theta f(x_{n+k}, \bar{y}_{n+k}) \tag{2.2}$$

where $\theta \neq 0$ and $\beta_k \neq \theta$. It is straightforward to verify that if (1.2) has order p and (2.1) has order $\geqq p-1$, then (2.2) will have order $\geqq p$. Moreover, to preserve the efficiency of the predictor-corrector pair, the predictor will be specifically constructed as in [1] so that if (2.1) is solved for $\bar{y}_{n+k}$ and (2.2) is solved for y_{n+k}, both using a modified Newton iteration scheme, then the coefficient matrix of both iteration schemes will be

$$I - \gamma h \frac{\partial f}{\partial y}, \tag{2.3}$$

that is, we require $\gamma = \bar{\beta}_k = \beta_k - \theta$.

Applying (2.1), (2.2) to the scalar test equation $y' = \lambda y$ where $\mathrm{Re}(\lambda) < 0$ yields the stability equation

$$\pi_s(r, q) \equiv \sum_{j=0}^{k} [(\alpha_j - q\beta_j)(\bar{\alpha}_k - q\bar{\beta}_k) - \theta q(q\bar{\beta}_j - \bar{\alpha}_j)] r^j = 0 \tag{2.4}$$

where $q = \lambda h$. When we denote the stability polynomial of the predictor (2.1) by

$$\bar{\pi}(r, q) = \bar{\rho}(r) - q\bar{\sigma}(r), \tag{2.5}$$

and that of the corrector (1.2) by

$$\pi(r, q) = \rho(r) - q\sigma(r), \tag{2.6}$$

from (2.4) the stability polynomial of the SLMM given in (2.1) and (2.2) is

$$\pi_s(r, q) = (\bar{\alpha}_k - q\bar{\beta}_k)\pi(r, q) + \theta q\bar{\pi}(r, q). \tag{2.7}$$

We assume, without loss of generality, that $\bar{\alpha}_k = 1$. Then with $\bar{\beta}_k = \beta_k - \theta$, (2.7) becomes

$$\pi_s(r, q) = \pi(r, q) - q\pi^*(r, q) \tag{2.8}$$

where

$$\pi^*(r, q) = \rho^*(r) - q\sigma^*(r) \tag{2.9}$$

and

$$\rho^*(r) = (\beta_k - \theta)\rho(r) - \theta\bar{\rho}(r), \tag{2.10a}$$

$$\sigma^*(r) = (\beta_k - \theta)\sigma(r) - \theta\bar{\sigma}(r). \tag{2.10b}$$

A(α)-stability requires that the roots of $\pi_s(r, q)$ be inside the unit circle when q is real and $q \to -\infty$; in addition, $L(\alpha)$-stability requires these roots approach zero as $q \to -\infty$. In this limit, the roots of $\pi_s(r, q)$ approach those of $\sigma^*(r)$, and so it is natural to select

$$\sigma^*(r) = c_k r^k, \tag{2.11}$$

which has all its roots at the origin. The SLMMs described by Cash [1] arise by splitting (basic) BDFs, that is, $\sigma(r)=\beta_k r^k$. With this choice, (2.10b) and (2.11) yield $\bar{\sigma}(r)=(\beta_k-\theta)r^k$ with $\bar{\rho}(r)$ determined so that the predictor (2.1) has order $k-1$. As with standard BDFs, the resulting SLMMs of stepnumber k possess order k and are listed in [1] for $k=1(1)6$ along with their regions of $L(\alpha)$-stability; in the sequel, they will be referred to as SBDF (k, k).

SLMMs of stepnumber k and order $k+1$ possessing comparable regions of $L(\alpha)$-stability arise by splitting (basic) AMFs, that is, $\rho(r)=r^k-r^{k-1}$; the resulting methods will be denoted SAMF $(k, k+1)$. From (2.10b) it follows that

$$\sigma^*(r)=\sum_{j=0}^{k}(\beta_j\bar{\beta}_k-\theta\bar{\beta}_j)r^j, \tag{2.12}$$

and, consequently, (2.11) requires that

$$\beta_j\bar{\beta}_k-\theta\bar{\beta}_j=0, \qquad j=0, 1, \cdots, k-1. \tag{2.13}$$

As an example, consider splitting the two-step third-order AMF

$$y_{n+2}-y_{n+1}=h[\beta_2 f_{n+2}+\beta_1 f_{n+1}+\beta_0 f_n] \tag{2.14}$$

where $\beta_0=-1/12$, $\beta_1=8/12$, and $\beta_2=5/12$. The second-order predictor has the form

$$\bar{y}_{n+2}-(1+a)y_{n+1}+ay_n=h[\bar{\beta}_2\bar{f}_{n+2}+\bar{\beta}_1 f_{n+1}+\bar{\beta}_0 f_n] \tag{2.15}$$

where $\bar{\beta}_2=\beta_2-\theta$, $\bar{\beta}_1=\beta_1-a/2+2\theta$, and $\bar{\beta}_0=\beta_0-a/2-\theta$. Using (2.13), we have

$$\beta_0\bar{\beta}_2-\theta\bar{\beta}_0=0, \tag{2.16a}$$

$$\beta_1\bar{\beta}_2-\theta\bar{\beta}_1=0. \tag{2.16b}$$

Since $\bar{\beta}_2\neq 0$ and $\theta\neq 0$, (2.16) requires $\beta_0\bar{\beta}_1-\beta_1\bar{\beta}_0=0$ or $a=-\frac{4}{3}\theta$. Use of this value of a in (2.16a) results in the quadratic equation

$$48\theta^2+24\theta-5=0 \tag{2.17}$$

with solutions $\theta=-\frac{1}{4}\pm\sqrt{6}/6$. In [3], the choice $\theta=-\frac{1}{4}+\sqrt{6}/6$ is used as it yields a smaller error constant for the predictor with the corresponding SLMM being L_0-stable. However, the choice $\theta=-\frac{1}{4}-\sqrt{6}/6$ gives a SLMM possessing a larger region of $L(\alpha)$-stability. This, along with other k-step methods, $k=1(1)5$, are listed in the next section.

3. Some particular formulas. In this section we list some particular formulas of order less than or equal to six based on standard Adams–Moulton formulas.

SAMF (1, 2).

Predictor:

$$\bar{y}_{n+1}-y_n=h[(\tfrac{1}{2}+\theta)f_n+(\tfrac{1}{2}-\theta)\bar{f}_{n+1}];$$

Corrector:

$$y_{n+1}-y_n=h[\tfrac{1}{2}f_n+(\tfrac{1}{2}-\theta)f_{n+1}]+\theta h\bar{f}_{n+1};$$

L.T.E. of predictor: $\theta h^2 y_n''+O(h^3)$;

L.T.E. of predictor-corrector pair:

$$\theta^2 h^3\frac{\partial f}{\partial y}y_n''-\frac{1}{12}h^3 y_n'''+O(h^4);$$

L-stable for $\theta=-\frac{1}{2}+1/\sqrt{2}$.

SAMF (2, 3).

Predictor:

$$\bar{y}_{n+2}-\left(1-\frac{4}{3}\theta\right)y_{n+1}-\frac{4}{3}\theta y_n$$

$$=h\left[\left(-\frac{1}{12}-\frac{\theta}{3}\right)f_n+\left(\frac{2}{3}+\frac{8}{3}\theta\right)f_{n+1}+\left(\frac{5}{12}-\theta\right)\bar{f}_{n+2}\right];$$

Corrector:

$$y_{n+2}-y_{n+1}=h[-\tfrac{1}{12}f_n+\tfrac{2}{3}f_{n+1}+(\tfrac{5}{12}-\theta)f_{n+2}]+\theta h\bar{f}_{n+2};$$

L.T.E. of predictor: $-(\theta/3)h^3y_n'''+O(h^4)$;
L.T.E. of predictor-corrector pair:

$$-\frac{\theta^2}{3}h^4\frac{\partial f}{\partial y}y_n'''-\frac{1}{24}h^4y_n^{iv}+O(h^5);$$

L-stable for $\theta=-\frac{1}{4}-\sqrt{6}/6$.

SAMF (3, 4).

Predictor:

$$\bar{y}_{n+3}-\left(1-\frac{56}{15}\theta\right)y_{n+2}-\frac{72}{15}\theta y_{n+1}+\frac{16}{15}\theta y_n$$

$$=h\left[\left(\frac{1}{24}+\frac{11}{45}\theta\right)f_n-\left(\frac{5}{24}+\frac{11}{9}\theta\right)f_{n+1}+\left(\frac{19}{24}+\frac{209}{45}\theta\right)f_{n+2}+\left(\frac{9}{24}-\theta\right)\bar{f}_{n+3}\right];$$

Corrector:

$$y_{n+3}-y_{n+2}=h\left[\frac{1}{24}f_n-\frac{5}{24}f_{n+1}+\frac{19}{24}f_{n+2}+\left(\frac{9}{24}-\theta\right)f_{n+3}\right]+\theta h\bar{f}_{n+3};$$

L.T.E. of predictor: $\frac{4}{5}\theta h^4y_n^{iv}+O(h^5)$;
L.T.E. of predictor-corrector pair:

$$\frac{4}{5}\theta^2h^5\frac{\partial f}{\partial y}y_n^{iv}-\frac{19}{720}h^5y_n^{v}+O(h^6);$$

$L(\alpha)$-stable, $\alpha>89.999°$, for $\theta=-\frac{15}{88}-\frac{3}{22}\sqrt{5}$.

SAMF (4, 5).

Predictor:

$$\bar{y}_{n+4}-\left(1-\frac{3312}{473}\theta\right)y_{n+3}-\frac{5184}{473}\theta y_{n+2}+\frac{2304}{473}\theta y_{n+1}-\frac{432}{473}\theta y_n$$

$$=h\left[-\left(\frac{19}{720}+\frac{95}{473}\theta\right)f_n+\left(\frac{53}{360}+\frac{530}{473}\theta\right)f_{n+1}-\left(\frac{11}{30}+\frac{120}{43}\theta\right)f_{n+2}\right.$$

$$\left.+\left(\frac{323}{360}+\frac{3230}{473}\theta\right)f_{n+3}+\left(\frac{251}{720}-\theta\right)\bar{f}_{n+4}\right];$$

Corrector:

$$y_{n+4}-y_{n+3}=h\left[-\frac{19}{720}f_n+\frac{53}{360}f_{n+1}-\frac{11}{30}f_{n+2}+\frac{323}{360}f_{n+3}+\left(\frac{251}{720}-\theta\right)f_{n+4}\right]+\theta h\bar{f}_{n+4};$$

L.T.E. of predictor: $\frac{1728}{2365}\theta h^5 y_n^v+O(h^6)$;
L.T.E. of predictor-corrector pair:

$$\frac{1728}{2365}\theta^2 h^6\frac{\partial f}{\partial y}y_n^v-\frac{3}{160}h^6 y_n^{vi}+O(h^7);$$

$L(\alpha)$-stable, $\alpha>85°$, for $\theta=-473/3600-\sqrt{1419}/150$.

SAMF (5, 6).

Predictor:

$$\bar{y}_{n+5}-\left(1-\frac{46944}{4265}\theta\right)y_{n+4}-\frac{17280}{853}\theta y_{n+3}+\frac{11520}{853}\theta y_{n+2}-\frac{4320}{853}\theta y_{n+1}+\frac{3456}{4265}\theta y_n$$
$$=h\left[\left(\frac{3}{160}+\frac{3699}{21325}\theta\right)f_n-\left(\frac{173}{1440}+\frac{23701}{21325}\theta\right)f_{n+1}+\left(\frac{241}{720}+\frac{66034}{21325}\theta\right)f_{n+2}-\left(\frac{133}{240}+\frac{109326}{21325}\theta\right)f_{n+3}+\left(\frac{1427}{1440}+\frac{195499}{21325}\theta\right)f_{n+4}+\left(\frac{95}{288}-\theta\right)\bar{f}_{n+5}\right];$$

Corrector:

$$y_{n+5}-y_{n+4}=h\left[\frac{3}{160}f_n-\frac{173}{1440}f_{n+1}+\frac{241}{720}f_{n+2}-\frac{133}{240}f_{n+3}+\frac{1427}{1440}f_{n+4}+\left(\frac{95}{288}-\theta\right)f_{n+5}\right]+\theta h\bar{f}_{n+5};$$

L.T.E. of predictor: $\frac{576}{853}\theta h^6 y_n^{vi}+O(h^7)$;
L.T.E. of predictor-corrector pair:

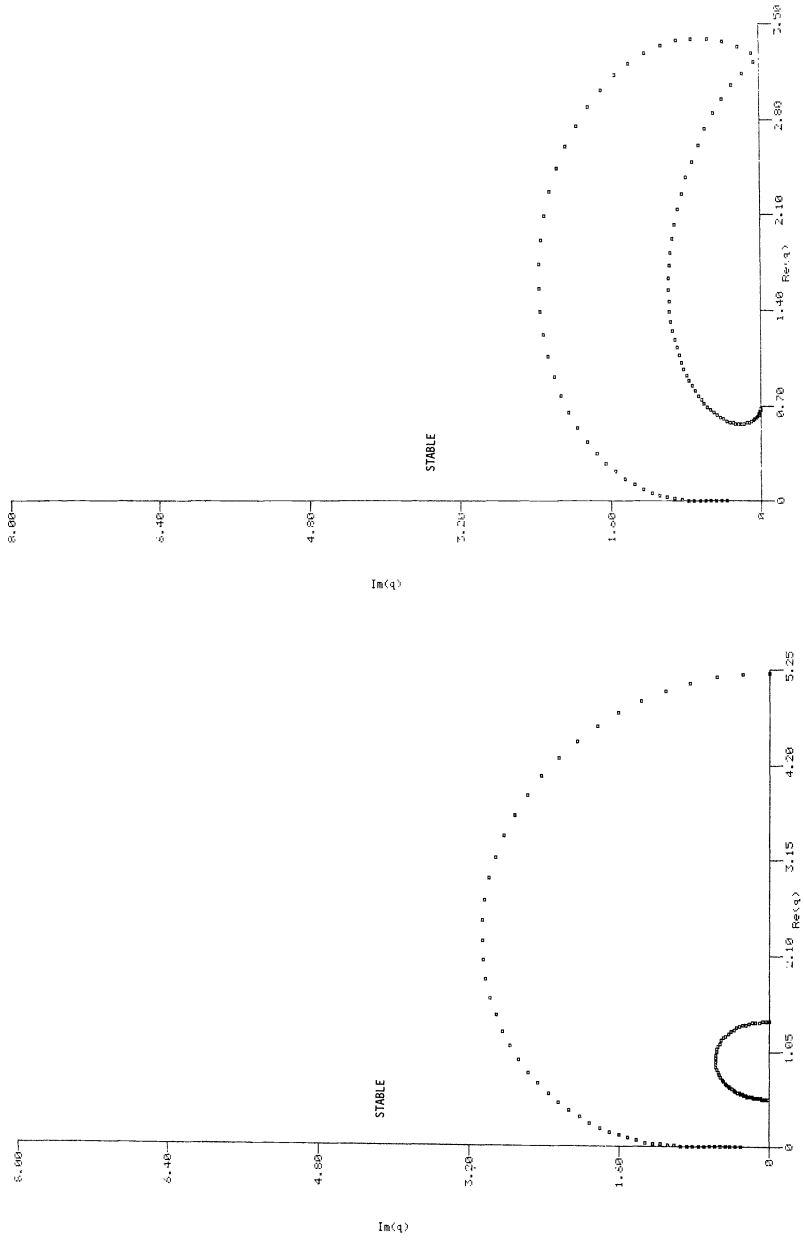
$$\frac{576}{853}\theta^2 h^7\frac{\partial f}{\partial y}y_n^{vi}-\frac{863}{60480}h^7 y_n^{vii}+O(h^8);$$

$L(\alpha)$-stable, $\alpha>71°$, for $\theta=-\frac{4265}{39456}-5\sqrt{5118/1644}$.

The L.T.E. of a SLMM of order p has the form

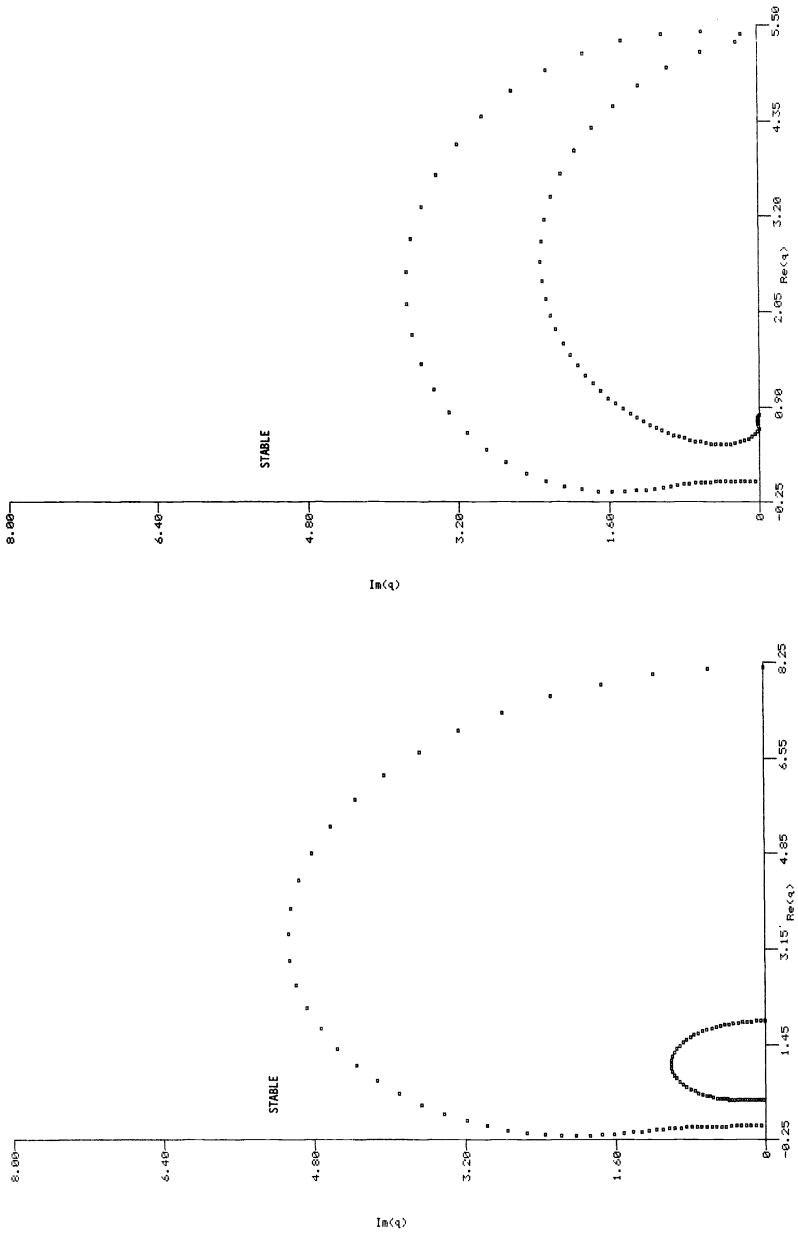
$$\theta^2\bar{C}_p h^{p+1}\frac{\partial f}{\partial y}y^{(p)}+C_{p+1}h^{p+1}y^{(p+1)}+O(h^{p+2}) \tag{3.1}$$

where $\theta\bar{C}_p$ and C_{p+1} are the error constants of the predictor and corrector, respectively. While requiring one less starting value, an SAMF possesses the additional feature that both of its error constants appearing in (3.1) are smaller in magnitude than those of a SBDF of the same order (see [1]). Moreover, the regions of absolute stability of SAMFs are similar to those of SBDFs; in particular, their regions of $L(\alpha)$-stability are nearly identical. This is quite remarkable due to the drastic differences in the stability regions of the unsplit (basic) AMFs and BDFs (see [6, pp. 131, 215–216]). The regions of absolute stability were found numerically using the boundary locus method, and appear in Figs. 3.1 and 3.2 for the fourth- and fifth-order SLMMs, respectively; the regions are symmetric about the real axis.



(a) SAMF (3, 4) (b) SBDF (4, 4)

FIG 3.1. *Stability boundaries of fourth-order* SLMMs.



(a) SAMF (4, 5) (b) SBDF (5, 5)

FIG 3.2. *Stability boundaries of fifth-order* SLMMs.

4. Numerical results. In this section, the results of using SLMMs on some test problems are given. Our first example is Ehle's linear problem [4]:

$$\begin{aligned} \dot{y} &= -y+95z, \quad y(0)=1, \\ \dot{z} &= -y-97z, \quad z(0)=1, \end{aligned} \qquad x\in[0,2]. \tag{4.1}$$

The results for the numerical integration of (4.1) using fixed stepsizes are given in Table 4.1. In this and subsequent tables, "Error" denotes the error in the infinity norm at the end of the interval of integration and "Ord" denotes the order of the method. Letting e_1 and e_2 be the maximum errors at $x=2$ using stepsizes h_1 and h_2, respectively, and assuming $e_i = Ch_i^p$, Table 4.1 also includes the observed orders of convergence calculated using

$$p = (\ln e_1 - \ln e_2)/(\ln h_1 - \ln h_2).$$

All required missing starting values for the SLMMs were provided using the exact solution of (4.1).

TABLE 4.1

	h	$\frac{1}{8}$		$\frac{1}{16}$		$\frac{1}{32}$		$\frac{1}{64}$	
Ord	Method	Error	p	Error	p	Error	p	Error	p
2	SAMF(1, 2)	3.83E−4	—	9.46E−5	2.02	2.35E−5	2.01	5.87E−6	2.00
	SBDF(2, 2)	3.42E−3	—	8.10E−4	2.07	1.98E−4	2.04	4.88E−5	2.02
3	SAMF(2, 3)	6.29E−4	—	8.72E−5	2.85	1.16E−5	2.91	1.50E−6	2.95
	SBDF(3, 3)	1.13E−5	—	6.18E−6	0.87	1.31E−6	2.24	1.99E−7	2.71
4	SAMF(3, 4)	8.01E−5	—	5.26E−6	3.93	3.38E−7	3.96	2.15E−8	3.98
	SBDF(4, 4)	1.71E−4	—	1.18E−4	3.86	7.73E−7	3.92	4.97E−8	3.97
5	SAMF(4, 5)	1.31E−5	—	3.93E−7	5.06	1.24E−8	4.99	3.88E−10	4.99
	SBDF(5, 5)	4.29E−5	—	1.31E−6	5.02	4.19E−8	4.97	1.32E−9	4.99
6	SAMF(5, 6)	2.13E−5	—	2.57E−7	6.37	5.05E−10	8.99	7.89E−12	6.00
	SBDF(6, 6)	5.61E−5	—	3.13E−7	7.49	2.25E−9	7.12	3.19E−11	6.14

From Table 4.1, we note that SAMFs are consistently more accurate than SBDFs for a given stepsize h, except for the third-order methods. As mentioned in § 2, using the other root of (2.17) does define a third-order SAMF that is more accurate than SBDF(3, 3), on this problem; however, it is not L-stable.

Next we consider a variable-step algorithm. An SLMM of order p generates two approximate solutions at x_{n+k}, namely, y_{n+k} and $\bar{y}_{n+k}$. The different orders of the two formulas lead to an estimate of the local trunction error in $\bar{y}_{n+k}$ as discussed by Shampine and Gordon [7]. Our implemation is based on local extrapolation in that the asymptotically more accurate solution y_{n+k} is carried forward. Defining

$$\text{EST} = \|y_{n+k} - \bar{y}_{n+k}\|_\infty,$$

we use the following stepsize control policy for a given local accuracy requirement ε:

If EST$>\varepsilon$, reject the solution and halve h.

If $\varepsilon >$ EST$> \varepsilon/32$, accept the solution and keep h fixed.

If EST$< \varepsilon/32$, accept the solution and double h.

The predicted value $\bar{y}_{n+k}$ and the corrected value y_{n+k} are defined implicitly in (2.1) and (2.2), respectively. We resolve this implicitness using a modified Newton method that for the predictor (2.1) has the form

$$(I - \bar{\beta}_k hJ)\Delta^{(p)}_{n+k} = g(\bar{y}^{(p)}_{n+k})$$

where

$$\Delta_{n+k}^{(p)} = \bar{y}_{n+k}^{(p+1)} - \bar{y}_{n+k}^{(p)}, \qquad J = \frac{\partial f}{\partial y}(x_{n+k}, \bar{y}_{n+k}^{(0)}),$$

$$g(\bar{y}_{n+k}^{(p)}) = -\bar{\alpha}_k \bar{y}_{n+k}^{(p)} + h\bar{\beta}_k f(x_{n+k}, \bar{y}_{n+k}^{(p)}) - \sum_{j=0}^{k-1} (\bar{\alpha}_j y_{n+j} - h\bar{\beta}_j f_{n+j}).$$

The Newton iteration scheme for the corrector is similar to that for the predictor, in particular, from (2.3), both possess the same iteration matrix. Only if the iteration fails to converge inside 10 times is this matrix updated, where the convergence criterion $\|\Delta_{n+k}^{(p)}\|_\infty < \varepsilon \cdot 10^{-2}$ was used. In all examples attempted, this iteration scheme was found to converge rapidly with the initial approximations $\bar{y}_{n+k}^{(0)} = y_{n+k-1}$ and $y_{n+k}^{(0)} = \bar{y}_{n+k}$.

To test our automatic step control policy we consider the nonlinear problem D5[5]:

$$\begin{aligned} \dot{y} &= 0.01 - (0.01 + y + z)(y^2 + 1001y + 1001), & y(0) &= 0, \\ \dot{z} &= 0.01 - (0.01 + y + z)(1 + z^2), & z(0) &= 0, \end{aligned} \qquad x \in [0, 100]. \tag{4.2}$$

The initial stepsize was taken to be $h = 0.125$, and the one-step method SAMF (1, 2) was used to provide the missing starting values initially, and during stepsize changes. As can be observed in Table 4.2, as the local accuracy requirement ε is decreased, the global accuracy obtained is also decreased. Note also that SAMFs return a more accurate solution than SBDFs in each case and, typically, in fewer steps as the local accuracy requirement is decreased.

Our final example is B5[5]:

$$\begin{aligned} \dot{y}_1 &= -10y_1 + 100y_2, & y_1(0) &= 1, \\ \dot{y}_2 &= -100y_1 - 10y_2, & y_2(0) &= 1, \\ \dot{y}_3 &= -4y_3, & y_3(0) &= 1, \\ \dot{y}_4 &= -y_4, & y_4(0) &= 1, \\ \dot{y}_5 &= -0.5y_5, & y_5(0) &= 1, \\ \dot{y}_6 &= -0.1y_6, & y_6(0) &= 1, \end{aligned} \qquad x \in [0, 20], \tag{4.3}$$

which is well known to create difficulties for standard BDFs due to its eigenvalues, $\lambda_{1,2} = -10 \pm 100i$, of large modulus lying close to the imaginary axis. For a given local accuracy requirement ε from Table 4.3 it can be seen that as the order of the SLMMs

TABLE 4.2

	ε	1.E−4		1.E−6		1.E−8	
Ord	Method	Error	Steps	Error	Steps	Error	Steps
3	SAMF (2, 3)	3.16E−5	48	3.00E−7	231	6.42E−10	1078
	SBDF (3, 3)	9.73E−5	22	1.51E−6	213	1.19E−8	1043
4	SAMF (3, 4)	1.43E−5	32	2.89E−7	98	1.07E−10	343
	SBDF (4, 4)	3.20E−5	22	8.44E−7	108	6.86E−9	379
5	SAMF (4, 5)	1.63E−5	30	1.89E−7	61	1.09E−9	178
	SBDF (5, 5)	3.43E−5	19	5.49E−7	67	3.33E−9	202
6	SAMF (5, 6)	8.22E−6	29	1.21E−8	51	1.76E−9	111
	SBDF (6, 6)	1.41E−5	27	1.88E−7	44	2.40E−9	123

TABLE 4.3

	ε	1.E−2		1.E−3		1.E−4	
Ord	Method	Error	Steps	Error	Steps	Error	Steps
3	SAMF (2, 3)	2.07E−4	224	3.21E−5	521	4.49E−6	1125
	SBDF (3, 3)	1.70E−5	231	2.42E−6	526	5.01E−7	1129
4	SAMF (3, 4)	2.29E−5	150	2.23E−6	299	1.05E−6	552
	SBDF (4, 4)	1.38E−5	186	4.06E−6	365	8.22E−7	617
5	SAMF (4, 5)	8.32E−6	131	8.58E−6	271	6.72E−7	488
	SBDF (5, 5)	3.06E−6	175	3.06E−6	355	1.50E−6	511
6	SAMF (5, 6)	4.14E−3	1209	8.25E−4	1246	3.75E−5	1314
	SBDF (6, 6)	6.23E−3	1343	1.77E−4	1487	7.91E−5	1588

increases to five, the integration is completed in fewer steps. The methods of order six are $L(\alpha)$-stable where $\alpha \approx 71°$ and, consequently, the eigenvalues $\lambda_{1,2}$ severely restrict the stepsize in this case. For this example, SAMFs complete the integrations in fewer steps than SBDFs for each of the specified local accuracy requirements.

These examples suggest SAMFs can be competitive with SBDFs while maintaining the desirable features of the Adams formulas. This relationship seems to warrant further investigation, in particular, to determine whether SAMFs can work in collusion with nonstiff Adams solvers in a type-insensitive code.

REFERENCES

[1] J. R. CASH, *Split linear multistep methods for the integration of stiff differential systems*, Numer. Math., 42 (1983), pp. 299-310.

[2] ———, *Two new finite difference schemes for parabolic equations*, SIAM J. Numer. Anal., 21 (1984), pp. 433-446.

[3] M. J. CASPER AND D. A. VOSS, *L_0-stable formulas for parabolic equations*, Comput. Math. Appl., to appear.

[4] B. L. EHLE, *On Padé approximations to the exponential function and A-stable methods for the numerical solution of initial value problems*, Res. Report CSRR2010, Department of AACS, University of Waterloo, Waterloo, Ontario, Canada, 1969.

[5] W. ENRIGHT, T. HULL, AND B. LINDBERG, *Comparing numerical methods for stiff systems of* O.D.E.'s, BIT, 15 (1975), pp. 10-48.

[6] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice Hall, Englewood Cliffs, NJ, 1971.

[7] L. F. SHAMPINE AND M. K. GORDON, *Compute Solution of Ordinary Differential Equations*, W. H. Freeman, San Francisco, 1975.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 1000-1009, September 1989
© 1989 Society for Industrial and Applied Mathematics
010

STABILITY PROPERTIES OF BACKWARD EULER MULTIRATE FORMULAS*

STIG SKELBOE† AND PER ULFKJAER ANDERSEN†

Abstract. Stability properties of multirate formulas cannot be analyzed by a scalar test equation but require at least one equation for each different steplength. This paper generalizes the concept of absolute stability and A-stability for backward Euler multirate formulas. Stability theorems for multirate methods with two and three different steplengths are given, while a general result for an arbitrary number of different steplengths is the topic of future research.

Key words. multirate formulas, stability, numerical integration of ODEs

AMS(MOS) subject classification. 65L20

1. Introduction. In this paper we consider the initial value problem for systems of ordinary differential equations that can be partitioned into m subsystems:

$$\begin{aligned} \mathbf{x}_1' &= \mathbf{f}_1(\mathbf{t}, \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m) \\ \mathbf{x}_2' &= \mathbf{f}_2(\mathbf{t}, \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m) \\ &\vdots \\ \mathbf{x}_m' &= \mathbf{f}_m(\mathbf{t}, \mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m) \end{aligned} \tag{1.1}$$

where

$$x_r \in R^{n_r}, \qquad \mathbf{f}_r : R^{N+1} \to R^{n_r} \quad \text{for } r = 1, 2, \cdots, m,$$

and

$$N = \sum_{r=1}^{m} n_r, \qquad t \in [t_a, t_b].$$

Let a linear multistep formula be defined by the operator

$$L_k[x(t); h] = \sum_{r=0}^{k} \alpha_r x(t - rh) + h\beta_r x'(t - rh).$$

Assume that there exist disjoint intervals $I_r \subset [t_a, t_b]$, $r = 1, 2, \cdots$ and an ordering of $\{1, 2, \cdots, m\}$ for each interval I_r denoted by $\{i_1, i_2, \cdots, i_m\}$ such that

$$\|L_k[\mathbf{x}_{i_1}(t); h]\| \gg \|L_k[\mathbf{x}_{i_2}(t); h]\| \gg \cdots \gg \|L_k[\mathbf{x}_{i_m}(t); h]\| \tag{1.2}$$

for $t \in I_r$. The operator L_k is applied to true solutions $\mathbf{x}_r$ of (1.1). In each of the intervals I_r, it is then possible, without increasing the maximum local truncation error, to integrate the system (1.1) with the formula L_k using increasingly longer steps in subsystems $i_1, i_2, \cdots, i_m$. An integration formula using different steplengths in different subsystems is called a multirate formula.

A very important class of applications for multirate formulas is the simulation of digital MOS circuits in connection with the waveform relaxation method. This method was first described in [1], and it is very thoroughly described in [2] including a comprehensive bibliography. The stability properties of implicit linear multirate formulas have been discussed previously in [3] and [4].

* Received by the editors July 11, 1988; accepted for publication (in revised form) October 26, 1988.
† Department of Computer Science, University of Copenhagen, Universitetsparken 1, 2100 Copenhagen 0, Denmark.

2. Backward Euler multirate formulas. Assume without loss of generality that (1.2) is fulfilled for the index set $\{1, 2, \cdots, m\}$ for $I_1 = [0, t_b]$. Let $\mathbf{x}_{r,s}$ denote a discrete solution point such that $\mathbf{x}_{r,s} \approx \mathbf{x}_r(t_s)$ and $t_s = sh$. Then the backward Euler multirate formulas can be expressed as the coupled system

$$
\begin{aligned}
&\mathbf{x}_{1,r_1} = \mathbf{x}_{1,r_1-1} + h\mathbf{f}_1(t_{r_1}, \mathbf{x}_{1,r_1}, \mathbf{x}_{2,r_2}, \mathbf{x}_{3,r_3}, \cdots, \mathbf{x}_{m,r_m}),\\
&\mathbf{x}_{2,r_2} = \mathbf{x}_{2,r_2-q_2} + q_2 h\mathbf{f}_2(t_{r_2}, \mathbf{x}_{1,r_2}, \mathbf{x}_{2,r_2}, \mathbf{x}_{3,r_3}, \cdots, \mathbf{x}_{m,r_m}),\\
&\mathbf{x}_{3,r_3} = \mathbf{x}_{3,r_3-q_2q_3} + q_2q_3 h\mathbf{f}_3(t_{r_3}, \mathbf{x}_{1,r_3}, \mathbf{x}_{2,r_3}, \mathbf{x}_{3,r_3}, \cdots, \mathbf{x}_{m,r_m}),\\
&\qquad\vdots\\
&\mathbf{x}_{m,r_m} = \mathbf{x}_{m,r_m-Q} + Qh\mathbf{f}_m(t_{r_m}, \mathbf{x}_{1,r_m}, \mathbf{x}_{2,r_m}, \mathbf{x}_{3,r_m}, \cdots, \mathbf{x}_{m,r_m})
\end{aligned}
\tag{2.1}
$$

where

$$
\begin{aligned}
r_1 &= 1 + s_1 + s_2q_2 + s_3q_2q_3 + \cdots + s_mQ,\\
r_2 &= \quad q_2 + s_2q_2 + s_3q_2q_3 + \cdots + s_mQ,\\
r_3 &= \qquad q_2q_3 + s_3q_2q_3 + \cdots + s_mQ,\\
&\vdots\\
r_m &= \qquad\qquad\qquad Q + s_mQ,
\end{aligned}
$$

and

$$
\begin{aligned}
s_1 &= 0, 1, \cdots, q_2 - 1,\\
s_2 &= 0, 1, \cdots, q_3 - 1,\\
s_3 &= 0, 1, \cdots, q_4 - 1,\\
&\vdots\\
s_m &= 0, 1, \cdots\\
Q &= q_2q_3 \cdots q_m
\end{aligned}
$$

and

$$q_2, q_3, \cdots, q_m \quad \text{are positive integers.}$$

In the first formula of (2.1)

$$\mathbf{x}_{2,r_2}, \mathbf{x}_{3,r_3}, \cdots, \mathbf{x}_{m,r_m} \quad \text{represent } \mathbf{x}_{2,r_1}, \mathbf{x}_{3,r_1}, \cdots, \mathbf{x}_{m,r_1}$$

obtained by an interpolation formula of order zero. Similarly for the second formula,

$$\mathbf{x}_{3,r_3}, \cdots \mathbf{x}_{m,r_m} \quad \text{represent } \mathbf{x}_{3,r_2}, \cdots, \mathbf{x}_{m,r_2}$$

obtained by zero-order interpolation, and analogously for the remaining formulas. Use of interpolation of order zero is sufficient to retain the overall order of one of the backward Euler formula.

The multirate discretization can be symbolized by the time axes of Fig. 1 where each axis corresponds to a subsystem and where the ticks correspond to the time instants where the solution of each individual subsystem is computed.

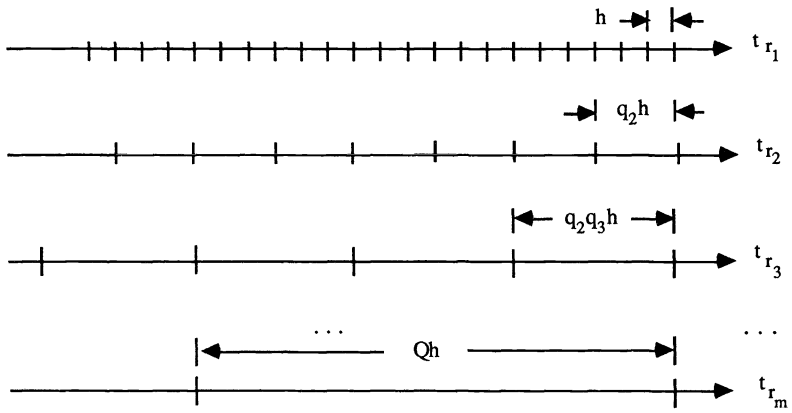


FIG. 1. *Discretization points for the multirate formula in* (2.1).

A compound step of the backward Euler multirate formula (2.1) is defined as one step of the last formula, q_m steps of the next-to-last formula, etc., including Q steps of the first formula. A compound step involves a total of

$$P = \sum_{r=1}^{m-1} n_r \prod_{s=r+1}^{m} q_s + n_m$$

algebraic equations to be satisfied. This number should be compared with N algebraic equations for the classical backward Euler formula for each of a total of Q steps.

Implicit multirate formulas would only be of practical interest in rare cases if the algebraic equations resulting from a compound step had to be solved with standard methods, say a Newton iteration. For the simulation of digital MOS circuits, the algebraic equations, however, can be solved efficiently by the waveform relaxation method that is outlined for a compound step:

(i) Compute an estimate of

$$\mathbf{x}_{1,r_1}, \mathbf{x}_{2,r_2}, \cdots, \mathbf{x}_{m,r_m}$$

$$\text{for } s_1 = 0, 1, \cdots, q_2 - 1, \quad s_2 = 0, 1, \cdots, q_3 - 1, \cdots, s_m = 0.$$

(ii) Use each of the formulas of (2.1) **independently** to compute new values of

$$\mathbf{x}_{1,r_1}, \mathbf{x}_{2,r_2}, \cdots, \mathbf{x}_{m,r_m}$$

$$\text{for } x_1 = 0, 1, \cdots, q_2 - 1, \quad s_2 = 0, 1, \cdots, q_3 - 1, \cdots, s_m = 0.$$

The estimated solution values from (i) or from a previous iteration are used to compute $\mathbf{x}_{1,r_1}$ from the first formula, $\mathbf{x}_{2,r_2}$ from the second formula, etc.

(iii) If the process has converged, a compound step is finished and another is initiated by going to (i). Otherwise the most recent solution values are used for another iteration of step (ii).

The waveform relaxation is usually applied to several consecutive compound steps at a time. The algorithm outlined above is analogous to Jacobi's iterative method for linear equations. It is obvious how to modify the algorithm for the analogue of the Gauss-Seidel method.

This paper deals with the stability properties of backward Euler multirate formulas, and this problem will be dealt with independently of the method chosen for the solution of the algebraic equations. Waveform relaxation methods may or may not be convergent

for the class of problems where the multirate formulas are stable, but this important issue will not be considered further in this paper.

3. Test problem. The stability properties of classical linear multistep formulas are characterized by the discretization of the simple scalar test equation $x' = \lambda x$ where λ represents an eigenvalue of the coefficient matrix of a linear system. A test problem for a multirate method requires at least one equation for each steplength, i.e., at least m equations for the method (2.1).

Let a linearization of (1.1) be represented by the linear block system (displayed eq. (3.1)), which is

$$\begin{pmatrix} \mathbf{x}_1' \\ \mathbf{x}_2' \\ \vdots \\ \mathbf{x}_m' \end{pmatrix} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \cdots & \mathbf{A}_{1m} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \cdots & \mathbf{A}_{2m} \\ \vdots & \vdots & & \vdots \\ \mathbf{A}_{m1} & \mathbf{A}_{m2} & \cdots & \mathbf{A}_{mm} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_m \end{pmatrix} \tag{3.1}$$

autonomous without loss of generality. The matrices $\mathbf{A}_{rs}$ are of dimension $n_r \times n_s$.

Let the system (3.1) be discretized by the multirate formula in (2.1). Then a compound step can be expressed as follows:

$$(\mathbf{x}_{1,Q}, \mathbf{x}_{2,Q}, \cdots, \mathbf{x}_{m,Q})^T = \mathbf{K}(\mathbf{x}_{1,0}, \mathbf{x}_{2,0}, \cdots, \mathbf{x}_{m,0})^T \tag{3.2}$$

where $\mathbf{K}$ is the analogue of the companion matrix for a classical multistep formula.

The discretization of (3.1) by the formulas (2.1) is called absolutely stable if $\rho(\mathbf{K}) < 1$ where ρ denotes spectral radius. The system of differential equations (3.1) is stable if $\alpha(\mathbf{A}) < 0$ where α is the spectral abcissa (the maximum real part of the eigenvalues of $\mathbf{A}$) and $\mathbf{A}$ is the total coefficient matrix in (3.1).

The stability properties of a discretization of (3.1) by a classical linear multistep formula is completely characterized by the eigenvalues of $\mathbf{A}$. However, this is not the case for multirate formulas where the eigenvectors of $\mathbf{A}$ also influence the stability of the discretization. The only exception is when $\mathbf{A}$ is block triangular or block diagonal.

Define the matrix $\mathbf{A}_b$ by

$$\mathbf{A}_b = \begin{pmatrix} \mu(\mathbf{A}_{11}) & \|\mathbf{A}_{12}\| & \cdots & \|\mathbf{A}_{1m}\| \\ \|\mathbf{A}_{21}\| & \mu(\mathbf{A}_{22}) & \cdots & \|\mathbf{A}_{2m}\| \\ \vdots & \vdots & & \vdots \\ \|\mathbf{A}_{m1}\| & \|\mathbf{A}_{m2}\| & \cdots & \mu(\mathbf{A}_{mm}) \end{pmatrix} \tag{3.3}$$

where μ is the logarithmic norm [5] based on the norm used in (3.3).

Define a linear system of differential equations by

$$\mathbf{y}' = \mathbf{A}_b \mathbf{y}. \tag{3.4}$$

Then the following lemma can be proved [6, Cor. 3]):

LEMMA. *Consider the solutions of* (3.1) *and* (3.4) *for* $t \geqq 0$ *assuming that*

$$\|\mathbf{x}_1(0)\| \leqq y_1(0), \qquad \|\mathbf{x}_2(0)\| \leqq y_2(0), \cdots, \|\mathbf{x}_m(0)\| \leqq y_m(0).$$

Then

$$\|\mathbf{x}_1(t)\| \leqq y_1(t), \quad \|\mathbf{x}_2(t)\| \leqq y_2(t), \cdots, \|\mathbf{x}_m(t)\| \leqq y_m(t) \quad \textit{for all } t \geqq 0.$$

The system of differential equations (3.3) is stable if the eigenvalues of $\mathbf{A}_b$ are in the left halfplane. It follows trivially from the lemma that stability of (3.4) implies stability of (3.1). The approach of this paper is to give sufficient conditions for stability of (3.4) and prove that these conditions are also sufficient for the stability of the discretization of (3.1) by (2.1).

The logarithmic norm will be used extensively in the following, and basic properties of this important tool can be found in [5] and [7]. The logarithmic norm is defined as

$$\mu(\mathbf{A}) = \lim_{h \to 0+} (\|\mathbf{I} + h\mathbf{A}\| - 1)/h$$

where $\|\cdot\|$ is the matrix norm on which μ is based. For the 1, 2, and ∞ norms μ is expressed as follows:

$$\mu_1(\mathbf{A}) = \max_j \left(a_{jj} + \sum_{\substack{i=1 \\ i \neq j}}^{N} |a_{ij}| \right),$$

$$\mu_2(\mathbf{A}) = \max_j \{\lambda_j((\mathbf{A} + \mathbf{A}^T)/2)\},$$

$$\mu_\infty(\mathbf{A}) = \max_i \left(a_{ii} + \sum_{\substack{j=1 \\ i \neq j}}^{N} |a_{ij}| \right).$$

The following properties of the logarithmic norm, which have all been proved in [5], are used in this paper:

$$-\|\mathbf{A}\| \leqq -\mu(-\mathbf{A}) \leqq \operatorname{Re}\{\lambda_i(\mathbf{A})\} \leqq \mu(\mathbf{A}) \leqq \|\mathbf{A}\| \quad \text{(for all } i\text{)}, \tag{3.5}$$

$$\mu(c\mathbf{A}) = c\mu(\mathbf{A}) \quad \text{for } c \geqq 0, \tag{3.6}$$

$$\mu(\mathbf{A} + c\mathbf{I}) = \mu(\mathbf{A}) + c \quad \text{for all } c \in R, \tag{3.7}$$

$$\|\mathbf{A}\mathbf{x}\| \geqq \max\{-\mu(-\mathbf{A}), -\mu(\mathbf{A})\}\|\mathbf{x}\|. \tag{3.8}$$

Consider a discretization of (3.1) by the backward Euler multirate formulas (2.1). The first step of the first equation reads

$$\mathbf{x}_{1,1} = \mathbf{x}_{1,0} + h(\mathbf{A}_{11}\mathbf{x}_{1,1} + \mathbf{A}_{12}\mathbf{x}_{2,q_2} + \cdots + \mathbf{A}_{1m}\mathbf{x}_{m,Q}).$$

Using the logarithmic norm, $\mathbf{x}_{1,0}$ can be bounded as follows:

$$\|\mathbf{x}_{1,0}\| \geqq (1 - h\mu(\mathbf{A}_{11}))\|\mathbf{x}_{1,1}\| - h\|\mathbf{A}_{12}\|\,\|\mathbf{x}_{2,q_2}\| - \cdots - h\|\mathbf{A}_{1m}\|\,\|\mathbf{x}_{m,Q}\| \tag{3.9}$$

since $\|(\mathbf{I} - h\mathbf{A}_{11})\mathbf{x}_{1,1}\| \geqq (1 - h\mu(\mathbf{A}_{11}))\|\mathbf{x}_{1,1}\|$ by (3.8) where $-\mu(-\mathbf{A})$ is chosen since $-\mu(-\mathbf{I} + h\mathbf{A}_{11}) > -\mu(\mathbf{I} - h\mathbf{A}_{11})$ for $h\mu(\mathbf{A}_{11}) < 1$, which is reasonable to expect if the system (3.4) is stable.

Define the following modification of $\mathbf{A}_b$ in (3.3):

$$\mathbf{B} = -h\mathbf{A}_b \tag{3.10}$$

where the matrix elements are denoted by b_{ij}.

The principle of relation (3.9) can now be continued by bounding $\|\mathbf{x}_{1,1}\|$ and substituting this relation into (3.9), etc.:

$$\begin{aligned} \|\mathbf{x}_{1,0}\| &\geqq (1 + b_{11})^2\|\mathbf{x}_{1,2}\| + ([(1 + b_{11})^2 - 1]/b_{11})(b_{12}\|\mathbf{x}_{2,q_2}\| + \cdots + b_{1m}\|\mathbf{x}_{m,Q}\|) \\ &\vdots \\ \|\mathbf{x}_{1,0}\| &\geqq (1 + b_{11})^{q_2}\|\mathbf{x}_{1,q_2}\| + ([(1 + b_{11})^{q_2} - 1]/b_{11})(b_{12}\|\mathbf{x}_{2,q_2}\| + \cdots + b_{1m}\|\mathbf{x}_{m,Q}\|) \end{aligned} \tag{3.11}$$

where the following relation has been used for the scalar b_{11}:

$$\sum_{r=0}^{q-1} (\mathbf{I} + \mathbf{B})^r = [(\mathbf{I} + \mathbf{B})^q - \mathbf{I}]\mathbf{B}^{-1}. \tag{3.12}$$

Introducing

$$c_1 = [(1+b_{11})^{q_2} - 1]/(b_{11}q_2) \tag{3.13}$$

we can write (3.11) as

$$\|\mathbf{x}_{1,0}\| \leqq (1 + q_2 c_1 b_{11})\|\mathbf{x}_{1,q_2}\| + q_2 c_1 (b_{12}\|\mathbf{x}_{2,q_2}\| + \cdots + b_{1m}\|\mathbf{x}_{m,Q}\|). \tag{3.14}$$

Similarly, from the second equation in (2.1) with $r_2 = q_2$, we have

$$\|\mathbf{x}_{2,0}\| \leqq (1 + q_2 b_{22})\|\mathbf{x}_{2,q_2}\| + q_2 (b_{21}\|\mathbf{x}_{1,q_2}\| + b_{23}\|\mathbf{x}_{3,q_2q_3}\| + \cdots + b_{2m}\|\mathbf{x}_{m,Q}\|). \tag{3.15}$$

The idea leading to (3.14) is now repeated for the pair (3.14) and (3.15) and the analogous pairs from $r_2 = 2q_2, \cdots, q_3q_2$, giving

$$\begin{pmatrix} \|\mathbf{x}_{1,0}\| \\ \|\mathbf{x}_{2,0}\| \end{pmatrix} \leqq (\mathbf{I}_2 + q_3\mathbf{C}_2\mathbf{B}_2) \begin{pmatrix} \|\mathbf{x}_{1,q_2q_3}\| \\ \|\mathbf{x}_{2,q_2q_3}\| \end{pmatrix} + q_2q_3\mathbf{C}_2 \begin{pmatrix} c_1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} b_{13} \cdots b_{1m} \\ b_{23} \cdots b_{2m} \end{pmatrix} \begin{pmatrix} \|\mathbf{x}_{3,q_2q_3}\| \\ \vdots \\ \|\mathbf{x}_{m,Q}\| \end{pmatrix} \tag{3.16}$$

where

$$\mathbf{B}_2 = q_2 \begin{pmatrix} c_1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

and $\mathbf{I}_m$ denotes an $m \times m$ identity matrix where ambiguity may occur. $\mathbf{C}_2$ is defined by (3.12):

$$\mathbf{C}_2 = \frac{1}{q_3}[(\mathbf{I}_2 + \mathbf{B}_2)^{q_3} - \mathbf{I}_2]\mathbf{B}_2^{-1}. \tag{3.17}$$

By repeated application of the calculations above, a compound step can be estimated as follows:

$$(\|\mathbf{x}_{1,0}\|, \|\mathbf{x}_{2,0}\|, \cdots, \|\mathbf{x}_{m,0}\|)^T \leqq (\mathbf{I}_m + \mathbf{B}_m)(\|\mathbf{x}_{1,Q}\|, \|\mathbf{x}_{2,Q}\|, \cdots, \|\mathbf{x}_{m,Q}\|)^T \tag{3.18}$$

where

$$\mathbf{B}_r = q_2q_3 \cdots q_r \begin{pmatrix} \mathbf{C}_{r-1} & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{C}_{r-2} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_2 \end{pmatrix} \cdots \begin{pmatrix} c_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{r-1} \end{pmatrix} \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & \cdots & b_{2r} \\ \vdots & \vdots & & \vdots \\ b_{r1} & b_{r2} & \cdots & b_{rr} \end{bmatrix}$$

for $r = 2, 3, \cdots, m$ and

$$\mathbf{C}_r = \frac{1}{q_{r+1}}[(\mathbf{I}_r + \mathbf{B}_r)^{q_{r+1}} - \mathbf{I}_r]\mathbf{B}_r^{-1}$$

for $r = 2, 3, \cdots, m-1$; c_1 is defined in (3.13). Note that $\mathbf{K}_b = (\mathbf{I}_m + \mathbf{B}_m)^{-1}$ is the companion matrix (see the definition in (3.2)) corresponding to the discretization of (3.4) by (2.1).

If $\mathbf{K}_b = (\mathbf{I}_m + \mathbf{B}_m)^{-1}$ has nonnegative elements it can be applied to both sides of (3.18) retaining the inequality, and consequently the discretization of (3.1) by (2.1) is stable if $\rho(\mathbf{K}_b) < 1$. Sufficient conditions for stability will be given in the next section for $m = 2$ and $m = 3$. $\mathbf{B}_2$ is given in (3.16) and $\mathbf{B}_3$ reads:

$$\mathbf{B}_3 = q_2q_3 \begin{pmatrix} \mathbf{C}_2 & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} c_1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} \tag{3.19}$$

where c_1 and $\mathbf{C}_2$ are defined in (3.13) and (3.17), respectively.

4. Stability analysis. The stability analysis will be based on the theory of M-matrices [8]. Let Z denote the class of all real square matrices whose off-diagonal elements are nonpositive.

A real square matrix $\mathbf{A}$ is of class M if:

(1) $\mathbf{A} \in Z$; and

(2) $\mathbf{A}$ fulfills one of the following conditions that can be proved to be equivalent [8]:

(a) All principal minors of $\mathbf{A}$ are positive.

(b) $\mathbf{A}^{-1}$ exists and $\mathbf{A}^{-1} \geqq 0$ (all elements of $\mathbf{A}^{-1}$ are nonnegative).

(c) The real part of each eigenvalue of $\mathbf{A}$ is positive.

Stability conditions equivalent to A-stability for $m=2$ and $m=3$ are now formulated in Theorem 1 and 2.

THEOREM 1. *The discretization of* (3.1) *for* $m=2$ *by the backward Euler multirate formula* (2.1) *is absolutely stable for all* $h>0$ *and integer* $q_2>0$ *if there exits a norm such that* $-\mathbf{A}_b$ *is of class M where* $\mathbf{A}_b$ *is defined in* (3.3).

Proof. By assumption, $-\mathbf{A}_b$ is of class M and so is $\mathbf{B}$ as defined in (3.10). $\mathbf{B}_2$ (defined in (3.16)) is the product of an M-matrix and a positive diagonal matrix and is therefore also an M-matrix [8]. This implies that $\mathbf{I}+\mathbf{B}_2$ is an M-matrix that has eigenvalues with real parts larger than one, implying that $\rho((\mathbf{I}+\mathbf{B}_2)^{-1})<1$ and $(\mathbf{I}+\mathbf{B}_2)^{-1} \geqq 0$ and thus stability.

THEOREM 2. *The discretization of* (3.1) *for* $m=3$ *by the backward Euler multirate formula* (2.1) *is absolutely stable for all* $h>0$ *and integer* $q_2, q_3>0$ *if there exists a norm such that* $-\mathbf{A}_b$ *is of class M where* $\mathbf{A}_b$ *is defined in* (3.3).

Proof. The proof of Theorem 2 is complicated by the fact that $\mathbf{B}_3$ as defined in (3.19) is not necessarily of class M since one of the elements of the vector

$$q_2 q_3 \mathbf{C}_2 \begin{pmatrix} c_1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} b_{13} \\ b_{23} \end{pmatrix}$$

may become positive. It must therefore be proved separately that $\mathbf{B}_3$ has eigenvalues only in the right halfplane and that the inverse of $\mathbf{I}+\mathbf{B}_3$ has nonnegative elements.

(1) $\operatorname{Re}\{\lambda_i(\mathbf{B}_3)\}>0$ for $i=1,2,3$. From the proof of Theorem 1 it is clear that

$$\tilde{\mathbf{B}}_2 = q_2 \begin{pmatrix} c_1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} \tag{4.1}$$

is of class M. The matrix

$$\tilde{\mathbf{C}}_2 = \begin{pmatrix} \mathbf{C}_2 & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix} \tag{4.2}$$

is also of class M if $\mathbf{C}_2$ is of class M.

$\mathbf{C}_2$ is defined in (3.17) and $\mathbf{B}_2 \in M$ according to the proof of Theorem 1. The eigenvalues therefore have positive real part and since $\mathbf{B}_2 \in \mathrm{M}$ is 2×2, both eigenvalues are real [8]. From (3.12) it its obvious that $\mathbf{C}_2$ is a polynomial in $\mathbf{B}_2$ with positive coefficients and since the eigenvalues of $\mathbf{C}_2$ are the eigenvalues of $\mathbf{B}_2$ mapped by the same polynomial, $\mathbf{C}_2$ has positive eigenvalues. Furthermore, from (3.12) it is clear that $\mathbf{C}_2 \in Z$ since $\mathbf{C}_2$ is 2×2, so altogether $\mathbf{C}_2 \in M$. (Multiplying 2×2 matrices of class Z and with positive diagonal elements results in new matrices of class Z, a property that does not carry over to matrices of higher dimensions.)

The real parts of the eigenvalues of $\mathbf{B}_3$ are positive if the real parts of the eigenvalues of $\tilde{\mathbf{C}}_2\tilde{\mathbf{B}}_2$ are positive since $\mathbf{B}_3 = q_3\tilde{\mathbf{C}}_2\tilde{\mathbf{B}}_2$. The symbolic manipulation program Maple [9] is used to calculate the coefficients p_0, p_1, p_2, p_3 in

$$|-\tilde{\mathbf{C}}_2\tilde{\mathbf{B}}_2 - \lambda \mathbf{I}_3| = p_3\lambda^3 + p_2\lambda^2 + p_1\lambda + p_0 = P(\lambda)$$

and the Routh–Hurwitz criterion (see [10]) is then applied to $P(\lambda)$ to determine that the zeros of $P(\lambda)$ are in the left halfplane implying that the eigenvalues of $\tilde{\mathbf{C}}_2\tilde{\mathbf{B}}_2$ are in the right halfplane. The coefficients of $P(\lambda)$ are expressed as functions of the elements of the M-matrices $\tilde{\mathbf{C}}_2$ and $\tilde{\mathbf{B}}_2$, and the following properties of M-matrices are used in proving the relations of the Routh–Hurwitz criterion:

- The principal minors are positive and accordingly the diagonal elements are positive.
- The off-diagonal elements are nonpositive.

Let the elements of $\tilde{\mathbf{C}}_2$ and $\tilde{\mathbf{B}}_2$ be denoted as follows:

$$\tilde{\mathbf{C}}_2 = \begin{pmatrix} \alpha_{11} & -\alpha_{12} & 0 \\ -\alpha_{21} & \alpha_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \tilde{\mathbf{B}}_2 = \begin{pmatrix} \beta_{11} & -\beta_{12} & -\beta_{13} \\ -\beta_{21} & \beta_{22} & -\beta_{23} \\ -\beta_{31} & -\beta_{32} & \beta_{33} \end{pmatrix}$$

where $\alpha_{ii}, \beta_{ii} > 0$ and $\alpha_{ij}, \beta_{ij} \geqq 0$ for $i \neq j$. Then the coefficients of $P(\lambda)$ can be expressed as follows:

$$p_0 = |\tilde{\mathbf{C}}_2||\tilde{\mathbf{B}}_2|,$$

$$p_1 = \alpha_{11}(\beta_{11}\beta_{33} - \beta_{13}\beta_{31}) + \alpha_{22}(\beta_{22}\beta_{33} - \beta_{23}\beta_{32}) + \alpha_{21}\beta_{13}\beta_{32} + \alpha_{21}\beta_{12}\beta_{33} + \alpha_{12}\beta_{21}\beta_{33} + \alpha_{12}\beta_{31}\beta_{23} + (\alpha_{11}\alpha_{22} - \alpha_{12}\alpha_{21})(\beta_{11}\beta_{22} - \beta_{12}\beta_{21}),$$

$$p_2 = \beta_{11}\alpha_{11} + \beta_{22}\alpha_{22} + \beta_{12}\alpha_{21} + \beta_{21}\alpha_{12} + \beta_{33}, \qquad p_3 = 1.$$

The Routh–Hurwitz criterion requires the following relations to be fulfilled:

$$p_0, p_1, p_2, p_3 > 0 \quad \text{and} \quad p_1p_2 - p_3p_0 > 0.$$

The coefficients of $P(\lambda)$ are all positive because of the assumptions:

p_0 is a product of positive determinants.

p_1 is a sum of positive and nonnegative terms. The factors in parentheses are principal minors that are positive by assumption.

p_2 is a sum of positive and nonnegative terms.

The positivity of $p_1p_2 - p_3p_0$ is established analogously by computing the expression in terms of α_{ij} and β_{ij}.

(2) $\mathbf{K}_b \geqq 0$. The property is proved in two parts. First it is proved that $\mathbf{K}_b \geqq 0$ except for the entries in positions (1, 3) and (2, 3), and in the second part these two values are also proved to be nonnegative.

$\mathbf{C}_2$ as defined in (3.17) can be expanded as follows:

$$q_3\mathbf{C}_2 = q_3\mathbf{I} + \mathbf{D}$$

where $\mathbf{D}$ is a polynomial in $\mathbf{B}_2$ with positive coefficients. The positive real eigenvalues of $\mathbf{B}_2$ are mapped by the same polynomial implying that $\mathbf{D}$ has positive eigenvalues, and since $\mathbf{D} \in Z$ ($\mathbf{D}$ is 2×2) we can conclude that $\mathbf{D} \in M$.

$$\mathbf{K}_b = \left[\mathbf{I}_3 + \begin{pmatrix} q_3\mathbf{I}_2 + \mathbf{D} & \mathbf{0} \\ \mathbf{0} & q_3 \end{pmatrix}\tilde{\mathbf{B}}_2\right]^{-1}$$

is now computed symbolically by Maple, and it can be demonstrated that all the entries of $\mathbf{K}_b$ except (1, 3) and (2, 3) are positive by using the facts that $\mathbf{D}$ and $\tilde{\mathbf{B}}_2$ are M-matrices (positive diagonal, nonnegative off-diagonal elements, and positive principal minors).

The entry of $\mathbf{K}_b$ in position (1, 1) is now given as an example of the elements of $\mathbf{K}_b$:

$$\mathbf{K}_{b,1,1} = [1 + q_3\beta_{22} + q_3\beta_{33} + \delta_{21}\beta_{12} + \delta_{22}\beta_{22} + q_3\delta_{21}\beta_{12}\beta_{33} + q_3\delta_{21}\beta_{13}\beta_{32} + (q_3^2 + \delta_{22}q_3)(\beta_{22}\beta_{33} - \beta_{23}\beta_{32})]/\Delta_b$$

where

$$\mathbf{D} = \begin{pmatrix} \delta_{11} & -\delta_{12} \\ -\delta_{21} & \delta_{22} \end{pmatrix} \in M,$$

the β-values are defined previously and the determinant is $\Delta_b = |\mathbf{I} + \mathbf{B}_3|$.

$\mathbf{K}_{b,1,1}$ is a sum of positive and nonnegative terms where the last term is made up of two factors. The first factor has two positive terms and the last factor is a principal minor of $\tilde{\mathbf{B}}_2$, which is an M-matrix, and therefore positive. The determinant Δ_b is positive since the eigenvalues of $\mathbf{B}_3$ are in the right halfplane. In conclusion, $\mathbf{K}_{b,1,1}$ is positive.

The elements of $\mathbf{K}_b$ in positions (1, 3) and (2, 3) are proved positive by studying yet another decomposition of $\mathbf{C}_2$:

$$\mathbf{C}_2 = \frac{1}{q_3}[(\mathbf{I}+\mathbf{E})(\mathbf{I}+\mathbf{B}_2) - \mathbf{I}]\mathbf{B}_2^{-1} \quad \text{where } \mathbf{E} = (\mathbf{I}+\mathbf{B}_2)^{q_3^{-1}} - \mathbf{I}.$$

From the previous discussions it is clear that $\mathbf{E} \in M$. Other exploited properties are that $(\mathbf{I}+\mathbf{E})$ and $(\mathbf{I}+\mathbf{B}_2)$ commute and that $\mathbf{B}_2$ is the upper left 2×2 part of $\tilde{\mathbf{B}}_2$. $\mathbf{K}_b$ is now expressed as

$$\mathbf{K}_b = \left[\mathbf{I}_3 + \begin{pmatrix} [(\mathbf{I}_2+\mathbf{E})(\mathbf{I}_2+\mathbf{B}_2) - \mathbf{I}_2]\mathbf{B}_2^{-1} & \mathbf{0} \\ \mathbf{0} & q_3 \end{pmatrix}\tilde{\mathbf{B}}_2\right]^{-1}$$

and $\mathbf{K}_b$ is computed symbolically by Maple in terms of q_3 and the elements of $\mathbf{E}$ and $\tilde{\mathbf{B}}_2$.

It is proved that the elements of $\mathbf{K}_b$ in positions (1, 3) and (2, 3) are nonnegative by using the facts that $\mathbf{E}$ and $\tilde{\mathbf{B}}_2$ are M-matrices and that $(\mathbf{I}+\mathbf{E})$ and $(\mathbf{I}+\mathbf{B}_2)$ commute.

This completes the presentation of the proof of Theorem 2. Full details can be found in [11].

5. Discussion. An obvious direction for the generalization of the results in this paper is the following.

CONJECTURE. *The discretization of* (3.1) *by the backward Euler multirate formula* (2.1) *is stable for all* $h > 0$ *and integer* $q_2, q_3, \cdots, q_m > 0$ *if* $-\mathbf{A}_b$ *is of class M where* $\mathbf{A}_b$ *is defined in* (3.3).

Numerical experiments trying to invalidate this conjecture have been conducted but so far without success. However, the proof technique used for Theorem 2 does not seem applicable to a general proof for an arbitrary value of m since it exploits properties of 2×2 matrices, e.g., that the product of two M-matrices is a Z-matrix. Besides, the elements of $\mathbf{K}_b$ are computed and positivity is established for each individual element. The problem is that although $\mathbf{K}_b^{-1}$ possesses some of the key properties of M-matrices, $\mathbf{K}_b^{-1}$ is in general **not** of class M.

Another obvious generalization is to include backward differentiation methods of order two. No attempts have been made in this direction since the dimension of the companion matrix is doubled compared with the method of order one.

The logarithmic norm, which is used for constructing $\mathbf{A}_b$, depends on the underlying norm, and depending on the choice of this norm, the theorems in this paper predict the correct stability properties of a discretization. It should be noted that the theorems give sufficient conditions for stability.

The logarithmic norm is related to the spectral abcissa as $\alpha(\mathbf{A}) \leqq \mu(\mathbf{A})$ by (3.5), and for the sake of the stability theorems, the logarithmic norm should be as close to, preferably equal to, the spectral abcissa as possible. This can by obtained by choosing the underlying norm of the logarithmic norm as follows:

$$\|\mathbf{x}\|_T = \|T\mathbf{x}\|_\infty \quad \text{and} \quad \|\mathbf{A}\|_T = \|\mathbf{T}\mathbf{A}\mathbf{T}^{-1}\|_\infty. \tag{5.1}$$

If $\mathbf{T}$ diagonalizes $\mathbf{A}$, then $\mu_T(\mathbf{A}) = \alpha(\mathbf{A})$.

Let the norms chosen for the logarithmic norms of $\mathbf{A}_{11}, \mathbf{A}_{22}, \cdots, \mathbf{A}_{mm}$ in (3.3) be denoted by $\|\cdot\|_1, \|\cdot\|_2, \cdots, \|\cdot\|_m$. Then the norms used for the off-diagonal elements are defined as follows [7]:

$$\|\mathbf{A}_{ij}\|_{ij} = \sup_{\mathbf{x}_j \neq \mathbf{0}} (\|A_{ij}\mathbf{x}_j\|_i)/\|\mathbf{x}_j\|_j. \tag{5.2}$$

The central condition of the stability theorems is $-\mathbf{A}_b \in M$, and this is only fulfilled if $\mathbf{A}_b$ has negative diagonal elements. This can be obtained if $\alpha(\mathbf{A}_{ii}) < 0$ by choosing the underlying matrix norms of μ according to (5.1). If possible, μ_i should be chosen such that $\mu_i(\mathbf{A}_{ii}) = \alpha(\mathbf{A}_{ii})$.

Section 4 offers the following definition of an M-matrix (among several). $\mathbf{A} \in Z$ and all principal minors of $\mathbf{A}$ are positive. The basic structure of $\mathbf{A}_b$ as defined in (3.3) implies $-\mathbf{A}_b \in Z$. If $\mu_i(\mathbf{A}_{ii}) < 0$ for all i, then the principal minors of order 1 of $-\mathbf{A}_b$ are positive. The higher-order principal minors also depend on the off-diagonal elements of $\mathbf{A}_b$ as computed by the norms in (5.2), and these norms are induced by the norms of the diagonal blocks.

Basing $\mathbf{A}_b$ on logarithmic norms where $\mu_i(\mathbf{A}_{ii}) = \alpha(\mathbf{A}_{ii})$ is expected to lead to sharper stability theorems in general. However, it will still be possible to construct systems of differential equations with stable discretizations without stability being guaranteed by Theorem 1 or 2.

REFERENCES

[1] E. LELARASMEE, A. E. RUEHLI, AND A. L. SANGIOVANNI-VINCENTELLI, *The waveform relaxation method for time-domain analysis of large scale integrated circuits*, IEEE Trans. Computer-Aided Design Integrated Circuits Systems, 1 (1982), pp. 131-145.

[2] A. E. RUEHLI, ED., *Circuit Analysis, Simulation and Design, Part* 2, North-Holland, Amsterdam, 1987.

[3] S. SKELBOE, *Stability properties of backwards differentiation multirate formulas*, Applied Numerical Mathematics, Special Issue on Stability and Error Control for Ordinary Differential Equations, 5 (1989), pp. 151-160.

[4] ———, *Stability properties of implicit linear multirate formulas*, in Proc. ECCTD 87, Paris, September 1-4, 1987, pp. 795-800.

[5] C. A. DESOER AND H. HANEDA, *The measure of a matrix as a tool to analyze computer algorithms for circuit analysis*, IEEE Trans. Circuit Theory, 19 (1972), pp. 480-486.

[6] G. DAHLQUIST, *On matrix majorants and minorants, with applications to differential equations*, Linear Algebra Appl., 52/53 (1983), pp. 199-216.

[7] T. STRÖM, *On logarithmic norms*, SIAM J. Numer. Anal., 12 (1975), pp. 741-753.

[8] M. FIEDLER AND V. PTAK, *On matrices with non-positive off-diagonal elements and positive principal minors*, Czech. Math. J., 12 (1962), pp. 382-400.

[9] B. W. CHAR, K. O. GEDDES, G. H. GONNET, AND S. M. WATT, *Maple User's Guide*, WATCOM Publication, 1985.

[10] J. D. LAMBERT, *Computational Methods in Ordinary Differential Equations*, John Wiley, New York, 1973.

[11] P. ULFKJAER ANDERSEN, *Aspects of the stability properties of implicit multirate formulas*, M.Sc. thesis, DIKU, Department of Computer Science, University of Copenhagen, Copenhagen, Denmark, 1988.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 1010-1023, September 1989
© 1989 Society for Industrial and Applied Mathematics
011

THE EFFECT OF CHANGING THE STEPSIZE IN LINEAR MULTISTEP CODES*

L. F. SHAMPINE† AND P. BOGACKI†

Abstract. In the usual convergence theory for linear multistep methods, a constant stepsize h is used. For a method of order p, the discretization error is proportional to h^{p+1}. In a variable step code, it is necessary to predict what the discretization error would be if the stepsize were changed to rh. It is usual to say that the observed error will be altered by a factor of r^{p+1}. Unfortunately this is not correct for multistep methods. The discrepancy arises in the fact that the usual theory does not model the way variable stepsize codes actually work. In this paper the correct behavior is determined for important classes of formulas and ways of changing stepsize.

Key words. ODE, multistep methods, discretization error, changing stepsize

AMS(MOS) subject classification. 65L05

C.R. classification. 5.17

1. Introduction.

We consider the approximation of the solution $y(x)$ of an initial value problem for a system of ordinary differential equations

$$y' = f(x, y), \qquad a \leqq x \leqq b,$$
$$y(a) \quad \text{given},$$

by a linear multistep method (LMM). In the classical convergence theory an integer N is chosen, a stepsize h is defined as $h = (b-a)/N$, and approximations are formed to $y(a+jh)$ for $j = 0, 1, \cdots, N$. An integration with stepsize h is regarded as one member of a sequence of integrations with the stepsize tending to zero. The discretization error of a formula measures how well it approximates the behavior of $y(x)$ locally. For a method of order p the discretization error is proportional to h^{p+1}. This implies that if the stepsize is changed from h to rh, the discretization error is changed by a factor of r^{p+1}.

Early codes used a constant stepsize, as in the classical model, but the advantages of changing the stepsize were quickly realized. Now a user provides the code with a tolerance. If the discretization error in a step is greater than this tolerance, the step is rejected, and another try is made with stepsize rh with $r < 1$. If the error is much smaller than the tolerance, it is predicted that a bigger stepsize could be used. The object is to approximate $y(x)$ throughout $[a, b]$, so increasing the stepsize from h to rh with $r > 1$ makes the integration more efficient. All the popular codes estimate the error made at each step and then deduce an appropriate value of r.

For methods that use previously computed solution values, as LMMs normally do, the effect of changing from h to rh in a code is different from that described earlier. This is unfortunate because the codes assume this behavior in their stepsize selection algorithms. The behavior we described earlier is what happens when a new stepsize rh is used *throughout* the integration. In a code the stepsize h is used up to the current x, and we ask what happens when we change to a new stepsize rh for the next step. The classical theory does not even discuss *how* to make such a change, much less its effect on the discretization error. Our goal in this paper is to investigate the effect of

* Received by the editors July 11, 1988; accepted for publication (in revised form) January 17, 1989.
† Mathematics Department, Southern Methodist University, Dallas, Texas 75275.

changing the stepsize on the discretization error of the formula. Our assumptions about stepsize changes are those of popular codes.

There are two approaches to changing the stepsize seen in popular codes. We find that they behave similarly, but not identically. It turns out that the effect of a change of stepsize is qualitatively different from what is expected in the LMM codes, but that with one important exception, this does not appear to cause serious trouble. The exception is when the stepsize is reduced greatly. From experiment the authors of some popular codes were aware that conventional stepsize selection algorithms are inefficient in this situation, and they found that a restart was a reasonable way to deal with the matter. Now we can understand what the difficulty with the discretization error is, and we can explain why a restart resolves the difficulty.

2. The task. We consider the approximation of the solution $y(x)$ of an initial value problem for a system of ordinary differential equations

$$y' = f(x, y), \qquad a \leqq x \leqq b,$$
$$y(a) \quad \text{given.}$$

Because there is no important difference in the matters we take up, we simplify the notation by working with a single equation rather than a system. Values y_j approximating $y(a+jh)$ will be computed by a linear multistep method (LMM) with stepsize h. Such a method defines a value y_{n+1} in terms of previously computed values $y_n, y_{n-1}, \cdots$ by a formula of the form

$$\sum_{j=-1}^{k} \alpha_j y_{n-j} = h \sum_{j=-1}^{k} \beta_j f(x_n - jh, y_{n-j}) \tag{2.1}$$

where $x_n = a + nh$. It is assumed that $\alpha_{-1} \neq 0$, $|\alpha_k| + |\beta_k| > 0$, and

$$\sum_{j=-1}^{k} \beta_j = 1.$$

How values $y_0, \cdots, y_{k-1}$ are obtained to start the integration is not relevant to the present investigation.

We restrict our attention to two classes of LMMs that are particularly important in practice. One class consists of those methods with $\beta_1 = \beta_2 = \cdots = \beta_k = 0$. We call such methods "BDF-like" methods because they represent a natural generalization of the backward differentiation formulas (BDFs). The other class consists of methods with $\alpha_1 = \alpha_2 = \cdots = \alpha_k = 0$. We call such methods "Adams-like" methods because they represent a natural generalization of Adams–Bashforth and Adams–Moulton formulas.

For sufficiently smooth $y(x)$, the discretization error E of the LMM (2.1) is

$$E = \sum_{j=-1}^{k} \alpha_j y(x_n - jh) - h \sum_{j=-1}^{k} \beta_j y'(x_n - jh).$$

A Taylor series expansion leads to

$$E = h^{p+1} c_{p+1} y^{(p+1)}(x_n) + \mathcal{O}(h^{p+2}) \tag{2.2}$$

when the formula is of order p. With our normalization of the coefficients, c_{p+1} here is the "error constant" of the method [5, p. 223]. Although the first codes performed integrations with a constant stepsize, the value of changing the stepsize was quickly recognized. When a different stepsize rh is used, it is seen from (2.2) that the discretization error

$$\mathscr{E} = (rh)^{p+1} c_{p+1} y^{(p+1)}(x_n) + \mathcal{O}(h^{p+2}),$$
$$\mathscr{E} = r^{p+1} E + \mathcal{O}(h^{p+2}). \tag{2.3}$$

This relationship has been the basis for selecting r in virtually all codes. However, it describes what happens when a different stepsize rh is *used*; it does not describe what happens when the stepsize is *changed*. The classical convergence theory for LMMs considers a particular integration with stepsize h as one member of a sequence of integrations with the constant stepsize being taken smaller and smaller. In contrast, a code might integrate with a stepsize h up to a point x_n and then change to a stepsize rh for the next step. In the classical theory the same stepsize is used throughout the integration so that the formula (2.1) can be used all the time. In contrast, a code must obtain, somehow, values $y^*_{n-1}, y^*_{n-2}, \cdots$ approximating $y(x_n - jrh)$ if the formula (2.1) is to be used to step from x_n with stepsize rh.

The first point we would make is that the issue raised here does not apply to methods with $k = 0$ because these methods make no use of $y_{n-1}, y_{n-2}, \cdots$, i.e., they are one-step methods. The result (2.3) is true for all one-step methods, which leads us to call it the "one-step hypothesis." This point is pertinent to codes based on LMMs because the BDF and Adams families that are so popular in practice make important use of members that are one-step formulas.

The discretization error is fundamental, but it is usually said that the codes estimate a related quantity, the local error. Here we do not investigate the relationship between these errors (which can be very complex when the order is varied). Correspondingly, we do not consider whether the error estimators in the codes are effective. Our investigation of the discretization error clarifies only one of the effects of changing the stepsize in codes.

Krogh [6] discusses a number of ways to change the stepsize, but only two are seen in popular codes. One that applies to all LMMs is to use interpolation to obtain the values needed if we are to continue the integration with the constant stepsize formula (2.1) and the new stepsize. We call this a "quasi-constant stepsize implementation," and for short refer to it as a QCSSI. The other approach is more special. The two most popular families, BDF and Adams, are based on polynomial interpolation. It is easy enough to define them for mesh points $x_{n+1}, x_n, x_{n-1}, \cdots$ that are not at a constant spacing. To keep down the cost of computing coefficients, it is typical that codes use a constant spacing as much as possible. For this reason the approach can be viewed as another way to change the stepsize. We call this a "fully variable stepsize implementation" of these families of methods and refer to it as an FVSSI.

3. Quasi-constant stepsize implementation (QCSSI). In this section we examine the behavior of the discretization error of an LMM on a change of stepsize in a quasi-constant stepsize implementation (QCSSI). Gear's well-known code DIFSUB [2] and its many descendants and relatives implement both the BDF and Adams-Moulton formulas in this way.

There are two situations to be considered that are quite distinct in principle: The user of a code specifies how much error is tolerable at each step. If the error in a step is too large, the step is rejected and repeated with a smaller stepsize. If the error in a step is too small, it is predicted that after accepting the step, it would be efficient to increase the size for the next step. To gain insight we analyze a number of limiting cases by analytical means. This is complemented with numerical investigations of some important formulas.

3.1. A failed step. In a QCSSI we are integrating with a constant stepsize h. Suppose that we have just computed $y_{n+1} \approx y(x_n + h)$ and found the discretization error E to be unacceptably large. We consider, then, the effect of changing to a stepsize rh with $r < 1$ to compute y^*_{n+1} with discretization error E^*. To accomplish a change of

stepsize, we need "fictitious" values

$$y^*_{n-j} \approx y(x_n - jrh), \quad y'^*_{n-j} \approx y'(x_n - jrh), \quad j = 0, 1, \cdots, k.$$

We use $y^*_n = y_n$, $y'^*_n = f(x_n, y_n)$, and determine the other values needed by interpolation. It is not necessary to approximate the $y'(x_n - jrh)$ in the case of BDF-like formulas, so we treat this case first. To approximate the $y(x_n - jrh)$, we use Lagrangian interpolation on $y_n, y_{n-1}, \cdots, y_{n-m}$ to obtain

$$y^*_{n-j} = \sum_{\tau=0}^{m} l_\tau(x_n - jrh) y_{n-\tau}, \qquad j = 1, \cdots, k.$$

Here $l_\tau(x)$ is a fundamental Lagrangian interpolating polynomial at the nodes $x_n, x_{n-1}, \cdots, x_{n-m}$.

A BDF-like formula has the form

$$\sum_{j=-1}^{k} \alpha_j y^*_{n-j} = rh \sum_{j=-1}^{0} \beta_j f(x_n - jrh, y^*_{n-j})$$

when the stepsize is rh and the fictitious values $y^*_{n-1}, \cdots$ are used. Substitution for the fictitious values shows how y^*_{n+1} depends on the values $y_{n-1}, \cdots$, that were actually calculated earlier in the integration:

$$\alpha_{-1} y^*_{n+1} + \alpha_0 y_n + \sum_{j=1}^{k} \alpha_j \left(\sum_{\tau=0}^{m} l_\tau(x_n - jrh) y_{n-\tau} \right) = rh\beta_{-1} f(x_n + rh, y^*_{n+1}) + rh\beta_0 f(x_n, y_n).$$

This implies that the discretization error is

$$\begin{aligned} E^* &= \alpha_{-1} y(x_n + rh) + \alpha_0 y(x_n) + \sum_{j=1}^{k} \alpha_j \left(\sum_{\tau=0}^{m} l_\tau(x_n - jrh) y(x_n - \tau h) \right) \\ &\quad - rh\beta_{-1} y'(x_n + rh) - rh\beta_0 y'(x_n) \\ &= (rh)^{p+1} c_{p+1} y^{(p+1)}(x_n) \\ &\quad - \sum_{j=1}^{k} \alpha_j \left[y(x_n - jrh) - \sum_{\tau=0}^{m} l_\tau(x_n - jrh) y(x_n - \tau h) \right] + \mathcal{O}(h^{p+2}). \end{aligned}$$

As might have been expected, the interpolation error

$$y(x_n - jrh) - \sum_{\tau=0}^{m} l_\tau(x_n - jrh) y(x_n - \tau h)$$

is of crucial importance. A standard result, after an obvious expansion about x_n, shows that this expression has the form

$$\frac{h^{m+1} y^{(m+1)}(x_n)}{(m+1)!} \prod_{\tau=0}^{m} (\tau - jr) + \mathcal{O}(h^{m+2}).$$

If $m > p$, we see that

$$\begin{aligned} E^* &= (rh)^{p+1} c_{p+1} y^{(p+1)}(x_n) + \mathcal{O}(h^{p+2}) \\ &= r^{p+1} E + \mathcal{O}(h^{p+2}). \end{aligned}$$

and the one-step hypothesis is valid. Unfortunately, we cannot count on $m > p$ past values being available, so we must, on practical grounds, restrict ourselves to $m \leqq p$.

An $m<p$ would reduce the order of the formula. For these reasons all the common codes take $m=p$, and we assume this from now on. We have then that

$$E^* = (rh)^{p+1} c_{p+1} y^{(p+1)}(x_n) - \frac{h^{p+1} y^{(p+1)}(x_n)}{(p+1)!} \sum_{j=1}^{k} \alpha_j \prod_{\tau=0}^{p} (\tau - jr) + \mathcal{O}(h^{p+2}),$$

$$E^* = E\left[r^{p+1} - \frac{1}{(p+1)!\, c_{p+1}} \sum_{j=1}^{k} \alpha_j \prod_{\tau=0}^{p} (\tau - jr) \right] + \mathcal{O}(h^{p+2}). \tag{3.1}$$

The analysis of Adams-like methods that have $\alpha_1 = \alpha_2 = \cdots = \alpha_k = 0$ is virtually identical. For such methods we need fictitious values

$$f^*_{n-j} \approx f(x_n - jrh, y(x_n - jrh)) = y'(x_n - jrh) \quad \text{for } j = 1, \cdots, k,$$

and we obtain them by Lagrangian interpolation to the previously computed values

$$f_{n-j} = f(x_n - jh, y_{n-j}) \quad \text{for } j = 0, \cdots, m.$$

Now the appropriate value of m is $p-1$. When we identify this case with a caret, a little calculation results in

$$\hat{E}^* = \hat{E}\left[r^{p+1} + \frac{r}{p!\, c_{p+1}} \sum_{j=1}^{k} \beta_j \prod_{\tau=0}^{p-1} (\tau - jr) \right] + \mathcal{O}(h^{p+2}). \tag{3.2}$$

It is obvious from (3.1) and (3.2) that the behavior on a change of stepsize is different from the one-step hypothesis. In the remainder of this section we try to understand just how different the behavior is.

For a variety of practical reasons, codes do not make "small" reductions of the stepsize. Nevertheless, we investigate $r \approx 1$ because it is so natural to hope that the one-step hypothesis will approximate the correct behavior in this case. Let us suppose that $r = 1 + \varepsilon$ for $|\varepsilon| \ll 1$. When $p \geqq k$, a little calculation shows that

$$E^* = r^{1+\nu} E + \text{h.o.t.} \tag{3.3}$$

where "h.o.t." is "higher order terms" and

$$\nu = p + \frac{1}{(p+1)!\, c_{p+1}} \sum_{j=1}^{k} j\alpha_j \prod_{\substack{\tau=0 \\ \tau \neq j}}^{p} (\tau - j).$$

Similarly, in the case of Adams-like methods,

$$\hat{E}^* = r^{1+\hat{\nu}} \hat{E} + \text{h.o.t.} \tag{3.4}$$

where

$$\hat{\nu} = p - \frac{1}{p!\, c_{p+1}} \sum_{j=1}^{k} j\beta_j \prod_{\substack{\tau=0 \\ \tau \neq j}}^{p-1} (\tau - j).$$

We have written (3.3) and (3.4) in such a way as to emphasize that the effect of reducing h to rh is qualitatively different from the one-step hypothesis (2.3). We have evaluated the coefficients ν for the BDF and $\hat{\nu}$ for the Adams–Moulton formulas of the stable orders. The "effective" order $\nu(\hat{\nu})$ turns out to be greater than p in every case. For the BDF, $\nu - p$ increases monotonically from 0.3 for $p=2$ to 3.2 for $p=6$. For the Adams–Moulton formulas, $\hat{\nu} - p$ increases monotonically from 0.3 for $p=3$ to 11 for $p=12$.

In general the correct behavior (3.3) or (3.4) is qualitatively different from the one-step hypothesis (2.3). For $r \approx 1$ the difference is small in absolute terms. More important is the fact that the effective order is greater than p in the cases of the BDF and Adams–Moulton formulas. This implies that the true error E^* is smaller than that predicted by the one-step hypothesis. The value of r selected using (2.3) will be smaller than necessary, but the step will succeed. In this range of r, the one-step hypothesis is safe for these formulas in a QCSSI.

Let us now investigate a "large" reduction of the stepsize. The important early code DIFSUB [2] allows a very large reduction, but current practice limits the reduction possible at any one try, e.g., $r \geqq 0.1$ is common. Suppose that a step of length h failed, another try of length $r_1 h$ failed, and we try yet again with a step of length $r_2 h$. For this last try, fictitious values approximating $y(x_n - jr_2 h)$ are formed by interpolating y_n and $y^*_{n-1}, y^*_{n-2}, \cdots$. The y^*_{n-j} were themselves formed by interpolating y_n and $y_{n-1}, y_{n-2}, \cdots$. By the uniqueness of polynomial interpolation, we get the same values approximating $y(x_n - jr_2 h)$ whether we reduce h directly to $r_2 h$ or indirectly by first reducing it to $r_1 h$ and then from $r_1 h$ to $r_2 h$. There may be repeated failures at x_n with the result that the code must resort to rh with r that are quite small. We have now seen that even though the code might limit the reduction at each try, the effect is the same as making a single "large" reduction.

Expanding (3.1) when $r \ll 1$ leads, for BDF-like methods, to

$$E^* = E\left[r^{p+1} + \frac{r}{(p+1)c_{p+1}} \sum_{j=1}^{k} j\alpha_j + \mathcal{O}(r^2)\right] + \mathcal{O}(h^{p+2}).$$

$$E^* = \mu r E + \text{h.o.t.} \tag{3.5}$$

where

$$\mu = \frac{1}{(p+1)c_{p+1}} \sum_{j=1}^{k} j\alpha_j.$$

Similarly, for Adams-like methods,

$$\hat{E}^* = \hat{\mu} r^2 \hat{E} + \text{h.o.t.} \tag{3.6}$$

where

$$\hat{\mu} = \frac{-1}{pc_{p+1}} \sum_{j=1}^{k} j\beta_j.$$

The results (3.5) and (3.6) have serious practical implications. They say that the one-step hypothesis (2.3) is too optimistic, and for p at all large, grossly so. A stepsize is selected to be about as large as possible while still being predicted to yield an acceptable error. The discretization error we discuss is only part of the error estimate computed in the codes. Still, the discrepancy between the one-step hypothesis and what we have found to be the correct behavior of the discretization error is so large when $r \ll 1$ that there will be many, unnecessary, step failures. The authors of the popular codes have coped with this without, apparently, understanding this source of difficulty. In the comments of DIFSUB, Gear says "... and try again unless there have [been] three failures. In that case the derivatives are assumed to have accumulated errors so a restart from the current values of Y is tried." Note that Gear's action cures the problem. His restart involves switching to a one-step method for which the one-step hypothesis is, of course, valid with the consequence that the predicted errors behave as the code expects. An interesting point here is that DIFSUB and later members of the Gear line of codes implement both the BDFs and Adams formulas. The same

tactics are used for both families of formulas, but we see from (3.5) and (3.6) that the formulas do not behave the same in this situation.

Shampine and Gordon [7, pp. 111ff.] raise the issue treated in this paper and implement a device in their suite of Adams codes intended to ameliorate the difficulties. However, they were not aware of this serious discrepancy in the case of a "large" stepsize reduction. Like Gear, they resort to a one-step formula, but they do so for a different reason. They wanted to construct a code that was provably stable with respect to variation of the stepsize and resorted to a formula without memory when a severe distortion of the mesh appeared necessary. Of course, they were aware from experiments that something must be done to make an Adams code robust in this situation. With hindsight we can see that resorting to a one-step formula is one way to bring the expected behavior of the discretization error into line with the observed behavior when $r \ll 1$.

For several families of formulas we have supplemented the limit analyses just presented with numerical computations. Let us define $m(r)$ by

$$E^* = m(r)E + \mathcal{O}(h^{p+2})$$

in (3.1), i.e., $m(r)$ is the term in brackets there, and similarly for (3.2). For any specific formula it is easy to evaluate $m(r)$ and so compare it to the one-step hypothesis of r^{p+1}. In the case of the Adams–Bashforth family this proves to be straightforward. The function $m(r)$ is positive for $r>0$ and increases monotonically between its limit behavior for $r \ll 1$ and $r \approx 1$. The cases of the Adams–Moulton and BDF are more interesting. In the range of orders considered, $m(r)$ has a single positive root in the interval (0.44, 0.96). One implication is that for r near this root, the analysis leading to (3.1) and (3.2) breaks down because we assume that higher order terms can be neglected. It is nice to have an unexpectedly small discretization error, but this does present a difficulty for the design of an effective stepsize selection algorithm. A more unpleasant observation is that for some of the higher order Adams formulas, there are r for which $|m(r)|>1$, which is to say that reducing the stepsize can lead to a *bigger* discretization error. Even for a modest reduction of the stepsize, the one-step hypothesis (2.3) can predict a behavior quite different from the true behavior (3.1) or (3.2). By way of an example, in Table 1 we provide the ratio $m(r)/r^{p+1}$ for the BDFs and Adams–Moulton formulas of the stable orders when $r=\frac{1}{2}$.

In [6] Krogh considers the change of stepsize for formulas written in terms of backward differences. In this context halving the stepsize is appealing. He presents a

TABLE 1
The ratio $m(r)/r^{p+1}$ for the BDF*s and Adams–Moulton formulas of the stable orders when $r=\frac{1}{2}$.*

p	BDF	AM
2	+0.25	+1.0
3	−2.75	0.0
4	−11.66	−3.9
5	−35.09	−15.6
6	−92.88	−46.3
7		−122.2
8		−302.7
9		−719.8
10		−1664.5
11		−3771.7
12		−8415.5

situation for which high order differences increase after halving the stepsize by interpolation. For this reason he warns that repeated step failures are possible and proposes modifications to ameliorate the difficulty. In [4, pp. 101ff] Hall gives an informal explanation for the phenomenon. These observations are closely related to the fact that $|m(\frac{1}{2})| > 1$ for the higher order Adams–Moulton formulas.

In codes, r is selected so that E^* will be a specified fraction of the error tolerance. The one-step hypothesis (2.3) makes it easy to find r. It is not so easy when we account properly for the effect of a change of stepsize, but in principle we could evaluate $m(r)$ from (3.1) or (3.2) and use a root finder to determine an appropriate r. It is not our goal here to propose a more effective scheme for predicting the discretization error after a failed step, rather to study the difficulty. It should be realized that stepsize algorithms usually require r to be rather less than one after a step failure; e.g., $r \leqq 0.9$ is common. Provided that a code can correctly estimate the discretization error, it can eventually find a stepsize that will succeed. The issue is more one of efficiency than reliability.

3.2. A successful step. When the step to $x_n + h = x_{n+1}$ is a success, the practical issue is to predict a suitable stepsize for continuing on from x_{n+1}. The discretization error for a BDF-like method on a step to x_{n+2} is

$$E^* = \sum_{j=-1}^{k} \alpha_j y(x_{n+1} - jh) - h \sum_{j=-1}^{0} \beta_j y'(x_{n+1} - jh)$$

$$= h^{p+1} c_{p+1} y^{(p+1)}(x_{n+1}) + \mathcal{O}(h^{p+2}) = E + \mathcal{O}(h^{p+2}).$$

This says that the observed error E in the step to x_{n+1} provides an estimate of the error that would be made in a subsequent step of length h. Obviously this is true of other LMMs as well. Now we ask what error would be made in a step from y_{n+1} of length rh.

It is usual to suppose that the one-step hypothesis is true, i.e., the discretization error E^* when the step from x_{n+1} is of length rh will satisfy

$$E^* \approx r^{p+1} E. \tag{3.7}$$

However, the arguments of the preceding section again lead to the result (3.1), and similarly for Adams-like methods to the result (3.2).

It is convenient that the mechanics of stepsize prediction are the same for both failed and successful steps. Later we shall see that this is not true for an FVSSI. Earlier we investigated $r \approx 1$ when the step is a failure. Now we realize that the same analysis applies when the step is a success. Indeed, we did not require $\varepsilon < 0$ earlier when we wrote $r = 1 + \varepsilon$ for just this reason. We commented then that codes do not make small reductions of stepsize after failures, and it is also true that they do not make small increases after successes. For these reasons the case $r \approx 1$ is not very interesting in practice. There is, however, one practical situation that it illuminates. Some codes, e.g., DE/STEP,INTRP [7], may decrease the stepsize a little after a successful step. Recall that for $r \approx 1$, the one-step hypothesis is qualitatively incorrect, but the difference is not important in practice.

Reflection about the role of previously computed approximate solutions in LMMs suggests that these methods will "look" most like one-step methods when the stepsize is increased greatly. For a variety of practical reasons, codes do not ordinarily permit a very large increase. The earliest codes in the line beginning with Gear's DIFSUB [2] permitted r as large as 10^4. Later codes in this line restrict r to 10 except in a

starting phase. An upper bound of 10 on r is not unusual in popular codes, but bounds of five or two are more common.

Expanding (3.1) when $r \gg 1$ leads for BDF-like methods to $E^* \sim \delta r^{p+1} E$, where

$$\delta = 1 + \frac{(-1)^p}{(p+1)!\, c_{p+1}} \sum_{j=1}^{k} \alpha_j j^{p+1}$$

and for Adams-like methods to $\hat{E}^* \sim \hat{\delta} r^{p+1} \hat{E}$, where

$$\hat{\delta} = 1 + \frac{(-1)^p}{p!\, c_{p+1}} \sum_{j=1}^{k} \beta_j j^p.$$

Even in this most favorable situation, the behavior is not that expected from the one-step hypothesis. It is perhaps worth comment that if we were to stay at x_{n+1} and repeatedly try to get on scale, the behavior would be what is expected because the effect of the factor δ ($\hat{\delta}$) cancels out then. Something along these lines is done by Gladwell [3] in finding an initial stepsize that is on scale.

Computation of the δ and $\hat{\delta}$ for the BDFs and the Adams–Moulton formulas of stable orders shows these factors to be smaller than one. This is important because it says that asymptotically the one-step hypothesis is pessimistic when increasing the stepsize. A stepsize selected by the conventional rule, then, will yield more accuracy than expected. This is perfectly acceptable. Indeed, one objection to attempting a "large" r at all is that limit arguments are scarcely credible, so that being conservative is appropriate. We have also studied numerically the behavior of $m(r)$ in

$$E^* = m(r)E + \mathcal{O}(h^{p+2}).$$

The limit analysis for $r \gg 1$ describes $m(r)$ well for surprisingly small r. The analysis for $r \approx 1$ and our computed results that $\nu > p$ for the BDFs and $\hat{\nu} > p$ for the Adams–Moulton formulas say that for "small" increases, the one-step hypothesis is a "little" optimistic. Numerical evaluation of $m(r)$ has shown that it can be a little bigger than r^{p+1} for some of these formulas with realistic increases of the stepsize, but for all practical purposes we can say that

$$m(r) \leqq r^{p+1}$$

for both the BDFs and Adams formulas. Because these are the formulas employed in popular codes like those of the Gear line, we conclude that the one-step hypothesis implemented in such codes will behave in an acceptable way when predicting increases of the stepsize.

4. Fully variable stepsize implementation (FVSSI). Because the BDFs and Adams methods are based on polynomial interpolation, they can be defined for irregular meshes. The well-known code EPISODE [1] implements both the BDFs and Adams–Moulton formulas in this way. Approximations y_j to $y(x_j)$ are sought on a mesh $x_0 < x_1 < \cdots$, and we define the (variable) stepsize $h_j = x_{j+1} - x_j$ for each j. Having reached x_n, a BDF defines an approximation y_{n+1} at x_{n+1} implicitly by letting $P(x)$ be the polynomial interpolating y_{n+1} and $y_n, \cdots, y_{n-k}$ and then requiring that

$$P'(x_{n+1}) = f(x_{n+1}, P(x_{n+1})). \tag{4.1}$$

If $P(x)$ is written in Lagrangian form, (4.1) becomes

$$\sum_{j=-1}^{k} \alpha_{j,n} y_{n-j} = h_n f(x_{n+1}, y_{n+1}).$$

Note that the coefficients $\alpha_{j,n}$ depend on the relative mesh spacing $h_n, h_{n-1}, \cdots$, so that there is an additional subscript. From interpolation theory it is found that the discretization error is

$$E = \sum_{j=-1}^{k} \alpha_{j,n} y(x_{n-j}) - h_n y'(x_{n+1})$$
$$= \frac{-y^{(k+2)}(\xi)}{(k+2)!} (x_{n+1} - x_n)^2 \prod_{j=1}^{k} (x_{n+1} - x_{n-j})$$

where $\xi \in [x_{n-k}, x_{n+1}]$. As in the case of constant stepsize we want to expand the derivative in a Taylor series about x_n, but we need to keep in mind what is small. Let us now define the span of the data $s_n = x_{n+1} - x_{n-k}$. Then

$$E = \frac{-y^{(k+2)}(x_n)}{(k+2)!} (x_{n+1} - x_n)^2 \prod_{j=1}^{k} (x_{n+1} - x_{n-j}) + \mathcal{O}(s_n^{k+3}). \tag{4.2}$$

In the case of a step failure, we repeat the step from x_n with a stepsize $h_n^* = rh_n$, $r < 1$. The discretization error then is

$$E^* = \frac{-y^{(k+2)}(x_n)}{(k+2)!} (x_{n+1}^* - x_n)^2 \prod_{j=1}^{k} (x_{n+1}^* - x_{n-j}) + \mathcal{O}(s_n^{k+3}).$$

Let us define $m(r)$ as

$$m(r) = \frac{(x_{n+1}^* - x_n)^2}{(x_{n+1} - x_n)^2} \prod_{j=1}^{k} \left(\frac{x_{n+1}^* - x_{n-j}}{x_{n+1} - x_{n-j}} \right) \tag{4.3}$$

so that

$$E^* \approx m(r) E. \tag{4.4}$$

The one-step hypothesis

$$E^* \approx r^{k+2} E \tag{4.5}$$

is used in FVSSI codes. It is harder to gain insight about the difference between (4.4) and (4.5) in the present context because we allow an arbitrary stepsize distribution. Here an inequality is helpful. If we substitute for x_{n+1}^* in (4.3), we are led to

$$m(r) = \frac{(rh_n)^2}{h_n^2} \prod_{j=1}^{k} \left(\frac{rh_n + x_n - x_{n-j}}{h_n + x_n - x_{n-j}} \right)$$
$$\geqq r^{k+2} \quad \text{for } r \leqq 1. \tag{4.6}$$

This says that for any choice of stepsizes $h_n, h_{n-1}, \cdots$, the one-step hypothesis (4.5) is too optimistic for a BDF on a reduction of stepsize. This means that a code based on (4.5) might experience step failures due to bad predictions. Just how bad a prediction can be is revealed by a limit analysis.

If $h_n^* \ll h_{n-1}$, then from (4.6) we have

$$m(r) \approx r^2 \prod_{j=1}^{k} \left(\frac{x_n - x_{n-j}}{x_{n+1} - x_{n-j}} \right) = \mu r^2.$$

It is easy to see that the coefficient μ satisfies $0 < \mu < 1$, but the important point is that this result states that

$$E^* \approx \mu r^2 E \tag{4.7}$$

rather than (4.5). Note that the power of r is different from that for the corresponding result (3.5) for a QCSSI. For our purposes this difference between the implementations is unimportant. The point is that for "large" reductions of stepsize, the observed behavior can be very different from that predicted by the one-step hypothesis (4.5).

The situation with Adams–Moulton formulas is much the same as with the BDFs. In this case a polynomial $P(x)$ interpolating $f(x_{n+1}, y_{n+1}), f(x_n, y_n), \cdots, f(x_{n-k}, y_{n-k})$ is integrated to obtain

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} P(t)\, dt$$

$$= y_n + h_n \sum_{j=-1}^{k} \beta_{j,n} f(x_{n-j}, y_{n-j}).$$

The discretization error $\hat{E}$ can be written for smooth $y(x)$ as

$$\hat{E} = \frac{-y^{(k+3)}(x_n)}{(k+2)!} I_D + \mathcal{O}(s_n^{k+4})$$

where

$$I_D = \int_{x_n}^{x_{n+1}} (x_{n+1} - t) \prod_{j=0}^{k} (t - x_{n-j})\, dt.$$

If the step from x_n is repeated with a stepsize $h_n^* = rh_n$, then

$$\hat{E}^* = \frac{-y^{(k+3)}(x_n)}{(k+2)!} I_N + \mathcal{O}(s_n^{k+4})$$

where

$$I_N = \int_{x_n}^{x_{n+1}^*} (x_{n+1}^* - t) \prod_{j=0}^{k} (t - x_{n-j})\, dt.$$

Thus

$$\hat{E}^* = \hat{m}(r) E + \mathcal{O}(s_n^{k+4})$$

where

$$\hat{m}(r) = I_N / I_D.$$

To bound $\hat{m}(r)$, let $t = u + x_n$ and define the nonnegative quantities w_i by

$$\prod_{j=0}^{k} (t - x_{n-j}) = \prod_{j=0}^{k} (u + x_n - x_{n-j}) = \sum_{i=1}^{k+1} w_i u^i.$$

Then

$$I_D = \int_0^{h_n} (h_n - u) \sum_{i=1}^{k+1} w_i u^i\, du = \sum_{i=1}^{k+1} \frac{w_i h_n^{i+2}}{(i+1)(i+2)}.$$

Similarly,

$$I_N = \sum_{i=1}^{k+1} \frac{r^{i+2} w_i h_n^{i+2}}{(i+1)(i+2)}.$$

For $r \leqq 1$, $I_N \geqq r^{k+3} I_D$, hence $\hat{m}(r) \geqq r^{k+3}$ and

$$\hat{E}^* \geqq r^{k+3} E + \mathcal{O}(s_n^{k+4}).$$

As with BDF we have found that the one-step hypothesis is optimistic when reducing the stepsize with an Adams-Moulton formula. It is also clear from the expressions derived that if $r \ll 1$, then with a suitable definition of $\hat{\mu}$, $\hat{m}(r) \approx \hat{\mu} r^3$ and $\hat{E}^* \approx \hat{\mu} r^3 \hat{E}$. Again the behavior differs from that of a QCSSI, but the important point is that the behavior is quite different from the one-step hypothesis that here would be $\hat{E}^* \approx r^{k+3} \hat{E}$.

There is an important distinction between the handling of a failed step and a successful one in a FVSSI. In the case of the BDFs, the discretization error of the step to x_{n+1} is the quantity E of (4.2). Now we are interested in continuing on to x_{n+2} with a stepsize $h_{n+1} = rh_n$ and usually $r \geqq 1$. The discretization error will be

$$E^* = \frac{-y^{(k+2)}(x_{n+1})}{(k+2)!}(x_{n+2} - x_{n+1})^2 \prod_{j=1}^{k} (x_{n+2} - x_{n+1-j}) + \text{h.o.t.}$$

Then

$$E^* = m(r)E + \text{h.o.t.}$$

where

$$m(r) = \left(\frac{x_{n+2} - x_{n+1}}{x_{n+1} - x_n}\right)^2 \prod_{j=1}^{k} \left(\frac{x_{n+2} - x_{n+1-j}}{x_{n+1} - x_{n-j}}\right). \tag{4.8}$$

What is different from the situation with a QCSSI is that the spacing of $x_n, x_{n-1}, \cdots, x_{n-k}$ for the step to x_{n+1} may not be the same as that of $x_{n+1}, \cdots, x_{n+1-k}$ pertinent to the step to x_{n+2}.

In the generality allowed, it is hard to relate the factor $m(r)$ of (4.8) to the one-step hypothesis that $E^* \approx r^{k+2}E$ used in the codes to select the stepsize increase factor r. We noted earlier that even with a FVSSI, the codes tend to work with constant stepsize. Because of this an interesting situation is when

$$r \geqq \frac{h_{j+1}}{h_j} \quad \text{for } j = n-k, \cdots, n. \tag{4.9}$$

(The case $j = n$ corresponds to the definition of r as $h_{n+1} = rh_n$.) This will hold for any $r \geqq 1$ if we are working at constant stepsize. It holds for all increases r that are at least as large as the largest increase in the current span of the data. With this assumption about r we have

$$\begin{aligned} x_{n+2} - x_{n+1-j} &= h_{n+1} + h_n + \cdots + h_{n+1-j} \\ &\leqq rh_n + \cdots + rh_{n-j} = r(x_{n+1} - x_{n-j}). \end{aligned}$$

This observation leads immediately to

$$m(r) \leqq r^{k+2},$$

which is to say that the one-step hypothesis is pessimistic in these circumstances for an increase of stepsize with a BDF. Of course the assumption (4.9) is true for "large" r, and it is easy to see from (4.8) that

$$m(r) \sim \mu r^{k+2} \quad \text{as } r \to \infty$$

with a suitable definition of μ. Although the matter is not as clear as in the case of a failed step, the one-step hypothesis is safe for "large" increases and in reasonable circumstances for smaller increases. To be sure that the hypothesis is safe, we need only delay increasing the stepsize until the code is working with a constant stepsize.

Unfortunately, it is not always the case that the one-step hypothesis is pessimistic for the BDFs in an FVSSI. If, e.g.,

$$r < \frac{h_{j+1}}{h_j} \quad \text{for } j = n-k, \cdots, n-1, \tag{4.10}$$

then

$$m(r) > r^{k+2}.$$

This situation is rather artificial, but it might occur during the starting phase as a code rapidly increases the stepsize from an initial guess that is much too small and then reduces the rate of increase to r as it approaches a value on scale.

Adams-Moulton methods in an FVSSI behave much like the BDFs do. The discretization error of a step to x_{n+2} is

$$\hat{E}^* = \frac{-y^{(k+3)}(x_n)}{(k+2)!} I_N + \text{h.o.t.}$$

where

$$I_N = \int_{x_{n+1}}^{x_{n+2}} (x_{n+2} - t) \prod_{j=0}^{k} (t - x_{n+1-j})\, dt.$$

Then

$$\hat{E}^* = \hat{m}(r)\hat{E} + \text{h.o.t.}$$

where

$$\hat{m}(r) = I_N / I_D.$$

Let $h_{n+1} = rh_n$ and suppose that (4.9) holds. In I_D let $t = u + x_n$ to get

$$I_D = \int_0^{h_n} (h_n - u) \prod_{j=0}^{k} (u + x_n - x_{n-j})\, du.$$

In I_N let $t = ru + x_{n+1}$ to get

$$I_N = r \int_0^{h_n} (h_{n+1} - ru) \prod_{j=0}^{k} (ru + x_{n+1} - x_{n+1-j})\, du.$$

As we saw earlier,

$$x_{n+1} - x_{n+1-j} \leqq r(x_n - x_{n-j})$$

so that

$$I_N \leqq r \int_0^{h_n} r(h_n - u) \prod_{j=0}^{k} r(u + x_n - x_{n-j})\, du = r^{k+3} I_D.$$

This implies that

$$\hat{m}(r) \leqq r^{k+3}$$

and that for r satisfying (4.9), the one-step hypothesis is pessimistic for Adams-Moulton methods. Just as with the BDFs, it is not always the case that the one-step hypothesis is pessimistic. Again, a choice of stepsizes satisfying (4.10) provides an example.

5. Summary. The discretization error E of a linear multistep method using a stepsize h is related to the error E^* when using stepsize rh by

$$E^* \approx r^{p+1} E. \tag{5.1}$$

Taken out of context, this behavior has been used in the codes to select r when repeating a failed step and when increasing the stepsize after a "very" successful step. This one-step hypothesis is valid for one-step methods but does not correctly describe the effect of a change of stepsize when a method has a memory.

A "small" change of stepsize results in a behavior qualitatively different from (5.1). This is not of much practical significance because the difference is small in an absolute sense, and LMM codes do not ordinarily make small changes of stepsize.

Reductions of the stepsize lead to a behavior that can be quite different from (5.1). Despite the complication of arbitrary mesh spacing, we showed that in a fully variable stepsize implementation (FVSSI) of the BDFs and Adams–Moulton formulas, the hypothesis (5.1) is always "optimistic." This means that the observed error will be larger than that predicted, with the consequence that the step might fail unless the stepsize is chosen conservatively. The matter is less clear with a quasi-constant stepsize implementation (QCSSI). Numerical studies have shown that this implementation behaves in a similar way in the cases of the BDFs and Adams formulas, but the detailed behavior is more complex. It can happen that the error actually gets worse when the stepsize is reduced. It can also happen that it is unexpectedly small. Firmer conclusions of broader applicability are possible for "large reductions," i.e., as r tends to zero.

A large reduction of the stepsize does occur in practice, and the correct behavior is then catastrophically different from (5.1) for both BDF-like and Adams-like methods. The behavior for the two ways of changing stepsize is different, but the important point is that the one-step hypothesis (5.1) is grossly optimistic. This is probably why codes have repeated step failures in this situation. The ad hoc device of restarting seen in popular codes is a remedy because restarting involves changing to a one-step formula for which the expected behavior (5.1) is true.

Fairly large increases of stepsize are seen in practice. A limit analysis says that the one-step hypothesis is pessimistic for BDFs and Adams–Moulton formulas. Numerical computations show that for all practical purposes, we may suppose that (5.1) is pessimistic whenever the stepsize is increased in a QCSSI of these formulas. The matter is more complex in a FVSSI. In useful circumstances we proved that (5.1) is pessimistic with this implementation, but we have also given examples to show that the contrary can be true.

REFERENCES

[1] G. D. BYRNE AND A. C. HINDMARSH, *A polyalgorithm for the numerical solution of ordinary differential equations*, ACM Trans. Math. Software, 1 (1975), pp. 71–96.

[2] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[3] 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.

[4] G. HALL AND J. M. WATT, EDS., *Modern Numerical Methods for Ordinary Differential Equations*, Clarendon Press, Oxford, 1976.

[5] P. HENRICI, *Discrete Variable Methods in Ordinary Differential Equations*, John Wiley, New York, 1962.

[6] F. T. KROGH, *Algorithms for changing the stepsize*, SIAM J. Numer. Anal., 10 (1973), pp. 949–965.

[7] L. F. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman, San Francisco, 1975.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 1024–1037, September 1989
© 1989 Society for Industrial and Applied Mathematics
012

A NEW ORDER SELECTION STRATEGY FOR ORDINARY DIFFERENTIAL EQUATION SOLVERS*

PETER TISCHER†

Abstract. The normal order selection strategy used by a variable order, variable stepsize ordinary differential equation solver is based on selecting the formula that can use the largest stepsize while meeting the accuracy constraint. This strategy does not take into account the stiffness or nonstiffness of the differential system. By considering how the stiffness of the differential system influences order selection, a new selection strategy is proposed that is biased in the presence of stiffness toward the lower-order formulas. This new order selection strategy has been incorporated into a version of a widely used ordinary differential equation solver LSODE. The modified solver is shown to be greatly more efficient for problems that the original solver cannot solve efficiently, namely, problems where the Jacobian of the differential system has some eigenvalues of large modulus close to the imaginary axis. On other problems that the unmodified solver solves efficiently, the modified solver still performs with comparable efficiency.

Key words. stiff, nonstiff, ordinary differential equations, order selection

AMS(MOS) subject classifications. 65L05, 65L20

1. Introduction. Codes using the backwards differentiation formulas (BDFs) are generally quite efficient for solving stiff ordinary differential equations (ODEs) but can perform poorly for problems where the Jacobian of the differential system has some eigenvalues that have a negative real part, large modulus, and are close to the imaginary axis. These difficulties have been ascribed to the relatively poor absolute stability regions of the higher-order BDFs in the vicinity of the imaginary axis, and a considerable amount of effort has been expended in deriving formulas that have better absolute stability properties than the BDFs [4]. Linear multistep formulas with order greater than two cannot be A-stable [2]. Methods other than linear multistep formulas can be A-stable with order greater than two, but these methods must do more work per step and codes based on these methods have not yet become widely available.

By considering when a formula of a particular order will be selected by a variable order, variable stepsize code, we propose a new measure of a formula's absolute stability properties, and investigate strategies for order selection that enable BDF-based codes to solve more efficiently problems where some eigenvalues of the Jacobian are large in modulus and close to the imaginary axis. We have implemented these strategies in modified versions of the widely available BDF-based code LSODE [6] and tested the modified codes on a number of problems, including problems where LSODE experiences stability difficulties.

2. Order and stepsize selection. Most codes that use linear multistep formulas (LMFs) for solving ODEs will choose the order of formula and stepsize in a similar manner, irrespective of whether the code is trying to solve stiff or nonstiff ODEs. The codes assume that when taking a step of size h with a pth-order LMF, the error will be dominated by $\|k_{p+1}h^{p+1}y^{(p+1)}(x_{n+1})\|$, where k_{p+1} is an error coefficient characteristic of the formula used, and $y^{(p+1)}(x_{n+1})$ is the $(p+1)$st derivative of the true solution of the differential system at the new point x_{n+1}. The codes choose an appropriate h for the formula by requiring that this error expression is less than some safety factor multiplied by a user supplied error tolerance.

* Received by the editors July 11, 1988; accepted for publication (in revised form) January 17, 1989.
† Department of Computer Science, Monash University, Clayton, 3168 Australia.

Most LMF codes implement a family of formulas at different orders. Given two LMFs, one pth order, and the other $(p+1)$st order, the code will choose between the formulas by comparing estimates of $\|k_{p+1}h^{p+1}y^{(p+1)}(x_{n+1})\|$ and $\|k_{p+2}h^{p+2}y^{(p+2)}(x_{n+1})\|$, where k_{p+1} and k_{p+2} are the error coefficients of the pth and $(p+1)$st formulas, respectively. As h gets arbitrarily small, the higher-order formula will provide greater accuracy. The codes try to pick the largest steplength h that enables the accuracy criterion to be met, and in the solution of stiff equations we are interested in taking steps with very large stepsizes. It is then interesting to ask under what conditions a code will choose the lower-order formula since lower-order formulas can have better stiff stability properties than higher-order formulas.

In the case of the linear problem $y' = \lambda y$, where λ is a constant, the dominant term for the error expansion for the lower-order formula will be smaller than the dominant term in the error expansion for the higher-order formula if

$$|\lambda h| > \left|\frac{k_{p+1}}{k_{p+2}}\right|.$$

One widely used code that uses the BDFs and is considered to be at least as efficient as any other code using the BDFs is the code LSODE due to Hindmarsh [6]. In the case of LSODE, there are different safety factors involved in changing the order up or down. The corresponding figures for changing the order up or down suggested by our model given that the LSODE safety factors are being used are given by Table 1.

TABLE 1
Order change values of λh for the BDF*s given the use of* LSODE*'s safety factors.*

	Order				
	1	2	3	4	5
Change down to this order	$\|\lambda h\| \geqq 2.20$	$\|\lambda h\| \geqq 1.73$	$\|\lambda h\| \geqq 1.63$	$\|\lambda h\| \geqq 1.64$	$\|\lambda h\| \geqq 1.69$
Change up from this order	$\|\lambda h\| \leqq 1.18$	$\|\lambda h\| \leqq 0.733$	$\|\lambda h\| \leqq 0.639$	$\|\lambda h\| \leqq 0.507$	$\|\lambda h\| \leqq 0.414$

Using this model as a guide, let us consider what happens when a code like LSODE is used to solve an ODE. If λh is small in magnitude, the problem is nonstiff and LSODE quickly changes to its highest-order formula. Many stiff probems have an initial transient region where the problem is locally nonstiff. As the problem becomes locally stiff, we would expect the stepsize in the stiff region to increase and that the code would automatically select a lower order formula, eventually bringing the order down to the lowest-order formula, order one. The first- and second-order BDFs are A-stable and their region of absolute stability includes the whole left complex halfplane.

This approach suggests that in a variable order, variable stepsize code, it is important to consider the absolute stability properties of the higher-order formulas when λh is within the half-disc centred at the origin and with radius r, $\{z: |z| < r, \text{Re}\,(z) \leqq 0\}$. For LSODE using the BDFs, the radius of interest is 1.73. If LSODE has managed to increase the stepsize so that $|\lambda h| > 1.73$, then the figures in Table 1 suggest that LSODE will lower the order to order one or two. These formulas are A-stable and the code should experience no stability difficulties.

DEFINITION. A formula for solving ODEs is said to have an *absolute stability radius r* if the absolute stability region for the formula applied to the problem $y' = \lambda y$ includes the half disc $\{z: |z| < r, \text{Re}\,(z) \leqq 0\}$.

Gupta [3] has considered the behaviour of the boundary of the absolute stability region of a linear multistep formula as applied to the linear model problem $y' = \lambda y$. He has shown that for a pth order linear multistep formula the behaviour of the boundary of the stability region near $\lambda h = 0$ depends on the signs of the error constants of the terms in the local truncation error expansion.

If the method is stable at the origin and $p \geqq 1$, then the boundary of the stability region will always include the origin $\lambda h = 0$. Let $y(x_n)$ denote the true solution at x_n of the linear model problem and let y_n denote the numerical solution at x_n. Then if

$$y(x_{n+1}) - y_{n+1} = k_{p+1}(h\lambda)^{p+1} y(x_{n+1}) + k_{p+2}(h\lambda)^{p+2} y(x_{n+1}) + \cdots$$

the boundary of the stability region will enter the left complex half plane immediately after leaving the origin if

- k_{p+2} is negative and $p \bmod 4 = 0$;
- k_{p+1} is negative and $p \bmod 4 = 1$;
- k_{p+2} is positive and $p \bmod 4 = 2$;
- k_{p+1} is positive and $p \bmod 4 = 3$.

The third-order BDF has $k_4 = 3/22$ while the fourth-order BDF has $k_6 = -356/3125$, and so both formulas actually have a zero radius of absolute stability.

In Table 2 we give the $A(\alpha)$-stability angles and absolute stability radii for the BDFs. These values were determined numerically. The absolute stability radii for the third- and fourth-order formulas as determined numerically confirm our assertion that these formulas have poor radii of absolute stability. The third- and fourth-order BDFs have large $A(\alpha)$-stability angles compared to the fifth- and sixth-order formulas. However, the absolute stability radii indicate that the third- and fourth-order BDFs may become unstable for arbitrarily small values of $|\lambda h|$ when $\mathrm{Re}\,(\lambda h) < 0$. Thus, these formulas may actually experience stability restrictions on the stepsizes they can use while solving problems that are locally nonstiff.

TABLE 2
$A(\alpha)$-stability angles and absolute stability radii for BDFs.

	Order					
	1	2	3	4	5	6
$A(\alpha)$ stability angle	90.0°	90.0°	86.0°	73.4°	51.8°	17.9°
Absolute stability radius	∞	∞	0.02	0.05	0.75	0.72

3. Some test results. To test our simple model we used LSODE to solve some constant Jacobian linear ODEs and traced the order used by the code. Our test problem was

$$y'(x) = A(y - g(x)) + g'(x)$$

where A is a constant matrix. This ODE has a constant Jacobian A. We chose A to be

$$U \begin{bmatrix} \beta_1 & -\beta_2 & 0 & 0 \\ \beta_2 & \beta_1 & 0 & 0 \\ 0 & 0 & \beta_3 & 0 \\ 0 & 0 & 0 & \beta_4 \end{bmatrix} U \quad \text{where } U = \frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix}.$$

The eigenvalues of A are $\beta_1 \pm \beta_2 i$, β_3 and β_4. We chose $g(x)$ to be either

$$g(x) = \begin{bmatrix} \cos(x) \\ \sin(x) \\ \tan(\cos(x)) \\ e^{\sin(x)} \end{bmatrix} \quad \text{or} \quad g(x) = \begin{bmatrix} 1-e^{-x} \\ 1+x \\ 1+x\,e^{-x} \\ (x+2)/(x+1) \end{bmatrix}.$$

The initial conditions were chosen so that the true solution was $y(x) = g(x) + e^{Ax}$. In the second case the solution in the stiff region is asymptotically linear and the code can increase the stepsize without bound while in the first case the solution in the stiff region is oscillatory and the stepsize in the stiff region is bounded by the accuracy constraint.

LSODE, using the option to approximate Jacobians numerically, was used to solve both problems at absolute error tolerances of 10^{-2}, 10^{-4}, and 10^{-6}. In the first case, the range of integration was chosen to be from zero to 10 while in the second it was chosen to be from zero to 1000. In Tables 3, 4, and 5 we list some representative results. More comprehensive results are available in Tischer [10]. For all problems β_3 was chosen to be -0.1 and β_4 was chosen to be -0.01. β_1 and β_2 were chosen to cause the Jacobian to have eigenvalues of $-0.1 \pm 0.1i$ (Table 3); $-500 \pm 500i$ (Table 4), and

TABLE 3
Oscillating solution in stiff region, eigenvalues $-0.1 \pm 0.1i$, -0.1, *and* -0.01.

10^{-2}			10^{-4}			10^{-6}		
Order	Step	$\lvert\lambda h\rvert$	Order	Step	$\lvert\lambda h\rvert$	Order	Step	$\lvert\lambda h\rvert$
1	1	0.0	1	1	0.0	1	1	0.0
2	3	0.0129	2	3	0.00123	2	3	0.000123
3	6	0.0258	3	12	0.00552	3	12	0.000174
4	36	0.0345	4	16	0.00845	4	20	0.00304
			5	26	0.0144	5	30	0.00432
						4	91	0.00645
						5	97	0.00665

TABLE 4
Oscillating solution in stiff region, eigenvalues $-500 \pm 500i$, -0.1, *and* -0.01.

10^{-2}			10^{-4} *			10^{-6} †		
Order	Step	$\lvert\lambda h\rvert$	Order	Step	$\lvert\lambda h\rvert$	Order	Step	$\lvert\lambda h\rvert$
1	1	0.0	1	1	0.0	1	1	0.0
2	3	0.198	2	3	0.0168	2	3	0.00167
3	6	0.293	3	6	0.0336	3	6	0.00334
2	18	0.932	4	10	0.0919	4	14	0.0376
1	21	1.89	5	20	0.222	5	27	0.0929
2	25	65.7	4	41	0.629	4	87	0.592
3	31	130.	3	46	0.932	3	92	0.834
2	53	176.	2	50	1.62	2	96	1.41
3	59	70.4	3	56	5.68	3	102	8.43
2	70	191.	4	67	33.7	4	110	15.9

* At 10^{-4}, 123 more steps and nine more order changes were required to reach the end of the interval.
† At 10^{-6}, 318 more steps and 25 more order changes were required to reach the end of the interval.

TABLE 5
Oscillating solution in stiff region, eigenvalues $-10 \pm 1000i$, -0.1, *and* -0.01.

10^{-2} ‡			10^{-4} §			10^{-6} ¶		
Order	Step	$\lvert\lambda h\rvert$	Order	Step	$\lvert\lambda h\rvert$	Order	Step	$\lvert\lambda h\rvert$
1	1	0.0	1	1	0.0	1	1	0.0
2	3	0.170	2	3	0.0167	2	3	0.00167
3	6	0.277	3	6	0.0333	3	6	0.00333
4	10	0.401	4	10	0.0879	4	14	0.0395
			5	20	0.202	5	24	0.0787
			4	1,325	0.800			
			3	1,330	0.150			
			4	1,334	0.299			
			3	1,351	0.613			
			2	1,364	0.583			
			3	1,367	0.487			
			4	4,759	0.304			

‡ At 10^{-2}, 19,729 more steps were required after the last order change to reach the end of the interval.
§ At 10^{-4}, 16,170 more steps were required after the last order change to reach the end of the interval.
¶ At 10^{-6}, 15,689 more steps were required after the last order change to reach the end of the interval.

$-10 \pm 1000i$ (Table 5). The tables list results for the three tolerances and lists the number of the step and the value of $|\lambda h|$ for the dominant eigenvalue λ at each change of order.

For the results reported in Table 3 the Jacobian of the ODE had no eigenvalues that were large in magnitude and these problems would normally be regarded as nonstiff. For all tolerances the code continually changed the order up and used the highest order available. The value of $|\lambda h|$ remained small and the code persisted with the higher-order formulas as predicted by our model.

For Table 4 the Jacobian of the differential system had a pair of complex eigenvalues $-500 \pm 500i$. For these problems the order rose in the initial phase. The code then persisted with a higher-order formula without changing order. As the stiff components were damped out, the value of $|\lambda h|$ increased and the order was lowered to order one or two. Having lowered the order as the stiffest component was damped out, the code then raised the order, possibly in response to the need to track the subdominant eigenvalue or forcing terms accurately. The order used in the stiff region tended to fluctuate while the code used a constant order for many steps in the nonstiff region. The fluctuations in the order during the stiff region may have been due to errors being reintroduced into the stiffest component and again influencing the order to change down.

For Table 5 the Jacobian of the differential system had a pair of eigenvalues $-10 \pm 1000i$ that were sufficiently close to the imaginary axis to cause LSODE to run into stability difficulties. Codes like LSODE that use the BDFs are expected to be very inefficient when solving such problems. In all cases the code began by raising the order to track the stiff components accurately. However, the code was never able to increase the value of $|\lambda h|$ sufficiently to change the order down to formulas which were A-stable. In all cases the code was trapped into using either the fourth- or fifth-order formula for very large numbers of steps. For the tolerance 10^{-4} the code did change down to second order but value of $|\lambda h|$ was 0.583, and the code changed up to order three at the next opportunity to change order.

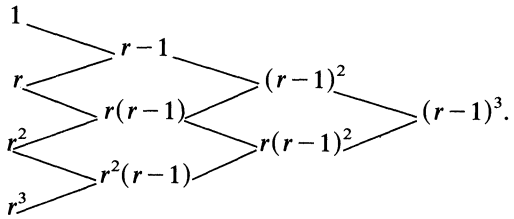
4. Order and stepsize selection—another model. In the previous section we proposed a simple model for understanding which formula a variable order, variable stepsize ODE solver using linear multistep formulas would select when solving a constant Jacobian linear ODE. This simple model can be criticised with regard to the following points:

(1) The simple model is based on behaviour of the true solution and not on behaviour of the numerical solution.

(2) The simple model assumes the code can use the derivative directly in stepsize estimation, whereas in general it has to approximate the derivative.

(3) The simple model assumes the dominant constraint on the stepsize is caused by the dominant eigenvalue of the Jacobian of the differential system. This may not be the case when the problem is locally nonstiff or when the stiff components have been largely damped out in the numerical solution.

We can try to construct what we hope will be a more accurate model of the order selection process by generalizing our approach in an attempt to accommodate some of these criticisms. Let us model the behaviour of the numerical solution by assuming that the numerical solution is varying at a constant rate from step to step. Without loss of generality let us represent the numerical solution generated using the constant stepsize h by $1, r, r^2, \cdots$. We then have the following relation holding for the higher-order backward differences:



From the diagram we can see that the ratio of successive backward differences at the same point is $(r-1)/r$. Let us assume that there have been enough constant sized steps for the pattern in the diagram to have been established. Suppose that a code estimates the pth higher derivative by the pth order backward difference at the current dependent variable value. That is, the pth higher derivative when the dependent variable equals r^q will be estimated to be

$$r^{q-p}(r-1)^p.$$

Let us assume that at the current stepsize, the lower-order formula, with order $p-1$ and error constant k_{p-1}, is estimated to be as accurate as the formula with order p and error constant k_p. Then we have

$$|k_{p-1}r^{q-p+1}(r-1)^{p-1}| = |k_p r^{q-p}(r-1)^p|. \tag{4.1}$$

For convenience, we consider only positive error constants and nonzero r in which case we have

$$\left|\frac{k_{p-1}}{k_p}\right| = \frac{|r-1|}{|r|}.$$

So far we have not assumed anything about the nature of the ODE. In the case of the linear model problem $y' = \lambda y$, r can be approximated by the dominant root of the characteristic polynomial of the linear multistep formula. It is a straightforward

exercise to plot values of $|(r-1)|/|r|$ for $\mathrm{Re}\,(h\lambda) \leqq 0$. In this graph large values indicate a large potential to change the order down. Note, for a stable formula $r \approx 1$ in the neighbourhood of the origin, $\lambda h = 0$, and $|(r-1)|/|r|$ is small, indicating that the higher-order formula will always be chosen in preference to the lower-order formula as λh gets arbitrarily small.

We will refer to $|(r-1)|/|r|$ as the *switching potential.* In Fig. 1 we give an example of a plot of the switching potential for the third-order BDF. Since the plot is symmetric we only graph the switching potential for the quadrant of the complex λh plane $\{\lambda h, \mathrm{Re}\,(\lambda h) \leqq 0, \ \mathrm{Im}\,(\lambda h) \geqq 0\}$. In Fig. 2 we give a similar plot for the fifth-order Adams-Moulton formula.

Examination of the plots of the switching potential for a number of formulas has shown that in those regions of the left complex halfplane where the numerical solution is a good approximation of the true solution, the switching potential contours are close to semicircular in shape. For example, the contour for 0.5 in both Figs. 1 and 2 is close

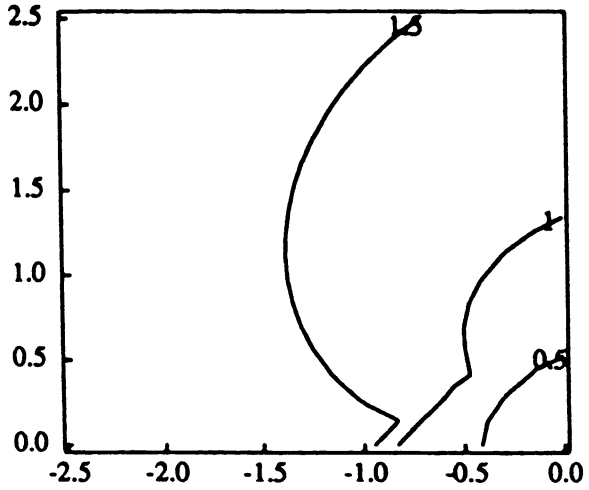


FIG. 1. *Switching potential plot for the third-order* BDF.

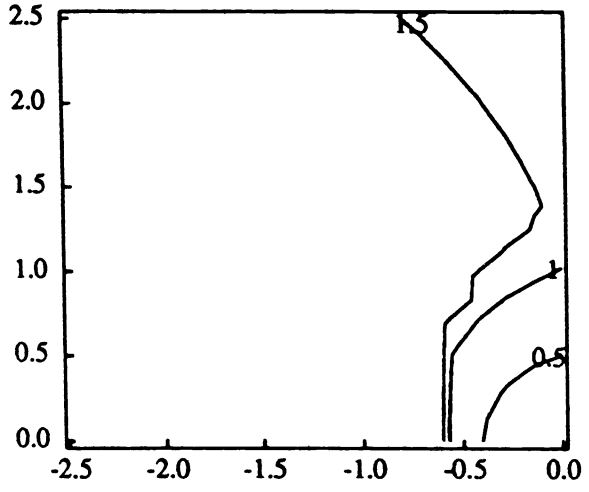


FIG. 2. *Switching potential plot for the fifth-order Adams-Moulton formula.*

to semicircular with a radius of 0.5. The fifth-order Adams-Moulton formula has a radius of absolute stability of 1.25, and the switching potential contours for 0.5 and 1 lie in the half-disc of absolute stability. The third-order BDF has a zero absolute stability radius, but for $|\lambda h| \leqq 0.55$ the third-order BDF is close to being absolutely stable in the sense that no root of the characteristic polynomial exceeds 1.01 in magnitude.

For λh close to the imaginary axis $e^{\lambda h}$ is close to 1 and the dominant root of the characteristic equation is likely to be the principal root, that root which approximates $e^{\lambda h}$. Along the negative real axis, the principal root decays and is unlikely to be the dominant root. In the vicinity of the negative real axis, the contours tend to come closer together and to come closer to the origin than they do as they approach the imaginary axis. This can be seen in the contours for 1.5 and 1 in both Figs. 1 and 2. Let us assume that the current stepsize being used by a code is h and the current order is p and that the code estimates that it can take steps of size R_{down}, R_{same}, and R_{up} for the formulas of order $p-1$, p, and $p+1$, respectively. Then under the assumptions we used to derive (4.1), we can obtain the following:

$$R_{\text{down}} = \left[\frac{k_p}{k_{p-1}} \frac{|r-1|}{|r|}\right]^{1/p} R_{\text{same}}^{(p+1)/p}, \tag{4.2}$$

$$S_{\text{down}} = \frac{R_{\text{down}}^{p} |k_{p-1}|}{R_{\text{same}}^{p+1} |k_p|} = \frac{|r-1|}{|r|}. \tag{4.3}$$

We can also find similar relationships linking R_{same} and R_{up}. An estimate of the switching potential provided by (4.3) is readily available in a variable stepsize, variable order code using linear multistep formulas, and this estimate may be used to influence the order selection strategy to shift the order down in the presence of stiffness.

5. A new order selection strategy. We have seen that for problems for which the BDFs remain absolutely stable, a BDF-code like LSODE will reduce the order as the stiffest components damp out. Seifert [8] has reported observing similar behaviour when testing the BDF-based ODE solver GEAR, which is an ancestor of LSODE, also written by Hindmarsh [5]. On some problems, particularly those where the Jacobian of the differential system has eigenvalues close to the imaginary axis, the code is not able to increase the stepsize sufficiently for the normal order selection process to force the order down and use formulas that are A-stable. Our considerations of the order selection process and our knowledge of the absolute stability properties of the BDFs suggest that if we were to modify the order selection process in LSODE to force it to switch the order down at smaller values of $|\lambda h|$, then the resulting code would exhibit better behaviour in the worst case and might be able to solve more problems efficiently than the original code. It might also be possible to achieve this increase in robustness without losing efficiency on problems that do not present difficulties for the original code.

Both the third- and fourth-order BDFs have a zero radius of absoulte stability, and thus for problems where the Jacobian of the differential system has eigenvalues sufficiently close to the imaginary axis the code may be trapped into using very small stepsizes, even if the problem appears locally nonstiff. The fifth- and sixth-order BDFs are absolutely stable for $|\lambda h| \leqq 0.72$, and we would expect no difficulties when using these formulas to solve problems that are locally nonstiff. The first- and second-order BDFs are A-stable, and we would expect no stability difficulties when using these formulas to solve problems that are locally stiff. This suggests that we should design

the order selection strategy to select fifth- or sixth-order if the problem appears locally nonstiff and use first or second order if the problem appears to be stiff. The third- and fourth-order formulas should be used sparingly, largely as a means of increasing the order in the nonstiff region and decreasing the order in the stiff region.

In our experiments we modified LSODE's normal order selection strategies to try to choose the order as given in Table 6.

TABLE 6
Order switching points for modified LSODE.

	Order				
	1	2	3	4	5
Change down to this order	$\lvert\lambda h\rvert \geqq 1.5$	$\lvert\lambda h\rvert \geqq 0.2$	$\lvert\lambda h\rvert \geqq 0.2$	$\lvert\lambda h\rvert \geqq 0.6$	$\lvert\lambda h\rvert \geqq 0.5$
Change up from this order	$\lvert\lambda h\rvert \leqq 0.8$	$\lvert\lambda h\rvert \leqq 0.2$	$\lvert\lambda h\rvert \leqq 0.2$	$\lvert\lambda h\rvert \leqq 0.6$	$\lvert\lambda h\rvert \leqq 0.6$

To accommodate the new order selection strategy we made a number of changes to LSODE to produce a code we call MLSODE1. The major changes were as follows.

Backward differences of the numerical solution were maintained and used to compute the stepsize change factors used in estimating the switching potential S_{down} and S_{up} (see (4.3)). The higher-derivative estimates used by LSODE seemed to be less effective in estimating S_{down} and S_{up} than the backward differences. This may be due to LSODE's derivative estimating formulas using values from more past points than the backward difference formulas use. By experiment we discovered that the geometric mean of the two estimates of the switching potential provided the most accurate estimate of the actual switching potential.

The estimate of the switching potential was used as an estimate of $|\lambda h|$, and the order of formula selected was chosen in accordance with Table 6. The stepsize for a given order was chosen by LSODE's normal means, and LSODE's order selection strategy was used after error failures, as in this case not enough backward differences had been collected for the new order section strategy.

We forced LSODE to take at least $2q+2$ constant sized steps with a formula of order q before considering a change of stepsize or order. Our model for order selection assumes constant sized steps and by taking this number of constant sized steps we are trying to limit the influence that stepsize changing might have on the backward differences. We decided to use the BDFs of orders 1-6, as both our order selection models indicate that the sixth-order formula would not cause stability problems for small values of $|\lambda h|$.

Skelboe [9] has also considered the behaviour of BDF-based codes on ODEs where the Jacobian has some eigenvalues close to the imaginary axis. He proposes a stability test that tries to detect terms in the numerical solution that are oscillatory and that grow in magnitude. If instability is detected by this test, the normal order selection strategy is bypassed in favour of a strategy that forces the order down. Skelboe's code implementing his strategy is considerably more efficient for problems where the Jacobian has some eigenvalues close to the imaginary axis than BDF-based codes that use the normal order selection process.

Our approach differs from Skelboe's in that we are trying to prevent instability in the numerical solution from arising in the first place. Our strategy relies only on estimating $|\lambda h|$ and does not try to determine whether λ is in some sense close to the

imaginary axis. Moreover, once an estimate of $|\lambda h|$ is available, it is natural to consider using this in a type-insensitive code to determine whether the ODE is locally stiff or nonstiff. Developments in this direction will be the subject of an uncoming paper.

5.1. Testing on linear, nonautonomous problems. We applied the new code MLSODE1 to the same test ODEs we had used in § 3. In each case Jacobians were approximated numerically, and the integration was aborted if it took more than 30,000 steps. For the test ODEs where the eigenvalues of the Jacobian were not close to the imaginary axis, the modified codes performed with comparable efficiency to LSODE without a clear trend emerging as to which code was most efficient. Results for these problems can be found in Tischer [10].

In Table 7 we present results for the test problem where the Jacobian of the differential system has a dominant eigenvalue pair that is at an angle of 89.4° to the negative real axis. The column labelled LSODE2 is LSODE with the maximum order set to two, i.e., LSODE only using formulas that are known to be A-stable. The code MLSODE2 is a version of MLSODE1 where the new order switching strategy is used only if the current order is three or four, and the estimate of the switching potential is at least as great as 0.25. We produced this code in order to test our conjecture that the third- and fourth-order BDFs are the most likely cause of difficulties when LSODE is used to solve problems where the Jacobian of the differential system has eigenvalues close to the imaginary axis. These problems can cause stability difficulties for LSODE if it uses any formula with order greater than two. If LSODE's maximum order is restricted to two then the code has difficulty meeting requests for high accuracy. For example, to solve the problem that has an asymptotically linear solution at a tolerance of 10^{-6} LSODE2 would require about 70,000 steps. The modified codes, however, are still able to solve these problems relatively efficiently.

5.2. Testing on nonlinear, nonautonomous problems. Our models of order selection behaviour for stiff problems involved linear, constant Jacobian, autonomous problems. MLSODE1 and MLSODE2 were able to solve linear, constant Jacobian, nonautonomous problems efficiently. To test these further we used a nonlinear ODE with a known solution [7, Problem 13]. This ODE has four equations and the eigenvalues of the Jacobian can be fixed by appropriate choices of the free parameters, β_1, β_2, β_3, and β_4. The eigenvalues of the Jacobian are $-(\beta_1+2)\pm\beta_2 i$, $-(\beta_3+2)$, and $-(\beta_4+2)$ at $t=0$ and are $-|\beta_1|\pm\beta_2 i$, $-|\beta_3|$, and $-|\beta_4|$ as $t\to\infty$. The interval of the integration was from $t=0$ to $t=1000$. The values of β used for the three test problems were $\beta_1=10$, $\beta_3=1000$, $\beta_4=0.01$ while β_2 took the values 100, 500, and 1000. For these values of β the dominant eigenvalues are a complex conjugate pairs where the angle formed with the negative real axis ranges from 83.2° to 84.3° ($\beta_2=100$); 88.6° to 88.8° ($\beta_2=500$); and 89.3° to 89.4° ($\beta_2=1000$). As β_2 increases the stiffness ratio also increases.

The nonlinear problem was solved for three tolerances: 10^{-2}, 10^{-4}, and 10^{-6}. The error control used was an equal weighting of absolute and relative error control as the solution varies greatly in magnitude over the interval of integration. The codes always approximated the Jacobian by numerical differencing.

The results of the numerical testing for test problems where $\beta_2=500$ and $\beta_2=1000$ are detailed in Tables 8 and 9. The results for the test problem where $\beta_2=100$ are not presented here as they follow the same trend as followed in the other test problems. The modified codes were again able to solve problems efficiently that LSODE could not solve efficiently due to stability difficulties and where LSODE2 could not meet the accuracy requirement efficiently.

TABLE 7
Performance of LSODE *and modified codes, Jacobian eigenvalues* $-10 \pm 1000i$, -0.1, *and* -0.01.

		Oscillatory				Linear			
		LSODE	LSODE2	MLSODE1	MLSODE2	LSODE	LSODE2	MLSODE1	MLSODE2
10^{-2}	Number of steps	19,739	762	809	772	30,000	2,254	2,284	3,032
	Number of functions	25,226	1,165	1,264	1,291	38,538	2,986	2,962	3,995
	Number of Jacobians	990	61	71	81	1,502	134	128	163
	Furthest x-value	10.0	10.0	10.0	10.0	22.61	1,000	1,000	1,000
10^{-4}	Number of steps	20,929	4,235	2,023	2,343	30,000	16,135	3,351	4,222
	Number of functions	26,914	5,795	2,786	3,217	38,579	20,549	4,296	5,429
	Number of Jacobians	1,055	254	130	139	1,504	836	184	226
	Furthest x-value	10.0	10.0	10.0	10.0	21.28	1,000	1,000	1,000
10^{-6}	Number of steps	15,713	21,089	4,391	4,714	30,000	30,000	7,249	8,035
	Number of functions	20,195	27,012	5,790	6,426	38,509	38,534	9,169	10,190
	Number of Jacobians	792	1,107	242	289	1,503	1,501	375	413
	Furthest x-value	10.0	10.0	10.0	10.0	17.48	0.22	1,000	1,000

TABLE 8
Results for problem with eigenvalues $-10 \pm 500i$ using mixed absolute and relative error criteria.

		LSODE	LSODE2	MLSODE1	MLSODE2
10^{-2}	Number of steps	30,000	2,661	3,057	2,380
	Number of functions	67,193	7,004	6,808	4,737
	Number of Jacobians	3,693	674	572	374
	Furthest time value	29.92	1,000	1,000	1,000
10^{-4}	Number of steps	30,000	11,196	6,152	6,129
	Number of functions	40,294	18,156	10,174	10,333
	Number of Jacobians	1,789	1,085	637	667
	Furthest time value	31.78	1,000	1,000	1,000
10^{-6}	Number of steps	30,000	30,000	8,510	8,969
	Number of functions	40,310	39,931	12,472	13,349
	Number of Jacobians	1,750	1,665	641	709
	Furthest time value	28.17	0.5752	1,000	1,000

TABLE 9
Results for problem with eigenvalues $-10 \pm 1000i$ using mixed absolute and relative error criteria.

		LSODE	LSODE2	MLSODE1	MLSODE2
10^{-2}	Number of steps	30,000	6,675	7,258	4,632
	Number of functions	67,866	17,432	16,113	9,397
	Number of Jacobians	4,093	1,696	1,364	753
	Furthest time value	13.19	1,000	1,000	1,000
10^{-4}	Number of steps	30,000	24,016	12,685	13,284
	Number of functions	42,711	40,025	21,055	22,202
	Number of Jacobians	2,119	2,507	1,332	1,427
	Furthest time value	17.88	1,000	1,000	1,000
10^{-6}	Number of steps	30,000	30,000	18,273	18,723
	Number of functions	43,044	38,956	27,197	27,900
	Number of Jacobians	2,145	1,530	1,452	1,479
	Furthest time value	11.55	0.3213	1,000	1,000

5.3. Results of testing on the stiff detest problem set. The stiff detest package of Bedet, Enright, and Hull [1] enables a code to be run against a battery of test problems. The problems in this test set have been critized as being atypical in that most problems are small systems of equations. Many of the problems have constant Jacobians and few problems have large stiffness ratios. Nevertheless, performance results for a large number of problem solvers on the stiff detest problem set are available, and the problem set does contain many problems that differ from the model problem in that they are nonlinear with a slowly varying Jacobian. The results of applying LSODE and the two modified codes to this problem set are detailed in Table 10. Results for problem B5 are presented separately since the Jacobian of this problem has a dominant eigenvalue pair of $10 \pm 100i$, and this problem is solved relatively inefficiently by BDF-based codes like LSODE.

The code MLSODE1 is competitive with LSODE at the tolerance of 10^{-3} and 10^{-6}, and for many problems at the most stringent tolerance it is more efficient than

TABLE 10
Performance of LSODE *and the modified codes on the Stiff Detest Problem Set.*

		All problems (except B5)			Problem B5		
		LSODE	MLSODE1	MLSODE2	LSODE	MLSODE1	MLSODE2
10^{-3}	Number of steps	3,075	3,732	3,571	2,301	223	205
	Number of functions	6,624	6,907	6,601	3,477	358	342
	Number of Jacobians	608	543	527	135	19	18
10^{-6}	Number of steps	8,425	10,327	9,438	2,576	496	507
	Number of functions	15,007	16,588	15,332	3,875	732	750
	Number of Jacobians	1,092	1,101	1,005	154	32	33
10^{-9}	Number of steps	22,361	32,079	21,175	3,554	1,893	1,541
	Number of functions	35,402	52,386	32,132	5,168	3,022	2,224
	Number of Jacobians	2,092	3,979	1,852	200	155	92
Overall	Number of steps	33,861	46,128	34,184	8,431	2,612	2,253
	Number of functions	57,033	75,879	54,065	489	4,112	3,316
	Number of Jacobians	3,792	5,623	3,384	12,520	206	143

LSODE because it can use the sixth-order BDF. However, for some problems at the most stringent tolerance it performs less efficiently than LSODE. In particular, for problem F4 it required 23,071 steps compared to LSODE's 8,552 steps. This may be due to a coding error but it may also be due to the order selection mechanism being deceived into lowering the order when high-order formulas are required to track the solution accurately. MLSODE2 was the more robust of the new codes and actually required few steps than LSODE to solve F4 at the most stringent tolerance.

6. Conclusions. We have considered the behaviour of the order selection process when a variable stepsize, variable order linear multistep code is applied to a stiff ODE. By consideration of a model problem, we have formulated an order selection strategy that tries to take the stiffness of the ODE into account. We have implemented this strategy in modified versions of a well-known BDF-based code and have shown how the modified codes perform more efficiently on problems that generally are solved very inefficiently by BDF-based codes. This strategy tries to avoid the use of the third- and fourth-order BDFs, and our results seem to indicate that it is these formulas that are the main cause of trouble when a BDF-based code experiences stability difficulties when the Jacobian of the differential system has eigenvalues close to the imaginary axis.

We believe our results suggest that BDF-based codes can be made significantly more robust and efficient. Further work needs to be done on testing the modified codes on problems where the Jacobian of the differential system has a number of eigenvalues close to the imaginary axis as the testing we have done has concentrated on problems where only the dominant eigenvalues of the Jacobian are close to the imaginary axis.

Our experience with the modified codes has lead to a number of ideas for a new code that would be a further improvement. Work is currently in progress on a code that is type-insensitive in that it tries to switch between two sets of formulas, depending on whether the program diagnoses the ODE as being locally nonstiff or stiff. An important feature of this code is that if the code diagnoses the ODE as being locally nonstiff, the formulas it uses all have relatively large radii of absolute stability.

Acknowledgment. This paper and the work described herein benefitted greatly from the comments of Professor Chris Wallace.

REFERENCES

[1] R. A. BEDET, W. H. ENRIGHT, AND T. E. HULL, STIFF DETEST: *A program for comparing numerical methods for stiff ordinary differential equations*, Tech. Report 81, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada 1975.

[2] G. G. DAHLQUIST, *A special stability property for linear multistep methods*, BIT, 3 (1963), pp. 27–43.

[3] G. K. GUPTA, *New multistep methods for the solution of* ODEs, Ph.D. thesis, Department of Computer Science, Monash University, Australia, 1975.

[4] G. K. GUPTA, R. D. SACKS-DAVIS, AND P. E. TISCHER, *A review of recent developments in solving* ODEs, ACM Comput. Surveys, 1 (1985), pp. 5–47.

[5] A. C. HINDMARSH, GEAR: *Ordinary differential equation system solver*, Report UICD-30001, Rev. 3, University of California, Lawrence Livermore Laboratories, Livermore, CA, 1974.

[6] ———, LSODE and LSODI, *Two new initial value ordinary differential equation solvers*, ACM-SIGNUM Newsletter, 15 (1980), pp. 10–11.

[7] F. T. KROGH, *On testing a subroutine for the numerical integration of ordinary differential equations*, J. Assoc. Comput. Mach., 20 (1973), pp. 545–562.

[8] P. SEIFERT, *Computational experiments with algorithms for stiff* ODE's, Computing, 38 (1987), pp. 163–176.

[9] S. SKELBOE, *The control of order and steplength for backward differentiation methods*, BIT, 17 (1977), pp. 91–107.

[10] P. E. TISCHER, *A new order selection strategy for ordinary differential equation solvers*, Tech. Report 102, Department of Computer Science, Monash University, Clayton, Australia, 1988.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 1038–1051, September 1989
© 1989 Society for Industrial and Applied Mathematics
013

VODE: A VARIABLE-COEFFICIENT ODE SOLVER *

PETER N. BROWN†, GEORGE D. BYRNE‡, AND ALAN C. HINDMARSH§

Abstract. VODE is a new initial value ODE solver for stiff and nonstiff systems. It uses variable-coefficient Adams-Moulton and Backward Differentiation Formula (BDF) methods in Nordsieck form, as taken from the older solvers EPISODE and EPISODEB, treating the Jacobian as full or banded. Unlike the older codes, VODE has a highly flexible user interface that is nearly identical to that of the ODEPACK solver LSODE.

In the process, several algorithmic improvements have been made in VODE, aside from the new user interface. First, a change in stepsize and/or order that is decided upon at the end of one successful step is not implemented until the start of the next step, so that interpolations performed between steps use the more correct data. Second, a new algorithm for setting the initial stepsize has been included, which iterates briefly to estimate the required second derivative vector. Efficiency is often greatly enhanced by an added algorithm for saving and reusing the Jacobian matrix J, as it occurs in the Newton matrix, under certain conditions. As an option, this Jacobian-saving feature can be suppressed if the required extra storage is prohibitive. Finally, the modified Newton iteration is relaxed by a scalar factor in the stiff case, as a partial correction for the fact that the scalar coefficient in the Newton matrix may be out of date.

The fixed-leading-coefficient form of the BDF methods has been studied independently, and a version of VODE that incorporates it has been developed. This version does show better performance on some problems, but further tuning and testing are needed to make a final evaluation of it.

Like its predecessors, VODE demonstrates that multistep methods with fully variable stepsizes and coefficients can outperform fixed-step-interpolatory methods on problems with widely different active time scales. In one comparison test, on a one-dimensional diurnal kinetics-transport problem with a banded internal Jacobian, the run time for VODE was 36 percent lower than that of LSODE without the J-saving algorithm and 49 percent lower with it. The fixed-leading-coefficient version ran slightly faster, by another 12 percent without J-saving and 5 percent with it. All of the runs achieved about the same accuracy.

Key words. stiff ODEs, variable-coefficient BDF methods

AMS(MOS) subject classifications. 65L05

1. Introduction.

In the early 1970s, one of the more heavily used ODE initial value solvers was the GEAR package [9], which uses (in the stiff case) a fixed-coefficient Backward Differentiation Formula (BDF) method. But it was found that GEAR was unable to cope with certain chemical kinetics problems that have sharp and frequent time variations in the coefficient functions and solutions. This motivated a project in which a variable-coefficient form of that BDF method was developed. The result was a solver called EPISODE [5], [13], [14], which was nearly identical to GEAR in external appearance, but quite different internally in the areas of coefficient evaluation and the estimation and control of local error. (In analogy with the nonstiff Adams method in GEAR, EPISODE was augmented with a variable-coefficient Adams method, so that nonstiff systems could be handled with a simple change of an input method

* Received by the editors on July 11, 1988; accepted for publication (in revised form) October 26, 1988. 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 supported by the Department of Energy Office of Energy Research, Applied Mathematical Sciences Research Program.

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

‡ Corporate Research, Exxon Research and Engineering Company, Annandale, New Jersey 08801 (gdbyrne@erenj.bitnet). The work of this author was supported by Exxon Research and Engineering Company.

§ Computing & Mathematics Research Division, L-316, Lawrence Livermore National Laboratory, Livermore, California 94550 (alanh@lll-crg.llnl.gov).

flag.) EPISODE was found to be more reliable on these difficult kinetics problems, but less efficient on smooth stiff problems, because of the frequent changing of the scalar coefficient in the Newton matrix [4]. Two variants were written soon afterward, mainly for the method-of-lines solution of PDEs with nonsmooth variations of the type EPISODE was designed to accommodate. These variants are EPISODEB, for systems with banded Jacobian matrices [6], and EPISODEIB, for problems in linearly implicit form with banded Jacobians, both in analogy to the GEAR variants GEARB and GEARIB.

In an independent development, the considerable experience and user feedback from the GEAR family of solvers motivated a joint effort among people in the U.S. Department of Energy Laboratories to redesign and improve some of the ODE initial value solvers being used heavily. The goal of the project was a systematized collection of IVP solvers, to be called ODEPACK. The initial phase was the development of a standard user interface for the solvers. This was accomplished [10], and the GEAR and GEARB packages were then rewritten as a single solver, LSODE [11]. While LSODE represents only modest improvements in the internal methods and algorithms of GEAR and GEARB, it reflects a much greater flexibility with respect to user controls and options, yet it is generally easier to use, it is much more portable, and it is far easier to install in a library environment. These properties of LSODE, and now also of the ODEPACK collection [12], have received virtually unanimous positive feedback from the user community.

VODE combines these two developments. That is, the EPISODE and EPISODEB solvers, with their fully variable-step methods, have been combined and rewritten as an initial value ODE solver called VODE, with a user interface that conforms to the standard developed for ODEPACK. Thus it has a highly flexible user interface nearly identical to that of LSODE, but it contains variable-coefficient methods that are more efficient on problems that require frequent and wide changes in stepsizes. In the process, several algorithmic improvements have also been made, as described below. In addition, we have studied an alternative approach, based on the fixed-leading-coefficient form of BDF methods [15], and have implemented these in a variant of VODE, described below.

2. Basic methods. A detailed mathematical description of the methods used in EPISODE, and hence in VODE, is given in [5], and will not be repeated here. But for the sake of completeness, a brief summary is given below.

We write the initial value problem as

$$\dot{y} = f(t, y), \quad y(t_0) = y_0, \quad y \in \mathbf{R}^N.$$

The basic linear multistep formulas for both the stiff and nonstiff cases have the form

$$\sum_{i=0}^{K_1} \alpha_{n,i} y_{n-i} + h_n \sum_{i=0}^{K_2} \beta_{n,i} \dot{y}_{n-i} = 0.$$

For use on nonstiff problems, the Adams formula is characterized by $K_1 = 1$ and $K_2 = q - 1$, and the order q varies between 1 and 12. For stiff problems, the BDF formula has $K_1 = q$ and $K_2 = 0$, and the order q varies between 1 and 5. The coefficients $\alpha_{n,i}, \beta_{n,i}$ are computed as functions of the current and past stepsizes $h_j = t_j - t_{j-1} (j = n - q + 1, \cdots, n)$. The past history is represented by the $N \times (q + 1)$ Nordsieck array,

$$z_n = [y_n, h_n \dot{y}_n, \cdots, h_n^q y_n^{(q)}/q!], \tag{2.1}$$

the scaled derivatives being those of the corresponding interpolating polynomial associated with the history data involved in the formula.

As with any implicit method, some iterative scheme must be applied at each step to solve a nonlinear system

$$G(y_n) \equiv y_n + h_n(\beta_{n,0}/\alpha_{n,0})f(t_n, y_n) + a_n = 0$$

for the advanced value y_n (a_n is a vector involving past values of y_k and $\dot{y}_k$). VODE offers a choice between functional iteration (where no matrices are involved) and a modified Newton iteration in which the Jacobian matrix $J = \partial f/\partial y$ is treated as either full or banded, and as either supplied by the user or approximated internally by difference quotients. In the Nordsieck representation, the predicted value of z_n is

$$z_{n(0)} = z_{n-1}A, \tag{2.2}$$

where A is the order q Pascal triangle matrix. The first two columns of this array are the predicted values of y_n and $h_n\dot{y}_n$, denoted $y_{n(0)}$ and $h_n\dot{y}_{n(0)}$, respectively. Then the nonlinear system takes the form

$$G(y_n) = y_n - y_{n(0)} - (h_n/\ell_1)[f(t_n, y_n) - \dot{y}_{n(0)}] \tag{2.3}$$

and the final correction to the Nordsieck array is

$$z_n = z_{n(0)} + [y_n - y_{n(0)}]\ell \tag{2.4}$$

in terms of a vector of coefficients $\ell = [\ell_0, \ell_1, \cdots, \ell_q] (\ell_0 = 1)$. The predictor and corrector have the same order, and in PEC notation the mode is $P(EC)^m$ with variable m.

Following a successful corrector iteration for y_n, the local error is estimated and tested. Regardless of its outcome, a change in stepsize is considered, either for the current step or the next one, depending on the error test. Periodically, a change in the order q is also considered, based on estimated local errors at orders $q-1$ and $q+1$. The details of these parts of the algorithm are given in [5].

3. Algorithmic improvements. In the course of rewriting the EPISODE and EPISODEB solvers as VODE, we took the opportunity to correct some of the known deficiencies in the algorithm, and also to introduce some improvements that have been inspired by the work of others. Below is a summary of the most significant of these changes, as far as they have been completed. More modifications of this type are in progress or are planned, and they will be reported later.

3.1. Step/order resetting. At the conclusion of a successful step, VODE selects a stepsize h' and an order q' for the next step, step $n+1$. Either or both of these may be the same as the values h and q used for step n. In the original EPISODE family and in LSODE, if either h or q is to be changed, the Nordsieck array z_n is adjusted accordingly (rescaled, and shortened or augmented) immediately following the decision to make the change. If interpolated values of $y(t)$, with t in (t_{n-1}, t_n), are called for and are then computed before proceeding, these may be based on the wrong interpolating polynomial and so be slightly in error, as was pointed out by Berzins [1]. On close inspection, one finds that EPISODE commits no such error if $q' = q+1$, because the augmented column in z_n is the zero vector. But in the case $q' = q-1$, the interpolant is incorrect by an amount proportional to the discarded

last column $h^q y^{(q)}/q!$. In particular, in the case of the Adams method of any order or the BDF method with $q = 2$ ($q' = 1$), the interpolant fails to be continuous at t_{n-1}.

This error is fairly easy to correct, and it has been corrected in VODE, by postponing the adjustment of z_n until step $n+1$ is about to be taken, after any interpolations have already been done. This entails the use of some extra flags in the internal state data, and careful logic to handle all the combinations of cases. These include stepsize and/or order changes for step $n + 1$ forced by the user's stopping conditions or a change in method parameters.

With this change, the interpolated values are guaranteed to be continuous (within roundoff error), even in the presence of iteration error in the solution for y_n. This can be seen by writing

$$z_n = z_{n-1}A + e_n\ell$$

and noting that by construction the vector ℓ satisfies

$$\sum_{j=0}^{q} \ell_j(-1)^j = 0.$$

The value of the interpolant $y(t_{n-1})$ obtained from $z_n = [z_n^0, \cdots, z_n^q]$ is

$$y(t_{n-1}) = \sum_{j=0}^{q} z_n^j(-1)^j.$$

It follows that this value is the same as that gotten from $z_{n-1}A$, and this is the same as that from z_{n-1}, namely, y_{n-1}. The presence of iteration error is reflected in the vector $e_n = y_n - y_{n(0)}$ and has no effect.

In general, the interpolant will not be C^1, and we have not made an attempt to make it C^1, as was done in [1]. Interestingly, in the case of an Adams method of order $q \geq 2$, the interpolant is always C^1, even with iteration error, by virtue of the identity

$$\sum_{j=1}^{q} j\ell_j(-1)^j = 0.$$

The fact that the mode is $P(EC)^m$, and so there is no final f evaluation, is critical to these results. For Adams methods in $PE(CE)^m$ mode, it is difficult to get even C^0 continuity.

3.2. Initial stepsize. In the initial version of VODE, an algorithm for selecting the initial stepsize was taken directly from the LSODE solver. It involves only the initial conditions, and the initial slope vector $\dot{y}_0 = f(t_0, y_0)$. Inspired by some of the work on this problem by Watts [20] and by Shampine [19], we have adopted a new algorithm, which uses somewhat more data. Both algorithms start from the following premise: The desired initial stepsize h should satisfy

$$\|h^2\ddot{y}/2\|_{\mathrm{WRMS}} = 1, \tag{3.1}$$

where WRMS denotes the weighted root-mean-square norm,

$$\|v\|_{\mathrm{WRMS}} = \sqrt{(1/N)\sum_{i=1}^{N}(v^i/w^i)^2}\ ,$$

with weights w^i defined in terms of the relative and absolute tolerances supplied by the user, and where the second derivative is evaluated at the initial point. The basis for this premise is that this equation is identical to the local error test to be made at the end of the first step, except that the second derivative is estimated by a difference expression. The difficulty with the initial step selection is that the initial second derivative is not readily available.

The algorithm in LSODE (and the other ODEPACK solvers) approximates the components of weighted first-order principal error function, $(1/2)\ddot{y}^i/w^i$, by the squares of the components of the weighted zeroth order principal error function, $\dot{y}^i/w^i$. An adjustment has to be made to protect against the case where the initial $\dot{y}$ vanishes.

The algorithm in VODE, like those in [20] and [19], makes a more genuine attempt to approximate the initial second derivative vector. This involves an iteration, as follows: If a guessed value $\bar{h}$ is available, then the initial second derivative is approximated as

$$\ddot{y} = [f(t_0 + \bar{h}, y_0 + \bar{h}\dot{y}_0) - \dot{y}_0]/\bar{h}.$$

This value is inserted into (3.1) from the error test and that equation is solved for h. This is the next guess.

To get an initial guess for the iteration, we form some simple lower and upper bounds. First, in terms of the machine unit roundoff u and the initial output time t_{out}, a reasonable lower bound is 100 times the roundoff level in the initial time t, or

$$h_L = 100u \max\{|t_0|, |t_{\text{out}}|\}.$$

For an upper bound, we first take a fraction of the first output interval,

$$h_U = 0.1\,|t_{\text{out}} - t_0|.$$

Then we reduce this value, if necessary, to guarantee that for every $i = 1, \cdots, N$ the corresponding change in y^i is not very large in either relative or absolute terms. Specifically we guarantee that h_U satisfies

$$h_U|\dot{y}_0^i| \le 0.1\,|y_0^i| + \text{ATOL}^i,$$

where ATOL is the input absolute tolerance array (or scalar). A generous relative change of 0.1 is used above, rather than the user input RTOL, because this zeroth order step control may be much more limiting than the first-order control for the same tolerances. The initial guess for the iteration is then taken as the geometric mean

$$\bar{h} = (h_L h_U)^{1/2}.$$

The convergence test on the iteration is quite loose, since we only need a crude approximation to the largest stepsize permissible on the first step. Thus we stop iterating if the new and previous values of h differ by less than a factor of 2. A maximum of four iterations is allowed. In the tests, no more that two have ever been observed. A few special situations are handled separately (such as when $h_L > h_U$ or the norm of $\ddot{y}$ is very small), and the proper sign is attached to h at the end of the algorithm. In our testing, we found that this algorithm is quite effective and yet quite efficient compared to other algorithms we have seen.

3.3. Jacobian saving. We have added to VODE a device that has been used by several other authors of ODE solvers, which greatly improves its efficiency on many, if not most, stiff problems. This is an algorithm for saving and reusing the Jacobian matrix J, as it occurs in the Newton matrix $P = I - \gamma J$ (I is the identity matrix and γ is the scalar h_n/ℓ_1). In the form used in VODE, this algorithm is essentially the same as what is used in LSODES, the sparse Jacobian variant of LSODE. There the matrix P and its LU factorization are necessarily stored separately, and so it is natural to reuse an old value of P rather than to recalculate it, if circumstances warrant this. A very similar algorithm is described in [8] and [18].

In both LSODES and VODE, a decision is made at the start of the step as to whether or not to update and refactor P. A decision to update may result from the step count (first step, or 20 steps taken since the last update), the change in γ (by 30 percent or more), or by a failure of the corrector iteration to converge on a previous attempt at the current step. If the decision is made to update, then a saved value of J is used instead of recalculating it, if either

(a) no convergence failure occurred on this step and J is less than 50 steps old, or

(b) a convergence failure occurred with an old J and the relative change in γ since the last update to P exceeds 0.2.

In both cases, the idea is to isolate the situations in which P requires updating because of changes in γ and not changes in J.

This feature in VODE differs from the corresponding feature of LSODES in two ways. First, the VODE version is simpler, in that it saves J directly, whereas LSODES saves P and must recover a new value $\hat{P} = I - \hat{\gamma} J$ from an old value $P = I - \gamma J$ (with careful consideration of roundoff effects). Second, VODE requires considerable extra storage for this feature, for the saved copy of J as well as the factors of P, whereas LSODES must separately store the two sparse matrix arrays anyway. A user with a large problem (even with an ordering that gives a minimal bandwith), may well be unable to afford that extra storage without expensive overhead. For this reason, VODE provides an option to suppress the J-saving feature, i.e., to force an evaluation of J whenever the decision is made to update P and to overwrite P on J (and the factors of P on P).

3.4. Linear system relaxation for stiff systems. When the ODE problem is stiff, the nonlinear system (2.3) typically must be solved using a modified Newton iteration. In this subsection, we discuss a relaxation idea due independently to Petzold [17] and Burrage, Butcher, and Chipman [3] for speeding up the convergence of the nonlinear iteration. A related relaxation factor, involving estimates of the extreme eigenvalues, is given in [16]. From (2.3), the nonlinear system has the general form

$$G(y) = y + a - \gamma f(t, y), \tag{3.2}$$

where $\gamma = h_n/\ell_1$, $a = y_{n(0)} - \gamma \dot{y}_{n(0)}$ and $t = t_n$. The Newton iteration with relaxation has the form

$$\begin{array}{l} \text{Solve } \hat{P} s(m) = -cG(y_{n(m)}) \\ \text{Set } y_{n(m+1)} = y_{n(m)} + s(m) \end{array} \quad (m = 0, 1, 2, \cdots), \tag{3.3}$$

where c is a scalar to be defined below, and $\hat{P}$ is an approximation to the Newton matrix

$$P = I - \gamma J(t, y_{n(0)}), \text{ with } J(t, y) = \frac{\partial f}{\partial y}(t, y).$$

Typically, $\hat{P} = I - \hat{\gamma}\hat{J}$, where $\hat{\gamma}$ and $\hat{J}$ are close to γ and J, respectively.

The iteration (3.3) will converge (under suitable conditions on G) if

$$\rho(I - c\hat{P}^{-1}P) < 1,$$

and the initial guess $y_{n(0)}$ is close enough to a root of (3.2). Here $\rho(A)$ denotes the spectral radius of a matrix A. The modified Newton iteration (3.3) normally converges at a linear rate, with $\rho(I - c\hat{P}^{-1}P)$ an estimate of the rate constant for the iteration. Thus, one would like to choose the scalar c so that it solves the minimization problem

$$\min_{c \in \mathbf{R}} \rho(I - c\hat{P}^{-1}P).$$

Since this is impractical in the general setting, we choose c to solve the following, more tractable problem. Suppose $f(t, y) = Ay$ for a matrix A whose eigenvalues all have negative real part. In this case, $P = I - \gamma A$ and $\hat{P} = I - \hat{\gamma}A$. If λ is an eigenvalue of A, then

$$\bar{\lambda} = 1 - d\,\frac{\beta - \lambda}{\hat{\beta} - \lambda}$$

is an eigenvalue of $I - c\hat{P}^{-1}P$, letting $d = c(\gamma/\hat{\gamma})$, $\beta = 1/\gamma$, and $\hat{\beta} = 1/\hat{\gamma}$. In general, we do not know the eigenvalues of A, so we choose d to minimize

$$\max_{Re(\lambda)<0} \left| 1 - d\,\frac{\beta - \lambda}{\hat{\beta} - \lambda} \right|.$$

The solution of this problem is given by

$$d = d^* \equiv 2/(1 + \frac{\beta}{\hat{\beta}}),$$

which then gives the optimal value for c as

$$c = c^* = \frac{\hat{\gamma}}{\gamma} d^* = \frac{2\hat{\gamma}}{\gamma + \hat{\gamma}}. \tag{3.4}$$

Note that when $\gamma = \hat{\gamma}$, we have $c = 1$. Thus, c attempts to speed up the convergence only when the iteration is using an old Newton matrix approximation from a previous step.

4. A fixed-leading-coefficient variant. In this section, we discuss the fixed-leading-coefficient (FLC) form of the BDF methods and the relevant details of implementing them in the VODE solver. We have based our FLC solver on the work of Jackson and Sacks-Davis [15]. To be consistent with the fully variable-coefficient (VC) implementation already in VODE, we will use a Nordsieck form of the FLC methods. We first describe the FLC method formulas and then discuss the necessary modifications to VODE for the FLC variant.

In the FLC methods, the prediction stage is based on the polynomial $\omega_n^P(t)$ of degree q or less that satisfies

$$\omega_n^P(t_{n-i}) = y_{n-i} \quad (i = 1, \cdots, q) \quad \text{and}$$
$$\dot{\omega}_n^P(t_{n-1}) = f_{n-1},$$

where $y_{n-i} \in \mathbf{R}^N$ for each i and $f_{n-1} = f(t_{n-1}, y_{n-1})$. Note that $\omega_n^P(t)$ satisfies the same interpolatory conditions as that for the VC methods. However, the interpolatory conditions for the FLC corrector polynomial differ from those satisfied by the VC corrector. The FLC corrector $\omega_n^C(t)$ of degree q satisfies the $q+2$ conditions

$$\begin{aligned}
&\omega_n^C(t_n) = y_n, \\
&\dot{\omega}_n^C(t_n) = f_n, \quad \text{and} \\
&\omega_n^C(t_n - ih_n) = \omega_n^P(t_n - ih_n) \quad (i = 1, \cdots, q),
\end{aligned}$$

where $h_n = t_n - t_{n-1}$. Thus, the FLC corrector $\omega_n^C(t)$ interpolates $\omega_n^P(t)$ at evenly spaced past points rather than interpolating the computed values $\{y_{n-i}\}$ at the gridpoints $\{t_{n-i}\}$, as the VC corrector does.

The predicted values are given by

$$y_{n(0)} \equiv \omega_n^P(t_n) \quad \text{and} \quad \dot{y}_{n(0)} = f_{n(0)} \equiv \dot{\omega}_n^P(t_n).$$

Using the above formulas, one can derive the relationship

$$\alpha_0(y_n - y_{n(0)}) + h_n(f_n - f_{n(0)}) = 0, \tag{4.1}$$

where $\alpha_0 = -\sum_{j=1}^{q} 1/j$. Equation (4.1) can be further reduced to the form

$$\alpha_0 y_n + \sum_{j=1}^{q} \alpha_{n,j}^* y_{n-j} + h_n f_n = 0.$$

This last equation differs from the analogous relationship for the VC methods in that the leading coefficient α_0 depends only on the order q, and not upon the previous gridpoints t_{n-i}.

The Nordsieck history array z_n is the same as that given in (2.1), with the prediction found using (2.2). To obtain z_n from $z_{n(0)}$, one uses the analogous formula for the VC methods, namely, (2.4). However, the components of the row vector ℓ in the FLC version of (2.4) are different from those in the VC methods. In particular, $\ell_1 = -\alpha_0$ for the FLC methods, and depends only on the method order and not upon the previous gridpoints t_{n-i}, as in the VC methods. As a result, the Jacobian matrix $P = I - \gamma J$ in the Newton iteration should change less with stepsize changes than in the VC formulas. Thus, the FLC methods have the potential to be more efficient in terms of Jacobian evaluations and factorizations than the VC methods on problems for which the stepsize varies smoothly, i.e., that class of problems for which the solver EPISODE is less efficient than GEAR.

When the order of the method is changed, the history array z_n must be adjusted. A decrease in order is handled in exactly the same way as for the VC methods. The array z_n must be modified to become an array z_n^*, which is based on the polynomial $\omega_{n+1}^*(t)$ of degree $q-1$, which, in turn, is defined by

$$\begin{aligned}
&\omega_{n+1}^*(t_{n-i}) = y_{n-i} \quad (i = 0, \cdots, q-2) \quad \text{and} \\
&\dot{\omega}_{n+1}^*(t_n) = f_n.
\end{aligned}$$

This amounts to adding scalar multiples of the last column of z_n to itself and can be written in the form

$$z_n^* = \bar{z}_n + \frac{h_n^q}{q!} y_n^{(q)} \cdot d,$$

where d is a row vector of the form $d = [d_0, d_1, d_2, \cdots, d_{q-1}]$ with $d_0 = d_1 = 0$, and $\bar{z}_n$ is the z_n array minus its last column.

An order increase for the FLC methods also requires a modification of the z_n array, unlike that for the VC methods. In this case, z_n must be modified to become an array z_n^* based on the polynomial $\omega_{n+1}^*(t)$ of degree $q+1$ defined by

$$\omega_{n+1}^*(t_{n-i}) = y_{n-i} \quad (i = 0, \cdots, q) \quad \text{and}$$
$$\dot{\omega}_{n+1}^*(t_n) = f_n.$$

This can be accomplished in a way similar to the order decrease and is given by

$$z_n^* = \tilde{z}_n + (y_n - y_{n(0)}) \cdot c,$$

where $\tilde{z}_n = [z_n, 0]$ and the row vector c is of the form $c = [c_0, c_1, c_2, \cdots, c_{q+1}]$, with $c_0 = c_1 = 0$. See [15] for more details.

5. Numerical tests. The variable-coefficient solver VODE as described above was tested on several problems, two of which are given here. As mentioned in the Introduction, VODE was written according to the standard ODEPACK [10] interface, and its overall structure is similar to that of LSODE [11]. So we will forego a detailed presentation of the solver and its usage. We have also implemented a second version of the VODE solver which uses the fixed-leading-coefficient form of the BDF methods. This version will be referred to as VODE-FLC.

Since both the linear system scaling and new initial h modifications were overall beneficial in improving the efficiency of the solver, we have elected not to specifically present test results regarding these enhancements. However, since the J-saving strategy is a user-controllable option, the test results presented below will compare its effect on the solution of the test problems. For both problems considered below, we compare statistics for LSODE, VODE, and VODE-FLC (with and without the J-saving strategy).

One important aspect of the implementation of the VC and FLC methods in the two versions of VODE is algorithmic tuning. For this preliminary study, we have elected to use the tuning recommended in [14] for VODE, and that used by Jackson and Sacks-Davis in [15] for VODE-FLC. Further testing may indicate modifications of these recommendations.

In the tables below, the following performance statistics will be helpful in comparing the individual solvers:

- NST = number of time steps,
- NFE = number of f evaluations,
- NJE = number of J evaulations,
- NLU = number of LU factorizations of $P = I - \gamma J$,
- R.T. = run time (in seconds) on a Cray-1, and
- E.O. = error overrun.

The error overrun, E.O., is defined as the maximum value of

$$|\text{error in } y^i|/(\text{RTOL} * |y^i| + \text{ATOL})$$

taken over all components and over all output times. The errors were calculated by running the problems at a tighter tolerance (RTOL = 10^{-7}) and saving the answers for use in the test runs at the looser tolerances.

TABLE 5.1
Results for test problem 1.

Solver	RTOL	MF	NST	NFE	NJE	NLU	R.T.	E.O.
VODE	10^{-3}	-25	644	2322	236	236	2.27	5.3
VODE	10^{-3}	25	696	1406	19	275	1.95	2.9
VODE	10^{-5}	-25	1126	3552	298	298	3.59	5.0
VODE	10^{-5}	25	1122	2207	26	298	2.90	1.9
VODE-FLC	10^{-3}	-25	731	2131	207	207	2.13	1.8
VODE-FLC	10^{-3}	25	835	1377	25	224	1.90	0.9
VODE-FLC	10^{-5}	-25	1211	3093	271	271	3.17	5.2
VODE-FLC	10^{-5}	25	1252	2007	32	286	2.76	9.1
LSODE	10^{-3}	25	982	3038	302	302	3.09	1.7
LSODE	10^{-5}	25	2050	5511	470	470	5.64	5.3

5.1. Test problem 1. Our first test problem is one that was used in [7]. It is a system derived from a one-dimensional diurnal kinetics-diffusion PDE system with two species. The PDEs have the form

$$\frac{\partial c^i}{\partial t} = \frac{\partial}{\partial z}\left[K(z)\frac{\partial c^i}{\partial z}\right] + R^i(c^1, c^2, t) \ (i = 1, 2),$$

$$K(z) = 10^{-8}e^{z/5},$$

$$R^1(c^1, c^2, t) = -k_1 c^1 - k_2 c^1 c^2 + k_3(t) \cdot 7.4 \cdot 10^{16} + k_4(t)c^2,$$

$$R^2(c^1, c^2, t) = k_1 c^1 - k_2 c^1 c^2 - k_4(t)c^2,$$

$$k_1 = 6.031, \quad k_2 = 4.66 \cdot 10^{-16},$$

$$k_3(t) = \begin{cases} \exp[-22.62/\sin(\omega t)], & \text{for } \sin(\omega t) > 0, \\ 0, & \text{for } \sin(\omega t) \le 0, \end{cases}$$

$$k_4(t) = \begin{cases} \exp[-7.601/\sin(\omega t)], & \text{for } \sin(\omega t) > 0, \\ 0, & \text{for } \sin(\omega t) \le 0, \end{cases}$$

$$\omega = \pi/43,200 \ ,$$

with $30 \le z \le 50$ km, $0 \le t \le 432,000$ seconds (five days), and are discretized by finite differencing on a mesh of size 50. Homogeneous Neumann boundary conditions are used with polynomial initial conditions. See [7] for complete details on the problem definition and a discussion of the physical aspects.

The results of solving this problem using all three solvers are given in Table 5.1. The method flag, MF, shown in the table is an input for all three solvers. It indicates which ODE method is used, if the Jacobian matrix is treated as a dense or banded matrix, and whether or not the Jacobian is obtained by finite differences or analytically. The value |MF| = 25 means that the solver uses a stiff method, the Jacobian is banded, and it is obtained by finite differencing. A positive MF value means that the J-saving strategy is used, while a negative value means it is not. The RTOL value is the user-specified relative error tolerance. The absolute error tolerance ATOL is then 100*RTOL. For the LSODE run at the loose tolerance (and in that case only), a maximum stepsize of six hours was used to avoid overshooting the daytime parts of the model.

From the table, it is immediately apparent that the J-saving strategy is very beneficial on this problem in reducing the number of J evaluations, at the cost of a

somewhat higher number of LU factorizations. All runs produce results with about the same accuracy, with values of the error overrun varying by less than a factor of 6 for each of the tolerances. Both versions of VODE are more efficient in terms of run time than LSODE at both tolerances. The VODE-FLC version is the overall winner with a 51 percent decrease in run time over LSODE for the tight tolerance, although in general VODE-FLC takes slightly more steps per run than VODE.

5.2. Test problem 2. This second test problem is a two-dimensional version of the first one and was also used in [7]. The PDEs have the form

$$\frac{\partial c^i}{\partial t} = K_h \frac{\partial^2 c^i}{\partial x^2} + \frac{\partial}{\partial z}\left[K_v(z)\frac{\partial c^i}{\partial z}\right] + R^i(c^1, c^2, t) \ (i = 1, 2),$$
$$K_h = 4 \times 10^{-6}, \quad \text{and} \quad K_v(z) = 10^{-8} e^{z/5},$$

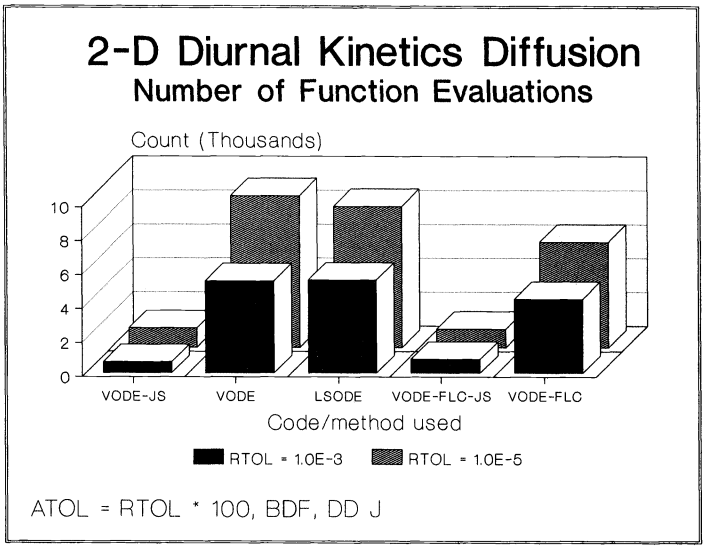
with $0 \le x \le 20$, $30 \le z \le 50$ km, $0 \le t \le 86,400$ seconds (1 day), and the other terms defined as in the first problem. We discretize the PDEs on a uniform mesh of size 20×20, and use homogeneous Neumann boundary conditions and polynomial initial conditions. Again, see [7] for further details.

The results for this problem are given in Table 5.2. Here, the additional MF values of ± 24 correspond to a banded Jacobian obtained analytically. The RTOL and ATOL values are the same as used in the first problem. The values of NFE and R.T. in the case MF $= \pm 25$ are also shown in the bar charts in Figure 5.1. In all runs, the accuracy of the answers was about the same, with the values of the error overrun varying within a factor of about 4 for each of the tolerances. It is clear that the J-saving strategy is also beneficial in reducing J evaluations and overall work for this problem. In addition, VODE and VODE-FLC are both more efficient than LSODE on this problem, with the biggest reductions in run time occuring when MF $= 25$ for LSODE and VODE-FLC. At the tighter tolerance, the run time is reduced by 63 percent!

6. Conclusion. We believe that VODE accomplishes the objective of incorporating the ODE methods used in EPISODE and EPISODEB in a flexible solver package that conforms to the ODEPACK structure. More than that, it improves on the older codes in several ways and includes some features that enhance its efficiency, in some cases by a considerable amount. The relative speedup resulting from Jacobian-saving can increase dramatically as the cost of evaluating f and J increases, especially when J is obtained by difference quotients. The new initial step algorithm and the linear system relaxation each contribute a modest additional gain in speed.

The relative merits of LSODE, VODE, and the FLC variant of VODE are not entirely clear at this time. It seems to remain true (as for EPISODE vs. GEAR) that on smooth problems, LSODE may be more efficient than VODE (either version). This can be attributed mainly to the impact of variable coefficients on the Newton matrix. On the other hand, the greater efficiency of VODE on many nonsmooth problems gives justification to the variable-coefficient methods. Furthermore, the FLC version of VODE seems to reduce the gap between VODE and LSODE on the smooth problems, while retaining the advantage for nonsmooth problems. More confident overall statements of this sort will have to await further refinement and broader testing.

Several further developments are planned for VODE. One is to improve the order selection to make it more reliable in the presence of large imaginary parts in the problem spectrum. Another is to study various heuristic (or tuning) issues and consider



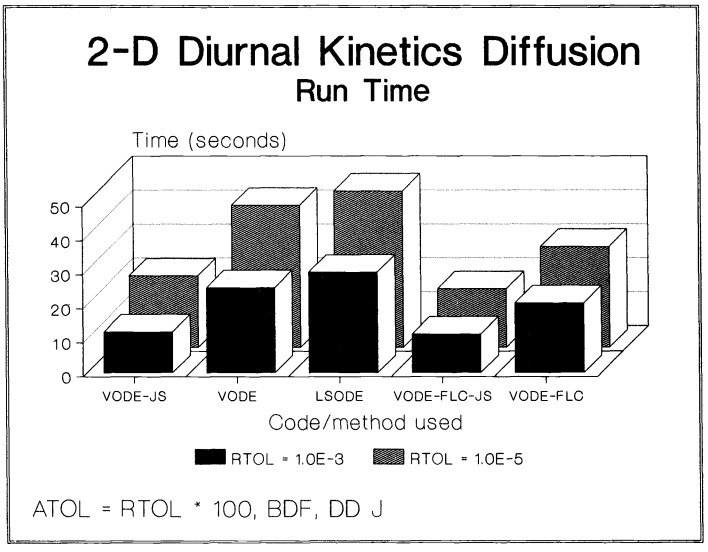


FIG. 5.1. NFE *and* R.T. *for test problem* 2 (*difference-quotient J*).

TABLE 5.2
Results for test problem 2.

Solver	RTOL	MF	NST	NFE	NJE	NLU	R.T.	E.O.
VODE	10^{-3}	-24	183	320	63	63	11.23	3.3
VODE	10^{-3}	24	185	323	4	63	11.19	3.3
VODE	10^{-3}	-25	183	5423	63	63	25.11	3.3
VODE	10^{-3}	25	185	647	4	63	12.07	3.3
VODE	10^{-5}	-24	331	589	103	103	19.04	1.7
VODE	10^{-5}	24	320	578	7	108	19.40	3.6
VODE	10^{-5}	-25	331	8932	103	103	41.86	1.7
VODE	10^{-5}	25	320	1145	7	108	21.08	3.6
VODE-FLC	10^{-3}	-24	230	307	50	50	9.43	0.8
VODE-FLC	10^{-3}	24	232	326	6	53	10.04	1.4
VODE-FLC	10^{-3}	-25	230	4357	50	50	20.56	0.8
VODE-FLC	10^{-3}	25	232	812	6	53	11.38	1.4
VODE-FLC	10^{-5}	-24	339	461	70	70	13.69	6.8
VODE-FLC	10^{-5}	24	342	469	7	73	14.14	2.8
VODE-FLC	10^{-5}	-25	353	6226	71	71	29.69	4.7
VODE-FLC	10^{-5}	25	375	1096	7	80	17.25	4.8
LSODE	10^{-3}	24	223	325	51	51	10.40	1.4
LSODE	10^{-3}	25	248	5480	63	63	29.64	1.7
LSODE	10^{-5}	24	482	708	98	98	20.99	2.4
LSODE	10^{-5}	25	481	8323	94	94	46.04	3.3

changes to these. Another is the development of a variant that includes preconditioned Krylov iteration methods for the algebraic system problem, in analogy with the LSODPK variant of LSODE [2].

REFERENCES

[1] M. BERZINS, *A C^1 interpolant for codes based on backward differentiation formulae,* Appl. Numer. Math., 2 (1986), pp. 109-118.

[2] P. N. BROWN AND A. C. HINDMARSH, *Reduced storage matrix methods in stiff* ODE *systems,* Lawrence Livermore National Laboratory Report UCRL-95088, Rev. 1, Livermore, CA, June 1987; J. Appl. Math. & Comp., to appear.

[3] K. BURRAGE, J. C. BUTCHER, AND F. H. CHIPMAN, *An implementation of singly-implicit Runge-Kutta methods,* BIT, 20 (1980), pp. 326-340.

[4] G. D. BYRNE, A. C. HINDMARSH, K. R. JACKSON, AND H. G. BROWN, *A comparison of two* ODE *codes:* GEAR *and* EPISODE, Comput. Chem. Engrg., 1 (1977), pp. 133-147.

[5] G. D. BYRNE AND A. C. HINDMARSH, *A polyalgorithm for the numerical solution of ordinary differential equations,* ACM Trans. Math. Software, 1 (1975), pp. 71-96.

[6] ——, EPISODEB*: An experimental package for the integration of systems of ordinary differential equations with banded Jacobians,* Lawrence Livermore National Laboratory Report UCID-30132, Livermore, CA, April 1976.

[7] ——, *Stiff* ODE *solvers: A review of current and coming attractions,* J. Comput. Phys., 70 (1987), pp. 1-62.

[8] A. R. CURTIS, *Solution of large, stiff initial value problems – The state of the art,* in Numerical Software – Needs and Availability, D. A. H. Jacobs, ed., Academic Press, London, 1978, pp. 257-278.

[9] A. C. HINDMARSH, GEAR*: Ordinary differential equation system solver,* Lawrence Livermore National Laboratory Report UCID-30001, Rev. 3, Livermore, CA, December 1974.

[10] ——, *A tentative user interface standard for* ODEPACK, Lawrence Livermore National Lab-

oratory Report UCID-17954, Livermore, CA, October 1987.

[11] A. C. HINDMARSH, LSODE *and* LSODI, *two new initial value ordinary differential equations solvers,* ACM SIGNUM Newsletter, 15, No. 4 (1980), pp. 10-11.

[12] ——, ODEPACK, *A systematized collection of* ODE *solvers,* in Scientific Computing, R. S. Stepleman, ed., North-Holland, Amsterdam, 1983, pp. 55-64.

[13] A. C. HINDMARSH AND G. D. BYRNE, *Applications of* EPISODE*: An experimental package for the integration of systems of ordinary differential equations,* in Numerical Methods for Differential Systems – Recent Developments in Algorithms, Software, and Applications, L. Lapidus and W. E. Schiesser, eds., Academic Press, New York, 1976, pp. 147-166.

[14] ——, EPISODE*: An effective package for the integration of systems of ordinary differential equations,* Lawrence Livermore National Laboratory Report UCID-30112, Rev. 1, Livermore, CA, April 1977.

[15] K. R. JACKSON AND R. SACKS-DAVIS, *An alternative implementation of variable step-size multistep formulas for stiff* ODE*s,* ACM Trans. Math. Software, 6 (1980), pp. 295-318.

[16] F. T. KROGH AND K. STEWART, *Asymptotic ($h \to \infty$) absolute stability for* BDF*s applied to stiff differential equations,* ACM Trans. Math. Software, 10 (1984), pp. 45-57.

[17] L. R. PETZOLD, *A description of* DASSL*: A differential/algebraic system solver,* in Scientific Computing, R. S. Stepleman, ed., North-Holland, Amsterdam, 1983, pp. 65-68.

[18] D. E. SALANE, *Improving the performance of a code for solving stiff systems of* ODE*s,* Sandia National Laboratories Report SAND86-2159, Albuquerque, NM, November 1986; Appl. Numer. Math., to appear.

[19] L. F. SHAMPINE, *Starting variable-order Adams and* BDF *codes,* Appl. Numer. Math., 3 (1987), pp. 331-337.

[20] H. A. WATTS, HSTART – *An improved initial step size routine for* ODE *codes,* Sandia National Laboratories Report SAND86-2633, Albuquerque, NM, November 1986.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 5, pp. 1053–1061, September 1989
©1989 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.

TO OVERLAP OR NOT TO OVERLAP: A NOTE ON A DOMAIN DECOMPOSITION METHOD FOR ELLIPTIC PROBLEMS*

PETTER E. BJØRSTAD† AND OLOF B. WIDLUND‡

Abstract. More than one hundred years ago, H.A. Schwarz introduced a domain decomposition method in which the original elliptic equation is solved on overlapping subregions, one after another, in an iterative process. A few years ago, Chan and Resasco, introduced a method that they classified as a domain decomposition method using nonoverlapping subdomains. In this note, it is shown that their method is an accelerated version of the classical method. It is also shown that the error propagation operator of the method can be expressed in terms of Schur complements of certain stiffness matrices and that techniques previously developed for the study of iterative substructuring algorithms can be used to derive estimates on the rate of convergence.

Key words. Schwarz's method, iterative substructuring methods, domain decomposition, elliptic equations, finite elements

AMS(MOS) subject classifications. 65F10, 65N30

1. Introduction. Recently there has been a strong revival of the interest in domain decomposition algorithms for elliptic problems; cf., e.g., Glowinski et al. [18] and Chan et al. [9]. This is to a large extent due to their potential on multiprocessor computing systems. It is therefore not surprising that much of the recent research effort has focused on the case of many subregions; cf., e.g., Bramble et al. [4], [5], [6], [7], [8]; Dryja [14], [15]; Dryja et al. [16], [17]; and Widlund [25]. However, in this note, we regress and consider only the case of two subregions. We show that a method recently developed by Chan and Resasco [11], [12] is identical to the classical Schwarz algorithm, cf. Schwarz [21], accelerated by the standard conjugate gradient method. We note that in a recent technical report, which we received after submitting this paper to SIAM, Chan and Goovaerts [10] give another description of our main result using a linear algebra framework similar to ours.

Domain decomposition algorithms are sometimes classified according to whether overlapping subregions are used or not. This distinction often makes sense. Thus, if overlapping is used, certain variables are associated with several subproblems, and the cost per iteration normally grows with the overlap. A decision of how to balance this

* Received by the editors September 9, 1988; accepted for publication (in revised form) March 22, 1989.

† Institutt for Informatikk, University of Bergen, Thormøhlens gate 55, N-5006 Bergen, Norway. This work was supported in part by the Norwegian Research Council for Science and the Humanities under grant D.01.08.054 and by The Royal Norwegian Council for Scientific and Industrial Research under grant IT2.28.28484.

‡ Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, New York 10012. This work was supported in part by the National Science Foundation under grant NSF-CCR-8703768, in part by the U.S. Department of Energy under contract DE-AC02-76ER03077-V at the Courant Mathematics and Computing Laboratory, and in part by the IBM Bergen Scientific Centre, while the author was a visiting scientist.

effect against an increased rate of convergence has to be made, while for a method that does not use overlapping, no such consideration arises. However, we demonstrate that the distinction is not all that clear and that the same technical tools can be used for the analysis of methods that can be regarded as members of the two different classes.

2. The Schwarz method: Several points of view. To simplify our presentation, we assume that the elliptic operator is the Laplacian and that we have a zero Dirichlet condition. Thus,

$$(1) \qquad \begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

The region Ω is bounded, two- or three-dimensional, with a Lipschitz continuous boundary. Our algorithms and results can be extended immediately to any linear, self-adjoint elliptic problem that can be formulated as a minimization problem. We use a variational formulation of the problem, which, as shown by Sobolev [22] more than fifty years ago, makes the maximum principle superfluous.

In variational form (1) is written as

$$a_\Omega(u,v) = \int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx = f(v) \,, \quad \forall v \in H_0^1(\Omega) \,,$$

where the solution $u \in H_0^1(\Omega)$, the closure in the Sobolev space $H^1(\Omega)$ of the space of smooth functions that vanish in the neighborhood of $\partial\Omega$. As always, the space $H^1(\Omega)$ is the subspace of $L_2(\Omega)$ for which $| u |^2_{H^1(\Omega)} = a(u,u)$ is finite. In our analysis we work exclusively with the inner product defined by $a(\cdot,\cdot)$. Thus in this paper orthogonality and symmetry always refer to this inner product.

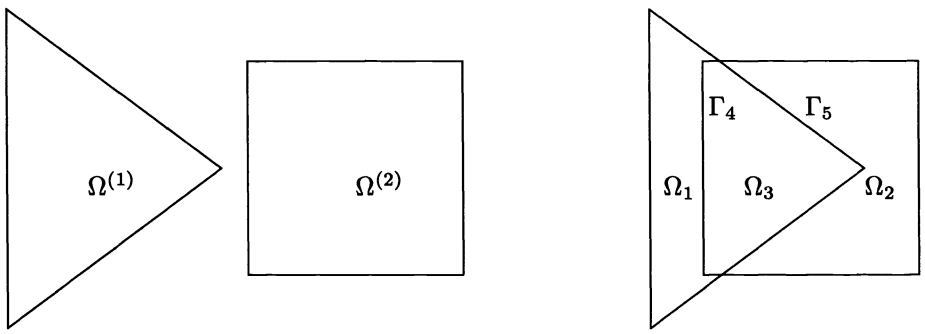


FIG. 1. *The partition of the domain.*

In this note we consider domain decomposition algorithms with two overlapping subregions $\Omega^{(1)}$ and $\Omega^{(2)}$. We also use the notations $\Omega_1 = \Omega \setminus \overline{\Omega}^{(2)}$, where $\overline{\Omega}^{(2)}$ is the closure of $\Omega^{(2)}$, $\Omega_2 = \Omega \setminus \overline{\Omega}^{(1)}$, and $\Omega_3 = \Omega^{(1)} \cap \Omega^{(2)}$. The region Ω is thus also divided into three nonoverlapping subregions Ω_1, Ω_2, and Ω_3, which are separated from each other by the curves (or surfaces) $\Gamma_4 = \overline{\Omega}_1 \cap \overline{\Omega}_3$ and $\Gamma_5 = \overline{\Omega}_2 \cap \overline{\Omega}_3$. In Fig. 1, we display separately the two subregions from which Ω is built on the left; the partitioning of Ω into the five subsets just defined is given on the right. We assume that Γ_4 and Γ_5

follow element boundaries, that they are Lipschitz, and that they intersect only in at most a few points (or along a few curves).

The problem is discretized by finite elements in the customary fashion; cf. Ciarlet [13]. The region Ω is triangulated, and a conforming finite element space $V^h \subset H_0^1(\Omega)$ is introduced. We assume that all the elements are shape regular, i.e., there is a bound on h_K/ρ_K that is independent of the number of degrees of freedom and of K. Here h_K is the diameter of the element K and ρ_K the diameter of the largest sphere that can be inscribed in K. The approximate solution $u_h \in V^h$ is defined by

$$a_\Omega(u_h, v_h) = f(v_h)\,, \quad \forall\, v_h \in V^h. \tag{2}$$

We return briefly to the continuous case and write down the Schwarz algorithm in its traditional form. There are two fractional steps. Let u^n be the nth iterate. Then the updated solution u^{n+1} is determined by

$$\begin{aligned} -\Delta u^{n+1/2} &= f && \text{in } \Omega^{(1)}, \\ u^{n+1/2} &= 0 && \text{on } \partial\Omega^{(1)} \cap \partial\Omega, \\ u^{n+1/2} &= u^n && \text{on } \Gamma_5, \end{aligned}$$

and

$$\begin{aligned} -\Delta u^{n+1} &= f && \text{in } \Omega^{(2)}, \\ u^{n+1} &= 0 && \text{on } \partial\Omega^{(2)} \cap \partial\Omega, \\ u^{n+1} &= u^{n+1/2} && \text{on } \Gamma_4. \end{aligned}$$

Following P. L. Lions [19], [20], who recently has obtained a number of interesting results on Schwarz's method, we transform these equations into variational form. This formulation is equally valid in the discrete case. The exact solution of (2) is denoted by u_h.

$$\begin{aligned} a_\Omega(u_h^{n+1/2} - u_h^n, v_h) &= f(v_h) - a_\Omega(u_h^n, v_h) \\ &= a_\Omega(u_h - u_h^n, v_h)\,, && \forall\, v_h \in V^h \cap H_0^1(\Omega^{(1)})\,, \\ a_\Omega(u_h^{n+1} - u_h^{n+1/2}, v_h) &= f(v_h) - a_\Omega(u_h^{n+1/2}, v_h) \\ &= a_\Omega(u_h - u_h^{n+1/2}, v_h)\,, && \forall\, v_h \in V^h \cap H_0^1(\Omega^{(2)})\,. \end{aligned}$$

Here $u_h^{n+1/2} - u_h^n \in H_0^1(\Omega^{(1)})$ and $u_h^{n+1} - u_h^{n+1/2} \in H_0^1(\Omega^{(2)})$, since the boundary values do not change from one fractional step to the next. Following Lions, these equations can conveniently be written as

$$\begin{aligned} u_h^{n+1/2} - u_h^n &= P_1(u_h - u_h^n)\,, \\ u_h^{n+1} - u_h^{n+1/2} &= P_2(u_h - u_h^{n+1/2})\,, \end{aligned}$$

where P_i, $i = 1, 2$, is the orthogonal projection of V^h into $V_i = V^h \cap H_0^1(\Omega^{(i)})$. The error $e_h^n = u_h^n - u_h$ satisfies the relationship

$$e_h^{n+1} = (I - P_2)(I - P_1)e_h^n = (I - (P_1 + P_2 - P_2P_1))e_h^n.$$

In a certain sense, Schwarz's method is therefore a straightforward iterative method of solving the equation

$$(P_1 + P_2 - P_2P_1)u_h = g_h,$$

for a certain right-hand side g_h. The operator is not symmetric, but there is an easy remedy, namely, the use of a third fractional step identical to the first. The resulting operator

$$I - (I - P_1)(I - P_2)(I - P_1) = P_1 + P_2 - P_1P_2 - P_2P_1 + P_1P_2P_1$$

is symmetric with respect to the energy inner product.

The method can be accelerated, e.g., with the conjugate gradient method. We can equally well regard the third fractional step as the first step of the next iteration, and the extra cost involved can therefore quickly be amortized. Since it can be shown that the spectra of $(I - P_2)(I - P_1)$ and $(I - P_1)(I - P_2)(I - P_1)$ are the same, it is easy to derive bounds for the rate of convergence of the different iterative methods at the same time.

The use of projections in the description of the algorithm has many merits. However, it is also useful to describe the projections and algorithms in a more concrete, traditional linear algebra framework. The symmetry required for the routine use of the conjugate gradient method can thus also be recovered by considering Schwarz's algorithm from this other point of view. From our experience with other domain decomposition methods, see, e.g., Bjørstad [1], Bjørstad and Hvidsten [2], Bjørstad and Widlund [3], and Widlund [24], [25], we have also learned that insight can be gained by expressing the algorithms in terms of stiffness matrices. With subvectors and subscripts corresponding to the degrees of freedom associated with the open sets Ω_1, Ω_2, and Ω_3 and the curves (surfaces) Γ_4 and Γ_5, the entire discrete problem can be written as

$$Kx = \begin{pmatrix} K_{11} & 0 & 0 & K_{14} & 0 \\ 0 & K_{22} & 0 & 0 & K_{25} \\ 0 & 0 & K_{33} & K_{34} & K_{35} \\ K_{14}^T & 0 & K_{34}^T & K_{44} & K_{45} \\ 0 & K_{25}^T & K_{35}^T & K_{45}^T & K_{55} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}. \tag{3}$$

As always, the elements of K are given by $a(\phi_i, \phi_j)$, where ϕ_i and ϕ_j are finite element basis functions. The zero blocks are a consequence of the fact that, when using standard finite element basis functions, there is no direct coupling between Ω_1 and Ω_2, etc.

Stiffness matrices are often computed by the so-called subassembly method. Since the bilinear form is defined in terms of an integral,

$$a_\Omega(\phi_i, \phi_j) = a_{\widetilde{\Omega}}(\phi_i, \phi_j) + a_{\Omega \setminus \widetilde{\Omega}}(\phi_i, \phi_j),$$

for any subset $\widetilde{\Omega} \subset \Omega$. We note that for a pair of basis functions associated with $\partial\widetilde{\Omega}$, we get contributions from both terms. If we, for example, choose $\widetilde{\Omega} = \Omega_2$, then we see that the quadratic form $x^T K y$ can be assembled by adding the quadratic forms corresponding to the matrices

$$\begin{pmatrix} K_{22} & K_{25} \\ K_{25}^T & K_{55}^{(2)} \end{pmatrix} \text{ and } \begin{pmatrix} K_{11} & 0 & K_{14} & 0 \\ 0 & K_{33} & K_{34} & K_{35} \\ K_{14}^T & K_{34}^T & K_{44} & K_{45} \\ 0 & K_{35}^T & K_{45}^T & K_{55}^{(3)} \end{pmatrix}. \tag{4}$$

Here, $K_{55}^{(2)}$ and $K_{55}^{(3)}$ represent the contributions to K_{55} attributable to the integrals over Ω_2 and Ω_3, respectively. Similarly, we can write $K_{44} = K_{44}^{(1)} + K_{44}^{(3)}$. We note

that the two matrices of (4) correspond to finite element models on $\Omega^{(1)}$ and Ω_2, respectively, with natural boundary conditions on Γ_5.

In the first fractional step, described above, we solve a homogeneous Dirichlet problem on $\Omega^{(1)}$. The corresponding linear system is

$$\begin{pmatrix} K_{11} & 0 & K_{14} \\ 0 & K_{33} & K_{34} \\ K_{14}^T & K_{34}^T & K_{44} \end{pmatrix} \begin{pmatrix} y_1 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_3 \\ c_4 \end{pmatrix}. \tag{5}$$

If we denote this coefficient matrix by $K^{(1)}$, and permute the unknowns appropriately, then we see that the projection P_1 corresponds to the mapping

$$y = P_1 x = \begin{pmatrix} K^{(1)^{-1}} & 0 \\ 0 & 0 \end{pmatrix} Kx.$$

Since it is easy to show that the residuals associated with interior nodes of the sets Ω_1, Ω_2, and Ω_3 are and remain zero after the first full step of the Schwarz algorithm, we can specialize to the case where $c_1 = 0$ and $c_3 = 0$. By block Gaussian elimination we can reduce the system (5) to

$$(S^{(1)} + S^{(3)})y_4 = c_4, \tag{6}$$

where the so called Schur complements $S^{(i)}$, $i = 1, 3$, are defined by

$$S^{(i)} = K_{44}^{(i)} - K_{i4}^T K_{ii}^{-1} K_{i4} \ , \quad i = 1, 3. \tag{7}$$

The Schur complement $S^{(2)}$ is associated with the entire region $\Omega^{(2)} = \Omega_2 \cup \Omega_3 \cup \Gamma_5$. It is given by the formula

$$S^{(2)} = K_{44}^{(3)} - \begin{pmatrix} 0 & K_{34}^T & K_{45} \end{pmatrix} \begin{pmatrix} K_{22} & 0 & K_{25} \\ 0 & K_{33} & K_{35} \\ K_{25}^T & K_{35}^T & K_{55} \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ K_{34} \\ K_{45}^T \end{pmatrix}.$$

The second fractional step makes all the residuals on $\Omega^{(2)}$ equal to zero. Upon the completion of any full iterative step, the only nonzero residuals are associated with Γ_4. We can therefore reduce the original system (3) to the equation

$$(S^{(1)} + S^{(2)})x_4 = b_4. \tag{8}$$

By Gaussian elimination, we show straightforwardly that the residual vector associated with Γ_4 is of the form

$$(S^{(1)} + S^{(2)})e_4^{(n)}$$

This is the right-hand side of (6), and the error propagation operator is therefore

$$I - (S^{(1)} + S^{(3)})^{-1}(S^{(1)} + S^{(2)}). \tag{9}$$

Chan and Resasco [11], [12] define their algorithm in terms of the same Schur complements. They reduce the original problem to (8), and since they use the same preconditioner, $S^{(1)} + S^{(3)}$, exactly the same iterates are produced if no acceleration procedure is used. It is also easy to show that the conjugate gradient version of their algorithm is identical to an accelerated version of the classical Schwarz method.

Chan and Resasco primarily consider cases where Ω is the union of two or three rectangles. The largest possible rectangular subregions of Ω are used to enhance the rate of convergence. The subproblems can then be solved by separation of variables. The preconditioner $S^{(1)} + S^{(3)}$ is diagonalized by using the fast Fourier transform (FFT), and the work involved is, therefore, independent of the size of the overlap in these special cases. In particular, Chan and Resasco demonstrate that if $\Omega^{(1)}$ is as large as possible a subset of Ω, the condition number of $(S^{(1)} + S^{(3)})^{-1}(S^{(1)} + S^{(2)})$ remains uniformly bounded, independent of the aspect ratios of $\Omega^{(1)}$ and $\Omega^{(2)}$. Such a result is not true for certain other algorithms, such as the Neumann-Dirichlet algorithm; cf. Bjørstad and Widlund [3]. However, the aspect ratios of the rectangles have to be extreme for this to be of any real importance.

3. Remarks on theory and other related methods. The convergence analysis of Schwarz's method can be carried out in a number of ways. Traditionally, the maximum principle is used to obtain an upper bound for the spectral radius; cf. Schwarz [21] or Lions [20]. Lions [19] has also derived a different upper bound that depends on the geometry through the maximum norm of the gradient of a certain function used in a partition of unity. Our analysis above has shown that the spectrum of the error propagation operator (9) can be expressed precisely in terms of a generalized eigenvalue problem involving three Schur complements. These Schur complements, defined through block Gaussian elimination of all variables not associated with the set Γ_4, are related to the strain energies of so-called discrete harmonic functions. Thus, the strain energy $x^T K x$ associated with the entire region Ω is equal to ${x_4}^T(S^{(1)} + S^{(2)})x_4$ if the components of Kx corresponding to the interior variables of Ω_1 and $\Omega^{(2)}$ vanish. It is known that the strain energy of a discrete harmonic function has a smaller strain energy than any other extension of the given boundary values. Similarly, since the space of extensions increases if the overlap Ω_3 increases, it can be shown that the rate of convergence of the Schwarz method increases with the overlap. In fact, every component of the expansion of the error into the eigenvectors of $(S^{(1)} + S^{(3)})^{-1}(S^{(1)} + S^{(2)})$ decreases faster.

To prove that the method is uniformly well conditioned, independent of the number of degrees of freedom, we establish uniform upper and lower bounds for the generalized Rayleigh quotient

$$ {x_4}^T(S^{(1)} + S^{(2)})x_4 / {x_4}^T(S^{(1)} + S^{(3)})x_4 \,, $$

by using

$$ {x_4}^T S^{(i)} x_4 \le C {x_4}^T S^{(j)} x_4 \,, \quad \forall x_4, \quad i,j = 1,2,3. $$

Proofs of this so-called extension theorem are given in [3] and [23]. The constants in these inequalities depend only on Ω_1, Ω_2 and Ω_3.

It is also of interest to relate Schwarz's algorithm to the Neumann-Dirichlet algorithm, cf. Bjørstad and Widlund [3], where a region is divided into two nonoverlapping subregions by an interface that we can identify with Γ_4. The Schur complement $S^{(1)}$ alone is used to precondition (8). The convergence rate, therefore, depends on the Rayleigh quotient

$$ {x_4}^T(S^{(1)} + S^{(2)})x_4 / {x_4}^T S^{(1)} x_4. $$

The same extension theorem establishes a uniform upper bound.

If the interface Γ_4 is straight, Ω_3 and its triangulation are mirror images of Ω_1 and its triangulation and the elements of Lagrangian type, then $S^{(3)} = S^{(1)}$. In that case, the Schwarz method and the Neumann-Dirichlet algorithm give exactly the same result if conjugate gradient acceleration is employed. In nonmodel cases, the two algorithms can differ. See Bjørstad and Hvidsten [2] for a discussion of a case where the Schwarz algorithm works much better.

Finally, we note that there is, in fact, no need to use subregions of Ω and the given triangulation in the construction of good preconditioners. By using the extension theorem, it is easy to show that the Schur complement with respect to the variables related to Γ_4 of a finite element model on any region, for which Γ_4 is an interface or a part of boundary, provides a preconditioner for the conjugate gradient method with a rate of convergence that is independent of the size of the linear system. There is no guarantee that the unaccelerated method will converge, since the strain energy of the new model might very well be smaller than that of the original problem.

4. Numerical experiments. We give a few numerical results to confirm the theoretical results and to give the reader an idea of the performance of the algorithms. For simplicity, Ω is the union of two rectangles.

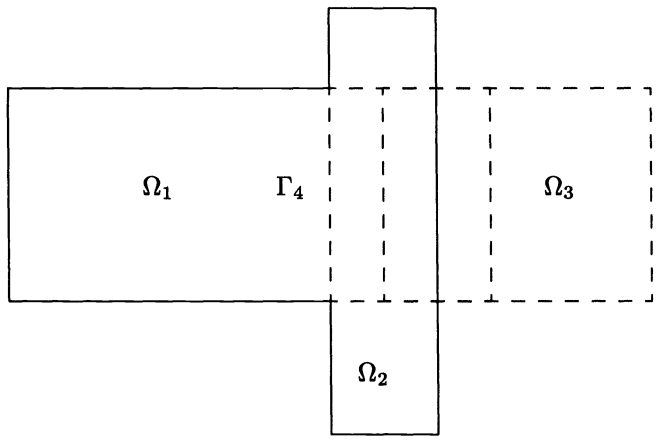


FIG. 2. *The computational domains.*

We use a mesh spacing $h = 1/256$ with the size of Ω_1 being .75 by .5, see Fig. 2. This gives a total of 40449 unknowns, with 127 along Γ_4. We consider the same problem as in [3] with the exact solution $u(x, y) = x^2 + y^2 - x \exp x \cos y$. The number of iterations required to reduce the initial error to the truncation error level (a factor of $3 \cdot 10^{-5}$) are reported both for the classical Schwarz method and for some of the other methods discussed in § 3. We create the overlap Ω_3 by extending the rectangle Ω_1 into and sometimes beyond Ω_2. Table 1 gives results for four different sizes of Ω_3: halfway into the second rectangle, all the way into the second rectangle, half the size of the mirror image of Ω_1 with respect to Γ_4, and equal to the mirror image. The second case is the Chan-Resasco algorithm. The last one, as indicated in the previous section, gives the same result as the Neumann-Dirichlet algorithm. The condition numbers relevant for the conjugate gradient method, are 1.29, 1.16, 1.177, and 1.184, respectively, in the four cases.

TABLE 1
Number of iterations.

Schwarz algorithms	Relative Overlap			
	1/2	1	3/2	3
Unaccelerated	14	11	11	11
Accelerated	6	5	5	5

REFERENCES

[1] P. E. BJØRSTAD, *Multiplicative and additive Schwarz methods; Convergence in the two domain case*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O.Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 147-159.

[2] P. E. BJØRSTAD AND A. HVIDSTEN, *Iterative methods for substructured elasticity problems in structural analysis*, in Proc. First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G.A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 301-312.

[3] 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. 1093-1120.

[4] 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.

[5] ———, *The construction of preconditioners for elliptic problems by substructuring*, II, Math. Comp., 49 (1987), pp. 1-16.

[6] ———, *The construction of preconditioners for elliptic problems by substructuring*, III, Math. Comp., 51 (1988), pp. 415-430.

[7] ———, *The construction of preconditioners for elliptic problems by substructuring*, IV, Tech. Report, Cornell University, Ithaca, NY, 1988; Math. Comp., to appear.

[8] ———, *An iterative method for elliptic problems on regions partitioned into substructures*, Math. Comp., 46 (1986), pp. 361-369.

[9] T. CHAN, R. GLOWINSKI, J. PÉRIAUX, AND O. WIDLUND, EDS., *Domain Decomposition Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[10] T. F. CHAN AND D. GOOVAERTS, *Schwarz = Schur: Overlapping versus nonoverlapping domain decomposition*, Tech. Report CAM 88-21, Department of Mathematics, University of California, Los Angeles, CA, 1988.

[11] T. F. CHAN AND D. C. RESASCO, *Analysis of domain decomposition preconditioners on irregular regions*, in Advances in Computer Methods for Partial Differential Equations, R. Vichnevetsky and R. Stepleman, eds., IMACS, 1987.

[12] ———, *An analysis of domain decomposition preconditioners on L-shaped and C-shaped regions*, Tech. Report /DCS/RR-534, Yale University, New Haven, CT, 1988.

[13] P. G. CIARLET, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.

[14] M. DRYJA, *An additive Schwarz algorithm for two- and three-dimensional finite element elliptic problems*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 168-172.

[15] ———, *A method of domain decomposition for three-dimensional finite element problems*, in Proc. First International Symposium on Domain Decomposition for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 43-61.

[16] M. DRYJA, W. PROSKUROWSKI, AND O. WIDLUND, *A method of domain decomposition with crosspoints for elliptic finite element problems*, in Optimal Algorithms, B. Sendov, ed., Bulgarian Academy of Sciences, Sofia, Bulgaria, 1986, pp. 97-111.

[17] M. DRYJA AND O. B. WIDLUND, *An additive variant of the Schwarz alternating method for the case of many subregions*, Tech. Report 339; Ultracomputer Note 131, Department of Computer Science, Courant Institute, New York, NY, 1987.

[18] R. GLOWINSKI, G. H. GOLUB, G. A. MEURANT, AND J. PÉRIAUX, EDS., *Domain Decomposition Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[19] P. L. LIONS, *On the Schwarz alternating method*, I, in Proc. First International Symposium on Domain Decomposition Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988, pp. 1-42.

[20] ———, *On the Schwarz alternating method*, II, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989, pp. 47-70.

[21] H. A. SCHWARZ, *Gesammelete Mathematische Abhandlungen*, Vol. 2, Springer-Verlag, Berlin, New York, 1890, pp.133-143. First published in Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich, Vol. 15 (1870), pp. 272-286.

[22] S. L. SOBOLEV, *The Schwarz algorithm in the theory of elasticity*, Dokl. Akad. Nauk, USSR, IV (1936), pp. 236-238. In Russian.

[23] O. B. WIDLUND, *An extension theorem for finite element spaces with three applications*, in Numerical Techniques in Continuum Mechanics, W. Hackbusch and K. Witsch, eds., Notes on Numerical Fluid Mechanics, Vol. 16, Friedr. Vieweg and Sohn, Braunschweig/Wiesbaden, 1987, pp. 110-122; Proc. 2nd GAMM-Seminar, Kiel, FRG, January 17-19, 1986.

[24] ———, *Iterative methods for elliptic problems on regions partitioned into substructures and the biharmonic Dirichlet problem*, in Computing Methods in Applied Sciences and Engineering, VI, R. Glowinski and J. L. Lions, eds., North–Holland, Amsterdam, New York, Oxford, 1984, pp. 33-45.

[25] ———, *Iterative substructuring methods: Algorithms and theory for elliptic problems in the plane*, in Proc. First International Symposium on Domain Decomposition for Methods for Partial Differential Equations, R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 113-128.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 6, pp. 1063-1075, November 1989
© 1989 Society for Industrial and Applied Mathematics

ON OPTIMAL INTERPOLATION TRIANGLE INCIDENCES*

E. F. D'AZEVEDO† AND R. B. SIMPSON†

Abstract. The problem of determining optimal incidences for triangulating a given set of vertices for the model problem of interpolating a convex quadratic surface by piecewise linear functions is studied. An exact expression for the maximum error is derived, and the optimality criterion is minimization of the maximum error. The optimal incidences are shown to be derivable from an associated Delaunay triangulation and hence are computable in $O(N \log N)$ time for N vertices.

Key words. optimal mesh, triangulation incidence, surface approximation

AMS(MOS) subject classifications. 65D05, 65L50

1. Introduction. In this paper, we study the question of an optimal choice of edge incidences for triangulating a given set of points. The study uses a model approximation problem, piecewise linear interpolation of a convex quadratic surface, and our optimality criterion is that the optimal choice of incidences minimizes the maximum error in any triangle. We establish that the optimal incidence problem can be transformed to an equivalent Delaunay triangulation problem, showing in particular that at least this model optimal incidence problem can be solved in time $O(N \log N)$ for N interpolation points.

General triangles have two independent length scales associated with them, e.g., the longest edge and the length of the perpendicular from this edge to the opposite vertex. It is common to regard the local error over a triangle T as depending on one length scale (the "size" of T, typically denoted "h") and to impose a geometric condition on the triangulation, i.e., small angles should be avoided. Strang and Fix [15] have developed an error bound that depends on the reciprocal of the sine of the minimum interior angle. However, Babuška and Aziz have shown [3] that actually small angles do not play a crucial role in approximation properties, but that limiting the largest angle is necessary and sufficient for convergence. Indeed, for a convex surface in which the curvature in the principal direction is markedly different from the curvature in the perpendicular direction, incidences producing triangles with small angles are appropriate, and are present in an optimal triangulation incidence, as shown in Example 2 below.

A commonly used incidence relation for a set of vertices is the geometrically defined Delaunay triangulation [9], [12], [13]. In § 1, we give an example of quadratic functions, and a series of sets of vertices, for which the Delaunay triangulation can be arbitrarily far from optimal. Representations of the error in linear interpolation for a general triangle and general quadratic function are surprisingly complicated (e.g., [4]-[7]). In § 2, we use a geometric argument to develop both analytic and geometric descriptions of this error, and of its maximum, over a general triangle. In § 3, the main result of the paper is presented, i.e., that an optimal triangle incidence for this interpolation problem can be obtained from a Delaunay triangulation of a transform of the given vertex set. For this result, we establish a new geometric optimality property of the Delaunay triangulation concerning minimizing the maximum circumcircle of the triangulation.

* Received by the editors May 9, 1988; accepted for publication (in revised form) October 6, 1988. This work was supported by the Natural Science and Engineering Council of Canada and the government of Ontario through the Information Technology Research Centre.

† Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.

For the quadratic model problem, the determination of optimal incidences is based on a criterion that is locally applied to a transform of the given vertices (see § 3). For more general smooth functions, this procedure can be used locally to determine appropriate triangle incidences if an adequate estimate for the Hessian matrix of the data function is available [1], [2], [14].

1.1. Example 1. To see the influence of the choice of triangle incidence on the accuracy of approximation, let us look at an example of piecewise linear approximation of the quadratic function

$$f(x, y) = \lambda_1 x^2 + \lambda_2 y^2, \qquad \lambda_1 \gg \lambda_2 > 0.$$

Consider the interpolation error over the square (see Fig. 1.1), if the y-axis is chosen as a diagonal, maximum interpolation error is

$$E_y = (\lambda_1 + \lambda_2)^2/(4\lambda_1)$$

and occurs at $P_1(P_2)$. (Expressions for the errors are developed in § 2 below.) If the x-axis is chosen, maximum interpolation error is

$$E_x = \lambda_1$$

at the origin (P_3) (see (2.5) below). The ratio of the error from these two incidences is

$$\frac{E_x}{E_y} = \frac{\lambda_1}{(\lambda_1+\lambda_2)^2/(4\lambda_1)} = \frac{4}{(1+\lambda_2/\lambda_1)^2} \leqq 4.$$

Consider the same interpolation problem with extra nodes, $(ih, 1)$, $(-ih, 1)$, $(ih, -1)$, $(-ih, -1)$, $i \in \{1, 2, \cdots, N\}$ and $h = 1/N$. A standard choice of triangle incidences for a given set of vertices is the Delaunay triangulation introduced in § 3, along with some of its geometric optimality properties. If a Delaunay triangulation incidence is chosen, including the x-axis as shown in Fig. 1.2, the maximum interpolation error is dominated by E_x. Consider the triangle incidences in Fig. 1.3. The maximum interpolation error for these triangle incidences is

$$E_h = (h^2\lambda_1 + 4\lambda_2)/4.$$

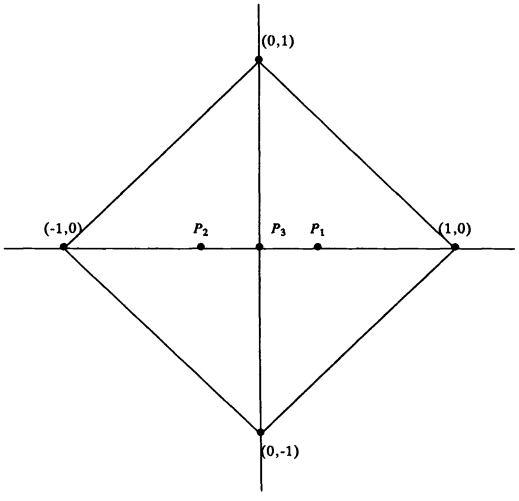


FIG. 1.1

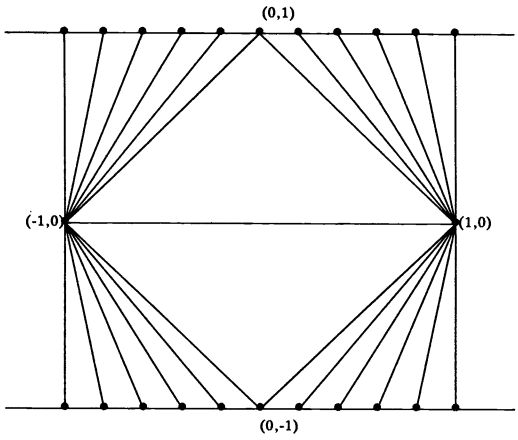


FIG. 1.2

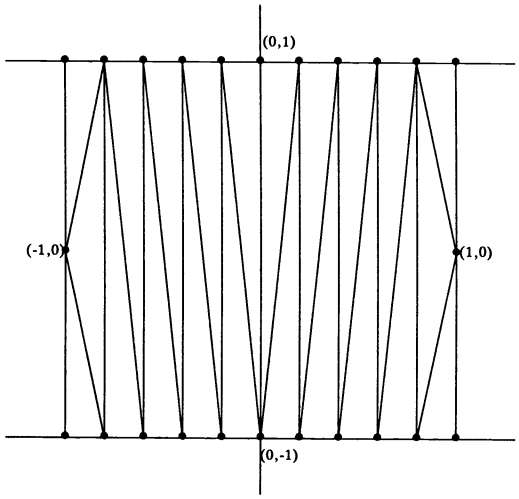


FIG. 1.3

Then the ratio of errors,

$$\frac{E_h}{E_x}=\frac{(h^2\lambda_1+4\lambda_2)}{(4\lambda_1)}=\frac{h^2}{4}+\frac{\lambda_2}{\lambda_1}.$$

For h small (N large),

$$\frac{E_h}{E_x}\approx\frac{\lambda_2}{\lambda_1}.$$

Hence the Delaunay triangle incidence can be arbitrarily far from optimal with respect to minimizing the maximum error.

1.2. Example 2. Here is a specific example of interpolation of a convex function with markedly different curvature. The data function to be interpolated is $f(x, y)=100x^2+y^2$. Figure 1.4 shows the Delaunay triangulation for an arbitrary set of points over the unit square and Fig. 1.5 shows the optimal triangulation incidence determined in this work. The errors of interpolation for both triangulations are displayed in Fig.

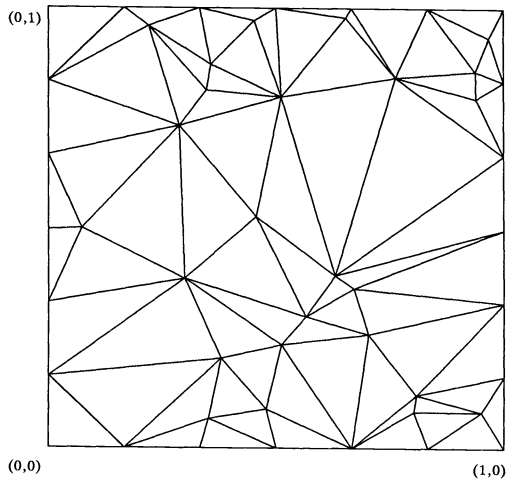


FIG. 1.4

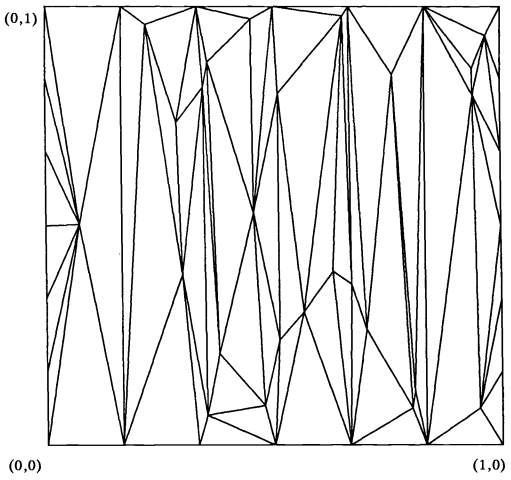


FIG. 1.5

1.6, where the maximum interpolation error over each triangle is plotted in ascending order. Note that the maximum error for the Delaunay triangulation is nearly six times larger than the one for the optimal triangulation.

2. Model problem error expressions. Error expressions to describe the interaction between the parameters of a general triangle and the relevant parameters of a general function for even piecewise linear interpolation are surprisingly complicated (e.g., [4]-[7]). In our discussion, we model a general function by a convex quadratic polynomial; i.e., let

$$f(\mathbf{x}) = a + \mathbf{d}^t\mathbf{x} + \mathbf{x}^t H\mathbf{x} \tag{2.1}$$

where H is symmetric with eigenvalues $\lambda_1 \geqq \lambda_2 > 0$, and without any further loss of generality, we may assume that the coordinate axes are aligned with the principal axes of H so that

$$f(x, y) = a + d_1 x + d_2 y + \lambda_1 x^2 + \lambda_2 y^2. \tag{2.2}$$

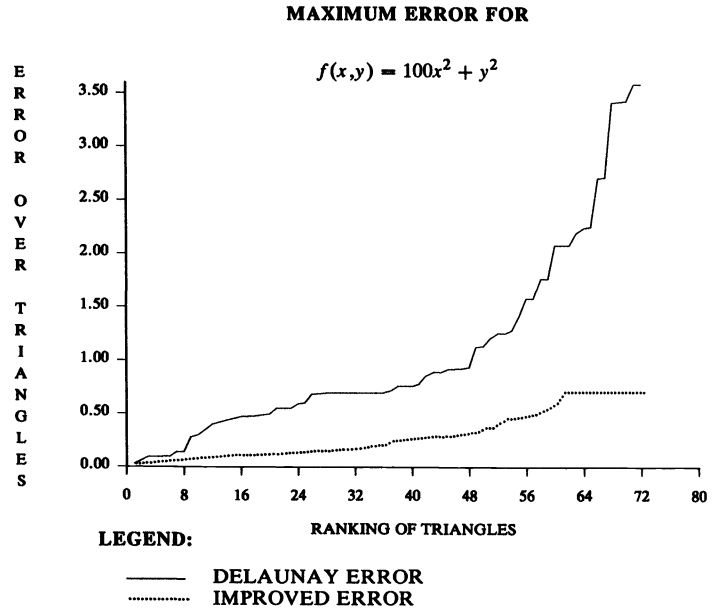


FIG. 1.6

The error function for the linear interpolation is a quadratic form in x, y with the same quadratic terms. We represent it as

$$E(x, y) = \lambda_1 x^2 + \lambda_2 y^2 + b_1 x + b_2 y + c \tag{2.3}$$

where the values of b_1, b_2, and c depend on the coordinates of the triangle vertices. Now $E(x, y)$ vanishes at the triangle vertices. Since the level curves of $E(x, y)$ are ellipses, then the zero error level curve of $E(x, y)$ is a circumscribing ellipse of the triangle. We will denote the ellipse as $e(T)$ for triangle T. The equation of the ellipse for the level curve of value $-K$ has the form

$$\begin{aligned} & E(x, y) = -K \\ & \Rightarrow \lambda_1 x^2 + \lambda_2 y^2 + b_1 x + b_2 y + c = -K \\ & \Rightarrow \lambda_1 \left(x + \frac{b_1}{2\lambda_1}\right)^2 + \lambda_2 \left(y + \frac{b_2}{2\lambda_2}\right)^2 = E - K \quad \text{where } E = \frac{b_1^2}{4\lambda_1} + \frac{b_2^2}{4\lambda_2} - c \\ & \Rightarrow \left(x + \frac{b_1}{2\lambda_1}\right)^2 \Big/ \left(\frac{(E-K)}{\lambda_1}\right) + \left(y + \frac{b_2}{2\lambda_2}\right)^2 \Big/ \left(\frac{(E-K)}{\lambda_2}\right) = 1. \end{aligned} \tag{2.4}$$

The parameters b_1, b_2, and c can be explicitly computed by requiring the ellipses for $K = 0$ to be the circumellipse of T. At the center, $(-b_1/(2\lambda_1), -b_2/(2\lambda_2))$ of $e(T)$, $|E(x, y)|$ attains a maximum value that can be expressed as follows:

$$E = \frac{(D_{12} D_{23} D_{31})}{16 \lambda_1 \lambda_2 A^2} \quad \text{where } D_{ij} = (\lambda_1 (x_i - x_j)^2 + \lambda_2 (y_i - y_j)^2) \tag{2.5}$$

and (x_i, y_i) $i = 1, 2, 3$ are vertices of the triangle, and A is area of the triangle. The details of the derivation can be found in Appendix A.

The maximum interpolation error for a triangle T will be denoted as

$$E_{\max}(T) = \max_{(x,y) \in T} |p_f(x, y) - f(x, y)| \tag{2.6}$$

where $p_f(x, y)$ is the linear interpolant of f at the vertices of T. In the case that the

center of $e(T)$ lies in T (including the boundary of T), $E_{\max}(T) = E$ and we note that the area of $e(T)$ is $A_1 = \pi E/\sqrt{\lambda_1\lambda_2}$, so that

$$E_{\max}(T) = E = \sqrt{\lambda_1\lambda_2}A_1/\pi. \tag{2.7}$$

If the center of $e(T)$ is not in T (see Fig. 2.1), $E_{\max}(T)$ is attained on an edge of T. Geometrically, $E_{\max}(T)$ is the level of the osculating level curve (ellipse) tangent to this side, $|E(x, y)| = E_{\max}(T)$. The area of the tangent ellipse is from (2.4):

$$\pi(E - E_{\max}(T))/\sqrt{\lambda_1\lambda_2} = A_2.$$

Thus $E_{\max}(T)$ can be expressed in terms of the ratio $\rho = A_2/A_1$ of these areas and E can be expressed as follows:

$$E_{\max}(T) = (1-\rho)E. \tag{2.8}$$

Consider a rescaling along x-axis by $(\lambda_1/\lambda_2)^{1/2}$ to transform the elliptical level curves to concentric circles (see Fig. 2.2). Since the area of an ellipse is directly proportional to the area of the corresponding circle in the transformed plane,

$$E_{\max}(T) = (1-\rho)E = \left(1 - \frac{\pi(OD)^2}{\pi(OA)^2}\right)E = \left(\frac{OA^2 - OD^2}{OA^2}\right)E = \left(\frac{AD^2}{OA^2}\right)E \tag{2.9}$$

where D is midpoint of AC tangent to the inner circle.

From this result in the transformed plane, we gain a geometric interpretation of the maximum error. Let $\bigcirc(T)$ denote the circumcircle of the transformed triangle T

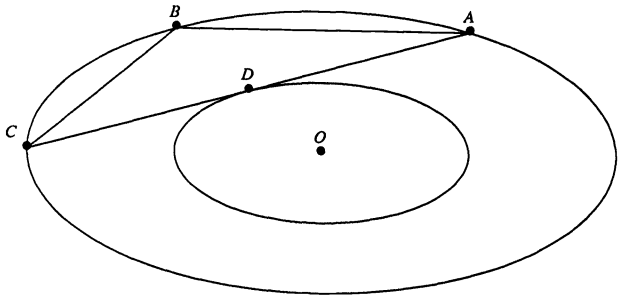


FIG. 2.1

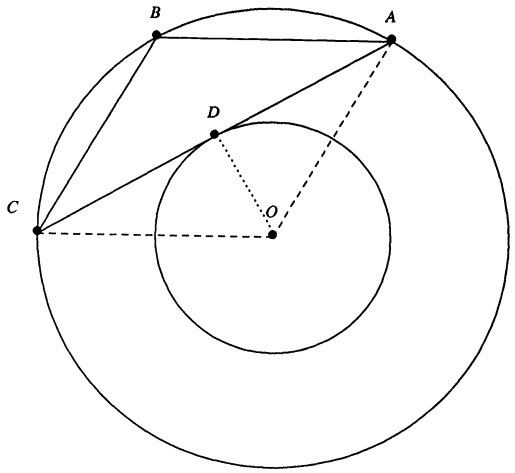


FIG. 2.2

and $|\bigcirc(T)|$ denote the area of this circumcircle. If the transformed image of a triangle has no obtuse angle, then its maximum error is proportional to the area of its circumscribing circle. If there is an obtuse angle in the transformed triangle, the maximum error occurs along the longest edge and the error is proportional to the area of circle with the longest edge as diameter.

3. Delaunay triangulation. The Delaunay triangulation of a fixed set of vertices and its related figure, the Voronoi diagram, are much studied geometric constructions (see [9], [12], [13]). Here, we briefly review some properties that are relevant for optimal interpolation incidence. The Delaunay triangulation selects triangle incidences that maximize the minimum angle in the triangulation. Lawson [9] has proposed an algorithm for converting an arbitrary triangulation to a Delaunay one by repeated application of a local edge-swapping procedure. In it, for each interior edge of the current triangulation, the two neighboring triangles are examined. If they form a convex quadrilateral, and if the replacement of the examined edge by the other diagonal of this quadrilateral would increase the minimimum angle, then a swap of diagonals is made. Lawson has shown that this criterion for picking the diagonal in a convex quadrilateral is characterized by the property that the circumcircles of either of the two triangles thus formed do not contain the fourth vertex of the quadrilateral.[1] This criterion is referred to as the empty circle criterion. Lawson has shown that the repeated application of his edge examination/edge-swapping procedure terminates in a Delaunay triangulation.

3.1. Optimal incidence for the model problem. We shall define a triangle incidence to be globally optimal if it minimizes the maximum interpolation error. Each interior edge in a triangulation is associated with a quadrilateral with that edge as diagonal. We shall also define a triangular mesh incidence to be locally optimal if for each convex quadrilateral associated by an interior edge in the triangulation, the incidence minimizes the maximum interpolation error over the quadrilateral.

Here we show that the problem of constructing a locally optimal mesh incidence for N points can be transformed to the problem of generating a Delaunay triangulation which provides an $O(N \log N)$ algorithm for solving this problem. We also show that a locally optimal incidence is globally optimal.

The rescaling of the x-axis introduced at the end of § 2, that results in error ellipses being mapped into circles will be used here to define the transform plane for which we do not explicitly introduce coordinates.

THEOREM 1. *A locally optimal interpolation triangulation incidence of N vertices is defined by a Delaunay triangulation in the transformed plane (and hence is computable in $O(N \log N)$ time, which is optimal).*

To simplify our discussion, we shall use the notation that all references to angles are based on the labelling in Fig. 3.1. Moreover, we shall always assume vertex A to be exterior to $\bigcirc(\triangle BCD)$. The diagonal BD is the incidence selected by the empty circle criterion. By elementary geometry, we have the following:

$$\angle CAD = \theta_1 < \theta_2 = \angle CBD, \qquad \angle BAC = \phi_1 < \phi_2 = \angle BDC,$$

$$\angle DCA = \gamma_1 < \gamma_2 = \angle DBA, \qquad \angle ACB = \eta_1 < \eta_2 = \angle ADB.$$

LEMMA 1. *Given a convex quadrilateral ABCD with vertex A exterior to $\bigcirc(\triangle BCD)$, then* $\max(|\bigcirc(\triangle ABC)|, |\bigcirc(\triangle ADC)|) \geqq \max(|\bigcirc(\triangle BCD)|, |\bigcirc(\triangle ABD)|)$.

[1] If the fourth vertex lies on the boundary of a circumcircle, then no change in the minimum angle occurs from edge swapping and either choice of edge results in a Delaunay triangulation.

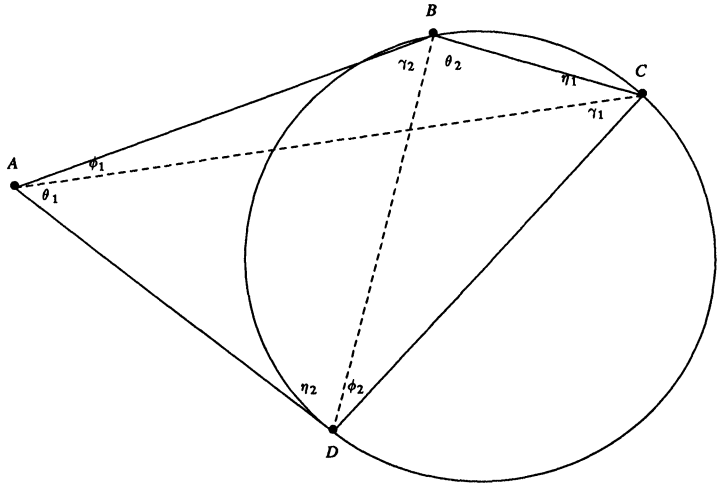


FIG. 3.1

Proof of Lemma 1.

Case 1. Assume that $\theta_2 \leqq \pi/2$. Let E be the midpoint of CD; let O_1, O_2 be centers for $\bigcirc(\triangle ADC)$ and $\bigcirc(\triangle BCD)$, respectively. The O_1 and O_2 lie on the perpendicular to DC through E, so O_1, O_2, and E are collinear. Let A', B' be points where this line meets $\bigcirc(\triangle ADC)$ and $\bigcirc(\triangle BCD)$ (see Fig. 3.2).

$$\angle EA'C = \frac{\theta_1}{2}, \qquad \angle EB'C = \frac{\theta_2}{2},$$

(3.1)

$$\tan\left(\frac{\theta_2}{2}\right) = \frac{EC}{EB'} = \frac{EC}{EO_2 + O_2B'} = \frac{EC}{EO_2 + O_2C}.$$

Now from $\triangle O_2EC$ we have

$$(O_2C)^2 = (O_2E)^2 + (EC)^2.$$

Thus (3.1) can be rewritten as follows:

$$\tan\left(\frac{\theta_2}{2}\right) = \frac{EC}{O_2C + ((O_2C)^2 - (EC)^2)^{1/2}}.$$

Similarly,

$$\tan\left(\frac{\theta_1}{2}\right) = \frac{EC}{O_1C + ((O_1C)^2 - (EC)^2)^{1/2}},$$

(3.2)

$$\theta_1 \leqq \theta_2 \leqq \frac{\pi}{2} \Rightarrow \tan\left(\frac{\theta_1}{2}\right) \leqq \tan\left(\frac{\theta_2}{2}\right).$$

Therefore

$$\frac{EC}{O_1C + ((O_1C)^2 - (EC)^2)^{1/2}} \leqq \frac{EC}{O_2C + ((O_2C)^2 - (EC)^2)^{1/2}}$$

(3.3)

$$\Rightarrow O_2C + ((O_2C)^2 - (EC)^2)^{1/2} \leqq O_1C + ((O_1C)^2 - (EC)^2)^{1/2}.$$

Since $x + (x^2 - a)^{1/2}$ is a monotone increasing function of x, (3.3) implies $O_2C \leqq O_1C$. Therefore $\bigcirc(\triangle ADC)$ has larger radius than $\bigcirc(\triangle BCD)$. Similarly, if $\phi_2 \leqq \pi/2$, then $\bigcirc(\triangle ABC)$ has larger radius than $\bigcirc(\triangle BCD)$.

Case 2. Assume $\theta_2 > \pi/2$. From $\triangle BCD$, we have $\phi_2 + \theta_2 < \pi$, and thus $\phi_2 \leqq \pi/2$ if $\theta_2 > \pi/2$. From the result in Case 1, $|\bigcirc(\triangle ABC)| \geqq |\bigcirc(\triangle BCD)|$. Similarly, by symmetry, C would be exterior to $\bigcirc(\triangle ABD)$.

Thus either $\bigcirc(\triangle ACD)$ or $\bigcirc(\triangle ABC)$ would be larger than $\bigcirc(\triangle ABD)$. Since Cases 1 and 2 are exhaustive, the lemma is proved.

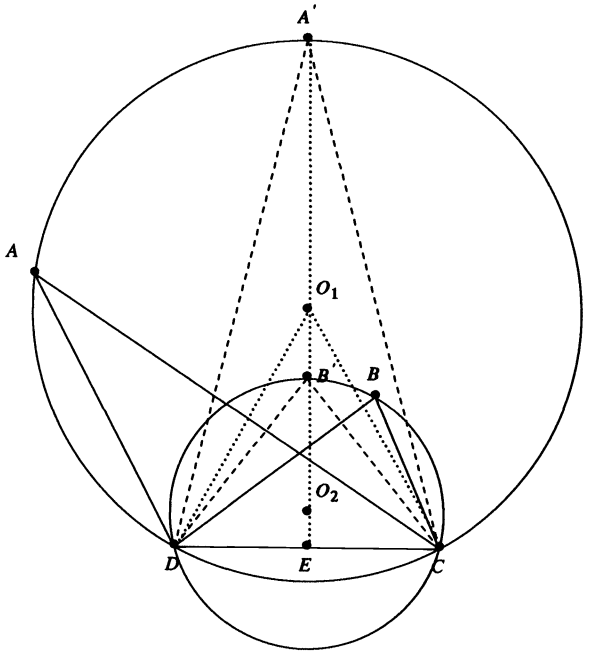


FIG. 3.2

COROLLARY. *The empty circle criterion applied to a convex quadrilateral selects the diagonal that minimizes the maximum circumcircle of the corresponding triangles.*

LEMMA 2. *Given a convex quadrilateral ABCD with vertex A exterior to* $\triangle BCD$ *the empty circle criterion selects the triangulation incidence that minimizes the area of the circle corresponding to the maximum interpolation error over the quadrilateral.*

Proof of Lemma 2. Recall from § 2 that the maximum interpolation error for a triangle is proportional to the area of the circumcircle of the transformed image of the triangle, if this image contains no obtuse angle. Otherwise, the maximum interpolation error is proportional to the area of the circle with diameter equal to the edge opposite the obtuse angle, i.e., the longest edge of the image triangle.

The first case was dealt with in Lemma 1 and its corollary in which we established that the diagonal BD is selected by the empty circle criterion. We now carry out a case-by-case study of configurations of image triangles with obtuse angles to show that the empty circle criterion also minimizes the maximum error circles. Hence we shall consider only cases where $\triangle ABC$ or $\triangle ACD$ contains an obtuse angle.

Case 1. Assume $\phi_1 \geqq \pi/2$. (See Fig. 3.3.) We show that BC is the longest edge and it determines the error circle, regardless of the choice of diagonals. The area of the error circle for $\triangle ABD$ is $\pi(BD/2)^2$, and for $\triangle BCD$ it is $\pi(BC/2)^2$. Note that $\phi_2 \geqq \phi_1 \geqq \pi/2$. $|\bigcirc(\triangle ABC)|$ is $\pi(BC/2)^2$, and $|\bigcirc(\triangle ADC)|$ is $\pi(AC/2)^2$. Now $\phi_1 < \phi_2$, and thus $|BC| > |BD|$. Since BD cannot be the longest chord, the lemma holds. By symmetry, the same applies if θ_1, η_1, γ_1 is obtuse.

Case 2. Assume $\angle ABC$ and $\angle ADC$ are both obtuse (see Fig. 3.4). The error circle for each of $\triangle ABC$ and $\triangle ACD$ would be the circle with AC as diameter. Let this

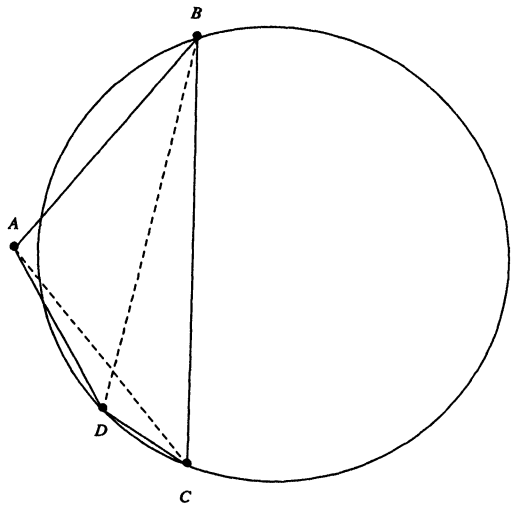


FIG. 3.3

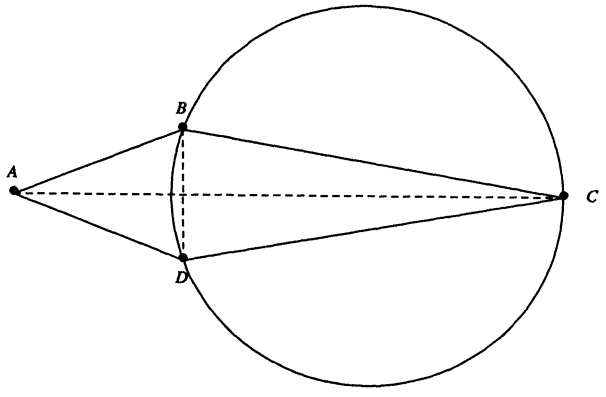


FIG. 3.4

circle be denoted by Γ. Both vertices B and C are contained in Γ. If triangle BCD has an obtuse angle, its error is indicated by its longest edge, which is entirely interior to Γ. If $\triangle BCD$ has no obtuse angle, its circumcircle is smaller than Γ. Hence error for $\triangle BCD$ is smaller than error for triangle ABC. The same reasoning applies for $\triangle ABD$. Therefore diagonal BD should be chosen, and the circle criterion holds.

Case 3. Assume $\angle ABC \leqq \pi/2$ is acute but $\angle ADC > \pi/2$ is obtuse, and ϕ_1, θ_1, η_1, γ_1 are all acute (see Fig. 3.5). The error for $\triangle ABC$ is proportional to the area of its circumcircle,

$$|\bigcirc(\triangle ABC)| = \frac{\pi(AC/2)^2}{\sin^2(\gamma_2+\theta_2)} = \frac{\pi(BC/2)^2}{\sin^2(\phi_1)} = \frac{\pi(AB/2)^2}{\sin^2(\eta_1)}. \tag{3.4}$$

If η_2 is obtuse, $|\bigcirc(\triangle ABD)|$ is $\pi(AB/2)^2 < |\bigcirc(\triangle ABC)|$; otherwise, it is bounded by the area of the circumcircle $\pi(AB/2)^2/\sin^2(\eta_1)$. If $\eta_2 \leqq \pi/2$, then $\eta_1 \leqq \eta_2$ and $\sin(\eta_1) \leqq \sin(\eta_2)$. Therefore, the error for $\triangle ABD$ is less than that of $\triangle ABC$.

Similarly, if ϕ_2 is obtuse, the area for $\bigcirc(\triangle BCD)$ is $\pi(BC/2)^2 < |\bigcirc(\triangle ABC)|$; otherwise, it is bounded by the area of its circumcircle $\pi(BC/2)^2/\sin^2(\phi_1)$. If $\phi_2 \leqq \pi/2$,

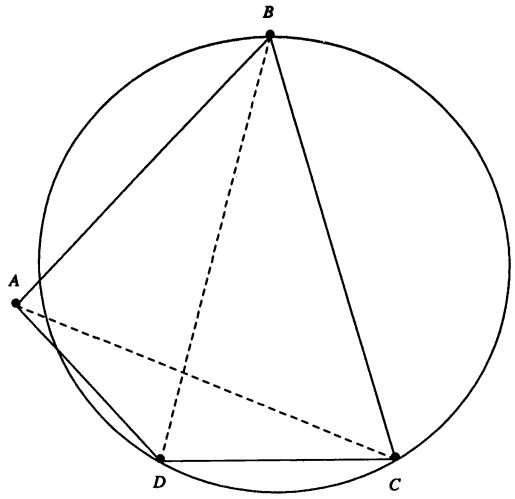


FIG. 3.5

then $\phi_1 \leqq \phi_2$ and $\sin(\phi_1) \leqq \sin(\phi_2)$. Therefore, the error for $\triangle BCD$ is less than that of $\triangle ABC$, and the empty circle criterion selects the incidence minimizing the maximum error.

Cases 1, 2, and 3 exhaust all possibilities for an obtuse angle in $\triangle ABC$ or $\triangle ACD$. The lemma is proved.

Proof of Theorem 1. For the image of the interpolation vertices in the transform plane, a triangulation that satisfies the empty circle criterion can be constructed in $O(N \log N)$ time [12, Thm. 5.18, p. 215]. The convex quadrilaterals of this Delaunay triangulation and of the triangulation induced on the interpolation vertices are in one-to-one correspondence. But by Lemmas 1 and 2, the incidences of the Delaunay triangulation minimize the interpolation error circles with respect to diagonal interchanges, and hence the induced triangulation is locally optimal.

COROLLARY. *The locally optimal triangle incidence of Theorem* 1 *defines a globally optimal triangle incidence.*

Proof. Starting with any globally optimal triangle incidence, we can apply the local edge-swapping procedure in the transformed plane to obtain a locally optimal incidence with the same maximum error. However, the size of the maximum error circle is uniquely determined for Delaunay triangulation in the transformed plane, so the maximum error in the original globally optimal incidence cannot be smaller than that of a locally optimal incidence, and hence a Delaunay triangulation also defines a globally optimal incidence. Note, however, a globally optimal triangulation incidence need not be a Delaunay triangulation.

The use of the transform plane appears in several related contexts in the literature. Nadler [10] uses it to establish the shape of triangulation for optimal L_2 linear interpolation for model quadratic data. Also in [11], Peraire et al. use it to support an adaptive remeshing scheme for the finite-element method for compressible flow computations.

Appendix A.

Derivation of interpolation error. Let the interpolation error over the triangle be given by

$$E(x, y) = \lambda_1 x^2 + \lambda_2 y^2 + b_1 x + b_2 y + c \tag{A1}$$

and by the interpolation condition

$$E(x_1, y_1) = E(x_2, y_2) = E(x_3, y_3) = 0$$

at the three vertices of the triangle. The unknowns b_1, b_2, c can easily be obtained by solving the following system of linear equations:

$$\text{(A2)} \qquad \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ c \end{bmatrix} = \begin{bmatrix} -r_1 \\ -r_2 \\ -r_3 \end{bmatrix}$$

where $r_i = \lambda_1 x_i^2 + \lambda_2 y_i^2$. By Cramer's rule,

$$\text{(A3)} \qquad b_1 = \frac{\det \begin{bmatrix} -r_1 & y_1 & 1 \\ -r_2 & y_2 & 1 \\ -r_3 & y_3 & 1 \end{bmatrix}}{D}, \quad b_2 = \frac{\det \begin{bmatrix} x_1 & -r_1 & 1 \\ x_2 & -r_2 & 1 \\ x_3 & -r_3 & 1 \end{bmatrix}}{D}, \quad c = \frac{\det \begin{bmatrix} x_1 & y_1 & -r_1 \\ x_2 & y_2 & -r_2 \\ x_3 & y_3 & -r_3 \end{bmatrix}}{D},$$

where A is area of triangle and

$$D = \det \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} = 2A.$$

Now by (2.4)

$$\text{(A4)} \qquad E_{\max} = \frac{b_1^2}{4\lambda_1} + \frac{b_2^2}{4\lambda_2} - c.$$

The substitution of (A3) into (A4) and its simplification was obtained through the algebraic computation system MAPLE [8]:

$$\text{(A5)} \qquad E = \frac{(D_{12} D_{23} D_{31})}{16 \lambda_1 \lambda_2 A^2} \quad \text{where } D_{ij} = (\lambda_1 (x_i - x_j)^2 + \lambda_2 (y_i - y_j)^2).$$

$E_{\max}$ represents the global maximum interpolation error obtained at $(-b_1/(2\lambda_1), -b_2/(2\lambda_2))$. It can be shown by calculus that the local maximum error along the boundary is attained at the midpoint of each edge. The maximum error along edge $(x_i, y_i), (x_j, y_j)$ is $|D_{ij}/4|$.

REFERENCES

[1] H. AKIMA, *A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points*, ACM Trans. Math. Software, 4 (1978), pp. 148–159.

[2] ———, *On estimating partial derivatives for bivariate interpolation of scattered data*, Rocky Mountain J. Math., 14 (1984), pp. 41–52.

[3] I. BABUŠKA AND A. K. AZIZ, *On the angle condition in the finite element method*, SIAM J. Numer. Anal., 13 (1976), pp. 214–226.

[4] R. E. BARNHILL AND J. A. GREGORY, *Interpolation remainder theory from Taylor expansions on triangles*, Numer. Math., 25 (1976), pp. 401–408.

[5] ———, *Sard kernel theorems on triangular domains with application to finite element error bounds*, Numer. Math., 25 (1976), pp. 215–229.

[6] J. H. BRAMBLE AND S. R. HILBERT, *Estimation of linear functionals on Sobolev spaces with application to Fourier transformations and spline interpolation*, SIAM J. Numer. Anal., 7 (1970), pp. 112–124.

[7] ———, *Bounds for a class of linear functionals with application to Hermite interpolation*, Numer. Math., 16 (1971), pp. 362-369.

[8] B. W. CHAR, K. O. GEDDES, G. H. GONNET, AND S. M. WATT, MAPLE: *Reference manual*, 1985.

[9] C. L. LAWSON, *Software for C^1 surface interpolation*, in Mathematical Software III, John R. Rice, ed., Academic Press, New York, 1977, pp. 161-194.

[10] E. NADLER, *Piecewise linear best L_2 approximation on triangulations*, in Approximation Theory V, J. D. Ward, ed., Academic Press, ISBN 0-12-174581-3, 1986, pp. 499-502.

[11] J. PERAIRE, M. VAHDATI, K. MORGAN, AND O. C. ZIENKIEWICZ, *Adaptive remeshing for compressible flow computations*, J. Comput. Phys., 72 (1987), pp. 449-466.

[12] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry: An Introduction*. Springer-Verlag, Berlin, New York, 1985.

[13] R. SIBSON, *Locally equiangular triangulations*, Comput. J., 21 (1978), p. 243-245.

[14] S. E. STEAD, *Estimation of gradients from scattered data*, Rocky Mountain J. Math., 14 (1984), pp. 265-279.

[15] G. STRANG AND G. FIX, *Analysis of the Finite Element Method*, Prentice Hall, Englewood Cliffs, NJ, 1973.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 6, pp. 1076-1096, November 1989
© 1989 Society for Industrial and Applied Mathematics

ON THE QUADRATIC CONVERGENCE OF THE SERIAL SINGULAR VALUE DECOMPOSITION JACOBI METHODS FOR TRIANGULAR MATRICES*

V. HARI†

Abstract. The quadratic convergence of the serial singular value decomposition (SVD) Jacobi methods for triangular matrices is proved. The obtained bounds are as sharp as those obtained by Wilkinson and Van Kempen for the symmetric Jacobi method. Special attention is paid to finding the structure of almost diagonal essentially triangular matrices with multiple singular values.

Key words. singular values, Jacobi method, quadratic convergence

AMS(MOS) subject classification. 65F15

CR classification. 5.14

1. Introduction. Jacobi methods for computing the singular value decomposition (SVD) of a square matrix A are iterative processes of the form

$$A^{(k+1)} = U_k^* A^{(k)} V_k, \qquad k \geqq 1 \tag{1.1}$$

where $A^{(1)} = A$ and U_k, V_k are unitary plane matrices. The matrices U_k, V_k are defined by a pair of (pivot) indices (l, m), $l < m$, called *pivot pair* and by 2×2 matrices

$$\hat{U}_k = \begin{bmatrix} u_{ll}^{(k)} & u_{lm}^{(k)} \\ u_{ml}^{(k)} & u_{mm}^{(k)} \end{bmatrix}, \qquad \hat{V}_k = \begin{bmatrix} v_{ll}^{(k)} & v_{lm}^{(k)} \\ v_{ml}^{(k)} & v_{mm}^{(k)} \end{bmatrix},$$

here called *pivot submatrices* of U_k, V_k, respectively. Except for the elements at positions (l, l), (l, m), (m, l), and (m, m), $U_k = (u_{ij}^{(k)})$ and $V_k = (v_{ij}^{(k)})$ are equal to the identity matrix I_n, where n is order of A. The elements of $\hat{U}_k$, $\hat{V}_k$ are determined from the requirement

$$a_{lm}^{(k+1)} = 0, \qquad a_{ml}^{(k+1)} = 0,$$

where $A^{(k)} = (a_{ij}^{(k)})$, $k \geqq 1$. The transformation $A^{(k)} \to A^{(k+1)}$ is called the *kth step* of the process (1.1). Each SVD Jacobi method is defined by an algorithm for computing the elements of $\hat{U}_k$, $\hat{V}_k$ and by a rule for selecting the pivot pairs (*pivot strategy*). Of special interest are the *cyclic* (*pivot*) *strategies.* Each one is defined by an ordering of the set $P_n = \{(i, j), 1 \leqq i < j \leqq n\}$. Among cyclic strategies the most common are the two *serial strategies*: the *column-* and the *row-cyclic.* The corresponding orderings of P_n are

$$(1, 2), (1, 3), (2, 3), \cdots, (1, n), (2, n), \cdots, (n-1, n)$$

and

$$(1, 2), (1, 3), \cdots, (1, n), (2, 3), \cdots, (2, n), \cdots, (n-1, n),$$

respectively. The sequence of

$$N = \tfrac{1}{2} n(n-1)$$

successive matrices, $A^{((r-1)N+1)}, \cdots, A^{(rN)}$, shall be referred to as the rth *cycle* of the process (1.1), and the transformation involved in passing from $A^{((r-1)N+1)}$ to $A^{(rN)}$, as the rth *sweep.*

* Received by the editors December 11, 1986; accepted for publication (in revised form) August 19, 1988.
† Department of Mathematics, University of Zagreb, 41001 Zagreb, Yugoslavia.

SVD Jacobi methods are also called "Kogbetliantz methods" for Kogbetliantz's very first contributions ([13], [14]). In the sequel we shall call them briefly SVD methods.

An SVD method is (*globally*) *convergent* if

$$\lim_k |A^{(k)}| = \Sigma_A$$

holds for (every initial matrix) A. Here $|A| = (|a_{ij}|)$ provided that $A = (a_{ij})$, and

$$\Sigma_A = \operatorname{diag}(\sigma_1, \cdots, \sigma_n),$$

where $\sigma_1, \cdots, \sigma_n$ is an ordering of the singular values of A. A useful measure for convergence is $\|\cdot\|_{\text{off}}$,

$$\|A\|_{\text{off}} = \|A - \operatorname{diag}(A)\|,$$

where $\|\cdot\|$ denotes the Euclidean (Frobenius) matrix norm and $\operatorname{diag}(A) = \operatorname{diag}(a_{11}, \cdots, a_{nn})$. This measure is called the departure from the diagonal form (see [16]) or simply the off-norm (see [2], [3], [18]).

An SVD method converges (asymptotically) *quadratically* if $\|A^{(k)}\|_{\text{off}}$ reduces quadratically per cycle, or, more precisely, if

$$\|A^{(N+1)}\|_{\text{off}} \leqq \text{const}\, \|A\|_{\text{off}}^2$$

holds for each A with sufficiently small $\|A\|_{\text{off}}$.

Although they can be well implemented on parallel computers (see [1]), SVD methods have some disadvantages. They work on full matrices and in the presence of multiple singular values the convergence is not quadratic (see [10], [18]). In theory, the global convergence under serial strategies has not been proved yet (see [4], [7], [8]).

The proper way to circumvent these shortcomings is to triangularize the initial matrix. In [12] and [17] it has been shown that the serial SVD methods do not essentially ruin the triangular form which then reduces the operation count and ensures the quadratic convergence (see [3]). This fact also makes the global convergence proof feasible (see [12], [8]). For implementation details, see [2] and [15].

Let A be upper-triangular. Then within an odd (even) cycle the nontrivial off-diagonal elements push downwards (upwards) so that after each odd (even) sweep the current matrix is lower- (upper-) triangular. At step k, $1 < k < N$. The pattern of the nontrivial elements of $A^{(k)}$ is presented in Fig. 1. The picture of $A^{(k+N)}$ is obtained if the matrix from Fig. 1 is transposed.

If A is lower-triangular, then after each odd (even) sweep the current matrix is upper- (lower-) triangular. Hence we can consider only the upper-triangular case.

In Fig. 1 the column-cyclic strategy has been assumed. For the row-cyclic strategy the distribution of the nonzero entries is quite analogous. In fact as far as the convergence of the serial SVD methods is concerned, we do not have to distinguish the row-cyclic from the column-cyclic strategy. Namely, Hansen [6] has proved that the matrices obtained after each sweep by the row- and column-cyclic methods are equal.

To save memory expenses, we can perform the whole process on an upper-triangular array. Let $T^{(k)} = (t_{ij}^{(k)})$, $k \geqq 1$ be the upper-triangular matrices defined by

$$T^{(k)} + T^{(k)T} = A^{(k)} + A^{(k)T}, \qquad k \geqq 1, \tag{1.2}$$

where $T^{(k)T}$ denotes the transpose of $T^{(k)}$. The transformation formulae for $T^{(k)}$,

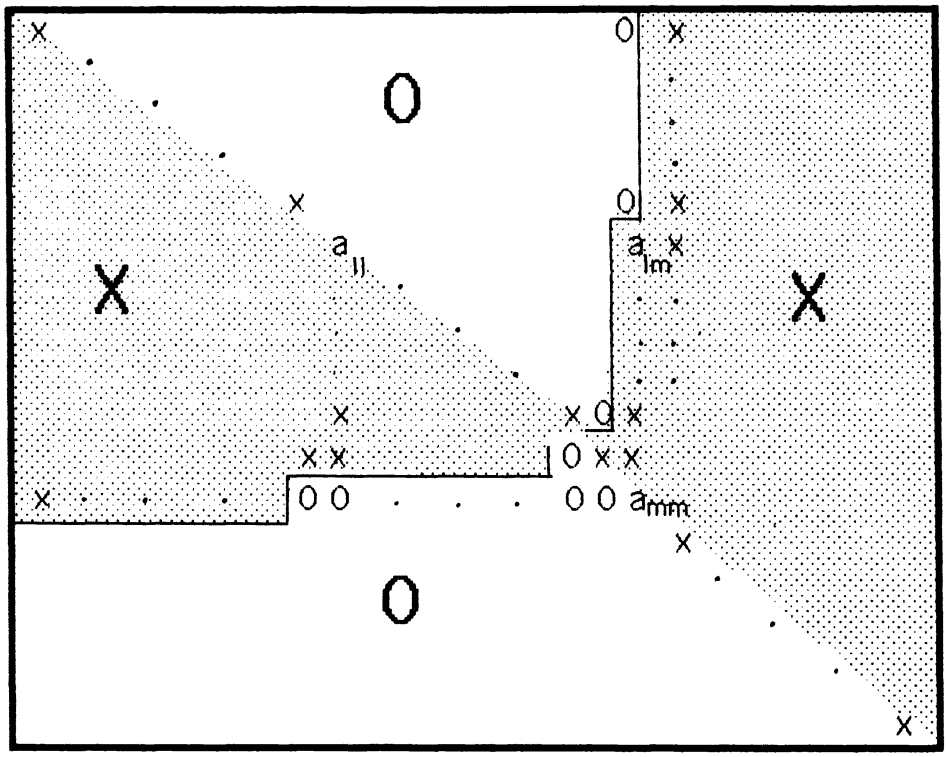


FIG. 1. *Matrix* $A^{(k)}$, $1 < k < N$.

$1 \leqq k < N$ are given:

$$
\left.\begin{aligned} t_{il}^{(k+1)} &= \bar{u}_{ll}^{(k)} t_{il}^{(k)} + \bar{u}_{ml}^{(k)} t_{im}^{(k)} \\ t_{im}^{(k+1)} &= \bar{u}_{lm}^{(k)} t_{il}^{(k)} + \bar{u}_{mm}^{(k)} t_{im}^{(k)} \end{aligned}\right\} \quad 1 \leqq i \leqq l-1,
$$

$$
\left.\begin{aligned} t_{li}^{(k+1)} &= v_{ll}^{(k)} t_{li}^{(k)} + v_{ml}^{(k)} t_{im}^{(k)} \\ t_{im}^{(k+1)} &= v_{lm}^{(k)} t_{li}^{(k)} + v_{mm}^{(k)} t_{im}^{(k)} \end{aligned}\right\} \quad l+1 \leqq i \leqq m-1, \tag{1.3}
$$

$$
\left.\begin{aligned} t_{li}^{(k+1)} &= \bar{u}_{ll}^{(k)} t_{li}^{(k)} + \bar{u}_{ml}^{(k)} t_{mi}^{(k)} \\ t_{mi}^{(k+1)} &= \bar{u}_{lm}^{(k)} t_{li}^{(k)} + \bar{u}_{mm}^{(k)} t_{mi}^{(k)} \end{aligned}\right\} \quad m+1 \leqq i \leqq n
$$

$$
t_{lm}^{(k)} = 0.
$$

The relation (1.3) holds for $A^{(k)}$ from an odd cycle. For even cycles roles of $\hat{U}_k$ and $\hat{V}_k$ in (1.3) are interchanged.

Since each pivot submatrix is triangular, the algorithms for computing $\hat{U}_k$ and $\hat{V}_k$ are also simplified. Therefore we call serial SVD methods for triangular matrix simply SVDT methods.

The aim of this paper is to derive sharp bounds for the quadratic convergence of a class of SVDT methods. In § 2 we investigate almost diagonal matrices with multiple singular values and in § 3 we consider SVDT methods on such matrices. The quadratic convergence results are proved in § 4.

Most of the results proved in this paper were part of our paper, *On the global and quadratic convergence of the serial* SVD *Jacobi methods for triangular matrices*, which was written in June 1986. Later a referee suggested that we concentrate on the quadratic

convergence, and so this paper came out. In the meantime, the paper of Charlier and Van Dooren [3] appeared. So we have laid stress on obtaining sharp bounds in the presence of multiple singular values which are not clustered.

2. Almost diagonal matrices. In considering the asymptotic behavior of Jacobi processes, it is essential to have insight into the structure of the underlying almost diagonal matrices. The matrices we deal with have a special property of being *hierarchically triangular* (see [12]), i.e., they are block-triangular matrices whose diagonal blocks are either triangular or again block-triangular. The final diagonal blocks are triangular. Such matrices are *permutationally similar to triangular* matrices (PST). A PST matrix A has two important properties: it is *essentially triangular* (ET), i.e.,

$$a_{ij}a_{ji}=0, \qquad i\neq j \tag{2.1}$$

holds, and its diagonal elements represent the eigenvalues of A. Here we consider almost diagonal PST matrices, but some results are formulated for ET matrices.

Let $A=(a_{ij})$ be a matrix of order n and let

$$\sigma_1=\cdots=\sigma_{s_1},\ \sigma_{s_1+1}=\cdots=\sigma_{s_2},\ \cdots,\ \sigma_{s_{p-1}+1}=\cdots=\sigma_{s_p} \tag{2.2}$$

be an ordering of its singular values. Then

$$n_i=s_i-s_{i-1} \qquad (s_0=0,\ s_p=n) \tag{2.3}$$

is the multiplicity of σ_{s_i} $(1\leqq i\leqq p)$. Here we have assumed that $p>1$ since otherwise A is proportional to a unitary matrix and unitary PST matrices are diagonal. Let

$$\sigma_{s_r}=\sigma_{\min}=\min_{1\leqq i\leqq n}\sigma_i, \tag{2.4}$$

$$3\nu_i=\min_{\substack{1\leqq j\leqq p\\ j\neq i}}|\sigma_{s_i}-\sigma_{s_j}|, \qquad 1\leqq i\leqq p, \tag{2.5}$$

and

$$\nu=\min_{1\leqq i\leqq p}\nu_i. \tag{2.6}$$

Let

$$A=D+E, \qquad D=\operatorname{diag}(A). \tag{2.7}$$

Then $\|A\|_{\text{off}}=\|E\|$ and if $\|E\|$ is sufficiently small, then $|a_{11}|,\cdots,|a_{nn}|$ approximate the singular values of A (see [5, § 8.3]). Hence there is a permutation matrix P such that

$$\tilde{A}=P^TAP=\begin{bmatrix}\tilde{A}_{11}&\cdots&\tilde{A}_{1p}\\ \vdots&\ddots&\\ \tilde{A}_{p1}&&\tilde{A}_{pp}\end{bmatrix}, \tag{2.8}$$

where $|\tilde{A}_{ii}|$ is approximated by $\sigma_{s_i}I_{n_i}$ $(1\leqq i\leqq p)$. The matrix P depends on A and on the assumed ordering (2.2). We say that the diagonal elements of $\tilde{A}_{ii}$ are affiliated with σ_{s_i}, or that $\tilde{A}_{ii}$ is associated with σ_{s_i}. Obviously, $\tilde{A}_{ii}$ is of order n_i $(1\leqq i\leqq p)$, n_i being defined by the relation (2.3).

In particular, if

$$\|E\|_2<\nu, \tag{2.9}$$

then P can easily be found since the perturbation theorem implies that the diagonal elements of $|\tilde{A}_{ii}|$ lie within the circle of radius ν around σ_{s_i} $(1\leqq i\leqq p)$. Here $\|E\|_2^2=$ spr E^*E where spr is the spectral radius.

LEMMA 2.1. *Let A be an essentially triangular matrix satisfying the condition* (2.9) *and let $\tilde{A}$ be defined by the relations* (2.8) *and* (2.2). *Then for $1 \leqq i \leqq p$, $i \neq r$*

$$\begin{aligned} \|\tilde{A}_{ii}\|_{\text{off}} &\leqq \sqrt{2}\,\|(|\tilde{A}_{ii}|+|\tilde{A}_{ii}^T|)/2-\sigma_{s_i} I_{n_i}\| \\ &\leqq \frac{\sqrt{2}}{4}\frac{1}{\nu_i}\left[\|\tilde{A}_{ii}\|_{\text{off}}^2+\sum_{\substack{j=1\\ j\neq i}}^{p}(\|\tilde{A}_{ij}\|^2+\|\tilde{A}_{ji}\|^2)\right]. \end{aligned} \tag{2.10}$$

The relation (2.10) *holds for $i = r$ provided that*

$$2\sigma_{\min} \geqq 3\nu_r \quad \textit{or} \quad \sigma_{\min} = 0. \tag{2.11}$$

Proof. We can assume that the diagonal elements of A are nonnegative since otherwise we can work with $\phi^* A$, $\phi = \text{diag}\,(e^{i\arg(a_{11})}, \cdots, e^{i\arg(a_{nn})})$.

Let us consider the hermitian matrix

$$H = Q^*\begin{bmatrix} 0 & \tilde{A} \\ \tilde{A}^* & 0 \end{bmatrix} Q = \frac{1}{2}\begin{bmatrix} \tilde{A}+\tilde{A}^* & \tilde{A}-\tilde{A}^* \\ -(\tilde{A}-\tilde{A}^*) & -(\tilde{A}+\tilde{A}^*) \end{bmatrix}$$

where

$$Q = \frac{\sqrt{2}}{2}\begin{bmatrix} I_n & -I_n \\ I_n & I_n \end{bmatrix}.$$

Since Q is orthogonal, the eigenvalues of H are $\sigma_1, \cdots, \sigma_n, -\sigma_1, \cdots, -\sigma_n$. Let P_i be a permutation matrix such that

$$P_i^T(H-\sigma_{s_i} I_{2n})P_i = \begin{bmatrix} K_i & G_i^* \\ \underbrace{G_i}_{2n-n_i} & \underbrace{H_{ii}}_{n_i} \end{bmatrix}\begin{matrix} \}\,2n-n_i \\ \}\,n_i \end{matrix}, \qquad 1 \leqq i \leqq p,$$

where

$$\begin{aligned} H_{ii} &= (\tilde{A}_{ii}+\tilde{A}_{ii}^*)/2-\sigma_{s_i} I_{n_i}, \\ 2G_i &= [\tilde{A}_{i1}+\tilde{A}_{1i}^*, \cdots, \tilde{A}_{i,i-1}+\tilde{A}_{i-1,i}^*, \tilde{A}_{i,i+1}+\tilde{A}_{i+1,i}^*, \cdots, \tilde{A}_{ip}+\tilde{A}_{pi}^*, \\ &\quad \tilde{A}_{i1}-\tilde{A}_{1i}^*, \cdots, \tilde{A}_{i,i-1}-\tilde{A}_{i-1,i}^*, \tilde{A}_{ii}-\tilde{A}_{ii}^*, \tilde{A}_{i,i+1}-\tilde{A}_{i+1,i}^*, \cdots, \tilde{A}_{ip}-\tilde{A}_{pi}^*]. \end{aligned}$$

Here $(\tilde{A}_{ij})$ is the partition (2.8) of $\tilde{A}$. If K_i is nonsingular, then the rank argument (see [23]) implies

$$H_{ii} = G_i K_i^{-1} G_i^*. \tag{2.12}$$

For $i \neq r$ we can estimate the eigenvalues of K_i as follows.

Let $\mathcal{J} = \{1, 2, \cdots, n\}$, $\mathcal{J}_i = \{s_{i-1}+1, \cdots, s_i\}$ and let $\kappa_j^{(i)}$, $j \in \mathcal{J}\backslash\mathcal{J}_i$, be the eigenvalues of K_i ordered so that $|\kappa_j^{(i)} - (\tilde{a}_{jj}-\sigma_{s_i})| \leqq \|K_i - \text{diag}\,(K_i)\|_2 \leqq \|H-\text{diag}\,(h)\|_2$, $j \in \mathcal{J}\backslash\mathcal{J}_i$. Here we have used the perturbation theorem for the eigenvalues of hermitian matrices (see [19, Fact 1-11]), and the fact that $\tilde{a}_{jj}-\sigma_{s_i}$, $j \in \mathcal{J}\backslash\mathcal{J}_i$, are the diagonal elements of K_i. The same perturbation theorem, together with the relations (2.2) and (2.8), implies

$$|(\sigma_j-\sigma_{s_i})-(\tilde{a}_{jj}-\sigma_{s_i})| \leqq \|H-\text{diag}\,(H)\|_2, \qquad j \in \mathcal{J}\backslash\mathcal{J}_i.$$

By the relations (2.7) and (2.9) we have

$$\|H-\text{diag}\,(H)\|_2 = \left\| Q^*\begin{bmatrix} 0 & P^TEP \\ P^TE^*P & 0 \end{bmatrix} Q \right\|_2 = \|P^TEP\|_2 = \|E\|_2 < \nu.$$

The last three relations imply

$$|\kappa_j^{(i)}| > |\sigma_j - \sigma_{s_i}| - 2\nu \geqq 3\nu_i - 2\nu \geqq \nu_i, \qquad j \in \mathcal{J}\backslash\mathcal{J}_i.$$

Thus K_i is nonsingular and the relation (2.12) holds. To sharpen the estimates for $|\kappa_j^{(i)}|$, $j \in \mathcal{J}\backslash\mathcal{J}_i$, we apply the perturbation theorem to $P_i^T(H - \sigma_{s_i} I_{2n})P_i$ and to its block-diagonal part. It implies that

$$|\kappa_j^{(i)} - (\tilde{\sigma}_j - \sigma_{s_i})| \leqq \left\| \begin{bmatrix} 0 & G_i^* \\ G_i & 0 \end{bmatrix} \right\|_2 = \|G_i\|_2 \leqq \|E\|_2 < \nu, \qquad j \in \mathcal{J}\backslash\mathcal{J}_i$$

holds for an ordering $\tilde{\sigma}_1, \cdots, \tilde{\sigma}_n$ of σ_j, $j \in \mathcal{J}$. Thus for each $j \in \mathcal{J}\backslash\mathcal{J}_i$, $|\kappa_j^{(i)}|$ is either larger than $3\nu_i - \nu = 2\nu_i$ or smaller than ν. Since the second case cannot occur, we have $|\kappa_j^{(i)}| > 2\nu_i$, $j \in \mathcal{J}\backslash\mathcal{J}_i$, and consequently

$$\|K_i^{-1}\|_2 < 1/(2\nu_i), \quad 1 \leqq i \leqq p, \quad i \neq r.$$

By the relation (2.12) we obtain

$$\|H_{ii}\| \leqq \|K_i^{-1}\|_2 \|G_i\|^2 \leqq \|G_i\|^2/(2\nu_i), \qquad i \neq r. \tag{2.13}$$

To estimate $\|G_i\|^2$ we can use the parallelogram law. We obtain

$$4\|G_i\|^2 = \|\tilde{A}_{ii} - \tilde{A}_{ii}^*\|_{\text{off}}^2 + 2 \sum_{\substack{j=1 \\ j \neq i}}^{p} (\|\tilde{A}_{ij}\|^2 + \|\tilde{A}_{ji}\|^2). \tag{2.14}$$

The ET property (2.1) is used only to replace $\|\tilde{A}_{ii} - \tilde{A}_{ii}^*\|_{\text{off}}^2$ by $2\|\tilde{A}_{ii}\|_{\text{off}}^2$, $\|H_{ii}\|$ by $\|(|\tilde{A}_{ii}| + |\tilde{A}_{ii}^T|)/2 - \sigma_{s_i} I_{n_i}\|$ and to estimate $\|\tilde{A}_{ii}\|_{\text{off}}$ by $\|H_{ii}\|$. Hence the relation (2.10) follows directly from the inequality (2.13), and the equality (2.14).

If the condition (2.11) holds, then the distance of σ_{s_r} to the nearest eigenvalue of H is $3\nu_r$ and therefore the above proof holds for $i = r$. □

LEMMA 2.2. *Let A be an essentially triangular matrix satisfying the condition* (2.9) *and let $(\tilde{A}_{ij})$ be the partition* (2.8) *of $\tilde{A}$. Then the inequality*

$$\|\tilde{A}_{ii}\|_{\text{off}} \leqq \sqrt{2}\|(|\tilde{A}_{ii}| + |\tilde{A}_{ii}^T|)/2 - \sigma_{s_i} I_{n_i}\| \leqq \frac{\sqrt{2}}{4} \frac{\|A\|_{\text{off}}^2}{\nu} \tag{2.15}$$

holds for $i \neq r$. It holds for $i = r$ provided that

$$2\sigma_{\min} \geqq 3\nu \quad \text{or} \quad \sigma_{\min} = 0. \tag{2.16}$$

Proof. The first assertion follows directly from Lemma 2.1. The second assertion follows from the proof of Lemma 2.1 and the fact that the distance of σ_{r_4} to the rest of the spectrum of H is not smaller then 3ν. □

To obtain useful bounds for $\|\,|\tilde{A}_{rr}| - \sigma_{s_r} I_{n_r}\|$ in the case

$$2\sigma_{\min} < 3\nu_r, \tag{2.17}$$

we require that

$$\|A\|_{\text{off}} < \nu. \tag{2.18}$$

LEMMA 2.3. *Let A be* PST *and let the conditions* (2.17) *and* (2.18) *hold. Then*

$$\|\,|\tilde{A}_{rr}| - \sigma_{s_r} I_{n_r}\|^2 \leqq \sum_{\substack{j=1 \\ j \neq r}}^{p} (\|\tilde{A}_{rj}\|^2 + \|\tilde{A}_{jr}\|^2). \tag{2.19}$$

Moreover, if

$$\tilde{A}_{rj} = 0, \quad 1 \leqq j \leqq p, \quad j \neq r \quad \text{or} \quad \tilde{A}_{jr} = 0, \quad 1 \leqq j \leqq p, \quad j \neq r, \tag{2.20}$$

then

$$\| |\tilde{A}_{rr}| - \sigma_{s_r} I_{n_r} \| \leqq \min \left\{ \frac{\sigma_{s_r}}{2\sqrt{2}-1}, \frac{1}{5\nu_r} \sum_{\substack{j=1 \\ j \neq r}}^{p} (\|A_{rj}\|^2 + \|A_{jr}\|^2) \right\}. \tag{2.21}$$

Proof. First we prove the inequality (2.19).

Let P_0 be a permutation matrix, such that

$$A' = P_0^T \tilde{A} P_0 = \begin{bmatrix} A_1 & B_1 \\ B_2 & A_0 \end{bmatrix} \begin{matrix} \} n - n_r \\ \} n_r \end{matrix} \tag{2.22}$$

$$\underbrace{}_{n-n_r} \quad \underbrace{}_{n_r}$$

where

$$A_0 = \tilde{A}_{rr},$$
$$B_1 = [\tilde{A}_{1r}^T, \cdots, \tilde{A}_{r-1,r}^T, \tilde{A}_{r+1,r}^T, \cdots, \tilde{A}_{pr}^T]^T,$$
$$B_2 = [\tilde{A}_{r1}, \cdots, \tilde{A}_{r,r-1}, \tilde{A}_{r,r+1}, \cdots, \tilde{A}_{rp}].$$

To simplify notation, we set

$$\sigma_0 = \sigma_{s_r}, \qquad F = A_0 - \sigma_0 I_{n_r}.$$

Now the relation (2.19) takes the following form

$$\|F\|^2 \leqq \|B_1\|^2 + \|B_2\|^2. \tag{2.23}$$

In proving the inequality (2.23), we can assume that A_1 is diagonal and that the diagonal elements of A_1 and A_0 are nonnegative. Namely, if $A_1 = U_1 D_1 V_1^*$ is the SVD of A_1 and $\phi_0 = \operatorname{diag}(e^{i \arg(\tilde{a}_{jj})})$, $s_{r-1}+1 \leqq j \leqq s_r$, then we can work with D_1, $U_1^* B_1$, $B_2 V$ and $\phi_0^* A_0$ instead of A_1, B_1, B_2, and A_0, respectively. Here $\tilde{A} = (\tilde{a}_{ij})$.

Let us consider the hermitian matrix

$$A'A'^* - \sigma_0^2 I_n = \begin{bmatrix} A_1^2 + B_1 B_1^* - \sigma_0^2 I_{n-n_r} & A_1 B_2^* + B_1 A_0^* \\ B_2 A_1^* + A_0 B_1^* & A_0 A_0^* + B_2 B_2^* - \sigma_0^2 I_{n_r} \end{bmatrix} = \begin{bmatrix} M & X \\ X^* & N \end{bmatrix}.$$

Its eigenvalues are $\sigma_1^2 - \sigma_0^2, \cdots, \sigma_n^2 - \sigma_0^2$. Hence, if M is nonsingular, then the rank argument implies

$$N = X^* M^{-1} X. \tag{2.24}$$

To estimate the eigenvalues $\lambda_j(M)$, $1 \leqq j \leqq n - n_r$ of M, we denote $A_1 = \operatorname{diag}(a_{11}^1, \cdots, a_{n-n_r,n-n_r}^1)$ and we make use of the inequalities (2.13) and (2.14) for $i \neq r$. Since A_1 is diagonal[1] $\|G_i\|^2 \leqq \|A\|_{\text{off}}^2/2$; hence by the condition (2.18) we have $\|H_{ii}\| < \nu/4$, $i \neq r$. Therefore,

$$a_{jj}^1 > \sigma_j - \nu/4 \geqq \sigma_0 + 3\nu_r - \nu/4 \geqq \sigma_0 + \tfrac{11}{4}\nu_r, \qquad 1 \leqq j \leqq n - n_r. \tag{2.25}$$

By the Weil monotonicity theorem we obtain

$$\lambda_j(M) \geqq (a_{jj}^1)^2 - \sigma_0^2 > [2\sigma_0 + \tfrac{11}{4}\nu_r][\tfrac{11}{4}\nu_r]$$
$$\geqq \tfrac{121}{16}\nu_r^2 + \tfrac{11}{2}\sigma_0\nu_r, \qquad 1 \leqq j \leqq n - n_r.$$

Thus A_1 and M are positive definite and the equality (2.24) holds. It can be written as

$$\sigma_0(F + F^*) + FF^* + B_2 B_2^* = \tilde{L}\tilde{L}^*, \tag{2.26}$$

[1] Note that A' with diagonal block A_1 is not necessarily ET, but the inequality (2.13) still holds.

where

$$\tilde{L} = \tilde{X}^* \tilde{M}^{-1/2},$$
$$\tilde{X} = A_1^{-1} X = B_2^* + A_1^{-1} B_1 A_0^*,$$
$$\tilde{M} = A_1^{-1} M A_1^{-1} = I_{n-n_r} - \sigma_0^2 A_1^{-2} + (A_1^{-1} B_1)(A_1^{-1} B_1)^*.$$

Since σ_0 is the minimal singular value of A and the nonnegative diagonal elements of A_0 represent the eigenvalues of A, we conclude that the diagonal elements of F are nonnegative. Hence, applying the trace to the both sides of the equality (2.26) we obtain

$$\|F\|^2 \leqq \|\tilde{L}\|^2 - \|B_2\|^2. \tag{2.27}$$

The relation (2.25) implies

$$\|A_1^{-1}\|_2 \leqq 4/(4\sigma_0 + 11\nu_r). \tag{2.28}$$

By the perturbation theorem for singular values we have

$$\|A_0\|_2 \leqq \sigma_0 + \left\| \begin{bmatrix} 0 & B_1 \\ B_2 & 0 \end{bmatrix} \right\|_2 \leqq \sigma_0 + \|A\|_{\text{off}} < \sigma_0 + \nu. \tag{2.29}$$

The relations (2.17) and (2.28) imply

$$\|\tilde{M}^{-1/2}\|_2^2 \leqq \frac{1}{1-(4\sigma_0/(4\sigma_0+11\nu_r))^2} \leqq \frac{1}{1-(1/(1+\frac{11}{4}\cdot\frac{2}{3})^2)} = \frac{284}{253}. \tag{2.30}$$

To estimate $\|A_1^{-1}\|_2 \|A_0\|_2$ we use the relations (2.29), (2.28), and (2.17). We obtain

$$\|A_1^{-1}\|_2 \|A_0\|_2 < \frac{4\sigma_0 + 4\nu}{4\sigma_0 + 11\nu_r} \leqq \frac{6\nu_r + 4\nu}{6\nu_r + 11\nu_r} \leqq \frac{10}{17}. \tag{2.31}$$

Here we have used the fact that $\sigma_0 \to (4\sigma_0+4\nu)/(4\sigma_0+11\nu_r)$ is an increasing function. Now the relations (2.27), (2.31), and (2.30) imply

$$\|F\|^2 \leqq \tfrac{289}{253}(\|B_2\| + \tfrac{10}{17}\|B_1\|)^2 - \|B_2\|^2.$$

Using the inequality $\|B_1\|\,\|B_2\| \leqq (\alpha\|B_2\|^2 + \|B_1\|^2/\alpha)/2$, $\alpha > 0$ we obtain

$$\|F\|^2 \leqq [(36+170\alpha)\|B_2\|^2 + (100+170/\alpha)\|B_1\|^2]/253.$$

The choice $\alpha = (16+\sqrt{16^2+85^2})/85$ implies

$$\|F\|^2 \leqq 0.95252(\|B_2\|^2 + \|B_1\|^2),$$

which proves the relation (2.23).

Next we prove the relation (2.21). The assumption (2.20) implies that either $B_1 = 0$ or $B_2 = 0$. We can assume $B_2 = 0$ since otherwise we can work with A'^T instead of A'. Thus A' from the relation (2.22) is block upper-triangular and we set $B = B_1$ so that $X = BA_0^*$.

The perturbation theorem for the eigenvalues of hermitian matrices implies either $\lambda_{\min}(M) \leqq \|X\|_2$ or

$$\begin{aligned} \lambda_{\min}(M) = \|M^{-1}\|_2^{-1} &\geqq \min_{\lambda_j \neq \sigma_0^2} \lambda_j(A'A'^* - \sigma_0^2 I_n) - \left\| \begin{bmatrix} 0 & X \\ X^* & 0 \end{bmatrix} \right\|_2 \\ &= \min_{\substack{i \\ i \neq r}} (\sigma_{s_i}^2 - \sigma_0^2) - \|X\|_2 \geqq 3\nu_r(2\sigma_0 + 3\nu_r) - \|X\|_2. \end{aligned} \tag{2.32}$$

From the relations (2.18) and (2.29) we obtain

$$\|X\|_2 \leqq \|B\|_2 \|A_0\|_2 < \nu_r(\sigma_0 + \nu).$$

Since we have already proved $\lambda_{\min}(M) \geqq \frac{121}{16}\nu_r^2 + \frac{11}{2}\sigma_0\nu_r$, the estimate $\lambda_{\min}(M) \leqq \|X\|$ is false. Therefore the relation (2.32) yields

$$\|M^{-1}\|_2 \leqq 1/(8\nu_r^2 + 5\nu_r\sigma_0).$$

We use this inequality together with the relations (2.27) and (2.18). Since $B_2 = 0$ and $\|\tilde{L}\|^2 = \operatorname{tr} \tilde{L}\tilde{L}^* = \operatorname{tr}(X^*M^{-1/2})(X^*M^{-1/2})^* = \|X^*M^{-1/2}\|^2$ we obtain

$$\|F\| \leqq \|X^*M^{-1/2}\| \leqq \frac{(\sigma_0 + \|F\|_2)\|B\|}{2\sqrt{2}\nu_r} \leqq \frac{\sigma_0 + \|F\|}{2\sqrt{2}},$$

whence

$$\|F\| \leqq \sigma_0/(2\sqrt{2} - 1). \tag{2.33}$$

With this estimate we can further sharpen the estimate for $\|M^{-1}\|_2$. We have

$$\begin{aligned} \|M^{-1}\|_2 &\leqq 1/[9\nu_r^2 + 6\nu_r\sigma_0 - (\sigma_0 + \|F\|_2)\|B\|_2] \\ &\leqq 1/[9\nu_r^2 + 6\nu_r\sigma_0 - 2\sqrt{2}\,\sigma_0\nu_r/(2\sqrt{2} - 1)] \\ &\leqq 1/[9\nu_r^2 + 4\sigma_0\nu_r]. \end{aligned} \tag{2.34}$$

On the other hand, the relation (2.26) and the ET property of F imply

$$\begin{aligned} 2\sigma_0^2\|F\|^2 \leqq \sigma_0^2\|F + F^*\|^2 &\leqq \|X^*M^{-1}X - FF^*\|^2 \\ &\leqq \|X^*M^{-1}X\|^2 + \|FF^*\|^2 \\ &\leqq \|X^*M^{-1}X\|^2 + \|F\|_2^2\|F\|^2. \end{aligned}$$

Hence if $\sigma_0 > 0$, we obtain

$$\|F\|^2 \leqq \|X^*M^{-1}X\|^2/(2\sigma_0^2 - \|F\|_2^2). \tag{2.35}$$

In the above estimates we have used the fact that the diagonal elements of F are real and that $X^*M^{-1}X$ and FF^* are positive semidefinite. From the relations (2.35), (2.33), (2.34), and (2.17) we finally obtain

$$\begin{aligned} \|F\|^2 &\leqq \frac{[\sigma_0 + \|F\|_2]^4\|B\|^4\|M^{-1}\|_2^2}{2\sigma_0^2 - \|F\|^2} \\ &\leqq \frac{[1 + (1/(2\sqrt{2} - 1))]^4\sigma_0^4\|B\|^4}{[2 - (1/(2\sqrt{2} - 1)^2)]\sigma_0^2[9\nu_r^2 + 4\nu_r\sigma_0]} \\ &\leqq \frac{3.367}{[9(\nu_r/\sigma_0) + 4]^2}\,\frac{\|B\|^4}{\nu_r^2} \leqq 0.03367\,\frac{\|B\|^4}{\nu^2}. \end{aligned} \tag{2.36}$$

The assertion (2.21) follows directly from the relations (2.33), (2.36), and the definition of F and B. □

If the condition (2.9) holds, we can define

$$\tau(A) = \left[\sum_{i=1}^{p}\sum_{\substack{j=1 \\ j\neq i}}^{p} \|\tilde{A}_{ij}\|^2\right]^{1/2},$$

where $(\tilde{A}_{ij})$ is the partition (2.8) of P^*AP and P is so chosen that the inequalities (2.10) hold.

If the diagonal elements of A, affiliated with the same singular value, occupy successive positions on the diagonal, we say that the *affiliation is proper.* In such a case we assume $P = I$, i.e., $\tilde{A} = A$.

We end this section by compiling the previous results in the following form.

THEOREM 2.4. *Let A be a* PST *matrix satisfying the condition* (2.18) *and the condition of proper affiliation. Then*

$$2\sum_{j=1}^{n} ||a_{jj}|-\sigma_j|^2+\sum_{i=1}^{p} \|A_{ii}\|_{\text{off}}^2 \leqq \frac{1}{4}\frac{\|A\|^4}{\nu^2}+\tau^2(A) \tag{2.37}$$

where $\sigma_1, \cdots, \sigma_n$ is an appropriate ordering of the singular values and (A_{ij}) is the partition (2.8) *of A.*

The term $\tau^2(A)$ can be omitted provided that the condition (2.16) *or the condition* (2.20) *holds.*

Proof. The inequality (2.37) follows directly from Lemma 2.2, Lemma 2.1, and Lemma 2.3. If X denotes the left-hand side of the inequality (2.37), then any of the conditions (2.16), (2.20) implies

$$X \leqq \frac{\sqrt{2}}{4}\frac{\|A\|_{\text{off}}^2}{\nu}\cdot\frac{\sqrt{2}}{4\nu}\left[\sum_{i=1}^{p}\|A_{ii}\|_{\text{off}}^2+\sum_{i=1}^{p}\sum_{\substack{j=1\\ j\neq i}}^{p}(\|A_{ij}\|^2+\|A_{ji}\|^2)\right]$$

$$\leqq \frac{1}{8}\frac{\|A\|_{\text{off}}^2}{\nu^2}\cdot 2\|A\|_{\text{off}}^2 = \frac{1}{4}\frac{\|A\|^4}{\nu^2}.$$

The rest of the proof follows by the first assertion of Lemma 2.3. □

In particular, if A is a triangular matrix satisfying the condition (2.18) and the condition of proper affiliation, then the inequality

$$2\sum_{j=1}^{n} ||a_{jj}|-\sigma_j|^2+\sum_{i=1}^{p} \|A_{ii}\|_{\text{off}}^2 \leqq \|A\|_{\text{off}}^4/(4\nu^2) \tag{2.38}$$

holds, provided that $r \in \{1, p\}$.

3. SVDT methods on almost diagonal matrices. Here we prove some auxiliary results for the SVDT methods when they are applied to almost diagonal triangular matrices. First we define the basic SVDT algorithms that are used for computing the elements of $\hat{U}_k$ and $\hat{V}_k$ $(k \geqq 1)$.

SVDT algorithms for 2×2 triangular matrices. Dropping the superscript k we can write

$$\hat{U}^*\hat{A}\hat{V} = \operatorname{diag}(a'_{ll}, a'_{mm}), \qquad l<m \tag{3.1}$$

where $\hat{A}$ is the triangular pivot submatrix. Assuming

$$\hat{U} = \begin{bmatrix} \cos\varphi & e^{i\alpha}\sin\varphi \\ -e^{-i\alpha}\sin\varphi & \cos\varphi \end{bmatrix}, \qquad \hat{V} = \begin{bmatrix} \cos\psi & e^{i\beta}\sin\psi \\ -e^{-i\beta}\sin\psi & \cos\psi \end{bmatrix},$$

the requirement (3.1) yields several sets of formulae for α, φ, β, and ψ (see [21], [2], [10]). For the case of real matrices, we take $\alpha = \beta = 0$ and the appropriate formulae can be found in, e.g., [21], [1], and [12]. Since the columns of $\hat{U}(\hat{V})$ represent the normalized eigenvectors of the hermitian matrix $\hat{A}\hat{A}^*$ $(\hat{A}^*\hat{A})$, different algorithms yield, in most cases, essentially the same matrices $\hat{U}$ and $\hat{V}$. Here, we use the formulae derived in [21].

Let $\hat{A}=(a_{ij})$, $i, j \in \{l, m\}$ be upper-triangular. If the right-hand transformation $\hat{V}$ is computed first, then we obtain

$$
\begin{aligned}
&\beta = \arg(a_{lm}\bar{a}_{ll}),\\
&\tan 2\psi = \frac{2|a_{lm}\bar{a}_{ll}|}{|a_{mm}|^2-|a_{ll}|^2+|a_{lm}|^2}, \qquad \psi \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right],\\
&\alpha = \arg(z),\\
&\tan\varphi = |z|, \qquad \varphi \in \left[0, \frac{\pi}{2}\right)
\end{aligned} \tag{3.2}
$$

where

$$
z = \frac{a_{lm}+a_{ll}\,e^{i\beta}\tan\psi}{a_{mm}} \qquad \left(= \frac{\bar{a}_{mm}\,e^{i\beta}\tan\psi}{\bar{a}_{ll}-\bar{a}_{lm}\,e^{i\beta}\tan\psi}\right)
$$

and

$$
a'_{ll} = \frac{\cos\psi}{\cos\varphi}\,a_{ll} - \frac{\sin\psi}{\cos\varphi}\,e^{-i\beta}a_{lm}, \qquad a'_{mm} = \frac{\cos\psi}{\cos\varphi}\,a_{mm}. \tag{3.3}
$$

The formula in parentheses for z can be alternatively taken for computational reasons. In particular, if $a_{mm}=0$ we assume $z=0$. This implies $\cos\varphi > 0$ and the validity of the formulae in the relation (3.3).

If $\hat{A}$ is real, the choice $\alpha=\beta=0$ implies $\tan 2\psi = 2a_{lm}a_{ll}/(a_{mm}^2-a_{ll}^2+a_{lm}^2)$ and $\tan\varphi = z$, so that $\varphi \in (-\pi/2, \pi/2)$.

The formulae (3.2) and (3.3) define an algorithm which we call UR (upper-triangular, right-first) algorithm.

In a similar way we can define UL algorithm where φ is computed first. The appropriate formulae are

$$
\begin{aligned}
&\alpha = \arg(a_{lm}\bar{a}_{mm}),\\
&\tan 2\varphi = \frac{2|a_{lm}\bar{a}_{mm}|}{|a_{mm}|^2-|a_{ll}|^2-|a_{lm}|^2}, \qquad \varphi \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right],\\
&\beta = \arg(z),\\
&\tan\psi = |z|, \qquad \psi \in \left[0, \frac{\pi}{2}\right),
\end{aligned} \tag{3.4}
$$

where

$$
z = \frac{a_{mm}\,e^{i\alpha}\tan\varphi - a_{lm}}{a_{ll}} \qquad \left(= \frac{\bar{a}_{ll}\,e^{i\alpha}\tan\varphi}{\bar{a}_{mm}+\bar{a}_{lm}\,e^{i\beta}\tan\varphi}\right)
$$

and

$$
a'_{ll} = \frac{\cos\varphi}{\cos\psi}\,a_{ll}, \qquad a'_{mm} = \frac{\cos\varphi}{\cos\psi}\,a_{mm} + \frac{\sin\varphi}{\cos\psi}\,e^{-i\alpha}a_{lm}. \tag{3.5}
$$

We shall refer to the angle φ or ψ which is computed first (and therefore is constrained to $[-\pi/4, \pi/4]$) as the "smaller angle." The other one is then called the "larger angle."

If $\hat{A}$ is lower-triangular, we can define LR and LL algorithms in a similar way wherein ψ and φ are the "smaller angles," respectively. An interesting feature is that UR and LL (UL and LR) algorithms coincide if they are expressed in terms of the elements of the upper-triangular matrix $\hat{T}$ defined by the relation (1.2), i.e., $\hat{T}+\hat{T}^T = \hat{A}+\hat{A}^T$.

Class J_n of SVDT methods. The four basic algorithms, UR, UL, LR, and LL, give rise to four SVDT methods: R, L, RL, and LR, which constitute the class J_n. The method R(L) uses only UR and LR (UL and LL) algorithms depending on the type of triangularity of the pivot submatrices. The method RL(LR) uses only UR and LL (UL and LR) algorithms. For the definiteness of these four methods we assume the column-cyclic strategy. Here we again recall the result of Hansen which implies that for our considerations it is irrelevant which one of the two serial strategies is assumed.

The importance of the RL and LR methods lies in the fact that their transformation formulae (1.3) together with the appropriate algorithms do not change with the parity of cycles, provided that they are expressed in terms of the elements of $T^{(k)}$ from the relation (1.2). This fact simplifies their computational code.

The global convergence of the methods from J_n has been proved in [12] and [8]. Here we prove their ultimate quadratic convergence.

Angle estimates. We start considering the asymptotic behavior of the methods from J_n. Let A be a triangular matrix of order n and let $A^{(1)}=A$, $A^{(2)}, \cdots$, be the sequence generated by a method from J_n. If $(A^{(k)}, k\geqq 1)$ is convergent $A^{(k)}$ will reach the stage indicated by the condition (2.18). For the sake of clarity and generality, we assume that A itself satisfies the condition (2.18).

LEMMA 3.1. *If $a_{ll}^{(k)}$ and $a_{mm}^{(k)}$ are affiliated with distinct singular values, then*

$$\max\{|\tan\varphi_k|, |\tan\psi_k|\} \leqq \frac{4}{7}\cdot\frac{1}{\nu}\cdot\max\{|a_{lm}^{(k)}|, |a_{ml}^{(k)}|\}. \tag{3.6}$$

Proof. Since $(\|A^{(k)}\|_{\text{off}}, k\geqq 1)$ is nonincreasing (see the relation (4.4)) we can use Lemma 2.1 and Lemma 2.2 with $A^{(k)}$ for any $k\geqq 1$. Suppose that $a_{ll}^{(k)}$ and $a_{mm}^{(k)}$ are affiliated with σ_{s_i} and σ_{s_j}, respectively.

First we consider the case $r\notin\{i,j\}$, i.e., neither σ_{s_i} nor σ_{s_j} is the minimal singular value.

From Lemma 2.2 we obtain

$$\big||a_{ll}^{(k)}|-\sigma_{s_i}\big| \leqq \|A^{(k)}\|_{\text{off}}^2/(4\nu) \leqq \|A\|_{\text{off}}^2/(4\nu) < \nu/4$$

and, in a similar way,

$$\big||a_{mm}^{(k)}|-\sigma_{s_j}\big| \leqq \nu/4.$$

Using the triangle inequality we obtain

$$\begin{aligned}
\min\{|a_{ll}^{(k)}|, |a_{mm}^{(k)}|\} &\geqq \min\{\sigma_{s_i}, \sigma_{s_j}\}-\nu/4\\
&\geqq 3\nu-\nu/4=\tfrac{11}{4}\nu,\\
\max\{|a_{ll}^{(k)}|, |a_{mm}^{(k)}|\} &\geqq \max\{\sigma_{s_i}, \sigma_{s_j}\}-\nu/4\\
&\geqq 6\nu-\nu/4=\tfrac{23}{4}\nu,\\
\big||a_{mm}^{(k)}|-|a_{ll}^{(k)}|\big| &\geqq |\sigma_{s_j}-\sigma_{s_i}|-\big||a_{mm}^{(k)}|-\sigma_{s_j}\big|-\big||a_{ll}^{(k)}|-\sigma_{s_i}\big|\\
&\geqq 3\nu-\nu/4-\nu/4=\tfrac{5}{2}\nu.
\end{aligned}$$

Since the formulae for LR and LL algorithms are essentially (i.e., when expressed in terms of the elements of $T^{(k)}$ from the relation (1.2)) the same as those for UL and UR algorithms, respectively, we consider only the UL and UR algorithms. For the UL algorithm we have

$$|\tan \varphi_k| \leqq \frac{1}{2}|\tan 2\varphi_k| \leqq \frac{|a_{lm}^{(k)}||a_{mm}^{(k)}|}{||a_{mm}^{(k)}|^2 - |a_{ll}^{(k)}|^2| - |a_{lm}^{(k)}|^2}.$$

If $\sigma_{s_i} > \sigma_{s_j}$ we have

$$|\tan \varphi_k| \leqq \frac{|a_{lm}^{(k)}|}{\frac{5}{2}\nu((|a_{ll}^{(k)}| + |a_{mm}^{(k)}|)/|a_{mm}^{(k)}|) - (|a_{lm}^{(k)}|^2/|a_{mm}^{(k)}|)} \leqq \frac{|a_{lm}^{(k)}|}{\frac{5}{2}\nu \cdot 2 - \frac{4}{11}\nu} = \frac{11}{51}\frac{|a_{lm}^{(k)}|}{\nu},$$

$$|\tan \psi_k| \leqq \frac{|a_{lm}^{(k)}| + |a_{mm}^{(k)}||\tan \varphi_k|}{|a_{ll}^{(k)}|} \leqq \left(\frac{4}{23} + \frac{11}{51}\right)\frac{|a_{lm}^{(k)}|}{\nu} \leqq \frac{2}{5}\frac{|a_{lm}^{(k)}|}{\nu}.$$

Otherwise we have

$$|\tan \varphi_k| \leqq \frac{|a_{lm}^{(k)}|}{\frac{5}{2}\nu - \frac{4}{23}\nu} \leqq \frac{1}{2}\frac{|a_{lm}^{(k)}|}{\nu},$$

$$|\tan \psi_k| \leqq \frac{|a_{ll}^{(k)}||\tan \varphi_k|}{|a_{mm}^{(k)}| - |a_{lm}^{(k)}||\tan \varphi_k|} \leqq \frac{|a_{ll}^{(k)}||\tan \varphi_k|}{|a_{ll}^{(k)}| + \frac{5}{2}\nu - \frac{1}{2}\nu}$$

$$\leqq |\tan \varphi_k| \leqq \frac{1}{2}\frac{|a_{lm}^{(k)}|}{\nu}.$$

For the UR algorithm the estimates for ψ_k and φ_k are obtained in the same way as above, only the formulae (3.4) are used instead of the formulae (3.2).

Next we consider the case $i = r$. We distinguish two possibilities: $2\sigma_{s_i} \geqq 3\nu$ and $2\sigma_{s_i} < 3\nu$.

Suppose $2\sigma_{s_i} \geqq 3\nu$. Then Lemma 2.2 implies

$$|a_{mm}^{(k)}| \geqq \sigma_{s_j} - \nu/4 \geqq \sigma_{s_i} + 3\nu - \nu/4 \geqq \tfrac{3}{2}\nu + \tfrac{11}{4}\nu = \tfrac{17}{4}\nu.$$

Hence the estimate $\max\{|\tan \varphi_k|, |\tan \psi_k|\} \leqq |a_{lm}^{(k)}|/(2\nu)$ is easily obtained in the same manner as above.

Now, suppose that $2\sigma_{s_i} < 3\nu$. Then

$$|a_{mm}^{(k)}| > \sigma_{s_j} - \nu/4 \geqq \sigma_{s_i} + \tfrac{11}{4}\nu$$

and by the perturbation theorem for singular values,

$$0 \leqq |a_{ll}^{(k)}| - \sigma_{s_i} \leqq \|A^{(k)}\|_{\text{off}} \leqq \|A\|_{\text{off}}.$$

We immediately obtain

$$|a_{mm}^{(k)}| - |a_{ll}^{(k)}| \geqq 3\nu - \nu - \nu/4 = \tfrac{7}{4}\nu$$

and

$$|a_{ll}^{(k)}| \leqq \sigma_{s_i} + \nu < \tfrac{5}{2}\nu.$$

Let us first consider the estimates corresponding to the UL algorithm.
If $|a_{ll}^{(k)}| < |a_{lm}^{(k)}|$, then $|a_{ll}^{(k)}|^2 + |a_{lm}^{(k)}|^2 < 2|a_{lm}^{(k)}|^2 < 2\nu^2$; hence

$$|\tan \varphi_k| \leqq \frac{|a_{lm}^{(k)}|}{|a_{mm}^{(k)}| - 2(|a_{lm}^{(k)}|^2/|a_{mm}^{(k)}|)} \leqq \frac{|a_{lm}^{(k)}|}{\frac{11}{4}\nu - 2\frac{4}{11}\nu} \leqq \frac{44}{89}\frac{|a_{lm}^{(k)}|}{\nu},$$

$$|\tan \psi_k| \leqq \frac{|a_{ll}^{(k)}||\tan \varphi_k|}{|a_{mm}^{(k)}| - |a_{lm}^{(k)}||\tan \varphi_k|} \leqq \frac{|a_{ll}^{(k)}||\tan \varphi_k|}{|a_{ll}^{(k)}| + \frac{7}{4}\nu - \frac{44}{89}\nu}.$$
$$\leqq |\tan \varphi_k| \leqq \frac{44}{89}\frac{|a_{lm}^{(k)}|}{\nu}.$$

Otherwise we have

$$|a_{ll}^{(k)}|^2 + |a_{lm}^{(k)}|^2 \leqq |a_{ll}^{(k)}|^2 + |a_{ll}^{(k)}|(|a_{mm}^{(k)}| - |a_{ll}^{(k)}|) = |a_{ll}^{(k)}||a_{mm}^{(k)}|;$$

hence

$$|\tan \varphi_k| \leqq \frac{|a_{lm}^{(k)}||a_{mm}^{(k)}|}{|a_{mm}^{(k)}|^2 - |a_{ll}^{(k)}||a_{mm}^{(k)}|} \leqq \frac{4}{7}\frac{|a_{lm}^{(k)}|}{\nu},$$
$$|\tan \psi_k| \leqq |\tan \varphi_k| \leqq \frac{4}{7}\frac{|a_{lm}^{(k)}|}{\nu}.$$

For the UR algorithm the estimates are more straightforward. We have

$$|\tan \psi_k| \leqq \frac{|a_{lm}^{(k)}||a_{ll}^{(k)}|}{(|a_{mm}^{(k)}| - |a_{ll}^{(k)}|)(|a_{mm}^{(k)}| + |a_{ll}^{(k)}|) + |a_{lm}^{(k)}|^2}$$
$$\leqq \frac{|a_{lm}^{(k)}|}{\frac{7}{4}\nu \cdot 2} = \frac{2}{7}\frac{|a_{lm}^{(k)}|}{\nu},$$
$$|\tan \varphi_k| \leqq \frac{|a_{lm}^{(k)}|}{|a_{mm}^{(k)}|} + \frac{|a_{ll}^{(k)}|}{|a_{mm}^{(k)}|}|\tan \psi_k| \leqq \left(\frac{\frac{7}{4}\nu + \frac{1}{2}|a_{ll}^{(k)}|}{|a_{mm}^{(k)}|}\right)\frac{4}{7}\frac{|a_{lm}^{(k)}|}{\nu}$$
$$\leqq \frac{4}{7}\frac{|a_{lm}^{(k)}|}{\nu}.$$

It remains to consider the case $j = r$. However, the bounds for this case can be obtained in the same way as for the case $i = r$. Only, the roles of $a_{ll}^{(k)}$ and $a_{mm}^{(k)}$ are interchanged. □

Lemma 3.1 is first used to prove that the affiliation of the diagonal elements is invariant under the methods from J_n.

LEMMA 3.2. *Let the condition* (2.18) *hold. Then there exists a permutation matrix* P *such that the relations* (2.8) *and* (2.10) *hold for each matrix* $P^T A^{(k)} P$, $k \geqq 1$.

Proof. Since $(\|A^{(k)}\|_{\text{off}}, k \geqq 1)$ is the nonincreasing sequence, Lemma 2.1 implies the existence of permutation matrices P_k, $k \geqq 1$, such that the relations (2.8) and (2.10) hold for each $P_k^T A^{(k)} P_k$, $k \geqq 1$. To prove that $P_k = P$ $(= P_1)$, $k \geqq 1$, we use induction with respect to k.

To step from k to $k+1$ we must prove that $a_{ll}^{(k)}$ and $a_{mm}^{(k)}$ cannot change their affiliation at the kth step.

Suppose that $a_{ll}^{(k)}$ and $a_{mm}^{(k)}$ are affiliated with σ_{s_i} and σ_{s_j}, respectively. If $i = j$, then any change in the affiliation of $a_{ll}^{(k)}$ and/or $a_{mm}^{(k)}$ would imply a change of the order of some diagonal blocks of $P_{k+1}^T A^{(k+1)} P_{k+1}$, and that is impossible.

If $i \neq j$, the same argument implies that $a_{ll}^{(k)}$ and $a_{mm}^{(k)}$ can at most interchange their affiliation. We shall prove that it cannot occur.

If $\sigma_{s_i} > \sigma_{s_j}$, then (see the proof of Lemma 3.1)

$$|a_{ll}^{(k)}| - |a_{mm}^{(k)}| > \tfrac{7}{4}\nu.$$

Hence, the interchange of the affiliation of $a_{ll}^{(k)}$ and $a_{mm}^{(k)}$ would imply

$$|a_{ll}^{(k+1)}| - |a_{mm}^{(k+1)}| < -\tfrac{7}{4}\nu, \tag{3.7}$$

so it suffices to prove that (3.7) fails.

Using the formulae (3.3) and (3.5), we obtain

$$|a_{ll}^{(k+1)}|-|a_{mm}^{(k+1)}| \geqq \frac{\cos x_k}{\cos y_k}(|a_{ll}^{(k)}|-|a_{mm}^{(k)}|) - \frac{|\sin x_k|}{\cos y_k}\max\{|a_{lm}^{(k)}|, |a_{ml}^{(k)}|\} \tag{3.8}$$

where x_k (y_k) is the "smaller angle" ("larger angle"). The relation (3.8) holds for all four algorithms.

Using the relations (3.8), (2.18), and (3.6), we obtain

$$\begin{aligned}|a_{ll}^{(k+1)}|-|a_{mm}^{(k+1)}| &\geqq \frac{\cos x_k}{\cos y_k}\cdot\frac{7}{4}\nu - \frac{|\sin x_k|}{\cos y_k}\nu \\ &= \frac{\cos x_k}{\cos y_k}\left(\frac{7}{4}-|\tan x_k|\right)\nu \\ &\geqq \sqrt{2}\left(\frac{7}{4}-\frac{4}{7}\right)\nu > 0.\end{aligned}$$

Thus the inequality (3.7) fails, so the proof is completed for the case $\sigma_{s_i} > \sigma_{s_j}$. Since the case $\sigma_{s_j} > \sigma_{s_i}$ is treated in the same way, Lemma 3.2 is proved. □

In the proof of Lemma 3.2 we have used only the formulae for $a_{ll}^{(k+1)}$, $a_{mm}^{(k+1)}$, and the angle estimates. Therefore, it holds for any pivot strategy that preserves the property of essential triangularity.

We end this section by proving that φ_k and ψ_k need not necessarily be tiny when $|a_{ll}^{(k)}|$ and $|a_{mm}^{(k)}|$ approximate the same singular value. For this end we provide the following example.

Example 3.3. The 3×3 matrix

$$A = A(t) = \begin{bmatrix} v_1(t) & 3t^2 v(t)/v_1(t) & -3tv^3(t)/v_1(t) \\ 0 & v_2(t)/v_1(t) & -3tv^3(t)/(v_1(t)v_2(t)) \\ 0 & 0 & 2/v_2(t) \end{bmatrix}$$

where

$$v(t)=\sqrt{1-t^2}, \quad v_1(t)=\sqrt{1+3t^2}, \quad v_2(t)=\sqrt{1+6t^2-3t^4}, \quad |t|<1$$

can be factored as follows

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_2 & s_2 \\ 0 & -s_2 & c_2 \end{bmatrix}\begin{bmatrix} c_1 & 0 & s_1 \\ 0 & 1 & 0 \\ -s_1 & 0 & c_1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ & & 2 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix}\begin{bmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix}.$$

Here $s=t$, $c=v(t)$, $s_1=-2t/v_1(t)$, $c_1=v(t)/v_1(t)$, $s_2=-2t/v_2(t)$, and $c_2=v(t)v_1(t)/v_2(t)$. Hence its singular values are 1, 1, and 2. The third-order approximation of $A(t)$ is

$$A_3(t) = \begin{bmatrix} 1+\frac{3}{2}t^2-\frac{9}{8}t^4 & 3t^2-6t^4 & -3t+\frac{15}{2}t^3 \\ 0 & 1+\frac{3}{2}t^2-\frac{57}{8}t^4 & 3t-\frac{27}{2}t^3 \\ 0 & 0 & 2-3t^2+\frac{75}{4}t^4 \end{bmatrix}.$$

For the UL and UR algorithms we obtain

$$\lim_{t\to 0}|\tan\varphi_1| = \lim_{t\to 0}|\tan\psi_1| = 1. \qquad \square$$

4. Quadratic convergence. Here we prove the ultimate quadratic convergence of the SVDT methods from J_n. The basic asymptotic assumptions are the following:

(A1) A is a triangular matrix of order n satisfying the condition (2.18).

(A2) The affiliation of the diagonal elements $a_{11}, \cdots, a_{nn}$ is proper.

The sequence $A^{(1)} = A, A^{(2)}, \cdots$ is assumed to be generated by any of the methods from J_n. Under these conditions, the results from §§ 2 and 3 hold for each matrix $A^{(k)}$ ($k \geqq 1$). Since the permutation matrix P from Lemma 3.2 is identity, these results refer to the partition (2.8) of $A^{(k)}$ ($= \tilde{A}^{(k)}$). It is assumed that A_{rr} is associated with the minimal singular value $\sigma_{s_r} = \sigma_{\min}$.

THEOREM 4.1. *Let the assumptions* (A1) *and* (A2) *hold. If the singular values of* A *are at most double, then*

$$\|A^{(N+1)}\|_{\text{off}} \leqq \tfrac{4}{7}\|A\|^2_{\text{off}}/\nu. \tag{4.1}$$

Proof. In [8] it has been proved that

$$\|A^{(N+1)}\|_{\text{off}} \leqq C\|A\|_{\text{off}} \tag{4.2}$$

holds where $C \geqq 0$,

$$C^2 = \begin{cases} 1 - \prod\limits_{\substack{k=1 \\ l<m-1}}^{N} \cos^2 \psi_k & \text{if } A \text{ is upper-triangular,} \\ 1 - \prod\limits_{\substack{k=1 \\ l<m-1}}^{N} \cos^2 \varphi_k & \text{if } A \text{ is lower-triangular.} \end{cases}$$

Here the products are taken over all k, $1 \leqq k \leqq N$ such that $l(k) < m(k) - 1$. By a variant of the Bernoulli's inequality we obtain

$$C^2 \leqq \begin{cases} \sum\limits_{\substack{k=1 \\ l<m-1}}^{N} \sin^2 \psi_k & \text{if } A \text{ is upper-triangular,} \\ \sum\limits_{\substack{k=1 \\ l<m-1}}^{N} \sin^2 \varphi_k & \text{if } A \text{ is lower-triangular.} \end{cases}$$

The assumption (A2), together with Lemma 3.2 and Lemma 3.1, implies

$$C^2 \leqq \frac{16}{49}\frac{1}{\nu^2} \sum_{\substack{k=1 \\ l<m-1}}^{N} \max\{|a_{lm}^{(k)}|^2, |a_{ml}^{(k)}|^2\}. \tag{4.3}$$

Iterating the equality

$$\|A^{(k+1)}\|^2_{\text{off}} = \|A^{(k)}\|^2_{\text{off}} - \max\{|a_{lm}^{(k)}|^2, |a_{ml}^{(k)}|^2\}, \qquad k \geqq 1, \tag{4.4}$$

we easily obtain

$$\sum_{k=1}^{N} \max\{|a_{lm}^{(k)}|^2, |a_{ml}^{(k)}|^2\} \leqq \|A\|^2_{\text{off}}. \tag{4.5}$$

The assertion (4.1) of Theorem 4.1 follows directly from the relations (4.2), (4.3), and (4.5). □

From Theorem 4.1 we see that the quadratic convergence bound for the SVDT methods is as sharp as the corresponding one for the symmetric/hermitian Jacobi method (cf. [22], [20], [10]).

Before considering the general case we show that the assumption (A2) of proper affiliation cannot be removed.

From Example 3.3 we see that large angles φ_k and ψ_k can arise whenever $|a_{ll}^{(k)}|$ and $|a_{mm}^{(k)}|$ approximate the same singular value. In the following example we investigate what happens with the off-norm reduction when the affiliation of the diagonal elements is not proper.

Example 4.2. The following 6×6 matrix has been generated using the floating point arithmetic with 16 decimal digits.

```
Matrix A.
 row 1:
-1.00000486975715E+000  -1.66559232016605E-003  -3.86229792838258E-006
 3.21198325729463E-003  -6.11835546775858E-006  -4.01586205765311E-003
 row 2:
 0.00000000000000E+000  -1.99999749882806E+000   1.74457418665018E-003
-5.49098863565807E-006  -1.29836599555287E-003   1.91263769977208E-006
 row 3:
 0.00000000000000E+000   0.00000000000000E+000  -1.00000344053744E+000
 1.76138149895389E-004   8.32301133463415E-007  -4.19146831904327E-003
 row 4:
 0.00000000000000E+000   0.00000000000000E+000   0.00000000000000E+000
-1.99999472299936E+000   2.34158685306338E-003  -8.89474219276185E-006
 row 5:
 0.00000000000000E+000   0.00000000000000E+000   0.00000000000000E+000
 0.00000000000000E+000  -1.00000120539952E+000   2.52103700941134E-004
 row 6:
 0.00000000000000E+000   0.00000000000000E+000   0.00000000000000E+000
 0.00000000000000E+000   0.00000000000000E+000   1.99998874686153E+000
```

The diagonal elements $a_{11}, \cdots, a_{66}$ are affiliated with the singular values 1, 2, 1, 2, 1, 2, respectively. Applying the methods from J_n to A we obtain the following off-norm reduction per cycle.

```
Off-norm/cycle.
Cycle       L              LR             RL             R
  0    7.55603E-003   7.55603E-003   7.55603E-003   7.55603E-003
  1    1.57043E-003   1.57043E-003   1.57043E-003   1.57043E-003
  2    3.57845E-004   3.57845E-004   3.57845E-004   3.57845E-004
  3    1.82972E-006   1.82972E-006   1.82972E-006   1.82972E-006
  4    2.08843E-008   1.45257E-008   1.62673E-008   2.50365E-009
  5    3.30315E-009   1.99751E-010   1.33865E-010   6.39299E-010
  6    2.48209E-011   3.43365E-013   3.67097E-014   9.62371E-026
  7    3.23894E-031   5.23891E-037   0.00000E+000
```

Here the first row (Cycle = 0) corresponds to the off-norm of A.

To substantiate the claim that large angles cause the failure of the quadratic convergence, we can find the maximal sum $|\tan\varphi_k|+|\tan\psi_k|$ in each cycle and display the appropriate pivot submatrices. In particular, for the L method and for the first three cycles we have obtained the following results.

```
Cycle 1.  (l,m) = (1,3).
tan("smaller angle") = -8.47085668656445E-001
tan("larger angle")  = -8.47090649101242E-001

[all  alm]   [-1.00000440738603  -5.79947032748864E-006]
[aml  amm] = [ 0.00000000000000  -1.00000344053744     ]

Cycle 2.  (l,m) = (1,5).

tan("smaller angle") = 7.11094880472538E-001
```

```
tan("larger angle") = 7.11094890304722E-001

[all  alm]   [-1.00000001307540000      0.00000000000000]
[aml  amm] = [ 1.30603336415503E-008  -1.00000000853571]

Cycle 3.  (l,m) = (1,5)

tan("smaller angle") = 9.41439251845755E-001
tan("larger angle")  = 9.41439251761372E-001

[all  alm]   [-1.00000000004268  -8.94693349081228E-011]
[aml  amm] = [ 0.00000000000000  -1.00000000004808      ]
```

Thus the SVDT methods are in that respect quite similar to the symmetric/hermitian Jacobi methods. □

To ensure the quadratic convergence in the general case, SVDT algorithms are modified to try to arrange the sequence $|a_{11}^{(k)}|, \cdots, |a_{nn}^{(k)}|$ monotonically (see [8, § 3]). Such algorithms necessarily bring $A_{rr}^{(k)}$ to the first/last position on the block-diagonal of $A^{(k)}$. Once the proper affiliation has been reached, the modified algorithms automatically reduce to the standard ones. Hence it is quite natural to assume that $r \in \{1, p\}$. However, for our quadratic convergence proof we shall require the following less restrictive assumption.

(A3) If the multiplicity of the minimal singular value is larger than two, then at least one of the following conditions holds:

(1) $\sigma_{\min} = 0$;

(2) $2\sigma_{\min} \geqq 3\nu$;

(3) $r \in \{1, p\}$.

This assumption ensures that the relation (2.15) holds for $i = r$. As a consequence the relation (2.38) holds.

THEOREM 4.3. *Let the assumptions* (A1), (A2), *and* (A3) *hold. Then*

$$\|A^{(N+1)}\|_{\text{off}} \leqq \sqrt{1 + L/4}\, \|A\|_{\text{off}}^2 / \nu \tag{4.6}$$

where L is the number of distinct singular values whose multiplicities are larger than two.

Proof. Let us first assume that $n_1 > 2$ and let us estimate the off-diagonal elements of the first block-row of $T^{(N+1)}$. To this end we define

$$a_i^{(k)} = [t_{i,i+1}^{(k)}, \cdots, t_{i,n_1-1}^{(k)}], \qquad 1 \leqq i \leqq n_1 - 2,$$

$$x_i^{(k)} = \begin{cases} [t_{in_1}^{(k)}, \cdots, t_{in}^{(k)}] & \text{if } 1 \leqq i \leqq n_1 - 1, \\ [t_{i,n_1+1}, \cdots, t_{in}^{(k)}] & \text{if } i = n_1, \end{cases}$$

$$y_{ij}^{(k)} = [t_{ij}^{(k)}, t_{i+1,j}^{(k)}, \cdots, t_{n_1-1,j}^{(k)}], \quad 2 \leqq i \leqq n_1 - 1, \quad n_1 + 1 \leqq j \leqq n,$$

and

$$q_i = q_{i-1} + (n - i), \quad 1 \leqq i \leqq n; \quad q_0 = 1.$$

Since the formulae (1.3) for the off-diagonal elements of $T^{(k)}$ $(k \geqq 1)$ are almost identical to the ones for the symmetric Jacobi method, we can make an analysis quite similar to that of Van Kempen [20]. Thus we assume the row-cyclic strategy. By the result of Hansen [6] after N steps we shall obtain $A^{(N+1)}$ from the relation (4.6).

Since the formualae (1.3) hold also for the row-cyclic strategy, we obtain

$$\|x_i^{(q_i)}\|^2 \leqq \|T^{(q_{i-1}+n_1-i)}\|_{\text{off}}^2 \sum_{j=n_1+1}^{n} \sin^2 \eta_{ij}$$

$$\leqq \|T^{(1)}\|_{\text{off}}^2 \sum_{j=n_1+1}^{n} \sin^2 \eta_{ij}, \qquad 1 \leqq i \leqq n_1,$$

where

$$\eta_{ij} = \begin{cases} \psi_{ij} & \text{if } A \text{ is upper-triangular,} \\ \varphi_{ij} & \text{if } A \text{ is lower-triangular,} \end{cases}$$

and $\varphi_{ij} = \varphi_k$, $\psi_{ij} = \psi_k$, $\eta_{ij} = \eta_k$ provided that $(i,j) = (l(k), m(k))$, $1 \leqq k \leqq N$.

By an analysis similar to that of Wilkinson [22] we obtain

$$|a_i^{(q_i)}| \leqq |a_i^{(q_{i-1}+n_1-i)}| + \sum_{j=n_1+1}^{n} |y_{i+1,j}^{(q_{i-1}+n_1-i)}||\sin \eta_{ij}|, \qquad 1 \leqq i \leqq n_1 - 2.$$

Here $|x| \leqq |y|$ denotes $|x_j| \leqq |y_j|$ for all j. Using the inequality $(\alpha+\beta)^2 \leqq 2\alpha^2 + 2\beta^2$ and afterwards the Cauchy-Schwarz inequality, we obtain

$$\|a_i^{(q_i)}\|^2 \leqq 2\|a_i^{(q_{i-1}+n_1-i)}\|^2 + 2\|T^{(q_{i-1}+n_1-i)}\|_{\text{off}}^2 \sum_{j=n_1+1}^{n} \sin^2 \eta_{ij}$$

$$\leqq 2\|a^{(q_{i-1}+n_1-i)}\|^2 + 2\|T^{(1)}\|_{\text{off}}^2 \sum_{j=n_1+1}^{n} \sin^2 \eta_{ij}, \qquad 1 \leqq i \leqq n_1 - 2.$$

Note that

$$\|a_i^{(q_i)}\|^2 + \|x_i^{(q_i)}\|^2 = \|a_i^{(k)}\|^2 + \|x_i^{(k)}\|^2, \quad q_i \leqq k \leqq N+1, \quad 1 \leqq i \leqq n_1 - 2,$$

$$\|x_{n_1-1}^{(q_{n_1}-1)}\|^2 + \|x_{n_1}^{(q_{n_1})}\|^2 = \|x_{n_1-1}^{(k)}\|^2 + \|x_{n_1}^{(k)}\|^2, \qquad q_{n_1} \leqq k \leqq N+1.$$

Hence

$$\|T_{11}^{(N+1)}\|_{\text{off}}^2 + \sum_{j=2}^{p} \|T_{1j}^{(N+1)}\|^2 = \sum_{i=1}^{n_1-2} \|a_i^{(q_i)}\|^2 + \sum_{i=1}^{n_1} \|x_i^{(q_i)}\|^2$$

$$\leqq 2 \sum_{i=1}^{n_1-2} \|a_i^{(q_{i-1}+n_1-i)}\|^2 + 3\|T^{(1)}\|_{\text{off}}^2 \sum_{\substack{1 \leqq i \leqq n_1 \\ n_1+1 \leqq h \leqq n}} \sin^2 \eta_{ih}.$$

The gist of this analysis is to recognize that $a_i^{(q_{i-1}+n_1-i)}$, $1 \leqq i \leqq n_1 - 2$ are (the parts of) the rows of $\tilde{T}_{11}$, which results from $T_{11}^{(1)}$ if the row-cyclic method is applied to it. Since $\|\tilde{T}_{11}\|_{\text{off}} \leqq \|T_{11}^{(1)}\|_{\text{off}}$, we obtain

$$\|T_{11}^{(N+1)}\|_{\text{off}}^2 + \sum_{j=2}^{p} \|T_{1j}^{(N+1)}\|^2 \leqq 2\|T_{11}^{(1)}\|_{\text{off}}^2 + 3\|T^{(1)}\|^2 \sum_{\substack{1 \leqq i \leqq n_1 \\ n_1+1 \leqq h \leqq n}} \sin^2 \eta_{ih}.$$

By an obvious generalization, for $1 \leqq i \leqq p$ we obtain

$$\|T_{ii}^{(N+1)}\|_{\text{off}}^2 + \sum_{j=i+1}^{p} \|T_{ij}^{(N+1)}\|^2 \leqq \begin{cases} \|T^{(q_{s_{i-1}})}\|_{\text{off}}^2 \cdot \sum_{t,h}' \sin^2 \eta_{th}, & n_i \leqq 2, \\ 2\|T_{ii}^{(q_{s_{i-1}})}\|_{\text{off}}^2 + 3\|T^{(q_{s_{i-1}})}\|_{\text{off}}^2 \sum_{t,h}' \sin^2 \eta_{th}, & n_i > 2 \end{cases}$$

where $\sum'$ denotes the sum over all $t < h$ such that $s_{i-1} + 1 \leqq t \leqq s_i$ and $s_i + 1 \leqq h \leqq n$. Hence

$$\|T^{(N+1)}\|_{\text{off}}^2 \leqq 2L \max_{\substack{1 \leqq i \leqq p \\ n_i > 2}} \|T_{ii}^{(q_{s_{i-1}})}\|_{\text{off}}^2 + 3\|T^{(1)}\|^2 \sum_{k}'' \sin^2 \eta_k$$

where $\sum''$ denotes the sum over all k $(1 \leqq k \leqq N)$ such that $t_{lm}^{(k)}$ lies off the diagonal blocks. By Lemma 3.1 and by the inequality (4.5) we have

$$\sum_{k}'' \sin^2 \eta_k \leqq \frac{16}{49} \frac{1}{\nu^2} \|T^{(1)}\|_{\text{off}}^2.$$

By Lemma 2.1, Lemma 2.3, and by the assumption (A3) we have

$$\max_{\substack{i \\ n_i>2}} \| T_{ii}^{(q_{s_{i-1}})} \|_{\text{off}}^2 \leqq \frac{1}{8} \frac{\| T^{(q_{s_{i-1}})} \|_{\text{off}}^4}{\nu^2} \leqq \frac{1}{8} \frac{\| T^{(1)} \|_{\text{off}}^4}{\nu^2}.$$

Hence

$$\| A^{(N+1)} \|_{\text{off}}^2 = \| T^{(N+1)} \|_{\text{off}}^2 \leqq \left(\frac{L}{4} + 3 \cdot \frac{16}{49} \right) \frac{\| T^{(1)} \|_{\text{off}}^4}{\nu^2}$$

$$\leqq \left(1 + \frac{L}{4} \right) \frac{\| A^{(1)} \|_{\text{off}}^4}{\nu^2},$$

and the proof of Theorem 4.3 is completed. □

The proof of Theorem 4.3 is not as simple as the one of Van Kempen [20] because we have estimated the contribution $\sum_i \| a_i^{(q_i)} \|^2$ which was neglected in [20]. This fact has been recently discovered by Charlier and Van Dooren (see [3, Remark 5]). Note that our result is sharper than the one of Charlier and Van Dooren [3]. We note that the bound $\sqrt{1+L/4}$ can be further decreased up to a constant <2, but the proof requires some additional investigations. Hence we intend to publish it elsewhere together with the new proof of the Van Kempen's result for symmetric matrices.

The question that arises is whether we could prove the quadratic convergence without the assumption (A3). In fact we can start as follows.

By Theorem 2.4 we have

$$\| A^{(N+1)} \|_{\text{off}}^2 = \sum_{i=1}^{p} \| A_{ii}^{(N+1)} \|_{\text{off}}^2 + \tau^2(A^{(N+1)})$$

$$\leqq \frac{1}{4} \frac{\| A^{(N+1)} \|_{\text{off}}^4}{\nu^2} + 2\tau^2(A^{(N+1)})$$

$$\leqq \frac{1}{4} \| A^{(N+1)} \|_{\text{off}}^2 + 2\tau^2(A^{(N+1)}).$$

Hence

$$\| A^{(N+1)} \|_{\text{off}}^2 \leqq (8/3)\tau^2(A^{(N+1)})$$

and $\| A_{rr}^{(N+1)} \|_{\text{off}}$ is not needed in the estimates. The difficult part of such a proof lies in estimating the contributions of

$$\| a_i^{(q_{s_{i-1}}+n_r-i)} \|^2, \qquad s_{r-1}+1 \leqq i \leqq s_r$$

to the sum

$$\sum_{j=r+1}^{p} \| A_{rj}^{(q_{s_r})} \|^2.$$

We end this paper by a brief discussion how to deal with close singular values forming clusters.

Suppose A is a triangular matrix having p (>1) clusters of singular values, some of which may contain only one element. Let 2ρ be the maximal width of these clusters. If $U \Sigma_A V^*$ is the SVD of A, then we can write $\Sigma_A = \Sigma' + \Sigma''$ where

$$\Sigma' = \operatorname{diag} (\underbrace{\sigma_1', \cdots, \sigma_1'}_{n_1}, \underbrace{\sigma_2', \cdots, \sigma_2'}_{n_2}, \cdots, \underbrace{\sigma_p', \cdots, \sigma_p'}_{n_p}),$$

and σ_1' is the center of the ith cluster.

Consequently, we have $A = A' + A''$ where A' has multiple singular values and $\|A''\|_2 = \rho$. Let $3\nu' = \min_{i \neq j} |\sigma'_i - \sigma'_j|$. If $\|A'\|_{\text{off}} < \nu'$ and $\rho < \|A'\|^2_{\text{off}} / \nu'$, then the structure of the almost diagonal matrix A is dominated by that of A'. So, we can derive most of the estimates for proving the quadratic reduction of $\|A'\|_{\text{off}}$ and hence of $\|A\|_{\text{off}}$. Note that A' is not necessarily ET; hence the estimates can be based on a modification of Lemma 2.1, which deals with full square matrices (see [9]).

Although Charlier and Van Dooren [3] have recently proved such results (using another approach), there still remains a question as to whether our approach could yield sharper bounds.

We note that the quadratic convergence bound for the serial SVD methods plays an important role in the quadratic convergence estimates for certain parallel SVD methods (see [11]).

Acknowledgments. The author is thankful to professor K. Veselić from the University of Hagen as well as to the anonymous referees of this journal for useful comments.

REFERENCES

[1] R. BRENT, F. LUK, AND C. VAN LOAN, *Computation of the singular value decomposition using mesh-connected processors*, J. VLSI Comput. Syst., 1 (1985), pp. 242-270.

[2] J. P. CHARLIER, M. VANBEGIN, AND P. VAN DOOREN, *Of efficient implementation of Kogbetliantz's algorithm for computing the singular value decomposition*, Numer. Math., 52 (1988), pp. 279-300.

[3] J. P. CHARLIER AND P. VAN DOOREN, *On Kogbetliantz's SVD algorithm in the presence of clusters*, Linear Algebra Appl., 95 (1987), pp. 135-160.

[4] G. FORSYTHE AND P. HENRICI, *The cyclic Jacobi method for computing the principal values of a complex matrix*, Trans. Amer. Math. Soc., 94 (1960), pp. 1-23.

[5] G. GOLUB AND C. VAN LOAN, *Matrix Computations*, North Oxford Academic, Oxford, 1985.

[6] E. R. HANSEN, *On Jacobi methods and block Jacobi methods for computing matrix eigenvalues*, Ph.D. thesis, Stanford University, Stanford, CA, 1960.

[7] V. HARI, *On the convergence of cyclic Jacobi-like processes*, Linear Algebra Appl., 81 (1986), pp. 105-127.

[8] ———, *On SVD Jacobi operators*, University of Zagreb, Zagreb, Yugoslavia, 1987, preprint.

[9] ———, *On almost diagonal square matrices with multiple singular values*, Rad. Mat., 4 (1988), pp. 209-225.

[10] ———, *On the quadratic convergence of Jacobi algorithms*, Rad. Mat., 2 (1986), pp. 127-146.

[11] ———, *On convergence of certain parallel Jacobi methods*, University of Zagreb, Zagreb, Yugoslavia, 1987, preprint.

[12] V. HARI AND K. VESELIĆ, *On Jacobi methods for singular value decompositions*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 741-754.

[13] E. KOGBETLIANTZ, *Diagonalization of general complex matrices as a new method for solution of linear equations*, Proc. Internat. Congress Math. Amsterdam, 2 (1954), pp. 356-357.

[14] ———, *Solution of linear equations by diagonalization of coefficient matrices*, Quart. Appl. Math., 13 (1955), pp. 123-132.

[15] F. LUK, *A triangular processor array for computing the singular value decomposition*, Linear Algebra Appl., 77 (1986), pp. 259-173.

[16] M. H. C. PAARDEKOOPER, *An eigenvalue algorithm based on normreducing transformations*, Ph.D. thesis, Technological University of Eindhoven, Eindhoven, The Netherlands, 1969.

[17] D. PAIGE, *Computing the generalized singular value decomposition*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 1126-1146.

[18] C. PAIGE AND P. VAN DOOREN, *A note on the convergence of Kogbetliantz's iterative algorithm for obtaining the singular value decomposition*, Linear Algebra Appl., 77 (1986), pp. 301-313.

[19] B. N. PARLET, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[20] H. P. M. VAN KEMPEN, *On the quadratic convergence of the special cyclic Jacobi method*, Numer. Math., 9 (1966), pp. 19-22.

[21] V. V. VOEVODIN, *Cislennye metody linejnoj algebry*, Nauka, Moscow, 1966. (In Russian.)

[22] J. H. WILKINSON, *Note on the quadratic convergence of the cyclic Jacobi process*, Numer. Math., 4 (1962), pp. 296-300.

[23] ———, *Almost diagonal matrices with multiple or close eigenvalues*, Linear Algebra Appl., 1 (1968), pp. 1-12.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 6, pp. 1097-1112, November 1989
© 1989 Society for Industrial and Applied Mathematics
003

COMPUTATION OF THE EIGENVALUES FOR PERTURBATIONS OF POISEUILLE FLOW USING A TWO-POINT BOUNDARY VALUE METHOD*

GERARDO A. ACHE†

Abstract. The decay rates for stationary perturbations of Poiseuille flow in channels and pipes are computed. The decay rates are found by solving eigenvalue problems of ordinary differential equations, where the eigenvalues give the rate of decay for the perturbation. A two-point boundary value method is used to compute the eigenvalues yielding efficient and accurate calculations. For the channel flow problem, the results are in agreement with previous calculations (e.g., Bramley [*J. Comput. Phys.*, 53 (1984), pp. 524-529]; Bramley and Dennis [*J. Comput. Phys.*, 47 (1982), pp. 179-198], [*J. Math. Anal. Appl.*, 101 (1984), pp. 30-38]; Gillis and Brandt [Report AF EOAR, SR-1, Weizmann Institute, Rehovot, Israel, 1964, pp. 63-73]; Wilson [*J. Fluid Mech.*, 38 (1968), pp. 793-806]), however, the problem of determining the rate of decay for a fluid motion in a pipe has been considered by Bramley [*J. Comput. Phys.*, 64 (1986), pp. 258-265]. It is proved that for the Stokes problem in a pipe the eigenvalues governing the rate of decay are complex. Computations for small and moderate Reynolds number are performed, as are high Reynolds number computations, to show the effectiveness of this method.

Key words. Navier-Stokes, eigenvalue problem, Poiseuille flow, PASVA3, Reynolds number, asymptotic

AMS(MOS) subject classifications. 65L15, 76D07

1. Introduction.

In this paper we are concerned with the eigenvalue problem that governs the rate of decay for a stationary perturbation of Poiseuille flow. We consider two-dimensional viscous motions in channels and axisymmetric viscous flow in pipes. We assume that the difference between the base flow and Poiseuille flow decays exponentially downstream (or upstream). It is then possible to seek solutions to the Navier-Stokes equations, far downstream (or upstream), that are a perturbation to the Poiseuille profile and that decay exponentially in the axial direction. The equations can then be linearized yielding an ordinary differential eigenvalue system where the eigenvalues determine the rate of decay for the stationary perturbation.

By use of the stream function formulation, it is possible to reduce the eigenvalue system to a single fourth-order differential eigenvalue equation for the decay of the stationary perturbation. In the two-dimensional case this differential equation is very similar to the Orr-Sommerfeld stability equation. Results regarding the computation of these eigenvalues, in the case of channel flow, have been presented previously (e.g., in [3], [4], [5], [8], [12]), however, when the fluid motion is considered axially symmetric, the problem of determining the rate of decay for the stationary perturbation has been considered by Bramley [6].

To compute the eigenvalues several methods have been used—for example, a spectral method by Bramley [3] and Bramley and Dennis [4], an initial value method by Bramley and Dennis [5], and a singular perturbation method by Wilson [12]. All

* Received by the editors October 7, 1987; accepted for publication (in revised form) October 4, 1988. This paper is a portion of the author's Ph.D. dissertation, carried out at the University of Wisconsin, Madison, Wisconsin. The final report was written while he was visiting the Center for the Mathematical Sciences, University of Wisconsin, Madison, Wisconsin. This work was supported in part by U.S. Army contract DAAG29-80-C-0041 and in part by Consejo de Desarrollo Científico y Humanístico, Universidad Central de Venezuela, Caracas, Venezuela.

† Facultad de Ciencias, Universidad Central de Venezuela, Apartado 4311, Carmelitas, Caracas 1010, Venezuela.

of these works have dealt with the two-dimensional channel flow problem, and with the exception of Bramley [3] all of them have considered the Navier-Stokes equations in the stream function formulation. The work of Bramley and Dennis [4] appears to be the most complete among all others mentioned above. They used an extension of the Orszag spectral method presented in [9]. In [3] Bramley used the same method to compute the eigenvalues using the primitive formulation for the Navier-Stokes equations, i.e., he used the velocity field and the pressure instead of the stream function. However the results of Bramley [3] are not in complete agreement with [4]. This disagreement may be the result of using the wrong number of boundary conditions. There are two disadvantages reported in [4] with respect to spectral methods; these are the computation of spurious eigenvalues and the loss of accuracy for high Reynolds numbers computations, especially for eigenvalues with negative real part. In [5] Bramley and Dennis used an initial value method to compute the eigenvalues. They said that using this method it is possible to overcome some of the difficulties of the spectral method.

In this paper we compute the eigenvalues for a perturbation of Poiseuille flow, in a channel and a pipe, using a more accurate and efficient method. This method can be described in two steps. First, we transform the eigenvalue problem into an equivalent two-point boundary value system; then we numerically solve this system using the two-point boundary value solver DVCPR from the IMSL library. For the two-dimensional problem our results are in full agreement with the previous work of Bramley and Dennis [4], for small and moderate Reynolds numbers, and with the asymptotic prediction of Wilson [12], for eigenvalues approaching zero at high Reynolds numbers. The numerical results presented in this paper show the efficiency of the calculations and the superiority of our method with respect to the other methods mentioned above. These eigenvalues are important for the derivation of boundary conditions; this problem was considered in [1].

2. Channel flow. Given a semi-infinite channel, the incompressible Navier-Stokes equations are given by

$$u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+\frac{\partial p}{\partial x}=\frac{1}{R}\nabla^2 u, \tag{2.1a}$$

$$u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+\frac{\partial p}{\partial y}=\frac{1}{R}\nabla^2 v, \tag{2.1b}$$

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0, \tag{2.1c}$$

with boundary conditions

$$u=v=0 \quad \text{at } y=\pm 1, \tag{2.1d}$$

$$u=F_1(y), \quad v=F_2(y) \quad \text{at } x=0, \tag{2.1e}$$

where F_i is a profile satisfying $F_i(\pm 1)=0$. Finally we have the regularity condition

$$(u, v)\to(\hat{u}, 0) \quad \text{as } x\to\infty. \tag{2.1f}$$

Here R represents the Reynolds number, $\nabla^2=\partial^2/\partial x^2+\partial^2/\partial y^2$, and $\hat{u}=\frac{3}{2}(1-y^2)$ is the Poiseuille parabolic profile.

Since we are interested in solutions of this system that decay exponentially, far downstream we seek an asymptotic solution to (2.1) in the form

$$u(x, y) = \tfrac{3}{2}(1-y^2) + W_1(y) \exp(-\lambda x), \tag{2.2a}$$

$$v(x, y) = W_2(y) \exp(-\lambda x), \tag{2.2b}$$

$$p(x, y) = -3R^{-1}x + R^{-1}\bar{q}(y) \exp(-\lambda x) + \bar{C}, \tag{2.2c}$$

where $\bar{C}$ is an arbitrary constant.

Substituting these expressions in equations (2.1a)-(2.1c), and neglecting nonlinear terms, we have that $(\lambda, W_1, W_2, \bar{q})$ satisfies the following eigenvalue system of ordinary differential equations:

$$\frac{d^2W_1}{dy^2} = -R\left\{\frac{3}{2}(1-y^2)\lambda W_1 + 3yW_2\right\} - \lambda\bar{q} - \lambda^2 W_1, \tag{2.3a}$$

$$\frac{dW_2}{dy} = \lambda W_1, \tag{2.3b}$$

$$\frac{d\bar{q}}{dy} = \frac{3}{2}R(1-y^2)\lambda W_2 + \lambda^2 W_2 + \lambda\frac{dW_1}{dy}, \tag{2.3c}$$

and boundary conditions

$$W_1(\pm 1) = W_2(\pm 1) = 0. \tag{2.3d}$$

We are interested in solving this two-point boundary value system to determine the rate of decay λ. For x large (i.e., downstream) the solution to (2.1) may be represented by

$$u(x, y) = \frac{3}{2}(1-y^2) + \sum_n W_{1,n}(y) \exp(-\lambda_n x), \tag{2.4a}$$

$$v(x, y) = \sum_n W_{2,n}(y) \exp(-\lambda_n x), \tag{2.4b}$$

and

$$q(x, y) = -3R^{-1}x + R^{-1}\sum_n \bar{q}_n(y) \exp(-\lambda_n x) + \bar{C}, \tag{2.4c}$$

$$Re(\lambda_n) \leqq Re(\lambda_{n+1}). \tag{2.4d}$$

The sum in (2.4a)-(2.4c) is taken over all the eigenvalues with positive real part (the eigenvalues with negative real part can be used to construct an asymptotic solution to (2.1) in a semi-infinite channel on the negative x-axis), $(W_{1,n}, W_{2,n}, \bar{q}, \lambda_n)$ being a solution to the system (2.3).

We may eliminate the perturbed pressure $\bar{q}$ from the system (2.3) by differentiating (2.3a) with respect to y, multiplying (2.3a) by $-\lambda$, adding the resulting expressions and using (2.3b). Then the system (2.3) is reduced into a single fourth-order differential equation in W_2 and λ very similar to the Orr-Sommerfeld equation (for simplicity we replace W_2 by W)

$$\frac{d^4W}{dy^4} + 2\lambda^2\frac{d^2W}{dy^2} + \lambda^4 W + \lambda R\left\{\frac{3}{2}(1-y^2)\left(\frac{d^2W}{dy^2} + \lambda^2 W\right) + 3W\right\} = 0, \tag{2.5a}$$

and boundary conditions

$$W = \frac{dW}{dy} = 0 \quad \text{at } y = \pm 1. \tag{2.5b}$$

We solved this equation in order to compare our method of solution with previous computations. Equations (2.5) are associated with the stream function formulation of the Navier-Stokes equations (see, e.g., [4]).

When the Reynolds number, R, is equal to zero we may find an explicit solution to (2.5) and therefore to (2.3). This solution is given in terms of the Papkovitch-Fad'le functions for the biharmonic equation in a semi-infinite strip. For $R=0$ (2.5a), (2.5b) become

$$\frac{d^4W}{dy^4}+2\lambda^2\frac{d^2W}{dy^2}+\lambda^4W=0, \tag{2.6}$$

and boundary conditions (2.5b). This equation is associated with the biharmonic equation when solutions of the form $w(x, y)=W(y)\exp(-\lambda x)$ are sought. There are two types of solutions for (2.6), one even and the other odd, which are given by

$$W^e(y)=(y-1)\sin(\lambda^e(y+1))+(y+1)\sin(\lambda^e(y-1)), \tag{2.7a}$$

and

$$W^o(y)=(y-1)\sin(\lambda^o(y+1))-(y+1)\sin(\lambda^o(y-1)), \tag{2.7b}$$

respectively. To satisfy the boundary conditions (2.5b) we need that the eigenvalues λ^e and λ^o satisfy the following transcendental equations:

$$\sin(2\lambda^e)+2\lambda^e=0, \tag{2.8a}$$

and

$$\sin(2\lambda^o)-2\lambda^o=0, \tag{2.8b}$$

respectively. We note that if λ is an eigenvalue satisfying (2.8) so are $-\lambda$ and $\pm\bar{\lambda}$. Also the function

$$w(x, y)=\sum_n \alpha_n W_n(y)\exp(-\lambda_n x), \tag{2.9}$$

where the sum is taken over the eigenvalues with positive real part, satisfies the biharmonic equation in a semi-infinite strip with transversal boundary conditions $w=\partial w/\partial y=0$.

In § 4 we discuss a method to solve the eigenvalue problems (2.3) and (2.5), where the solution to the Stokes problem (2.6) plays an important role. Also in § 5 we present numerical results involving the computation of these eigenvalues for arbitrary Reynolds numbers.

3. Axially symmetric flow. In this section we consider the incompressible Navier-Stokes equations in cylindrical coordinates and dimensionless form, set in a semi-infinite pipe. These equations can be written as

$$u\frac{\partial u}{\partial r}+v\frac{\partial u}{\partial z}+\frac{\partial p}{\partial r}=\frac{1}{R}\left(\nabla^2u-\frac{u}{r^2}\right), \tag{3.1a}$$

$$u\frac{\partial v}{\partial r}+v\frac{\partial v}{\partial z}+\frac{\partial p}{\partial z}=\frac{1}{R}\nabla^2v, \tag{3.1b}$$

$$\frac{1}{r}\frac{\partial}{\partial r}(ru)+\frac{\partial v}{\partial z}=0, \tag{3.1c}$$

and boundary conditions

$$u=v=0 \quad \text{at } r=1. \tag{3.1d}$$

Since the fluid motion is symmetric the conditions at the center line are

$$u = \frac{\partial v}{\partial r} = 0 \quad \text{at } r = 0. \tag{3.1e}$$

We also specify the entry condition

$$u = F_1(r), \quad v = F_2(r) \quad \text{at } z = 0, \tag{3.1f}$$

and finally we have the regularity condition,

$$(u, v) \to (0, \hat{v}) \quad \text{as } z \to \infty. \tag{3.1g}$$

Here $\nabla^2 = 1/r\, \partial/\partial r\, (r\partial/\partial r) + \partial^2/\partial z^2$, and $\hat{v} = 1 - r^2$.

Similar to the channel flow problem we seek an asymptotic solution to (3.1) in the form

$$u(r, z) = W_1(r) \exp(-\lambda z), \tag{3.2a}$$

$$v(r, z) = 1 - r^2 + W_2(r) \exp(\cdots \lambda z), \tag{3.2b}$$

and

$$p(r, z) = -4R^{-1}z + R^{-1}\bar{q}(r) \exp(-\lambda z) + \bar{C}, \tag{3.2c}$$

with $\bar{C}$ an arbitrary constant.

Substituting these expressions in (3.1a)-(3.1c), and neglecting nonlinear terms, we obtain an eigenvalue system in W_1, W_2, $\bar{q}$, and λ similar to (2.3), i.e.,

$$\frac{dW_1}{dr} = \lambda W_2 - \frac{W_1}{r}, \tag{3.3a}$$

$$\frac{d^2W_2}{dr^2} = -R\{2rW_1 + \lambda(1 - r^2)W_2\} - \frac{1}{r}\frac{dW_2}{dr} - \lambda^2 W_2 - \lambda\bar{q}, \tag{3.3b}$$

$$\frac{d\bar{q}}{dr} = R(1 - r^2)\lambda W_1 + \lambda\frac{dW_2}{dr} + \lambda^2 W_1, \tag{3.3c}$$

with boundary conditions,

$$W_1(0) = \frac{dW_2}{dr}(0) = 0 \quad \text{and} \quad W_1(1) = W_2(1) = 0. \tag{3.3d}$$

By solving this system we may represent the solution to (2.1), for z far downstream, in the same way as for the channel flow problem, i.e., by expressions similar to (2.4a)-(2.4d).

In a way similar to the channel flow problem, we solved the system (3.3) numerically, using the solution to the system for $R = 0$ (the Stokes problem). These solutions may be found as follows. When $R = 0$ the system (3.3) becomes,

$$\frac{dW_1}{dr} = \lambda W_2 - \frac{W_1}{r}, \tag{3.4a}$$

$$\frac{d^2W_2}{dr^2} = -\frac{1}{r}\frac{dW_2}{dr} - \lambda^2 W_2 - \lambda\bar{q}, \tag{3.4b}$$

$$\frac{d\bar{q}}{dr} = \lambda\frac{dW_2}{dr} + \lambda^2 W_1, \tag{3.4c}$$

and boundary conditions (3.3d).

To find an explicit solution to this system differentiate (3.4c) with respect to r, multiply (3.4a) by λ^2 and (3.4b) by λ, then add the resulting expressions and using (3.4c) we obtain the following differential equation in $\bar{q}$:

$$\frac{d^2\bar{q}}{dr^2}+\frac{1}{r}\frac{d\bar{q}}{dr}+\lambda^2\bar{q}=0. \tag{3.5}$$

This equation has the particular solution,

$$\bar{q}(r)=J_0(\lambda r), \tag{3.6}$$

where J_0 is the Bessel function of first kind and order zero. This is the only solution, up to a multiplicative constant, which is finite when r is equal to zero. Substituting $\bar{q}(r)$ in (3.4b), we obtain the differential equation

$$\frac{d^2W_2}{dr^2}+\frac{1}{r}\frac{dW_2}{dr}+\lambda^2 W_2=-\lambda J_0(\lambda r), \tag{3.7a}$$

with boundary conditions,

$$\frac{dW_2}{dr}(0)=W_2(1)=0. \tag{3.7b}$$

This two-point boundary value problem has the solution,

$$W_2(r)=\bar{b}J_0(\lambda r)-\tfrac{1}{2}rJ_1(\lambda r), \tag{3.8}$$

where $\bar{b}$ is a constant to be determined by $W_2(1)=0$. When we use (3.4a), W_1 is given as follows:

$$W_1(r)=\frac{1}{\lambda}\left[\frac{\lambda}{2}rJ_0(\lambda r)-(1-\lambda\bar{b})J_1(\lambda r)\right]. \tag{3.9}$$

Since the solutions (3.8) and (3.9) need to satisfy the boundary conditions (3.3d), we have that

$$\lambda W_1(1)=\frac{\lambda}{2}J_0(\lambda)-(1-\lambda\bar{b})J_1(\lambda)=0, \tag{3.10a}$$

and

$$W_2(1)=\bar{b}J_0(\lambda)-\tfrac{1}{2}J_1(\lambda)=0. \tag{3.10b}$$

By eliminating $\bar{b}$ from these two equations, we obtain

$$\frac{1\pm\sqrt{1-\lambda^2}}{2\lambda}J_0(\lambda)-\frac{1}{2}J_1(\lambda)=0. \tag{3.11}$$

To solve this equation we observe that (3.11) can be transformed, after some algebraic manipulations, into the nonlinear equation

$$J_1(\lambda)J_0(\lambda)-\frac{\lambda}{2}(J_1^2(\lambda)+J_0^2(\lambda))=0. \tag{3.12}$$

LEMMA 3.1. *All nontrivial solutions to the equation* (3.12), *that is those with* $\lambda\neq 0$, *are complex.*

Proof. We make use of the following integral formula involving the Bessel functions:

$$\int xJ_1^2(x)\,dx=\frac{1}{2}x^2(J_0^2(x)+J_1^2(x))-xJ_0(x)J_1(x). \tag{3.13}$$

This formula is obtained by integrating by parts Lommel's integral formula with $\alpha = 1$ (see [2, p. 10]). Then assuming that some λ satisfying (3.12) is real, we get the contradiction that

$$0 < \int_0^{\lambda} xJ_1(x)^2\,dx = \frac{1}{2}\lambda^2(J_0^2(\lambda)+J_1^2(\lambda)) - \lambda J_0(\lambda)J_1(\lambda) = 0,$$

where we have assumed without loss of generality that $\lambda > 0$. This contradiction proves the lemma.

It is known that the Bessel functions have the asymptotic representations

$$J_0(\lambda) = \left(\frac{2}{\pi\lambda}\right)^{1/2}\left\{\cos\left(\lambda - \frac{\pi}{4}\right) + \varepsilon_1(\lambda)\right\}, \tag{3.14a}$$

and

$$J_1(\lambda) = \left(\frac{2}{\pi\lambda}\right)^{1/2}\left\{\sin\left(\lambda - \frac{\pi}{4}\right) + \varepsilon_2(\lambda)\right\}, \tag{3.14b}$$

where $\varepsilon_1(\lambda)$, $\varepsilon_2(\lambda)$ can be regarded as small-order terms for $Re(\lambda)$ large. Substituting these asymptotic expressions in (3.12), and neglecting $\varepsilon_1(\lambda)$ and $\varepsilon_2(\lambda)$, we have that the λ's can be approximated by solutions of the transcendental equation

$$\cos(2\tilde{\lambda}) + \tilde{\lambda} = 0. \tag{3.15}$$

Solutions of this equation can be used as initial iterates for a Newton method to obtain the roots of (3.12). We will present the computational details in § 5.

In a way similar to the channel flow problem we have used the solution to the Stokes problem to numerically solve the system (3.3) for arbitrary Reynolds numbers.

4. Method of solution. In this section we discuss a method to solve the two-point boundary value problems for the systems (2.3), (3.3) and equation (2.5) and determine their eigenvalues. The method can be described as follows. First the eigenvalue problem is transformed into an equivalent two-point boundary value first-order system of the form

$$\vec{\mathbf{Z}}' = \mathbf{A}(\lambda, R, t)\vec{\mathbf{Z}}, \qquad a \leqq t \leqq b, \tag{4.1a}$$

$$\lambda' = 0, \tag{4.1b}$$

with boundary conditions

$$\mathbf{B}_a\vec{\mathbf{Z}}(a) + \mathbf{B}_b\vec{\mathbf{Z}}(b) = 0, \tag{4.1c}$$

$$z_k(b) = 1, \tag{4.1d}$$

for some k, $1 \leqq k \leqq N$. Here $\vec{\mathbf{Z}} = (z_1, z_2, \cdots, z_N)$, and $\mathbf{A}$, $\mathbf{B_a}$, $\mathbf{B_b}$ are $N \times N$ matrices. The prime in (4.1a) indicates derivatives with respect to t. The condition (4.1d) has to be chosen in a way that is not in conflict with (4.1c). In the case of systems (2.3), (3.3) and equation (2.5) we have used for condition (4.1d) $W_2'(1) = 1$, $W_1'(1) = 1$, and $W''(1) = 1$, respectively. To solve numerically the system (4.1) we have used the two-point boundary value system code DVCPR from the IMSL library (this code is also known as PSAVA3); it is a standard solver for first-order systems of ordinary differential equations with conditions at two endpoints. It uses a variable step, with an automatic criterion to select a nonuniform grid, and a variable order of accuracy, with an excellent correspondence between the requested tolerance (tol) and the actual global error in the numerical solution (see [10]). Since the systems considered are nonlinear we used as initial iterate in PASVA3 the solution to the corresponding Stokes problem ($R = 0$), which is known and for which the boundary conditions are satisfied,

to generate solutions for arbitrary Reynolds numbers. This procedure is called successive continuation. The disadvantages of this method are that we have to compute the eigenvalues one at a time; however, the eigenvalues are computed together with their eigenfunctions. Also we cannot compute the solution at a critical Reynolds number, i.e., a Reynolds number for which the solution to the system is not isolated; however, it is possible to compute solutions for Reynolds numbers close to a critical value. By using this method we avoid the computation of spurious eigenvalues, and also for high Reynolds numbers the solution can be computed in a very accurate manner.

5. Numerical results. We now present some numerical results concerning the eigenvalue problems described in §§ 2 and 3. For the channel flow problem our results are in full agreement with the previous computation of Bramley and Dennis [4]. Also we obtain the same answer by solving the system (4.1) corresponding to the primitive formulation, i.e., equations (2.3), and the system (4.1) corresponding to the stream function formulation, i.e., equation (2.5).

We present about the same number of results as in [4]. We present our results graphically concerning the behavior of these eigenvalues in the complex plane. We regard these eigenvalues as a function of Reynolds numbers in the form $\lambda(R) = \alpha(R) + i\beta(R)$ and plot the pairs $(\alpha(R), \beta(R))$ in the λ plane. We computed even and odd branches of eigenvalues and eigenvalues for axisymmetric flow. Note that for the axisymmetric flow there is no associated parity, such as even or odd; thus, when we refer to an even or odd solution it means an even or odd solution to the channel problem.

For the channel flow problem we have found that these eigenvalues behave as follows (see Figs. 1.1(a) and 1.1(b)). Let $(\lambda_k(0), \vec{\mathbf{Z}}_k(t))$ be an eigenpair solution for $R = 0$ where $|Re(\lambda_k)| < |Re(\lambda_{k+1})|$, $k = 1, 2, \cdots$. For $0 \leqq R < R^k_{c,0}$ $(\lambda_k(R), \vec{\mathbf{Z}}_k(t))$ remains complex, its conjugate also being an eigenpair solution. At $R = R^k_{c,0}$ this complex solution and its conjugate coalesce on the real axis and, then for R larger

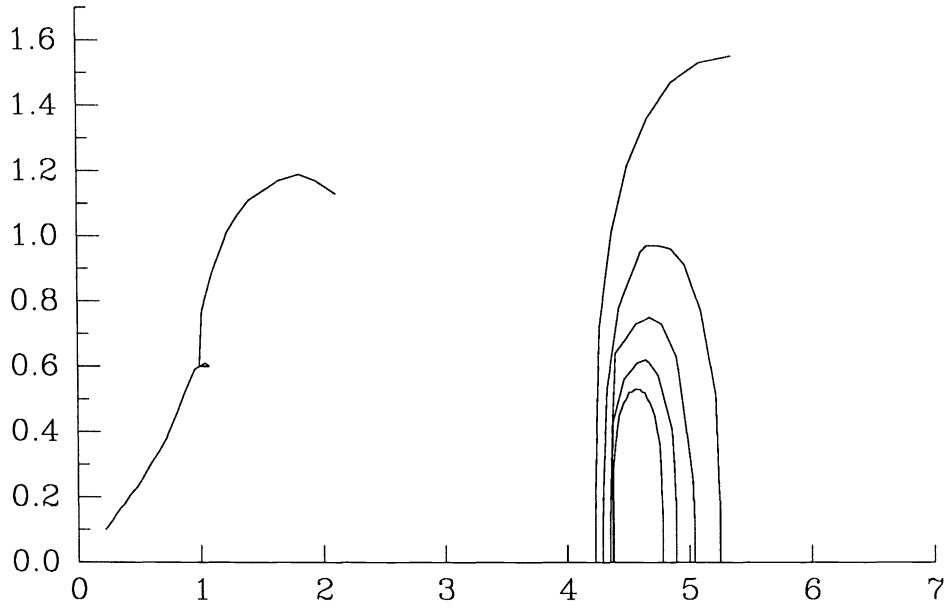


FIG. 1.1(a). *Even eigenvalues in the λ plane.*

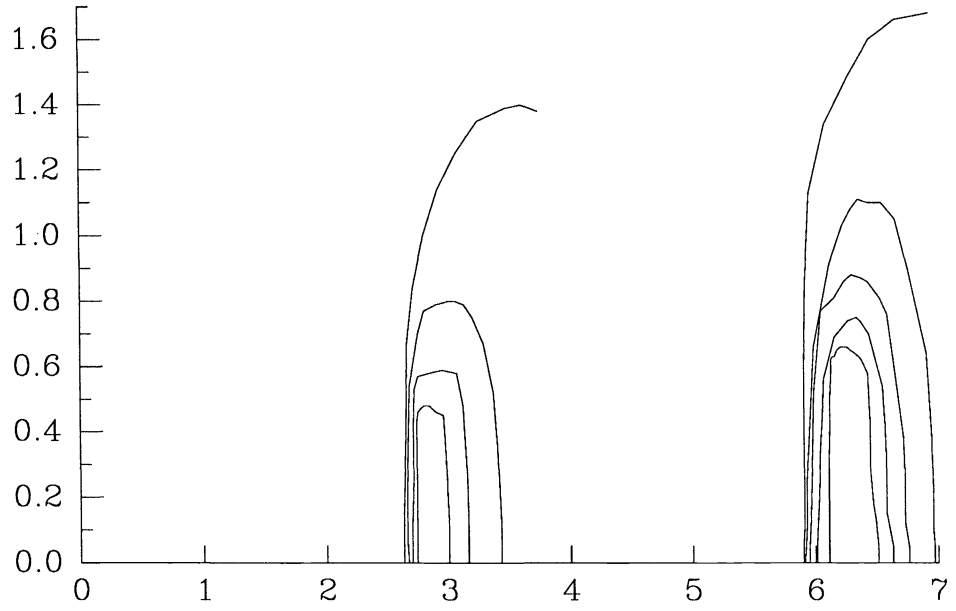


FIG. 1.1(b). *Odd eigenvalues in the λ plane.*

than $R_{c,0}^k$ they split into two branches of real solutions, one with increasing eigenvalues and the other with decreasing eigenvalues. For solutions with positive eigenvalues the increasing real solution $(\lambda_k(R), \vec{\mathbf{Z}}_k(t))$ coalesces with the decreasing real solution $(\lambda_{k+1}(R), \vec{\mathbf{Z}}_{k+1}(t))$ of the next branch at $R = R_{c,1}^k$. Then, for larger values of R these two solutions split into a complex solution and its conjugate, and again they remain complex for $R_{c,1}^k < R < R_{c,2}^k$. Then at $R = R_{c,2}^k$ the complex solution and its conjugate again coalesce on the real axis, and the cycle is repeated. Notice that $R_{c,m}^k$, $m = 1, 2, \cdots$, are the critical Reynolds numbers. There is an exception to this cycling phenomenon—the first even branch of solution remains complex and approaches zero as R increases. Also the first real branch of even and odd decreasing eigenvalues approaches zero as R increases. None of the previous researchers have reported the second type of critical Reynolds number at which two real solutions coalesce and split into a complex solution and its conjugate. Only Wilson [12] has pointed out that the positive increasing real eigenvalues approach a fixed value on the real axis. As can be seen from Figs. 1.1(a) and 1.1(b), the cycles of the eigenvalues appear to be converging to a value on the real axis.

For eigenvalues with negative real part the situation is different. Solutions with eigenvalues that decrease in magnitude either approach zero as R increases, but at a much slower rate than solutions with positive real eigenvalues, or they approach a fixed value. In this case, the real eigenvalues increase or decrease so slowly that for the range of the Reynolds number for which we made the computations, we did not observe the coalescence of real eigenvalues with negative real part.

We have found that for the axisymmetric problem the solutions behave in the same manner as the odd solutions for the channel problem. That is, the solutions are initially complex; the conjugate pairs coalesce on the real axis; the real solutions, with positive eigenvalues, coalesce, becoming complex; etc.

Figures 1.1(a) and 1.1(b) show the eigenvalues with positive real part in the λ plane for the first and second branches of even and odd eigenvalues, respectively, without including the complex conjugate. Also we do not include graphical displays for the axisymmetric case since it is similar to the odd case. These graphics have been obtained for a range of values of R that is different for each curve. For the even eigenvalues the first curve, corresponding to the complex branch, was made for $0 \leqq R \leqq 10^6$ while the second branch was for $0 \leqq R \leqq 100$. For odd eigenvalues the first curve corresponds to $0 \leqq R \leqq 90$, and the second for $0 \leqq R \leqq 100$. These differences arise because we wanted to illustrate the cycles for each branch. All the computations were done on the VAX 11/780 at the Mathematics Research Center at the University of Wisconsin, Madison, Wisconsin.

As in [4] we present several tables that give the computed eigenvalues. For $R = 0$, in the channel case, the eigenvalues may be computed by solving the transcendental equations (2.8a) and (2.8b). Results concerning the solutions of these equations can be found in [7] and [11]. However for the axisymmetric case, since there are no previous results for the Stokes problem, we may start by computing the first few eigenvalues, with positive real part, which correspond to solutions of equation (3.12). As we said in § 3, to solve equation (3.12) we used solutions to (3.15) as initial iterates for the Newton method. At the same time, to solve (3.15) we used the Newton method and as initial iterates the expressions

$$\lambda_n = \gamma_n - \frac{\log(2\gamma_n)}{4\gamma_n} + \frac{i}{2}\log(2\gamma_n), \tag{5.1}$$

with $\gamma_n = (2n+1)\pi/2$, $n = 1, 2, \cdots$, since these expressions are asymptotic solutions to (3.15). In each case we have found that only two or three iterations were needed for the Newton method to find solutions to (3.12) and (3.15) with six decimal places of accuracy. The first several solutions to the equation (3.12) are given in Table 1. Also, we are including only those tables for which we have obtained results that are not reported in [4] (the complete set of tables regarding all the numerical results can be found in [1]), for example, in Tables 2–6 we show real and complex eigenvalues for the axisymmetric case, in Tables 8 and 9 we give the critical Reynolds numbers, and eigenvalues, corresponding to graphs in (1.1a) and (1.1b), and in Tables 10 and 11 we show the eigenvalues for high Reynolds number calculations.

As an example of successive continuation, we have that to compute the first branch of even eigenvalues at $R = 100{,}000$ the following continuation steps were made: $R = 0$;

TABLE 1
Solutions to equation (3.12).

λ
4.46630+1.46747i
7.69410+1.72697i
10.87457+1.89494i
14.03889+2.02006i
17.19557+2.11995i
20.34797+2.20312i
23.49772+2.27441i
26.64572+2.33678i
29.79248+2.39223i
32.93833+2.44214i

TABLE 2
Axisymmetric complex eigenvalues with positive real part for $0 \leqq R \leqq 100$.

R	λ	λ	λ
0	4.46620+1.46747i	7.69410+1.72697i	10.87457+1.89494i
1	4.29200+1.42646i	7.52551+1.69447i	10.70765+1.86725i
5	3.76255+1.06197i	6.95388+1.31088i	10.11519+1.47824i
15	4.24793+0.78668i	7.31772+1.10920i	10.40317+1.28557i
25	3.70720+0.50937i	7.77552+0.31462i	10.82173+0.95050i
50	3.79197+0.45240i	7.44769+0.69724i	10.31659+0.90353i
100	4.17041+0.27261i	7.48688+0.33362i	10.22035+0.64556i

TABLE 3
Complex eigenvalues with smallest positive real part for $R > 100$.

R	λ^e(even)	λ^o(odd)	λ(axi-symm)
250	2.78306+0.27633i	0.72120+0.37930i	3.80091+0.06650i
500	2.71137+0.15264i	0.65637+0.33642i	3.97900+0.17319i
1000	2.70753+0.15583i	0.59650+0.29956i	3.86180+0.15698i
2500	2.67659+0.12999i	0.38104+0.18322i	3.50706+0.09679i
5000	2.59490+0.08062i	0.34571+0.16572i	

TABLE 4
Axisymmetric complex eigenvalues with negative real part for $0 \leqq R \leqq 10$.

R	λ	λ	λ
0.25	−4.5126+1.4738i	−7.7378+1.7309i	−10.9174+1.8976i
0.50	−4.5601+1.4784i	−7.7821+1.7331i	−10.9607+1.8986i
1.00	−4.6584+1.4823i	−7.8727+1.7318i	−11.0486+1.8949i
2.50	−4.9823+1.4417i	−8.1593+1.6744i	−11.3231+1.8331i
5.00	−5.6481+1.0772i	−8.6844+1.2420i	−11.8148+1.4338i
10.00	−8.3161+1.5370i	−11.4575+1.4277i	−14.4887+1.5590i

TABLE 5
Real positive axisymmetric eigenvalues for $R \leqq 5{,}000$.

R	λ	λ	λ	λ	λ
10	2.84882				
25	1.23820				
50	0.63541	1.86985			
100	0.32000	0.95032	1.89301	3.21182	
250	0.12827	0.38230	0.76379	1.27228	1.90837
500	0.06415	0.19134	0.38250	0.63760	0.95652
1000	0.03208	0.09568	0.19133	0.31900	0.47867
5000	0.00642	0.01914	0.03827	0.06381	0.09576

TABLE 6
Real negative axisymmetric eigenvalues for $R \leqq 10^5$.

R	λ	λ	λ
10	−4.94606		
25	−4.49291	−8.13980	−12.42763
50	−4.31816	−7.77838	−11.24747
100	−4.19980	−7.57708	−10.92172
250	−4.09219	−7.40731	−10.68291
500	−4.03427	−7.31888	−10.56491
1000	−3.99010	−7.25228	−10.47775
2500	−3.94679	−7.18740	−10.39379
5000	−3.92235	−7.15090	−10.34684
10000	−3.90323	−7.12238	−10.31023
50000	−3.87316	−7.07752	−10.25277
100000	−3.86452	−7.06463	−10.23627

TABLE 7
Odd complex eigenvalues with negative real part for $R \geqq 10$.

R	λ^o (tol $= 10^{-6}$)	λ^o (tol $\geqq 10^{-8}$)
10	−10.458682 + 1.694815i	−10.458682 + 1.694815i
25	−16.941646 + 2.763071i	−16.941646 + 2.763071i
50	−24.288883 + 4.070337i	−24.288883 + 4.070337i
100	−34.664217 + 5.966789i	−34.664089 + 5.966881i
250	−55.230448 + 9.710819i	−55.230451 + 9.710823i
500	−113.589146 + 11.159176i	−113.589146 + 11.159176i
1000	−111.168228 + 19.879302i	−111.168232 + 19.879303i
2500	−176.178035 + 31.692081i	−176.178035 + 31.692081i
5000	−362.979226 + 36.296213i	−362.979211 + 36.296142i
10000	−352.923824 + 63.768959i	−352.923830 + 63.768960i

TABLE 8
Critical Reynolds numbers and critical odd eigenvalues.

$R^k_{c,0}$	λ^o_c	$R^k_{c,1}$	λ^o_c
8.4606	2.63126	6.2983	5.89598
9.1576	3.42632	6.8691	6.97998
25.6282	2.67267	17.1490	5.91199
26.3405	3.15830	18.0316	6.76558
50.8847	2.70372	31.6603	5.95620
51.3489	2.95271	32.6501	6.63307
84.1917	2.74161	49.3053	6.00915
		50.7706	6.51354
		71.4975	6.19109

TABLE 9
Critical Reynolds numbers and critical even eigenvalues.

$R^k_{c,0}$	λ^e_c
6.8846	4.23405
7.5805	5.25272
19.5660	4.29246
20.5465	5.04126
37.1756	4.36997
38.2524	4.87873
59.8274	4.37883
60.7830	4.81713
87.3756	4.38935

TABLE 10
Even complex eigenvalues for high Reynolds numbers.

R	λ^e	λ_0
1000	0.59650+0.29956i	1.60023+0.80362i
10000	0.43318+0.20964i	1.61472+0.78146i
100000	0.31358+0.15003i	1.62420+0.77709i
1000000	0.22651+0.10819i	1.63013+0.77865i

TABLE 11
Numerical versus asymptotic eigenvalues at high Reynolds numbers.

R	λ^e (numer)	λ^e (asympt)
1000	0.59650+0.29956i	0.59625+0.28725i
10000	0.43317+0.20964i	0.43317+0.20780i
50000	0.34571+0.16572i	0.34572+0.16552i
100000	0.31358+0.15003i	0.31358+0.15004i
500000	0.24986+0.11934i	0.24986+0.11950i
750000	0.23592+0.11269i	0.23592+0.11271i
1000000	0.22651+0.10819i	0.22651+0.10819i

5; 20; 50; 100; 500; 1,000; 5,000; 10,000; 20,000; 50,000; and 100,000. The program was run on the Vax 11/780 and it took about 15 minutes of CPU time.

As pointed out in [4], for large values of R the eigenvalues that have large modulus tend to be less accurate. Since the complex eigenvalues with negative real part have larger modulus than those with positive real part they may be a sensitive quantity in the computation. We computed the first branch of odd complex eigenvalues for $R \geqq 10$ with different tolerances (the tolerance tol is a parameter in PASVA3 that controls the growth of the estimated errors). We have found that the accuracy of the eigenvalues increased considerably while we decreased the tolerance, e.g., for $\text{tol} = 10^{-6}$ and 10^{-8} the eigenvalues agree up to six decimal places for Reynolds numbers in the range $10 \leqq R \leqq 50$ and they have at least three decimal places in common for $R > 50$, while for $\text{tol} = 10^{-8}$ and 10^{-10} the eigenvalues agree up to six decimal places for Reynolds numbers in the range $10 \leqq R \leqq 100{,}000$. We present the corresponding results in Table 7.

6. Eigenvalues at critical Reynolds numbers. As we have pointed out before, this method fails to compute eigenvalues at critical Reynolds numbers. In [4] Bramley and Dennis computed some eigenvalues that they suggested occur at critical Reynolds numbers, e.g., at $R = 6.3$ they computed the eigenvalue $\lambda \approx 6.0$, which corresponds to the first critical eigenvalues of the second odd branch (see Fig. 1.1(b)). We have found, according to our computation, that $R = 6.3$ is not exactly a critical Reynolds number but it is very close to it, i.e., we computed two eigenvalues for that R, one corresponding to the increasing solution with $\lambda = 6.01267$ (which, apparently, agrees with the value given by Bramley and Dennis) and the other corresponding to the decreasing solution, i.e., $\lambda = 5.85346$. A similar situation happens for a critical eigenvalue reported in [8] by Gillis and Brandt. They computed the eigenvalue $\lambda \approx 2.632$, presumably a critical one, at $R = 8.461$. We compute eigenvalues $\lambda = 2.62085$ and $\lambda = 2.63875$ at that Reynolds number. We believe that it is difficult to determine exactly a critical Reynolds number and its critical eigenvalue. Here we introduce a device, which may be useful to compute better approximations to critical Reynolds numbers and their eigenvalues. First we assume that the eigenvalues, near a critical Reynolds number, may be represented by the following formula:

$$\lambda = \lambda_c + \bar{a}\sqrt{R - R_c}, \tag{6.1}$$

where R_c is the critical Reynolds number, λ_c the critical eigenvalue, and $\bar{a}$ is a constant. For Reynolds numbers R_1, R_2, R_3, near R_c we can rewrite (6.1), after some algebraic manipulation, as

$$R_i = R_1 + A(\lambda_i - \lambda_1) + B(\lambda_i - \lambda_1)^2, \tag{6.2}$$

with A and B given by $A = (1/\bar{a})^2$ and $B = (1/\bar{a})^2(2\lambda_1 - 2\lambda_c)$, respectively, and $i = 2, 3$. Then, using the values of R_i, λ_i, we solved the system (6.2) in terms of A, B. We can compute λ_c by the formula

$$\lambda_c = \lambda_1 - \frac{A}{2B}, \tag{6.3}$$

while R_c is obtained using (6.1). This approach seems to work well, whenever the eigenvalues are close to λ_c. In Tables 8 and 9 we show all the critical eigenvalues, corresponding to Figs. 1.1(a) and 1.1(b), computed by this approach.

7. Asymptotic behavior for large R. It is interesting to see the behavior of these eigenvalues for high Reynolds numbers. By high Reynolds number we mean that the system (4.1) becomes a stiff system of ordinary differential equations. In [12], Wilson developed a theory based on a singular perturbation analysis to investigate these eigenvalues. He considered equations (2.5), corresponding to a channel flow problem in the stream function formulation. For high Reynolds numbers, Wilson found that the eigenvalues approaching zero downstream fell into two categories, one in which the eigenvalues approached zero as $O(R^{-1})$ and in the other as $O(R^{-1/7})$. From Table 5 we see that the eigenvalues behaving like $O(R^{-1})$ correspond to positive real decreasing eigenvalues, which according to Wilson may be written as $\lambda = \lambda_0 R^{-1} + O(R^{-3})$ with λ_0 being a constant. By neglecting smaller terms we have that $\lambda_0 \cong \lambda R$. Then it is possible to use the numerical data to compute λ_0. We did that for $R = 10^5$ and the λ_0's that we found agree with those given by Wilson (see [12, Table 3]).

The second type of eigenvalues approaching zero downstream, according to our computation, are the even complex eigenvalues (first branch). By assuming that these eigenvalues can be written as

$$\lambda = \lambda_0 R^{-1/7} + \cdots, \tag{7.1}$$

we have that for R large enough, neglecting smaller perturbations, we may compute the λ_0's, using the numerical data, by $\lambda_0 \approx \lambda R^{1/7}$. In Table 10 we present the corresponding results.

Note that λ_0 tends to a constant as R increases; it seems to be that R has to be quite large before we can neglect any smaller perturbation in the asymptotic expression for λ. Unfortunately, Wilson did not mention how to obtain the smaller order terms in (7.1). Using the numerical data we may attempt to compute the second lower-order term in (7.1) by assuming that λ can be expanded as

$$\lambda = bR^{-1/7} + cR^{-d} + \cdots, \tag{7.2}$$

with b, c, d constants to be determined. We have then for four consecutive eigenvalues $\lambda_1, \lambda_2, \lambda_3$, and λ_4, at Reynolds numbers of the form $R_k = 10R_{k-1}$, the following formula:

$$10^{-1/7}\alpha_2 - \alpha_3 = Re(c)10^{-d}(10^{-1/7} - 10^{-d})R_1^{-d}, \tag{7.3a}$$

and

$$10^{-1/7}\alpha_3 - \alpha_4 = Re(c)10^{-2d}(10^{-1/7} - 10^{-d})R_1^{-d}, \tag{7.3b}$$

where $\lambda_k = \alpha_k + i\beta_k$, i.e., we have that d is given by

$$d = \log\left(\frac{10^{-1/7}\alpha_2 - \alpha_3}{10^{-1/7}\alpha_3 - \alpha_4}\right) \Big/ \log 10, \tag{7.4}$$

while b can be determined, using again (7.2), by

$$b = \frac{10^{-d}\lambda_3 - \lambda_4}{(10^{-d} - 10^{-1/7})R_1^{-1/7}}, \tag{7.5}$$

and c is determined using (7.2) with λ replaced by λ_4.

We found the following values for b, c and d: $b = 1.6400 + 0.7813i$, $c = -0.1655 - 0.0435i$, and $d = 0.34676$. We used λ_k corresponding to $R_k = 10^3$, 10^4, 10^5, 10^6. Therefore, by substituting these values in (7.2), it is possible to obtain a better asymptotic prediction. Note that d is approximately $\frac{5}{14}$, i.e., the eigenvalues behave, approximately, as

$$\lambda = (1.64 + 0.7813i)R^{-1/7} + O(R^{-5/14}). \tag{7.6}$$

In Table 11 we show a comparison between the numerical results of the eigenvalue calculations and the asymptotic results using (7.2). The agreement is satisfactory; note that for the Reynolds numbers used to derive (7.2) the agreement is better.

8. Conclusion. The eigenvalues governing the rate of decay of perturbations of Poiseuille flow have been studied. The procedure to compute the eigenvalues is based on expressing the eigenvalue system as a two-point boundary value problem. The eigenvalues are computed one at a time jointly with the eigenfunctions, and the results presented are in agreement with previous computations [4] and [12]. For high Reynolds numbers we show that the first branch of even eigenvalues has an asymptotic representation of the form (7.5). We think that our procedure is simpler, more direct and more efficient than previous methods, and it can be used with relative ease on a computer. Due to the high accuracy obtained on using this method, we strongly recommend it for computations involving eigenvalue systems of ordinary differential equations.

Acknowledgments. I would like to thank Professor John C. Strikwerda for his valuable advice and for the encouragement I received during the preparation of this research. I also thank the referees and the editor for their valuable suggestions.

REFERENCES

[1] G. A. ACHE, *Incompressible viscous steady flow in channel and pipes*, Ph.D. thesis, University of Wisconsin, Madison, WI, 1985.

[2] F. BOWMAN, *Introduction to Bessel Functions*, Dover, New York, 1958.

[3] J. S. BRAMLEY, *Note on the calculation of eigenvalues for stationary perturbation of Poiseuille flow*, J. Comput. Phys., 53 (1984), pp. 524–529.

[4] J. S. BRAMLEY AND S. C. R. DENNIS, *The calculation of eigenvalues for the stationary perturbation of Poiseuille flow*, J. Comput. Phys., 47 (1982), pp. 179–198.

[5] ———, *The calculation of eigenvalues for the stationary perturbation of Poiseuille flow using initial value methods*, J. Math. Anal. Appl., 101 (1984), pp. 30–38.

[6] J. S. BRAMLEY, *The calculation of eigenvalues for the stationary perturbation of symmetrical pipe Poiseuille flow*, J. Comput. Phys., 64 (1986), pp. 258–265.

[7] V. T. BUCHWALD, *Eigenfunctions of plane elastostatics*, I. *The strip*, Proc. Roy. Soc. London Ser. A, 277 (1964), pp. 385–400.

[8] J. GILLIS AND A. BRANDT, *The numerical integration of the equation of motions of a viscous fluid*, Report AF EOAR, SR-1, Weizmann Institute, Revohot, Israel, 1964, pp. 63–73.

[9] S. A. ORSZAG, *Accurate solution to the Orr–Sommerfeld stability equation*, J. Fluid Mech., 50 (1971), pp. 689–703.

[10] V. PEREYRA, PASVA3 *and adaptive finite difference Fortran program for first order nonlinear ordinary boundary value problems*, in Code for Boundary Value Differential Equations, B. Childs, M. Scott, J. W. Daniel, F. Denman, and P. Nelson, eds., Lecture Notes in Computer Science, 76, Springer-Verlag, New York, 1979, pp. 67–88.

[11] D. A. SPENCE, *A class of biharmonic end-strip problems arising in elasticity and Stokes flow*, IMA J. Appl. Math., 30 (1983), pp. 107–139.

[12] S. WILSON, *The development of Poiseuille flow*, J. Fluid Mech., 38 (1968), pp. 793–806.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 6, pp. 1113-1134, November 1989
© 1989 Society for Industrial and Applied Mathematics
004

DISTRIBUTED ORTHOGONAL FACTORIZATION: GIVENS AND HOUSEHOLDER ALGORITHMS*

ALEX POTHEN†‡ AND PADMA RAGHAVAN†

Abstract. Several algorithms for orthogonal factorization on distributed memory multiprocessors are designed and implemented. Two of the algorithms employ Householder transformations, a third is based on Givens rotations, and a fourth hybrid algorithm uses Householder transformations and Givens rotations in different phases.

The arithmetic and communication complexities of the algorithms are analyzed. The analyses show that the sequential arithmetic terms play a more important role than the communication terms in determining the running times and efficiencies of these algorithms. The hybrid algorithm is the fastest algorithm overall, since its arithmetic cost is lower than the Householder algorithms and its communication cost does not increase with the column length of the matrix. The observed execution times of the implementations on an iPSC-286 agree quite well with the complexity analyses. It is also shown that the efficiencies can be approximated using only the arithmetic costs of the algorithms.

Key words. Givens rotations, Householder transformations, distributed algorithms, hypercube multiprocessors, least squares problems, orthogonal factorization, parallel computations

AMS(MOS) subject classifications. 65F20, 65F25

1. Introduction. Orthogonal factorization is a fundamental matrix decomposition in numerical linear algebra, and it provides numerically stable methods for computing coordinate transformations, projections, and solutions to least squares and optimization problems. Since sequential algorithms for computing this factorization have a high ($\Theta(MN^2)$) arithmetic complexity, the development of parallel algorithms is of considerable interest. Several parallel orthogonal factorization algorithms have been designed for various machine architectures. We cite just a few: Sameh (1985) and Luk (1986) for systolic arrays; Bischof and Van Loan (1987) for vector pipelined machines; Dongarra, Sameh, and Sorensen (1986) and Lord, Kowalik, and Kumar (1983) for shared memory machines; Chamberlain and Powell (1986) and Pothen, Jha, and Vemulapati (1987) for distributed memory machines.

In this paper we design several medium grained parallel algorithms for computing this factorization on distributed memory multiprocessors. Two of the algorithms employ Householder transformations, a third uses Givens rotations, and a fourth is a hybrid algorithm that employs both Householder transformations and Givens rotations in different phases in the algorithm.

Previously reported distributed orthogonalization algorithms in Chamberlain and Powell (1986) and Pothen, Jha, and Vemulapati (1987) have been based on Givens rotations. Orthogonalization algorithms based on Givens rotations have twice the number of arithmetic operations as Householder algorithms, but the latter require twice as many memory references as Givens algorithms. On traditional sequential machines, the cost of a floating point operation dominates the cost of a memory reference, and Householder algorithms are preferred for factoring dense matrices. Dongarra, Sameh, and Sorensen (1986) show that on the Denelcor HEP, a shared memory multiprocessor, memory access times dominated flop times, and hence Givens algorithms were twice as fast as Householder algorithms.

* Received by the editors August 24, 1987; accepted for publication (in revised form) November 23, 1988.
† Computer Science Department, Pennsylvania State University, University Park, Pennsylvania 16802.
‡ The work of this author was supported by the National Science Foundation under grant CCR-8701723.

Our goal in this paper is to study the requirements and properties of the Givens and Householder algorithms we design. In particular, we wish to elucidate the features that influence the running times and efficiencies of these algorithms. Such an understanding is necessary for the proper choice of an orthogonalization algorithm in a particular context, such as the design of the numerical phase in sparse orthogonal factorization, or in optimization algorithms. Note that an application might impose constraints on the choice of an orthogonalization algorithm. For instance, the matrix may have to be mapped by rows for ease of updating the factorization as the matrix changes by a row or a column in a quadratic programming problem. Thus we believe that a comparative study of different algorithms is more beneficial than the description of one algorithm that is faster than others in a class of algorithms.

We analyze the arithmetic and communication complexity of each of the algorithms, and discuss their expected performances over a range of matrix sizes. The predicted behavior is compared with execution times from our implementation on an Intel iPSC-286 hypercube multiprocessor. We also derive expressions for the efficiencies of the algorithms, and provide comparisons with observed efficiencies.

1.1. The hypercube multiprocessor. We assume some familiarity with a hypercube multiprocessor; the proceedings edited by Heath ((1986), (1987)) contain two collections of articles on hypercube machines and algorithms. We provide a brief description of the machine for completeness. The n-dimensional hypercube is an MIMD machine with 2^n identical loosely coupled processors, each with its private memory. The processors are assigned identification numbers (id's) from 0 to $2^n - 1$. A processor is linked by bidirectional communication links to n neighbors; these neighbor processors differ from the binary representation of its id in precisely one bit. There is no shared memory, and processors share information by sending and receiving messages.

The operating system provides two primitives for message passing: *send* and *recv*. A processor can send a message to any other processor by invoking *send* with the id of the destination processor. When it invokes a *recv*, a processor receives the message and the id of the sending processor. A type variable is used to distinguish between messages a processor may receive. For more details the reader should consult Heath (1986), (1987), or a manual from a hypercube manufacturer.

We now mention some details specific to an Intel iPSC-286 hypercube since these features will influence the analyses of our algorithms. A processor of the iPSC-286 consists of an 80286 chip, an attached 80287 floating point processor, and 512 KB of RAM. The buffer space needed to store messages is included in the RAM. Each processor has several communication coprocessors, one for regulating message traffic across each communication link. The communication coprocessors on a processor have direct access to the processor memory, but only one of them can access the memory at a given time.

A processor communicates to a nonneighboring processor by routing the message along a path formed by adjacent processors. The number of hops in the path is the number of links a message must cross to reach its destination. Messages are routed in the *store and forward* mode, i.e., an intermediate processor stores the entire message in its memory upon receipt, and then forwards it to another neighbor. The time needed by a message to reach its destination is thus the product of the number of hops from the sender and the time for a hop.

The bottleneck for direct memory access at a processor has two implications. First, when several processes on a processor have to communicate at a given time, they have to access processor memory sequentially, thus slowing the messages considerably. This

is exacerbated by the store and forward mode of communication, since messages also could be slowed down at intermediate processors. Second, even though communication links are bidirectional, a processor cannot effectively send and receive at the same time.

Therefore, it behooves the designer of a distributed algorithm to map the problem on the multiprocessor such that processors required to communicate in the algorithm are as close as possible. In all but one of our algorithms, only neighboring processors need to communicate, so all messages are single hop. In the other, a broadcast is necessary, but broadcast can be efficiently implemented on a hypercube topology by embedding a spanning tree in the hypercube.

1.2. Outline. This paper has five additional sections. In § 2 we provide detailed descriptions of the Householder and hybrid algorithms and briefly describe the Givens algorithms. Analyses of the arithmetic and communication complexity of these algorithms and a discussion of their expected execution time requirements are provided in § 3. Discussions of our computational experiments and results are included in § 4. In § 5, we compare the observed efficiencies of our distributed algorithms with values from analytic expressions. The final section contains remarks about enhancements, other algorithms, and sparse orthogonal factorization.

Now we discuss some matters of notation. We write the orthogonal factorization as

$$QA = R,$$

where Q is orthogonal and R is upper trapezoidal. We assume that the matrix A has full column rank and let M denote the number of rows and N the number of columns. The number of processors in the multiprocessor will be denoted by P, and we assume that $P \ll M, N$. We use τ to denote the time needed for a flop, and assume that the time needed to send a message of m words between neighboring processors is $m\alpha + \beta$, where α is the word-transfer time, and β is the start-up time. This ignores the fact that the iPSC uses message packets of size 1024 bytes. In the communication complexity of an algorithm, the term involving α will be called the *transfer term*, and the term involving β will be called the *start-up term*. All logarithms in this paper are computed with base two.

2. Distributed algorithms. In this section, we describe in detail a column Householder algorithm and a hybrid algorithm, and briefly mention the Givens algorithms. In the column Householder algorithms, the matrix is partitioned by columns, and in the hybrid and the greedy Givens algorithms, the matrix is partitioned by rows. In all cases, we employ wrap mapping to balance the arithmetic work (to first-order terms) on all processors.

In algorithms that require ring communication, a ring is embedded in the hypercube by means of a binary reflected Gray code. Processors in the ring are numbered from 0 to $P-1$, according to their order in the ring, beginning with the processor whose id is zero in the hypercube. In wrap mapping, columns or rows are then wrapped on the ring of processors.

2.1. The pipelined Householder algorithm (p_house). We now design a distributed column Householder algorithm, p_house, that employs ring communication to compute the factorization. The b_house algorithm is similar except that it uses broadcast communication.

Householder orthogonalization can be described in terms of the computation of a sequence of matrices $\{A_j : j = 0, \cdots, N-1\}$, where $A_0 = A$ and $A_{N-1} = R$. Prior to

the jth step of the factorization, the first $(j-1)$ columns of R have been computed, and the intermediate matrix A_j has the form:

$$A_j = \begin{bmatrix} R_{j-1} & r_j & B_{N-j} \\ 0 & c_j & D_{N-j} \end{bmatrix}.$$

At this step a Householder vector u_j and a scalar π_j are computed from the column c_j; these define a Householder reflector $U_j = I - (u_j u_j^t / \pi_j)$, and the vector $U_j c_j$ is a multiple of the first coordinate vector. The next matrix in the sequence A_{j+1} is obtained by replacing c_j and D_{N-j} by $U_j c_j$ and $U_j D_{N-j}$, respectively. Good descriptions of the sequential algorithm may be found in Golub and Van Loan (1983) and Stewart (1973).

We refer to the computation of the Householder vector u_j and the scalar π_j from the column c_j as the *cdiv* (j) operation. The modification of a column k of D_{N-j} by the Householder vector u_j will be called a *cmod* (k, j) operation. These terms have been used by George, Heath, and Liu (1986), and by Geist and Heath (1986) to describe analogous tasks in Cholesky factorization and Gaussian elimination.

The columns of the matrix are wrapped on a ring of P processors. Each processor holds an M by $\lceil N/P \rceil$ submatrix. At the jth step of the algorithm, a processor has completed the transformation of all of its columns numbered lower than j. When the processor holding column j completes the modification of column j by all previous columns, it can begin the execution of the jth step. Then this processor performs *cdiv* (j), sends the Householder vector u_j to its successor on the ring, and modifies its higher numbered columns by u_j. The latter processor receives the Householder vector, sends it to its successor, and modifies its columns numbered higher than j. Thus the Householder vector u_j is pipelined around the ring at the jth step.

The pipelined column Householder algorithm has some features in common with the pipelined column Cholesky algorithm of Geist and Heath (1986). Both algorithms wrap columns on the processors in a ring, use ring communication, and are described in terms of the *cdiv* and *cmod* operations (these subtasks are different in the two algorithms). However, there is an additional constraint in Householder orthogonalization. The different *cmod* (k, j) operations on a column j do not commute, and must be performed in the order $k = 1, \cdots, j-1$. The reason is that the multiplier of column j in the *cmod* (k, j) operation is proportional to the inner product of columns k and j, and hence depends on column j.

To enhance parallelism in the distributed algorithm, we have designed a function update (j) that performs *cmod* operations with a Householder vector j. Let *first* denote the lowest numbered column, among the columns that need to be updated, on a processor. The function update modifies *first* with the Householder vector j and checks to see if all *cmod* operations on this column have been completed.

If so, it computes the Householder vector corresponding to *first* and communicates the vector to its successor in the ring. Columns numbered higher than *first* are then modified by Householder vectors j and *first*. If all *cmod* operations on *first* have not been completed yet, the processor performs the rest of the *cmod* operations on higher numbered columns with the Householder vector j.

Let *ncols* denote the number of columns on a processor, *ndivs* the number of *cdivs* that a processor has performed, and *nextcol* the next Householder vector a processor has to send or receive. The distributed algorithm is described in Fig. 1. In Fig. 1 and the rest of this section, subsequent to the *cdiv* operation on a column j, we shall refer to the Householder vector u_j as column j.

In the broadcast version of the Householder algorithm, the Householder vector u_j is broadcast to all the processors. The broadcast is done along an embedded spanning

```
begin
        nextcol := 0; {the next Householder vector I need to recv}
        ndivs := 0; {count of cdivs that I have performed}
        if (column 0 is one of my columns)
                then
                cdiv (0); ndivs := ndivs + 1;
                send column 0 to successor; nextcol := nextcol + 1;
                update(0);
        fi
        while (ndivs < ncols) do
                recv nextcol from predecessor;
                if (I am not the last processor to receive nextcol)
                        then send nextcol to successor;
                fi
                nextcol := nextcol + 1;
                update(nextcol - 1); {update with the vector just received}
        end while
end
function update (j)
{perform cmods with the jth Householder vector}
begin
                {the first column is modified separately}
                let first be my lowest numbered column to be updated;
                cmod (first, j);
                if (first has been updated by all previous columns)
                        then
                                cdiv (first); ndivs := ndivs + 1;
                                send first to successor; nextcol := nextcol + 1;
                                for each of my columns k, k > first do
                                        cmod (k, j); cmod (k, first);
                                rof
                        else
                                for each of my columns k, k > first do
                                        cmod (k, j);
                fi
end
```

FIG. 1. *The pipelined Householder algorithm.*

tree of height log P rooted at the broadcasting processor. The *cmod* operations are performed exactly as in the pipelined algorithm. Thus the only difference between the two algorithms is in the method employed to communicate the Householder vectors. We omit a more detailed description of the broadcast algorithm.

The broadcast scheme has the important advantage that it delivers the Householder vector to all processors in the hypercube in time proportional to log P. However, it has two potential disadvantages. First, when a broadcast is used, the processor holding column $j+1$ (and hence in greatest need of column j) may be one of the last processors to receive column j. Second, even after receiving column j, this processor may have to participate in forwarding column j to other processors before it can start modifying its columns.

2.2. The hybrid algorithm (hybrid). In the distributed column Householder algorithm processors communicate columns. On the other hand, in the Givens algorithms which are row oriented, processors would communicate rows. When the

matrix A is highly overdetermined, communication costs in Givens algorithms could become substantially lower than the Householder algorithms. This observation leads us to the design of an algorithm that has the low arithmetic complexity of a Householder algorithm and also the low communication complexity of a Givens algorithm. Later, when we present the complexity analyses of our algorithms, it will be seen that this algorithm also has a lower sequential arithmetic cost than the column Householder algorithm. We call this algorithm *the hybrid algorithm* since it employs both Householder transformations and Givens rotations.

The matrix is partitioned by rows numbered from 0 to $M-1$, and a ring of P processors numbered from 0 to $P-1$ is employed. The first N rows are wrapped on the processors such that row 0 is on processor 0, row 1 on processor 1, etc. The rest of the $M-N$ rows can be equally distributed among the processors in any manner. The portion of a column on a processor will be called a *subcolumn.* Each processor has a submatrix consisting of about $\lceil M/P \rceil$ rows and N subcolumns.

The matrix is transformed by columns, beginning from left to right. Each column is transformed in two phases: the internal reflections phase (IP) and a recursive elimination phase (RP). In the internal reflections phase, elements of a subcolumn are zeroed by Householder transformations. In the recursive elimination phase, processors communicate with each other to annihilate the remaining subdiagonal elements (one on each processor), by means of Givens rotations. The distributed hybrid algorithm is shown in Fig. 2.

Now let us consider the transformation of the jth column in detail. Prior to this step, columns $0, \cdots, j-1$ have been completely transformed, and the processing of rows $0, \cdots, j-1$ have also been completed. Define the *top row* of a processor at this step to be its smallest row numbered greater than or equal to j. The *active submatrix*

```
for subcol j = 0 to N − 1 do
    1. {internal reflection phase}
       cdiv (j);
       for subcol k = j + 1 to N do
           cmod (k, j);
       rof
    2. {recursive elimination phase}
       diag_node := processor holding diagonal row j;
       zeroed := false; stage := 1;
       while (not zeroed & stage ≤ log P) do
           {perform a stage of RP}
           i := stage;
           partner := processor differing from self
                      in ith most significant bit;
           send top row to partner;
           recv row from partner;
           if (self != diag_node in ith most significant bit)
                      then zeroed := true; {my element is zeroed}
           fi
           update top row;
           stage := stage + 1;
       end while;
       if (self = diag_node) then top := top + 1; fi
rof
```

FIG. 2. *The hybrid algorithm.*

on a processor is the submatrix that remains to be processed at this step and includes the top row and rows numbered higher, and subcolumns j through N. Hereafter a row or a subcolumn will refer to a row or subcolumn of the active submatrix.

During IP, each processor performs computations on the submatrix it holds, and there is no communication in this phase. It computes a Householder transformation with its jth subcolumn, to zero all elements of this subcolumn below its top row. We refer to the computation of a Householder vector from subcolumn j as the *cdiv* (j) operation. We call the modification of subcolumn k by the Householder vector j as the *cmod* (k, j) operation. When this phase has been completed, the jth subcolumn on a processor is zero below its top row.

In RP, processors cooperate pairwise to complete the transformation. Let diag_node denote the processor that holds the diagonal row j. This phase is divided into log P stages. In the ith stage, processors that differ in the ith most significant bit pair up to exchange their top rows. The processor that differs from the diag_node in the ith most significant bit zeros its element, while the processor that has the same bit as the diag_node updates its element. After its element has been zeroed, a processor takes no further part in RP on column j, and can proceed with the internal reflections phase of column $j+1$. The processor whose element is updated continues with the next stage. After log P such stages, the only surviving nonzero in the jth column will be in the diagonal row j, which is held by diag_node.

Processor pairing in the various stages of RP is illustrated in Fig. 3 for a ring of eight processors. The binary reflected Gray code is used to embed the ring on the hypercube. The first column in the figure shows the processor id's and processors that pair up at a stage are joined by an oval line. In the example, diag_node, the processor holding the diagonal row is processor 0. Since all processors that communicate with each other are neighbors in the hypercube, RP can be efficiently implemented on a hypercube architecture.

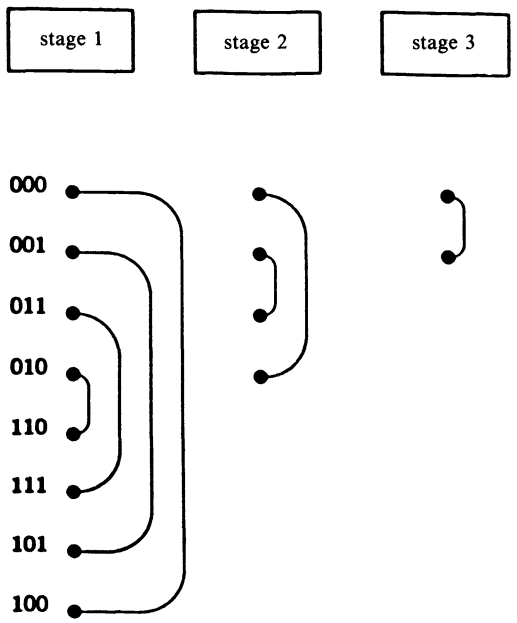


FIG. 3. *Processor pairing in recursive elimination.*

In the implementation of RP that we have described, all processors need to know which processor holds the diagonal row corresponding to a column. In the wrap mapping employed by us, this can be computed from the column number j, since diag_node is the processor numbered $j \bmod P$ in the ring. If some other mapping scheme is used, this information either must be computable by all processors, or must be explicitly communicated by the host. At the completion of this phase, all subdiagonal nonzeros in the jth column will be annihilated.

2.3. The Givens algorithms. In Pothen, Jha, and Vemulapati (1987), three distributed Givens algorithms were designed. We have implemented the sgs and ggs algorithms from that paper, but only the results of the ggs algorithm will be reported here.

The sgs algorithm is based on the standard Givens sequence previously studied by Gentleman (1975) and Sameh and Kuck (1978). This algorithm has a higher communication cost than other distributed Givens algorithms and is not an attractive algorithm in this context. In Pothen and Raghavan (1987), we have implemented this algorithm and verified that this is, indeed, the slowest of the algorithms.

The greedy Givens sequence arises naturally when we consider a distributed Givens algorithm that is efficient with respect to communication costs. The ggs algorithm is quite similar to the hybrid algorithm and differs from it only in the internal phase. Indeed, we designed the ggs algorithm first, and then designed the hybrid algorithm in attempting to reduce the arithmetic cost (the parallel arithmetic term) of ggs, while preserving its low sequential arithmetic term and communication costs.

Thus the matrix is mapped by rows, and the rows are wrapped on a ring of processors. In the internal phase, each processor performs Givens rotations between its rows until the only nonzero in a subcolumn is in the top row. Processors then take part in a recursive elimination phase to zero the nonzeros in the top rows.

Modi and Clarke (1984) and Cosnard, Muller, and Robert (1986) have studied a greedy algorithm to compute the orthogonal factorization by Givens rotations. This algorithm is inspired by the standard Givens sequence and, as in the latter algorithm, the rotations are organized in groups involving disjoint rows. In each group, as many rotations as possible that involve disjoint rows are included. For instance, the first group includes $\lfloor M/2 \rfloor$ rotations that introduce that many zeros into the first column. Cosnard, Muller, and Robert (1986) showed that this algorithm minimizes (asymptotically) the number of groups over other Givens sequences. However, on a distributed multiprocessor with a static initial assignment of rows, processors, in general, will have unequal numbers of rotations to perform in each group. Thus the time required to compute a group of rotations cannot be treated as a constant. Further, in this algorithm, different elements in a row have to be zeroed by rotations with different rows to maximize the number of rotations in each group, and the choice of the latter row depends on the group. Hence the communication requirements of this algorithm will be much less uniform than the ggs algorithm.

A third algorithm in Pothen, Jha, and Vemulapati (1987), the recursive fine partition algorithm, partitioned the matrix into square submatrices, and wrapped these submatrices on a rectangular grid of processors. In each step of the algorithm, the factorization of a block of columns is completed. Each processor introduces zeros into a subcolumn block it holds to make it upper trapezoidal; then the transformation of a column block is completed by a recursive elimination phase, in which the residual upper trapezoidal submatrices are eliminated. Our complexity analyses indicated that this algorithm would be competitive with the ggs algorithm when the number of

processors in the multiprocessor are large. Since the number of processors in the hypercube available to us was only 16, we did not implement this algorithm.

3. Arithmetic and communication complexity. We proceed to analyze the arithmetic and communication complexities of the column Householder and hybrid algorithms. The arithmetic and communication complexities of all the algorithms are tabulated in Table 1. (The first two entries correspond to sequential algorithms.)

TABLE 1
Arithmetic (A) and communication (C) complexities.

Algorithm		Complexity
Givens	A	$2N^2(M-N/3)\tau+2N(3M-2N)\tau$
Householder	A	$N^2(M-N/3)\tau+N(M+N/2)\tau$
sgs	A	$\frac{2N^2}{P}(M-N/3)\tau+N(3M+N/2)\tau$
	C	$\frac{2N^2}{P}(M-N/3)\alpha+N(3M+N/2)\alpha+\frac{2N}{P}(2M-N)\beta+2(M+N)\beta$
ggs	A	$\frac{2N^2}{P}(M-N/3)\tau+N^2\log P\tau$
	C	$N^2\log P\alpha+2N\log P\beta$
p_house	A	$\frac{N^2}{P}(M-N/3)\tau+3N(M-N/2)\tau$
	C	$N(M-N/2)\alpha+MP\alpha+(N+P)\beta$
b_house	A	$\frac{N^2}{P}(M-N/3)\tau+3N(M-N/2)\tau$
	C	$N(M-N/2)\log P\alpha+N\log P\beta$
hybrid	A	$\frac{N^2}{P}(M-N/3)\tau+N^2\log P\tau+\frac{3}{2}N^2\tau+\frac{MN}{P}\tau$
	C	$N^2\log P\alpha+2N\log P\beta$

In analyzing the arithmetic complexity of these algorithms, we pay careful attention to the first- and second-order arithmetic terms. It will turn out that the second-order arithmetic terms contribute more significantly to running times than communication terms. Thus we need to take greater pains to compute these terms accurately.

We denote the arithmetic complexity by t_A, and the communication complexity by t_C. Recall from the discussion in the introduction that a processor cannot send and receive on a communication link at the same time. Hence, the time needed by a pair of neighboring processors to exchange two rows of m elements each is $2(m\alpha+\beta)$.

Since we have described the Householder and hybrid algorithms in terms of the *cdiv* and *cmod* operations, it will be convenient to compute the cost of these operations. Recall that columns are numbered from 0 to $N-1$, processors from 0 to $P-1$, and that in the jth step of the factorization, we compute the jth Householder vector and the *cmods* with this vector. For ease of notation, we write $\mu \equiv M-j$, and $\nu \equiv N-j$.

The *cdiv* (j) operation involves the computation of the length of column j. A careful implementation that avoids unnecessary underflows and overflows costs $2\mu+2$ flops, and a square root. We shall treat the latter operation as equivalent to a flop. For a column $k>j$, *cmod* (k, j) involves the subtraction of a multiple of column j from column k. The multiplier is the inner product of the two columns divided by π_j. Hence, the *cmod* operation costs $2\mu+1$ flops.

3.1. The p_house algorithm. Assume for simplicity, that $N = kP$, for some integer k. Since columns are wrapped on the processors, the last column is on the last processor in the ring (numbered $P-1$). When the last processor computes the *cdiv* operation on the last column, the algorithm terminates. Hence, the time complexity of the distributed algorithm is the time taken by the last processor.

The last processor begins computation when it receives the first Householder vector. This initial delay includes the time taken by the first processor to compute this vector and the time it takes to reach the last processor after crossing $P-1$ edges of the ring. This delay, δ, is

$$\delta = (2M+3)\tau + (P-1)(M\alpha+\beta).$$

To compute the arithmetic complexity of the algorithm correctly, we need an accurate count of the number of columns the last processor updates in a given step. To perform such an analysis, it will be convenient to group the steps of the algorithm into *rounds*. Within each step in a round, the last processor modifies the same number of columns.

Thus round 0 consists of steps 0 to $P-2$, and in this round the last processor modifies k columns. It performs $k-1$ *cmods* in round 1, which consists of steps $P-1$ to $2P-2$. The ith round consists of steps $iP-1$ to $(i+1)P-2$, and this processor modifies $k-i$ columns.

Let us now compute the arithmetic work associated with the *cmods* in the ith round. Since *cmod* (j) costs $2\mu+1$ flops, and there are $k-i$ *cmods* in this round, this cost is

$$(k-i)\sum_{j=iP-1}^{(i+1)P-2}(2\mu+1)\tau = (2MP-2P(P-1)i-(P^2-4P-2))(k-i)\tau.$$

The total cost of the *cmods* performed by the last processor, t'_A, is the above quantity summed over rounds 0 to $k-1$. Thus we obtain

$$t'_A = \frac{N^2}{P}(M-N/3)\tau + N(M-N/2)\tau + \Theta(NP).$$

The computation of *cdiv* (j) introduces a delay of $2\mu+3$ flops in the jth step, since only one processor performs this computation, and all other processors have to wait to receive it. Hence, the total delay introduced by the *cdivs* is

$$D = 2N(M-N/2)\tau,$$

and the arithmetic complexity of the algorithm is

$$t_A = \frac{N^2}{P}(M-N/3)\tau + 3N(M-N/2)\tau + \Theta(NP).$$

In analyzing communication complexity, we can dispense with the concept of rounds and instead work with steps. At the jth step, the last processor receives the jth column, sends the column to its successor, and then modifies its columns. Thus the communication in this step costs $\mu\alpha+\beta$. The total communication cost is obtained by adding the initial delay to the cost over all steps. We obtain

$$t_C = N(M-N/2)\alpha + MP\alpha + (N+P)\beta.$$

The arithmetic complexity of the b_house algorithm is equal to that of the p_house algorithm. The communication complexity of the b_house algorithm differs from that of p_house owing to the use of broadcast communication.

3.2. The hybrid algorithm. Each processor has a submatrix of approximately $\lceil M/P \rceil$ rows and N subcolumns. The arithmetic cost of this algorithm is the cost of the Householder transformation in the internal phase and the cost of Givens rotations in the recursive elimination phase summed over all columns.

We assume again that $N = kP$, for some integer k. The algorithm terminates when the recursive elimination phase is completed on the last column, and hence the time complexity of the algorithm is the time taken by the diagonal processor for the last column. By the mapping of rows this is the last processor on the ring.

Consider the jth step of the internal phase. The jth subcolumn has length at most $\lceil \mu/P \rceil$, and *cdiv* (j) costs $2\lceil \mu/P \rceil + 3$ flops. Since there are $(\nu - 1)$ subcolumns to be modified, the c mods cost $(2\lceil \mu/P \rceil + 1)(\nu - 1)$ flops.

Summing over all the steps, the total cost of *cmods* in the internal phase is

$$\frac{N^2}{P}(M - N/3)\tau + \frac{3}{2}N^2\tau - \frac{MN}{P}\tau + \frac{N^2}{P}\tau,$$

and the cost of the *cdivs* is

$$\frac{2MN}{P}\tau - \frac{N^2}{P}\tau.$$

Computing the cost of the recursive elimination phase requires careful consideration, since a processor participates in a differing number of phases in RP, depending on the column number. The following two observations help in computing the cost of this phase correctly. First, each time a processor becomes the diagonal processor for a column j, it cannot finish RP before all other processors have completed this phase. This is because the diagonal row j, at completion of this phase, is a linear combination of the top rows on all processors. Thus even though it might have been ahead of the other processors (since it had less work to do in the recursive phase on previous columns), when RP on this column is completed, the diagonal processor cannot be ahead of the other processors. Second, the diagonal processor takes $\log P$ stages to complete RP on a column, and takes time $2\nu \log P\tau$.

To compute the cost of RP, we organize the columns into rounds, where a round ends when the last processor on the ring becomes the diagonal processor. The ith round includes columns $(i-1)P$ to $iP-1$. Therefore, the time required by the last processor to complete the ith round of RP is

$$\sum_{j=(i-1)P}^{iP-1} 2\nu \log P\tau.$$

Summing this cost over all the rounds, we get the cost of the recursive elimination phase to be

$$\sum_{i=1}^{k} \sum_{j=(i-1)P}^{iP-1} 2\nu \log P\tau = \sum_{j=0}^{kP-1} 2\nu \log P\tau = N^2 \log P\tau.$$

The total arithmetic cost of the hybrid algorithm is then

$$t_A = \frac{N^2}{P}(M - N/3)\tau + N^2 \log P\tau + \frac{3}{2}N^2\tau + \frac{MN}{P}\tau.$$

There is no communication in the internal phase; the communication cost of the jth step of the recursive elimination is $2(\nu\alpha + \beta)\log P$, as a row of size ν elements is

communicated in each of the $\log P$ stages. Summed over all steps, this yields a communication cost of

$$t_C = N^2 \log P\alpha + 2N \log P\beta.$$

3.3. Givens algorithms. The arithmetic and communication complexities of the ggs and sgs algorithms are included in Table 1. These complexities have been computed in Pothen and Raghavan (1987).

The greedy Givens has the same communication complexity as the hybrid algorithm but differs in its arithmetic complexity, since Givens rotations are used in the internal phase.

It turns out that it is fairly tedious to estimate the arithmetic complexity of the sgs algorithm. Conceptually, the computations in this algorithm are organized into $M+N-2$ groups of Givens rotations. We bound the time required for a group of rotations and then multiply the resulting time by the total number of groups. The time required per group depends on the number of rotations in the group. The number of rotations in each group can be accurately computed by partitioning the algorithm into three phases: in the *start phase*, the number of rotations in successive groups increases from 1 to N; in the *central phase*, each group has N rotations, one in each column; and in the *end phase*, the number of rotations in successive groups decreases from N to 1.

3.4. Discussion. The arithmetic and communication complexities of the Householder, hybrid, and ggs algorithms are listed in Table 1.

We consider the arithmetic complexity of each algorithm as being made up of two terms: a *parallel term*, that has a $1/P$ dependence, and a lower order *sequential term*, that does not have this dependence. The former term represents the part of the arithmetic work done in parallel by each processor, and the latter results from the inherently sequential arithmetic work and load imbalances in the algorithm.

In the hybrid and ggs algorithms, the parallel term represents the cost of the internal phase, and the sequential term is the cost of the recursive elimination phase. In the Householder algorithms, the former term is the cost of the *cmod* operations, and the latter term is comprised of two parts. Two-thirds of it is the cost of the sequential *cdivs*, and one-third of it comes from the slight inequality in load balance when the matrix is mapped by columns.

To understand the relative importance of the parallel and sequential arithmetic terms and the communication terms for the range of matrix sizes we consider, we need values for the parameters α, β, and τ. Reasonable values of these parameters for the iPSC-286 multiprocessor are tabulated in Table 2. We have used three different values of τ: one for the Givens algorithms, another for the Householder algorithms, and a third for the hybrid algorithm. This is because address arithmetic overheads are included

TABLE 2
iPSC-286 *parameters.*

Parameter	Value
α	16 μs
β	900 μs
τ_{ggs}	77.7 μs
τ_{hyb}	70.3 μs
$\tau_{\text{Householder}}$	90.6 μs

in the value of a flop, and these overheads are different for the three classes of algorithms. The values for τ were obtained from equivalent sequential algorithms (implemented on one processor) that had the same address arithmetic overheads as the distributed algorithms, but without the communication primitives.

The exact values of the parameters are not important for the analysis in this section, but their relative magnitudes are. The terms τ and α are of similar size (τ is a factor of four to six greater), but β is much larger (a factor of 50) than α. We will report our results on matrices with values of N in the hundreds, and values of M ranging from hundreds to thousands. In Fig. 4(a), using complexities from Table 1 and parameters from Table 3, we plot the arithmetic times of the ggs, Householder, and hybrid algorithms for a matrix with 100 columns. The communication times of these algorithms are plotted in Fig. 4(b).

Despite β being much greater than α, the transfer term is the major contributor to the communication cost of each algorithm, since its complexity is an order of magnitude higher than that of the start-up term. Further, the arithmetic complexity of each algorithm is an order of magnitude greater than communication complexity. Hence, for these algorithms, arithmetic costs contribute dominantly to the running times, and communication costs are less important. This is clear from a comparison of the scales of the ordinates in Figs. 4(a) and 4(b).

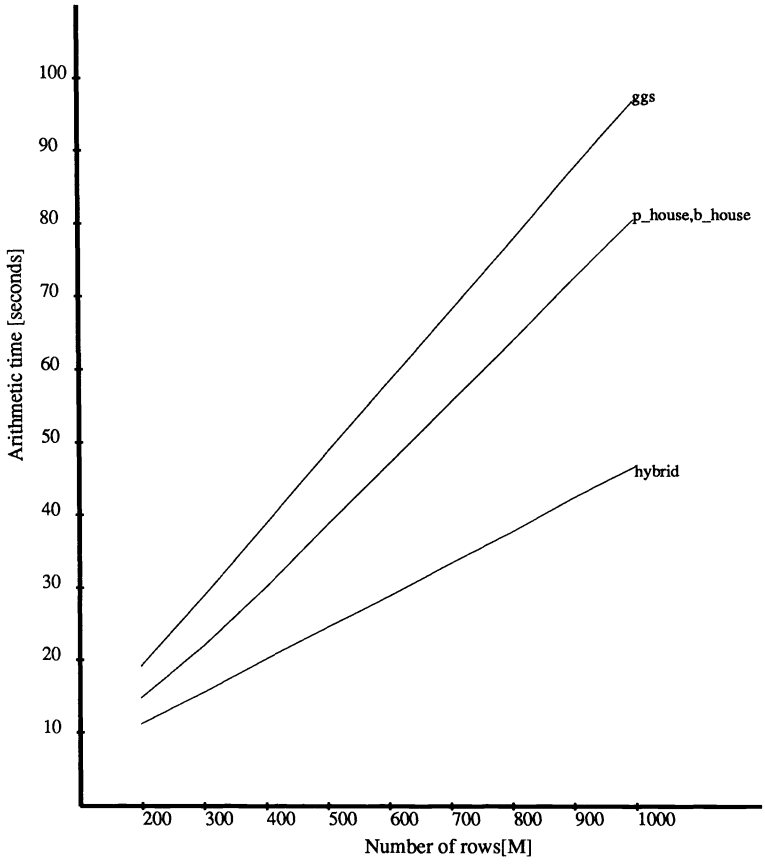


FIG. 4(a). *Predicted arithmetic times of* ggs, p_*house*, b_*house*, *and hybrid algorithms for a matrix with* 100 *columns. The flop times used for each were obtained from implementations of equivalent sequential algorithms on the* iPSC-286.

TABLE 3
Total times (seconds) for a matrix with 200 columns.

M	ggs	b_house	p_house	hybrid
300	99	71	71	79
400	137	101	102	101
800	289	223	224	180
1200	442	345	346	251
1600	595	466	468	320
2000	747	588	590	386

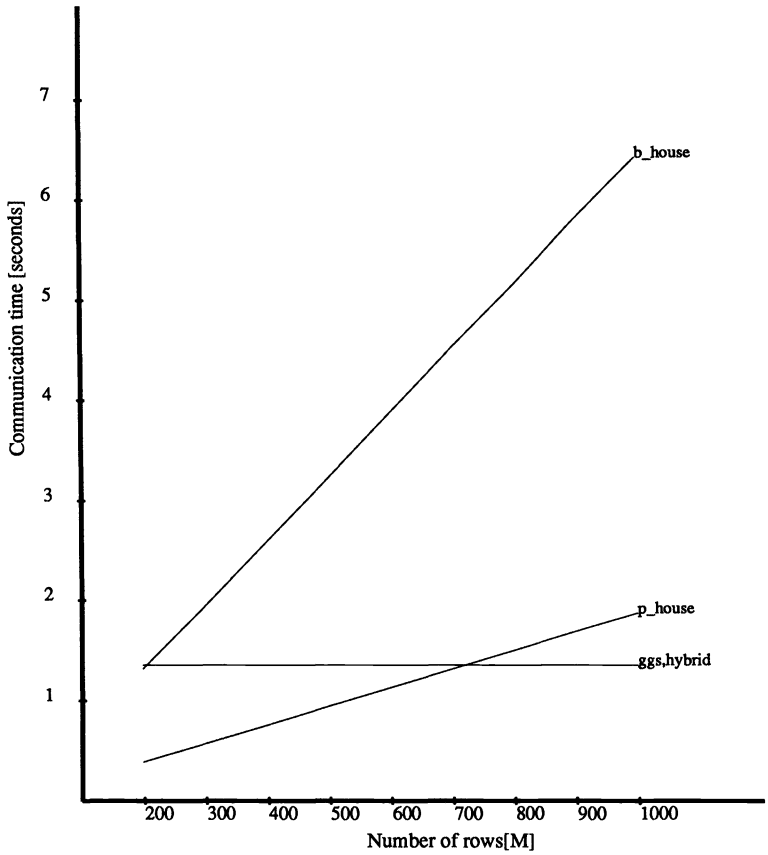


FIG. 4(b). *Predicted communication times of* ggs, p_*house*, b_*house, and hybrid algorithms for a matrix with* 100 *columns. Note that the predicted communication times are much lower than the predicted arithmetic times for these algorithms.*

The parallel arithmetic term in the ggs algorithm is optimal (considering distributed Givens algorithms), since the sequential Givens algorithm has an arithmetic cost of $2N^2(M - N/3)$ flops. Similarly the parallel terms of p_house, b_house, and hybrid are also optimal. Optimality shows that these distributed algorithms succeed in balancing the arithmetic work among the P processors and is the result of the wrap mapping of rows onto processors in the case of the ggs and hybrid algorithms and that of

columns in the case of the Householder algorithms. If block mapping were employed, this term would not be optimal, as can be easily seen from a complexity analysis.

The Householder algorithms have sequential arithmetic terms that are $\Theta(MN)$, and ggs and the hybrid, $\Theta(N^2)$. The communication terms of these respective algorithms also have the same complexity. Since the magnitude of τ is about four to six times that of α, the second-order arithmetic terms make a more significant contribution to the execution times than the communication terms. Indeed, we shall show in § 5.1 that the first- and second-order arithmetic terms suffice to approximate the observed efficiencies of these algorithms.

From Figs. 4(a) and 4(b), we can conclude that because of its greater arithmetic complexity, ggs is the slowest when compared with the Householder and hybrid algorithms. The hybrid algorithm is the fastest, since arithmetic times are smaller in the hybrid than in the Householder algorithms. There are two causes for this: its smaller sequential arithmetic term and its lower value for flop time τ. In p_house and b_house, the sequential term comes from the *cdivs*, and increases with the column length. In the hybrid, the primary contributor to this term is the sequential recursive elimination phase and is independent of column length. Thus as the matrix becomes more and more overdetermined, the hybrid becomes increasingly superior.

4. Experiments and results.

4.1. Experimental conditions. The four algorithms described as well as the standard Givens algorithm (sgs) were implemented on an iPSC-286 hypercube with 16 processors. The version we used was the *memory machine* in which each processor is outfitted with a memory board of four megabytes. The node program and the data reside on the memory board. The node operating system was NX Release 3.1. All experiments were run in the "large" memory model.

Our programs were written in the programming language C, and we used the cc optimizing compiler. Double precision (eight bytes) arithmetic was employed. The matrices were generated using a pseudorandom number generator. The correctness of the factorization was verified by solving a system of equations $Ax = b$, with a known solution vector x.

We stored the matrix in a two-dimensional array. Since this array is stored by rows in C, for ggs, storing the matrix permits sequential access to the rows. For the Householder and hybrid algorithms, we stored the transpose of the matrix for sequential access to columns.

Householder orthogonalization algorithms require twice as much memory access and address arithmetic as Givens algorithms. Address arithmetic costs can be significant on the iPSC-286 if integer multiplication is involved, since the latter operation is slow on the 80286 chip. Therefore, in order to compare both Householder and Givens algorithms fairly, we must keep address arithmetic costs low.

For Givens algorithms, a function that computes Givens rotations between two rows must be provided the addresses of the rows (addresses of the first elements in each row). Then each element can be accessed by an integer addition. Instead, if each element in the two rows is accessed by its row and column index, then an integer multiplication and an integer addition is required. We observed that the orthogonal factorization of a matrix with 100 rows and 50 columns on one processor was slowed down by 70 percent by the address arithmetic overhead in the latter scheme. A similar observation can be made about column addresses in Householder algorithms.

4.2. Distributed algorithms. Experiments were performed for several overdetermined systems with 50 to 500 variables and 200 to 2,000 equations. We observed the

total time for the completion of the algorithm, and the time required by the internal and recursive phases of the ggs and hybrid algorithms. We report the time taken by the processor that completes last. The observed times were reproducible to within a percent. The total times (in seconds) are plotted in Fig. 5 for a matrix with 100 columns. We also tabulate results from experiments for a matrix with 200 columns in Table 3.

The results show good agreement with the complexity analysis. The higher parallel arithmetic term of ggs makes it the algorithm with the highest running time, the p_house and b_house algorithms are next, and the hybrid is the fastest algorithm.

Note that the ratio between ggs and the Householder algorithms is less than two. There are three factors that contribute to this lower ratio. The lower order arithmetic terms are the primary cause, since they increase with M for the Householder algorithms, whereas they are independent of M for ggs. A similar observation can be made about the communication costs. Finally, the ggs algorithm has half the address arithmetic cost of the Householder algorithms.

The Householder algorithms, p_house and b_house, are quite close in performance to each other. Indeed, in Fig. 5, the times for both algorithms are shown by a single line. Our analysis predicts slightly smaller communication times for the pipelined algorithm which results in this algorithm being slightly superior. This is not observed in our results and could be caused by a more efficient implementation of the broadcast

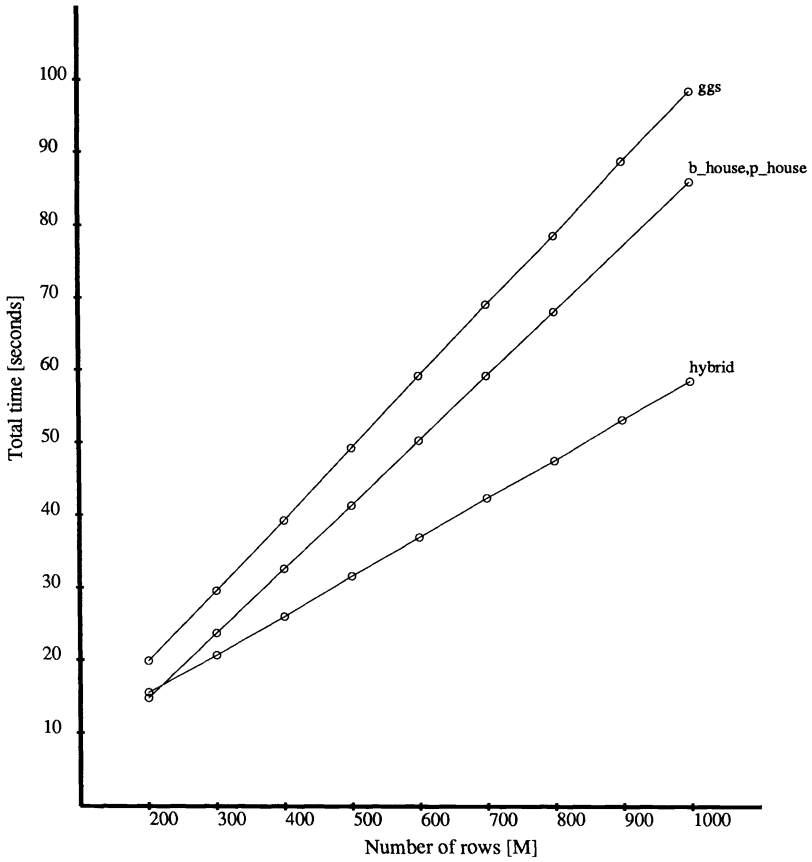


FIG. 5. *Experimentally observed values of total times of the* ggs, p_*house*, b_*house, and hybrid algorithms for a matrix with* 100 *columns.*

than pipelined ring communication. We used a broadcast facility implemented in software on the iPSC-286 and wrote the pipelined communication scheme ourselves.

The hybrid algorithm is slightly slower than the Householder algorithms for $M = 200$, but for larger values of M it is the fastest algorithm. The initial slowdown of hybrid is caused by the nonsequential access it performs in the recursive phase to assemble elements of a row of the matrix. Recall that the transpose of the matrix is stored to permit sequential access to the columns in the internal phase. By measuring the times required by the recursive phase, we confirmed that this phase required more time in the hybrid than in the ggs algorithm.

There are two reasons for the superiority of the hybrid algorithm as M increases. The hybrid has a second-order arithmetic term that is $\Theta(N^2)$ while the corresponding terms in the Householder algorithms are $\Theta(MN)$. The communication terms also show the same behavior.

5. Analysis of efficiencies. From our expressions for the complexities of parallel and sequential algorithms, we can compute theoretical expressions for the efficiencies of these algorithms. We compare these theoretical values with experimentally observed values to gauge the importance of the sequential terms viz. sequential arithmetic terms, communication terms, load imbalances, etc., on the efficiency of parallel algorithms.

The usual definition of speed-up is $s = t_1/t_P$, where t_P is the time taken by the multiprocessor with P processors, and t_1 is the time taken by the best sequential algorithm on one processor. The parallel efficiency of the algorithm is then $e = s/P$. The practical difficulties associated with this definition of efficiency have been pointed out by Geist and Heath (1986). The storage required to solve a problem large enough to be of interest may be available on a multiprocessor, but not on one processor. A serial code may have had many levels of code optimization, but it is clearly impractical to employ the same effort into optimizing the parallel code. Furthermore, since we are comparing two classes of orthogonalization algorithms (Householder and Givens) with different numbers of arithmetic operations, memory accesses, and address arithmetic costs, it is more informative to compare each parallel algorithm with a sequential algorithm in its class, to expose the loss in parallelism due to sequential terms.

To circumvent this difficulty, we define t_1 to be the time that would be taken by an equivalent sequential algorithm on one processor. Thus we compare the performance of a parallel Householder algorithm with a sequential Householder algorithm, and a parallel Givens algorithm with a sequential Givens algorithm. Both parallel and serial codes then have the same level of code optimization, and have similar memory access and address arithmetic overheads. Since storage constraints prohibit the solution of large problems on a single processor, we compute t_1 from the number of flops in the sequential algorithm and the flop times in Table 2. The flop times in Table 2 were obtained from sequential orthogonalization algorithms implemented (on one processor) to match the structure of the distributed algorithms but without communication primitives. Thus the sequential and distributed algorithms have similar address arithmetic overheads.

Note that with this definition of efficiency, a more efficient algorithm is not necessarily the better parallel algorithm. This definition helps us measure the relative importance of the sequential parts in an algorithm in relation to its parallel part.

We have derived the following expressions for efficiency from the complexities in Table 1. For simplicity, we have ignored the communication terms in these expressions, since the arithmetic complexity is an order of magnitude greater than communication complexity and the flop time is four to six times the word transfer

time. The expressions are

$$e_{\text{ggs}} = 1 + \frac{3}{N} - \frac{P \log P + 2}{2M},$$

$$e_{\text{hou}} = 1 - \frac{(3P-1)}{N} + \frac{(3P+5)}{6M},$$

$$e_{\text{hyb}} = 1 - \frac{1}{6M}(6P \log P + 9P - 3).$$

These expressions represent the effect of sequential arithmetic terms on the efficiency of the distributed algorithms and are derived in Pothen and Raghavan (1987). Only one expression is given for p_house and b_house, since the arithmetic costs of these algorithms are the same.

The theoretical and experimentally observed efficiencies are shown in Fig. 6 as a function of M for $N = 200$. The agreement between predicted and experimental values is quite good for ggs and the Householder algorithms (efficiencies of p_house are shown). Experimental efficiencies are lower than theoretical values for the hybrid algorithm.

There are three reasons for this discrepancy. First, since the hybrid algorithm has the lowest running times, the communication term is a relatively more significant fraction of the running time. Second, the recursive phase in hybrid requires more time than in the ggs algorithm, since assembling rows for Givens rotations requires non-sequential access into the two-dimensional array. But the major cause is the imbalance in the arithmetic work in the internal phase of the hybrid algorithm.

As the factorization proceeds, when a top row on a processor is completely transformed, it becomes a row of R and drops out of the computation. Thus the number of rows in the active submatrices on different processors can differ by one. This causes a slight imbalance in the arithmetic work in the internal phase. Such an imbalance is present in the ggs algorithm also, but it is much less significant because IP in ggs takes roughly twice the time it takes in the hybrid. Note that as the problem size increases, IP requires more arithmetic work, and this imbalance is much less important. Hence, the fit in Fig. 6 improves as the number of rows increases.

We can conclude that efficiencies of these algorithms are lower than one primarily because of the sequential arithmetic term. To first order, these efficiencies are independent of the communication terms.

Moler (1987) has recently introduced the notion of *effective parallel algorithms.* Let α denote the inherently sequential nonparallelizable fraction of a sequential algorithm. Then the time required by the parallel algorithm can be expressed as

$$t_P = \frac{(1-\alpha)}{P} t_1 + \alpha t_1.$$

If n denotes the size parameter of a problem, then an algorithm is *effective* if $\alpha(n)$ tends to zero as n tends to infinity. In this case, for large problem sizes, linear speedup can be achieved, and hence efficiencies tend to one.

In our algorithms, the problem size parameters are M and N. From our expressions for efficiencies, it is easy to see that the ggs, Householder, and the hybrid algorithms are effective algorithms. On the other hand, from Table 1 it can be seen that sgs is not an effective algorithm, since its communication complexity is of the same order of magnitude as its arithmetic complexity, and causes a constant loss of parallel efficiency.

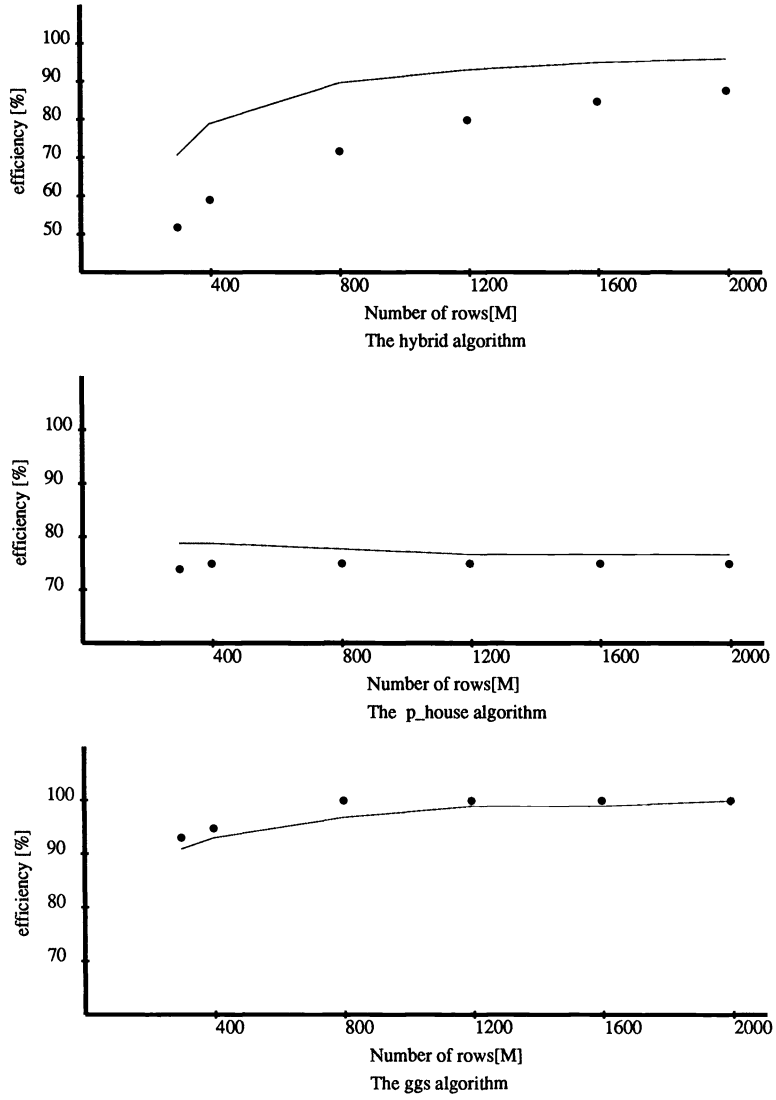


FIG. 6. *Efficiency of the hybrid, Householder, and* ggs *algorithms for a matrix with* 200 *columns. The solid line indicates theoretically predicted values, and the filled circles indicate experimentally observed values.*

6. Concluding remarks. We have considered four distributed algorithms for computing the orthogonal factorization of an overdetermined matrix. In two of these, p_house and b_house, the matrix is mapped by columns, and in the other two, the hybrid and the ggs, the matrix is mapped by rows. The first two rely on column Householder transformations but differ in communication schemes they employ. The ggs algorithm and the hybrid algorithm both transform a column in two phases: the internal phase and the recursive elimination phase. They differ in the internal phase, in which the hybrid employs Householder transformations and the ggs employs Givens rotations.

An important feature that emerges from our analyses is the role the second-order arithmetic term plays in determining execution times and efficiencies of these

algorithms. This term represents the cost of the sequential parts in the arithmetic work. In the Householder algorithms, this term represents the cost of computing the Householder vectors, and in the hybrid and the ggs, this is the cost of the recursive elimination phase. Similarly, the first-order arithmetic term represents the parallel term. In the Householder algorithms, this is the cost of the column modifications with the Householder vector, and in hybrid and ggs, this is the cost of the internal phase.

The communication terms in these algorithms have the same complexity as the sequential arithmetic terms. Since the flop time is four to six times greater than the word transfer time, the latter costs are more significant than the former.

Several enhancements can be incorporated into the algorithms we have described. Our goal in this paper was to compare fairly simple implementations of these algorithms to understand the fundamental features that determine their execution times and efficiencies. We are currently incorporating some of these enhancements into our algorithms.

Loop unrolling can be used to reduce integer arithmetic in the Householder algorithms. In b_house and p_house, a processor performs *cmod* operations with one Householder vector at a time. Instead, a processor can accumulate a set of $k > 1$ Householder vectors, and then perform *cmod* operations simultaneously with all k of these vectors. The resulting algorithm requires temporary storage for k Householder vectors. The code becomes more complex; however, integer arithmetic is reduced. Since the decrease in integer arithmetic becomes smaller as k increases, and the increase in code complexity leads to greater execution times, the best results are obtained for small values of k. Geist and Heath (1986) and Geist and Romine (1987) have used this technique to improve the efficiency of distributed LU factorization algorithms.

6.1. Other algorithms. Johnsson (1982) has described a row-oriented Householder transformation in designing a computational array in VLSI for computing the orthogonal factorization. George and Liu (1987) have used this row Householder transformation to compute the orthogonal factorization of sparse matrices. This algorithm is competitive with a sparse Givens algorithm in terms of intermediate fill and arithmetic operations.

The Householder transformations in our distributed algorithms are the more traditional column-oriented versions. Row-oriented versions of p_house and b_house algorithms have the advantage that their communication costs are independent of the row dimension of the matrix. A row-oriented version of the Householder transformation can be used in the internal phase of the hybrid algorithm as well and might lead to better data flow since the recursive phase uses Givens rotations and is row oriented. The design of row-oriented distributed Householder orthogonalization algorithms merits further work.

When the matrix A does not have full column rank, pivoting needs to be incorporated into the QR factorization. The development of distributed algorithms for QR factorization that include pivoting is an important research problem.

6.2. Sparse orthogonal factorization. Communication costs in distributed sparse orthogonal factorization will be more significant than it has been here. The communication requirements of sparse algorithms are extremely irregular. Information about which processors a given processor needs to communicate with is unknown a priori, and a symbolic phase is necessary to obtain this information.

Hence, in general, a processor will need to communicate with a subset of processors, most of whom are not its neighbors. The density of messages also increases, since there is less arithmetic to do between communication steps due to sparsity. Thus

the store and forward mode of communications on the iPSC-286 could lead to poor performance of sparse matrix algorithms.

The newly announced 80386 based iPSC/2 replaces store and forward with "worm-hole routing," a mode in which a communication coprocessor can route a message by looking at the destination id contained in the first few bits, without storing the message in memory. In this routing scheme, the bottleneck for direct memory access at an intermediate processor does not slow down a message. However, this bottleneck is still there, and it can cause messages to collide at a processor. That is, if several messages having one processor as their destination arrive concurrently at the processor, it has to receive them sequentially. Hot spots can develop also, if a path is heavily used for forwarding messages between several processors. Still, this new scheme should lead to more efficient communications in sparse matrix algorithms.

Acknowledgments. We thank Professor Mary Jane Irwin for several helpful conversations, Dan Ehrlich of the Computer Systems Lab for help with the PostScript language, and Chris Bischof for pointing out an error in an earlier version of this paper. Our computational experiments were run on Intel hypercubes at the Theory Center, Cornell University, Ithaca, NY. We are grateful to Ashley Doerschug, a staff member there, for her cheerful assistance throughout this work.

REFERENCES

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

R. M. CHAMBERLAIN AND M. J. D. POWELL (July 1986), QR *factorization for linear least squares problems on the hypercube*, Report CCS 86/10, Chr. Michelsen Institute, Bergen, Norway.

M. COSNARD, J.-M. MULLER, AND Y. ROBERT (1986), *Parallel* QR *decomposition of a rectangular matrix*, Numer. Math., 48, pp. 239–249.

J. J. DONGARRA, A. H. SAMEH, AND D. C. SORENSEN (1986), *Implementation of some concurrent algorithms for matrix factorization*, Parallel Comput., 3, pp. 25–34.

G. A. GEIST AND M. T. HEATH (1986), *Matrix factorization on a hypercube multiprocessor*, in Hypercube Multiprocessors 1986, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 161–180.

G. A. GEIST AND C. H. ROMINE (1987), LU *factorization algorithms on distributed memory multiprocessor architectures*, Tech. Report ORNL/TM-10383, Oak Ridge National Laboratory, Oak Ridge, TN.

W. M. GENTLEMAN (1975), *Error analysis of* QR *decomposition by Givens transformations*, Linear Algebra Appl., 10, pp. 189–197.

A. GEORGE, M. T. HEATH, AND J. W. H. LIU (1986), *Parallel Cholesky factorization on a shared memory multiprocessor*, Linear Algebra Appl., 77, pp. 165–187.

A. GEORGE AND J. W. H. LIU (1987), *Householder reflections versus Givens rotations in sparse orthogonal decomposition*, Linear Algebra Appl., 88, pp. 223–238.

G. H. GOLUB AND C. F. VAN LOAN (1983), *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD.

M. T. HEATH, ED. (1986), *Hypercube Multiprocessors* 1986, Society for Industrial and Applied Mathematics, Philadelphia, PA.

———, ED. (1987), *Hypercube Multiprocessors* 1987, Society for Industrial and Applied Mathematics, Philadelphia, PA.

L. JOHNSSON (1982), *A computational array for the* QR *method*, Proc. 1982 Conference on Advanced Research in VLSI, Massachusetts Institute of Technology, Cambridge, MA, pp. 123–129.

R. E. LORD, J. S. KOWALIK, AND S. P. KUMAR (1983), *Solving linear algebraic equations on an* MIMD *computer*, J. Assoc. Comput. Mach., 30, pp. 103–117.

F. T. LUK (1986), *A rotation method for computing the* QR *decomposition*, SIAM J. Sci. Statist. Comput., 7, pp. 452–459.

J. J. MODI AND M. R. B. CLARKE (1984), *An alternative Givens ordering*, Numer. Math., 43, pp. 83–90.

C. MOLER (1987), *A closer look at Amdahl's law*, Technical Note, Intel Scientific Computers, Beaverton, OR.

A. POTHEN, S. JHA, AND U. VEMULAPATI (1987), *Orthogonal factorization on a distributed memory multiprocessor*, in Hypercube Multiprocessors 1987, M. T. Heath, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, pp. 587–596.

A. POTHEN AND P. RAGHAVAN (July 1987), *Distributed orthogonal factorizations: Givens and Householder algorithms*, Tech. Report CS-87-24, Computer Science, Pennsylvania State University, University Park, PA.

A. H. SAMEH AND D. J. KUCK (1978), *On stable parallel linear system solvers*, J. Assoc. Comput. Mach., 25, pp. 81–91.

A. H. SAMEH (1985), *Solving the linear least squares problem on a linear array of processors*, in Algorithmically Specialized Computers, L. Snyder, ed., Academic Press, New York, pp. 191–200.

G. W. STEWART (1973), *Introduction to Matrix Computations*, Academic Press, New York.

SPECIAL SECTION ON SPARSE MATRIX ALGORITHMS ON SUPERCOMPUTERS

It has long been recognized that the efficient solution of sparse linear equations is a key element for solving large scientific applications problems. A recent study lists several major scientific application areas which require the numerical solution of linear systems: lattice-gauge theory, structural analysis, circuit simulation, device simulation, quantum chemistry, computational fluid dynamics, and geodetic networks. The availability of supercomputers and breakthroughs in algorithm design have led to the feasibility of larger and more refined models in these scientific areas, yet new and more demanding models remain to be solved. Most researchers involved in these fields believe that only the combined efforts of advanced algorithm technology and new supercomputing hardware will enable us to solve the challenging problems ahead. However, scientists involved with supercomputing applications are often unaware of the latest algorithmic advances, and algorithm researchers are often unaware of pressing applications needs.

In order to address some of these challenging problems, and in order to make a step towards interdisciplinary cooperation, NASA Ames Research Center, Boeing Computer Services, and Cray Research, Inc. sponsored a three day workshop from March 27–March 30, 1988, at the Chaminade Conference Center near Santa Cruz. The goal of the workshop was to bring together all three groups involved in the progress of supercomputing: hardware designers, algorithm developers, and researchers in scientific applications, and thus to provide a better interaction between the various groups involved in supercomputing. The numerical solution of sparse linear systems was the unifying theme of the workshop.

The workshop was attended by more than 70 scientists and software developers from a wide range of applications areas. There were eleven invited talks and a large number of informal workshop presentations. The editors of SISSC are pleased to publish a selection of the papers from this workshop.

The selected papers are indicative of the wide variety of tools and methods used by researchers in the field of sparse matrix algorithms on advanced architectures. The papers by Lewis, Peyton, and Pothen and by Liu show the deep involvement in techniques rooted in discrete mathematics, which are necessary to develop efficient orderings for sparse matrices on parallel machines. The paper by van der Vorst demonstrates the intricacies of an efficient implementation of preconditioned methods on vector architectures. And Young et al. show how to combine theoretical results and practical implementations to solve large scale industrial problems using sparse matrix techniques. Finally, Saad has contributed an excellent survey paper, which demonstrates the depth of current research activities in the sparse matrix community.

We wish to thank Tony Chan and Ahmed Sameh, both of whom helped in overseeing the review process, which followed the normal SIAM policies for selecting referees and making recommendations. We also wish to thank Chris Hsiung from Cray Research, who originated the idea for such a workshop and whose organization also provided the initial funding for getting the project started. Further thanks are due to Ralph Blohm from the Supercomputer Integration Program (SCIP) at Boeing Computer Services and Carter Rich from the GE Western Systems Division, who provided additional funding, and Phuong Vu from Cray Research, who supported the organization.

C. W. Gear
H. D. Simon

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 6, pp. 1136-1145, November 1989
© 1989 Society for Industrial and Applied Mathematics
005

THE MINIMUM DEGREE ORDERING WITH CONSTRAINTS*

JOSEPH W. H. LIU†

Abstract. A hybrid scheme for ordering sparse symmetric matrices is considered. It is based on a combined use of the top-down nested dissection and the bottom-up minimum degree ordering schemes. A separator set is first determined by some form of incomplete nested dissection. The minimum degree ordering is then applied subject to the constraint that the separator nodes must be ordered last. It is shown experimentally that the quality of the resulting ordering from this constrained scheme exhibits less sensitivity to the initial matrix ordering than that of the original minimum degree ordering. An important application of this approach to find orderings suitable for parallel elimination is also illustrated.

Key words. sparse matrices, ordering, minimum degree, nested dissection, separator

AMS(MOS) subject classifications. 65F50, 65F25

1. Introduction. It is well known that matrix reordering is an important phase in the direct solution of sparse symmetric positive definite linear systems [3], [8]. One of the reordering schemes is the *minimum degree* ordering, a symmetric version of the Markowitz scheme [19]. Indeed, the minimum degree ordering has been widely recommended as a general purpose fill-reducing reordering strategy. Its wide acceptance is largely due to its effectiveness in reducing fill and its efficient implementations. Many sparse matrix software packages have included the minimum degree ordering as a standard reordering scheme. Over the last two decades, many important enhancements have been made to the implementation of the minimum degree ordering. The survey paper by George and Liu [9] contains a comprehensive account of the development.

Another popular fill-reducing reordering scheme is the *nested dissection* ordering [6]. Central to this scheme is the notion of *separators*: a separator of a connected graph is a subset of nodes whose removal renders the remaining graph disconnected. From an algorithmic point of view, the nested dissection scheme orders the graph from a top-down approach by recursive use of separators. On the other hand, the minimum degree algorithm performs the ordering bottom-up using degree information.

In this paper, we consider a hybrid reordering scheme based on a combined use of these two different approaches. A separator set is first determined by some form of an *incomplete* nested dissection ordering [7]. This set is then used as *node constraints* for the minimum degree ordering; that is, the ordering algorithm is only allowed to select nodes of minimum degree outside this constraint set, and the nodes from this set are to be ordered last.

The initial motivation of this constrained ordering algorithm is to attempt to remove one undesirable feature of the basic minimum degree ordering: that its quality depends heavily on the initial ordering of the given matrix problem. This shortcoming has been attributed to the lack of an effective *tie-breaking* strategy for the ordering algorithm. A difference of over 25 per cent in factor nonzeros and over 55 per cent in factor operation counts has been reported [9] for the current SPARSPAK implementation of the minimum degree ordering [15] when applied to the same grid problem with different initial orderings. Indeed, Berman and Schnitger [1] have shown that there is

* Received by the editors October 5, 1988; accepted for publication (in revised form) January 31, 1989.

† Department of Computer Science, York University, North York, Ontario, Canada M3J 1P3. This research was partially supported by Natural Sciences and Engineering Research Council of Canada grant A5509 and by Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

an initial ordering of the square grid for which the minimum degree ordering can have fills an order of magnitude more than the optimal.

In the constrained minimum degree ordering scheme introduced in this paper, we will show that with suitably selected constraint sets from some form of incomplete nested dissection, this sensitivity to the initial matrix ordering is reduced at the expense of more reordering time. We will also demonstrate by experimental results that sometimes the quality of the ordering can even be improved over the original minimum degree algorithm. Another important application of this modified algorithm is in the context of parallel factorization. We will show by experiments that orderings obtained from this scheme are suitable for parallel elimination. The number of parallel node elimination steps is reduced by 10 per cent to over 30 per cent, when compared with existing orderings.

An outline of this paper follows. In § 2, the constrained minimum degree algorithm is described. Section 3 contains experimental results on its use for grid problems. Using separators from an incomplete nested dissection [7] of the grid, we demonstrate that sensitivity to the initial matrix ordering is reduced.

In § 4, we apply the constrained algorithm to devise the *bootstrap* minimum degree ordering. The idea hinges on the use of the basic minimum degree ordering scheme to determine good separator sets as described in [17]. In other words, the separator sets for the incomplete nested dissection are obtained from the original minimum degree algorithm. These separator sets are then used in connection with the constrained algorithm. In some sense, we are attempting to improve on the minimum degree ordering based on itself, and hence the term *bootstrap* is used. Experimental results of its performance on some problems from the Harwell-Boeing sparse matrix collection are provided in § 4. In § 5, we show that the orderings obtained from the bootstrap algorithm are also relevant for parallel elimination. Section 6 contains the concluding remarks.

In this paper, the reader is assumed to be familiar with basic graph-theoretic notions related to sparse matrix computation. Familiarity with the minimum degree and nested dissection orderings is also assumed. The reader is referred to [3] or [8].

2. Constrained minimum degree ordering.

2.1. The basic minimum degree algorithm. Let G be an undirected graph and v a node of G. Throughout this paper, we will use the notation $\mathrm{Adj}_G(v)$ to refer to the set of nodes adjacent to v in G. The degree of the node v in G is simply $|\mathrm{Adj}_G(v)|$. For convenience, we will also use G to refer to the node set in the graph G.

The basic minimum degree ordering can be best described in terms of *elimination graphs*. Following Rose [20], we use the notation G_v to represent the elimination graph obtained after the elimination of the node v from the graph G. The graph G_v can be obtained by deleting the node v and its incident edges from G and then adding edges to make the nodes adjacent to v into a clique.

The basic algorithm selects a node y of minimum degree from the graph G and numbers the node y next in the ordering. The graph is then replaced by the elimination graph G_y, whereby the selection process is repeated. Many enhancements have been developed through the last two decades to improve on the performance of this basic algorithm. The survey paper by George and Liu [9] discusses most of the practical enhancements and their effectiveness.

2.2. The constrained algorithm. The constrained version of the minimum degree algorithm is a simple extension of the basic ordering scheme. Let G be the undirected graph and K be a given set of *constraint* nodes. The constrained algorithm orders the

graph G by the minimum degree ordering subject to the condition that nodes from K are to be numbered last.

ALGORITHM 2.1 (Constrained minimum degree ordering).

```
begin
  G := given symmetric graph;
  while G − K ≠ ∅ do
    begin
      Select a node y of minimum degree |Adj_G (y)| from G − K
        and order y next;
      G := G_y;
    end;
  {the elimination graph G now contains only nodes in K}
  Order the remaining nodes in G;
end.
```

It is worth noting that although nodes from the subset K cannot be selected for elimination in the **while**-loop, they participate in the degree calculation of their adjacent nodes in $G - K$. Indirectly, they affect the node selection process. Also note that if the subset K is empty, Algorithm 2.1 is simply the basic minimum degree algorithm. All the various enhancements described in [9] can be applied to this constrained version.

In the algorithm, we have not specified the order in which nodes within the constraint set K are to be numbered. A reasonable approach is to re-apply minimum degree ordering on the remaining elimination graph with only the constraint nodes (at the end of the **while**-loop). However, as we shall see in later sections, the way in which the constraint set is obtained usually provides an implicit ordering on the nodes. In such cases, we will simply retain the relative ordering of the nodes in the constraint set.

3. Case study: The regular grid.

3.1. Use of grid separators as constraints. In Algorithm 2.1, even if some nodes in the constraint subset K are of the current minimum degree, they do not participate in the next round of selection. In effect, it forces the algorithm to select minimum degree nodes outside the subset K. In this section, we use k-by-k regular grid model problems to study the effect of using separators as the constraint set. For this problem, we assume the nine-point difference operator so that nodes associated with each small square are pairwise connected.

On the k-by-k grid, the well-known *nested dissection* algorithm by George [6] produces fill-reducing orderings from a top-down approach. It is based on a recursive application of the following partitioning scheme. Consider a given s-by-t subgrid. If the subgrid is sufficiently small, say $\max\{s, t\} \leqq 2$, number all the nodes associated with it row-by-row. Otherwise, if $s \leqq t$, choose S to be the set of nodes in a (roughly) middle vertical grid line. Choose S to correspond to a middle horizontal grid line for the case $s > t$. The set S forms a separator of the subgrid whose removal partitions it into two smaller subgrids U and V. Labeling nodes in S last will ensure that there is no edge between any pair of nodes from U and V in the filled graph. A recursive application of this idea to the subgrids U and V gives the nested dissection ordering.

In the literature, there are slight variants of this basic dissection scheme for the grid problem. See, for example, the paper by Duff, Erisman, and Reid [2]. In [7], George, Poole, and Voigt consider orderings obtained by the premature termination of the full nested dissection scheme. They use the term *incomplete* nested dissection to refer to such orderings.

3.2. Experimental results. The (vertical or horizontal) grid separator partitions a given grid into two roughly equal halves. The original objective is to limit fills so that they cannot occur between nodes in the two remaining subgrids. Such separators are good candidates for the constraint set in the constrained minimum degree ordering. We may view the approach as using the set of grid separators from the incomplete nested dissection as constraints. In this section, we provide experimental results to demonstrate the effectiveness of the combined use of incomplete nested dissection with the constrained minimum degree ordering.

We use a parameter "*level*" that indicates the dissection level. For *level* = 0, that means no dissector is used so that the scheme is simply the original minimum degree ordering (without any constraint set). When *level* = 1, we are using one level of separators; this implies the constraint set consists of a single grid line. We illustrate the cases when *level* = 1, 2, 3, in Fig. 3.1.

In Table 3.1, we provide statistics gathered from runs using this ordering combination on a 100×100 regular grid (with nine-point connectivity). The sensitivity of the ordering quality of the minimum degree implementations to the initial orderings has been demonstrated in [2] and [9]. The objective of Table 3.1 is to demonstrate the use of separators from incomplete nested dissection in reducing the sensitivity.

For each case (with a varying *level* value), we obtain the constrained minimum degree ordering for the initial row-by-row ordering and for twenty randomly permuted initial matrix orderings. Table 3.1 contains the number of nonzeros in the factor matrices. For the 20 randomly permuted problems, we tabulate the best, the worst, and the average number of factor nonzeros. For comparison, we also include the statistics using the initial row-by-row ordering. The orderings on the nodes in the constraint set are retained from the incomplete nested dissection.

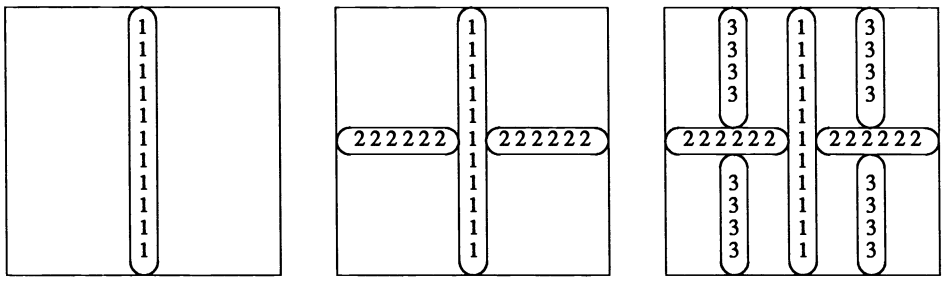


FIG. 3.1. *Constraint sets for level* = 1, 2, 3.

TABLE 3.1

Factor nonzeros from constrained minimum degree ordering on a 100×100 *grid using incomplete nested dissection.*

Separator level	Row-by-row initial ordering	20 randomly permuted initial orderings		
		Best	Worst	Average
0	304163	365002	383570	374852
1	295420	350770	361266	356389
2	285559	339476	349806	344442
4	289050	318923	327440	322504
6	288042	302350	304824	303612
8	287807	290555	291445	291008

As expected, the results from Table 3.1 show that the orderings tend to improve as the separator level increases. The constraint set from incomplete nested dissection reduces the subset from which the next node can be selected by the ordering algorithm. This partially removes the dependence of the ordering quality on the initial matrix ordering. It is relevant to point out that a complete nested dissection ordering on a 100×100 grid (in the manner described in § 3.1) will give a Cholesky factor with 287020 number of off-diagonal nonzeros. The numbers in Table 3.1 are approaching this value as the separator level increases. These statistics show that the nested dissection ordering is actually doing better than the minimum degree ordering. This can be attributed to the observation that the nested dissection ordering on the grid with horizontal and vertical grid separators can be viewed as a minimum degree ordering where ties are broken almost perfectly during the ordering.

4. Bootstrapping the minimum degree ordering.

4.1. Finding separators from the minimum degree ordering. The results from the previous section on the study of the grid model problem strongly suggest the viability of the approach of using separators as constraint sets for the constrained minimum degree ordering. Such separators are readily available for any rectangular subgrids. However, for general undirected graphs, the problem of finding good separators is still under active research [5], [10], [12], [13], [14].

Recently, this author [17] has devised a graph partitioning method to determine node separators for undirected connected graphs. The method depends, in turn, on a minimum degree ordering. Schematically, the algorithm can be described as follows.

ALGORITHM 4.1 (Separator determination).
begin
  $S := \emptyset$;
  Find an initial separator S_{new} of a given connected graph G;
  while $S_{\text{new}} \neq S$ **do**
    begin
      $S := S_{\text{new}}$;
      $S_{\text{new}} :=$ Improve (S);
    end;
  Return separator S
end.

In the scheme, the initial separator is obtained from a minimum degree ordering on the given connected graph. Note that for any subgraph U, its adjacent set $S = \text{Adj}_G(U)$ separates the nodes in this set from the remaining set of nodes $G - S - U$ (provided nonempty). The objective then is to obtain a subgraph of size close to half the graph size and with a small adjacent set.

The relevance of the minimum degree ordering is as follows. At each step of the ordering, consider the connected components of the subgraph of G induced by the nodes that have already been labeled. Numbering the next node has the effect of enlarging these labeled connected components. The criterion used by the minimum degree ordering ensures that

$$\max \{|\text{Adj}_G(C)| : |C \text{ labeled connected component}\}$$

is minimized locally. Thus the minimum degree ordering has the desirable effect of controlling locally the sizes of the adjacent sets of labeled connected components.

Let n be the number of nodes in the given connected graph. We obtain the initial separator for Algorithm 4.1 by monitoring the labeled connected components in the

minimum degree ordering algorithm. We find a labeled connected component $\tilde{C}$ such that

$$|\tilde{C}| = \max\{|C| : |C| < n/2\}.$$

The separator is then given by $\mathrm{Adj}_G(\tilde{C})$.

Then an iterative improvement strategy is applied on this initial separator $\mathrm{Adj}_G(\tilde{C})$. The improvement scheme *Improve* (S) is based on some established results in bipartite graph matching. Let U be the larger portion remaining after the removal of the current separator S, and $V = G - S - U$ be the smaller portion. If we can find a subset Y of the current separator S, such that the adjacent set of Y in U is smaller than Y itself, we can reduce the size of the separator by replacing Y by the adjacent set of Y in U. More specifically, let $\mathrm{Adj}(Y, U)$ be the adjacent set of Y in U with $|\mathrm{Adj}(Y, U)| < |Y|$. The three node subsets defining the new partitioning will be given by

$$S_{\mathrm{new}} = (S - Y) \cup \mathrm{Adj}(Y, U),$$
$$U_{\mathrm{new}} = U - \mathrm{Adj}(Y, U),$$
$$V_{\mathrm{new}} = V \cup Y.$$

We can view this heuristic scheme as using a local search for an improved separator set. The iterative scheme in Algorithm 4.1 then proceeds to improve on this new separator S_{new} using the larger of the two portions U_{new} and V_{new}. It stops when no such improvement can be obtained. For more details on this separator determination algorithm and the relevance of bipartite graph matching, the reader is referred to [17].

4.2. The bootstrap minimum degree algorithm. The results in [17] show that the scheme in Algorithm 4.1 is quite effective in returning good separators. We can therefore use such separators as constraint sets for the constrained minimum degree ordering. Since the initial separator used by Algorithm 4.1 is obtained by a minimum degree ordering, we may view the entire process as *bootstrapping* the minimum degree ordering.

We formulate this bootstrapping approach in Algorithm 4.2. A parameter r is used that is similar to the "level" parameter in the incomplete nested dissection ordering. On exit, we obtain an ordering using a constraint set K, whose removal leaves no component with more than $n/2^r$ nodes.

ALGORITHM 4.2 (Bootstrap minimum degree ordering).

begin
  $K := \emptyset$;
  while there is a connected component C in $G - K$ with $|C| \geqq n/2^r$ **do**
    begin
      Find a separator S of subgraph C using Algorithm 4.1;
      $K := K \cup S$.
    end;
  Apply the constrained minimum degree ordering of Algorithm 2.1
    using K and return the ordering;
end

Note that the step to find a separator S for a connected component C using Algorithm 4.1 also employs a minimum degree ordering. For our implementation, we actually use a constrained minimum degree ordering for C, with the current K being the constraint set. To illustrate the separators from this algorithm, we display in Fig. 4.1 the constraint set obtained on the 50-by-50 grid problem with dissection value $r = 5$. Those gridpoints with values from one to five belong to the constraint set, and each

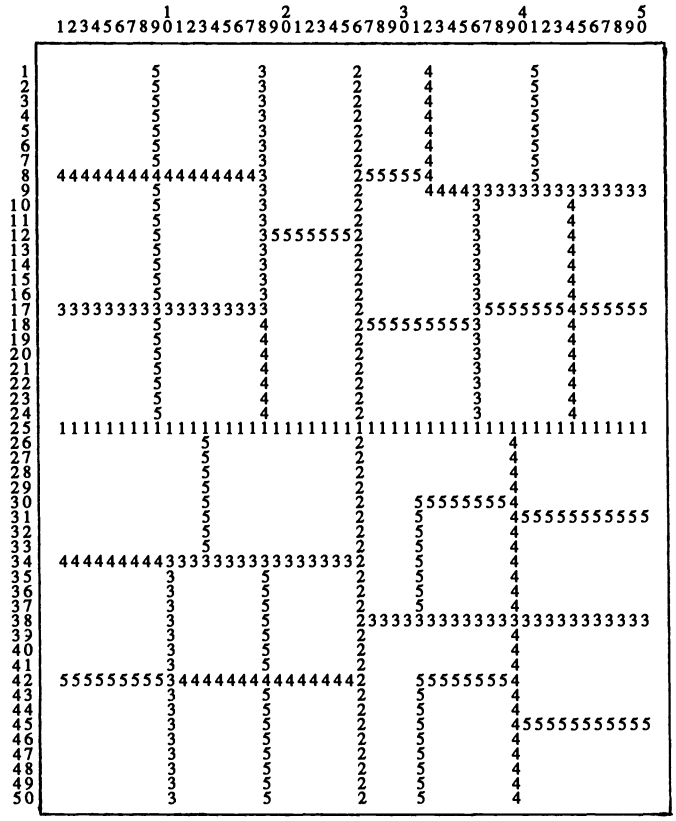


FIG. 4.1. *Constraint set on a* 50×50 *grid for bootstrap minimum degree ordering.*

assigned value represents the associated level during the incomplete nested dissection ordering.

4.3. Experimental results on grid problem and selected Harwell–Boeing problems. To see the effectiveness of the bootstrap minimum degree approach of Algorithm 4.2, we apply it to the 100-by-100 grid problem. In Table 4.1, we obtain statistics for different values of the dissection parameter r.

TABLE 4.1
Factor nonzeros for bootstrap minimum degree ordering on a 100×100 *grid.*

Separater level	Row-by-row initial ordering		20 randomly permuted initial orderings		
	Time	Nonz	Best	Worst	Average
0	14.16	304163	365002	383570	374852
1	21.16	296320	346121	361127	353920
2	28.54	290088	337855	363152	345361
4	43.58	295083	322187	354238	330595
6	58.06	298285	308695	347836	318748
8	70.84	298606	302599	345298	313132

The separators used in Table 4.1 are obtained from Algorithm 4.1. Although they are not as good as those grid separators used in § 3, the experimental results show behavior similar to those in Table 3.1. The sensitivity to the initial matrix ordering is likewise reduced. It is interesting to point out that the bootstrap separators used in Table 4.1 are not as effective at reducing sensitivity to the initial ordering as the grid horizontal and vertical separators used in Table 3.1. It appears that the bootstrap separators have inherited from the minimum degree ordering some of its sensitivity to the initial ordering.

We have also included in Table 4.1 the ordering time in CPU seconds on a SUN 3/280 (the codes are in FORTRAN). The ordering time for $r=0$ is simply the basic minimum degree ordering time with no constraints. For this particular problem, the penalty to pay in ordering time for each additional dissection level r is about half the basic minimum degree time.

To obtain general performance results of the bootstrap minimum degree ordering, we apply it to selected problems from the Harwell-Boeing collection of sparse matrix problems [4]. In Table 4.2, we give the off-diagonal nonzero counts of the Cholesky factors. For detailed description of the problems, the reader is referred to [4] or [17]. Statistics for the column "Minimum Degree" are obtained using the minimum degree ordering implementation as described in [15].

TABLE 4.2
Factor nonzero counts on selected Harwell-Boeing problems.

Problem	Minimum degree	Dissection value r				
		$r=1$	$r=2$	$r=3$	$r=4$	$r=5$
BCSPWR09	4592	4615	4709	4770	4797	4838
BCSPWR10	22757	22756	23235	23958	24269	24320
BCSSTK08	29973	28538	31003	31258	31643	31744
BCSSTK13	269668	284128	279035	265807	264692	262327
BCSSTM13	43491	43491	41283	40252	41721	41321
BLCKHOLE	51570	51169	48690	50137	50148	49960
CAN 1072	19329	19783	20175	20743	20719	20778
DWT 2680	50070	50758	54783	57103	57754	58648
LSHP3466	83116	80941	81694	82206	82654	82632

From Table 4.2, we note that there is only a modest reduction in the number of factor nonzeros in five of the nine problems, and an increase in the other four problems. With the extra reordering time, the bootstrap minimum degree approach can hardly be recommended for fill reduction. However, if matrices of the same matrix structure must be factored many times, it might be desirable to execute the bootstrap minimum degree ordering for several values of r and select the best ordering.

5. Orderings for parallel elimination. A more important application of the bootstrap minimum degree approach is in the determination of orderings for sparse matrices appropriate for parallel machines. Its suitability comes from the divide-and-conquer paradigm inherited from the use of separators. As a measure of parallelism, we will use the number of parallel node elimination steps. This quantity is related to an important structure for sparse factorization, called the *elimination tree.* We refer the reader to [16] for a discussion of the role of elimination tree structures in modeling

TABLE 5.1
Number of parallel elimination steps on selected Harwell-Boeing problems.

Problem	Equivalent minimum degree	Dissection value r				
		$r=1$	$r=2$	$r=3$	$r=4$	$r=5$
BCSPWR09	50	50	43	44	44	42
BCSPWR10	110	102	103	98	101	100
BCSSTK08	197	197	180	177	177	178
BCSSTK13	639	613	572	521	507	501
BCSSTM13	242	283	217	206	210	202
BLCKHOLE	210	222	188	174	175	173
CAN 1072	134	134	119	119	119	119
DWT 2680	264	268	202	191	175	175
LSHP3466	273	279	256	208	203	203

parallel sparse factorization. The number of parallel elimination steps used here is simply the height of the corresponding elimination tree.

In Table 5.1, we tabulate the parallel steps for the respective orderings. The column "Equivalent Minimum Degree" provides the number of parallel steps for the reordering obtained by the minimum degree ordering as described in [15], followed by an equivalent sparse matrix reordering that preserves the filled graph but minimizes the number of parallel steps. The postprocessing algorithm used to determine such equivalent reordering is by Jess and Kees [11] and the implementation used is from [18]. The number of steps given in this column represents the best possible based on the given minimum degree ordering.

From Table 5.1, using a dissection value r of three or four gives consistently better results. The reduction in the number of parallel elimination steps is quite substantial. It ranges from 10 per cent to over 30 per cent. In the context of parallel factorization, the extra reordering time in the bootstrap minimum degree approach seems to be worthwhile in obtaining a better ordering for parallel elimination.

6. Concluding remarks. We have introduced the idea of using constraint sets in conjunction with the minimum degree ordering algorithm. It has been demonstrated that when we use appropriate separators in the constrained ordering, the resulting ordering quality exhibits less sensitivity to the initial matrix ordering than that of the basic minimum degree ordering.

Combining the separator determination algorithm from [17] with the constrained ordering, we have devised the bootstrap minimum degree ordering algorithm. It is a general purpose reordering a scheme that produces orderings suitable for parallel elimination on multiprocessing machines. It is interesting to point out that the same ordering approach can be used if different separator algorithms (such as those from [5], [12], [13]) are used in the bootstrap minimum degree ordering.

Acknowledgment. The author thanks the two referees for their careful reading of the original manuscript and for their many constructive suggestions.

REFERENCES

[1] P. BERMAN AND G. SCHNITGER, *On the performance of the minimal degree heuristic for Gaussian elimination*, Tech. Report, Department of Computer Science, Pennsylvannia State University, University Park, PA, 1987.

[2] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *On George's nested dissection method*, SIAM J. Numer. Anal., 13 (1976), pp. 686-695.

[3] ———, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1987.

[4] I. S. DUFF, R. GRIMES, J. LEWIS, AND W. POOLE, *Sparse matrix test problems*, SIGNUM Newsletter, 17 (1982).

[5] C. M. FIDUCCIA AND R. M. MATTHEYSES, *A linear-time heuristic for improving network partition*, in 19th Design Automation Conference, Las Vegas, NV, June 14-16, 1982, pp. 175-181.

[6] J. A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345-363.

[7] J. A. GEORGE, W. G. POOLE JR., AND R. G. VOIGT, *Incomplete nested dissection for solving n by n grid problems*, SIAM J. Numer. Anal., 15 (1978), pp. 662-673.

[8] J. A. GEORGE AND J. W. H. LIU, *Computer Solution of Large Sparse Poisitive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[9] ———, *The evolution of the minimum degree ordering algorithm*, SIAM Rev. 31 (1989), pp. 1-19.

[10] J. R. GILBERT AND E. Z. ZMIJEWSKI, *A parallel graph partitioning algorithm for a message-passing multiprocessor*, Tech. Report TR 87-803, Department of Computer Science, Cornell University, Ithaca, New York, 1987.

[11] J. A. G. JESS AND H. G. M. KEES, *A data structure for parallel* L/U *decomposition*, IEEE Trans. Comput., 31 (1982), pp. 231-239.

[12] B. W. KERNIGHAN AND S. LIN, *An efficient heuristic procedure for partitioning graphs*, Bell System Tech. J., 49 (1970), pp. 291-307.

[13] C. E. LEISERSON AND J. G. LEWIS, *Orderings for parallel sparse symmetric factorization*, in Third SIAM Conference on Parallel Processing for Scientific Computing, December 1987.

[14] R. J. LIPTON AND R. E. TARJAN, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177-189.

[15] J. W. H. LIU, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141-153.

[16] ———, *Computational models and task scheduling for parallel sparse Cholesky factorization*, Parallel Comput., 3 (1986), pp. 327-342.

[17] ———, *A graph partitioning algorithm by node separators*, Tech. Report CS-88-01, Department of Computer Science, York University, North York, Ontario, Canada, 1988. ACM Trans. Math. Software, to appear.

[18] J. W. H. LIU AND A. MIRZAIAN, *A linear reordering algorithm for parallel pivoting of chordal graphs*, SIAM J. Discrete Math., 2 (1989), pp. 100-107.

[19] H. M. MARKOWITZ, *The elimination form of the inverse and its application to linear programming*, Management Sci., 3 (1957), pp. 255-269.

[20] D. J. ROSE, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. C. Read, ed., Academic Press, New York, 1972, pp. 183-217.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 6, pp. 1146-1173, November 1989
© 1989 Society for Industrial and Applied Mathematics
006

A FAST ALGORITHM FOR REORDERING SPARSE MATRICES FOR PARALLEL FACTORIZATION*

JOHN G. LEWIS†, BARRY W. PEYTON‡, AND ALEX POTHEN§

Abstract. Jess and Kees [*IEEE Trans. Comput.*, C-31 (1982), pp. 231-239] introduced a method for ordering a sparse symmetric matrix A for efficient parallel factorization. The parallel ordering is computed in two steps. First, the matrix A is ordered by some fill-reducing ordering. Second, a parallel ordering of A is computed from the filled graph that results from symbolically factoring A using the initial fill-reducing ordering. Among all orderings whose fill lies in the filled graph, this parallel ordering achieves the minimum number of parallel steps in the factorization of A. Jess and Kees did not specify the implementation details of an algorithm for either step of this scheme. Liu and Mirzaian [*SIAM J. Discrete Math.*, 2 (1989), pp. 100-107] designed an algorithm implementing the second step, but it has time and space requirements higher than the cost of computing common fill-reducing orderings.

A new fast algorithm that implements the parallel ordering step by exploiting the *clique tree* representation of a chordal graph is presented. The cost of the parallel ordering step is reduced well below that of the fill-reducing step. This algorithm has time and space complexity linear in the number of *compressed subscripts* for L, i.e., the sum of the sizes of the maximal cliques of the filled graph. Running times nearly identical to Liu's heuristic composite rotations algorithm, which approximates the minimum number of parallel steps, are demonstrated empirically.

Key words. chordal graph, clique tree, Jess and Kees method, parallel ordering, parallel sparse Cholesky factorization, perfect elimination ordering, sparse matrix

AMS(MOS) subject classifications. 65F50, 65F25, 68R10

1. Introduction.

An important problem in the design of parallel algorithms for the direct solution of sparse, symmetric, positive definite systems is reordering the given matrix A so that the reordered matrix can be factored efficiently in parallel. Jess and Kees [14] suggest a modular approach to this problem. First, a permutation matrix P is computed such that the factor L is sparse, where $PAP^T = LL^T$; the adjacency graph of the resulting filled matrix $F = L + L^T$ is a chordal graph. Second, a parallel ordering is computed from the chordal filled graph by successively removing maximum independent sets of simplicial (nondeficient) nodes from the graph. If F were viewed as a matrix to be factored, then the computed ordering permits the parallel factorization of F in the minimum number of steps, under the constraint that no additional fill in F is permissible. Alternatively, the computed ordering achieves the minimum number of parallel steps in factoring A for all orderings whose fill lies in the filled graph resulting from the original sequential ordering.

Liu [18] shows that the Jess and Kees method finds a parallel ordering that has the shortest elimination tree over all perfect elimination orders of the filled graph. The

* Received by the editors October 5, 1988; accepted for publication (in revised form) February 22, 1989.

† Scientific Computing and Analysis Division, Boeing Computer Services, P.O. Box 24346, Seattle, Washington 98124. The research of this author was supported by the U.S. Air Force Office of Scientific Research under grant F49620-87-C-0037.

‡ Scientific Computing and Analysis Division, Boeing Computer Services, P.O. Box 24346, Seattle, Washington 98124. Present address, Mathematical Sciences Section, Oak Ridge National Laboratory, P.O. Box 2009, Building 9207-A, Oak Ridge, Tennessee 37831-8083. The research of this author was supported by the U.S. Air Force Office of Scientific Research under grant F49620-87-C-0037.

§ Department of Computer Science, Pennsylvania State University, University Park, Pennsylvania 16802. The research of this author was supported by National Science Foundation grant CCR-8701723; National Science Foundation equipment grant CCR-8705110; U.S. Air Force Office of Scientific Research grant AFOSR-88-0161; and by Oak Ridge National Laboratory and the University of Tennessee, Knoxville, during their numerical linear algebra year.

elimination tree captures the dependencies among the columns of A, and its height is an appropriate measure of parallel complexity in a computing environment in which the elimination of a column can be assumed to be a constant-time operation. In addition, the elimination tree height is closely related to other measures of parallel complexity (Liu [17]) and provides a simple but effective criterion for parallelism in several models of parallel factorization.

This modular approach only provides the minimum number of parallel steps for parallel orderings with fill constrained by the initial fill-reducing ordering. Some justification for this restricted approach is provided in [23], where it is shown that the problem of computing an ordering of A with the shortest elimination tree is NP-complete, and hence intractable.

In the Jess and Kees approach, the parallel recordering algorithm is applied as a postprocessing to an initial fill-reducing ordering. It is important that this postprocessing does not markedly increase the complexity of the overall reordering scheme. Jess and Kees did not specify the implementation details of an algorithm for the second step of their scheme. Liu and Mirzaian [20] designed an algorithm that implements this step, but its complexity is unsatisfactory. Its time and space requirements are linear in the size of the filled matrix $F = L + L^T$, and the postprocessing is often more expensive than the rest of the ordering process. Indeed, more storage may be needed for the postprocessing than for the factorization of the matrix. Recognizing these requirements as unacceptably large, Liu [18] designed a heuristic composite rotations algorithm that attains *nearly* minimum tree height. This algorithm is based on the concept of elimination tree rotations and has space and time requirements that are almost linear in the size of the original matrix A. In [18] Liu empirically compares the Liu and Mirzaian implementation of the Jess and Kees method with the tree rotations heuristic and demonstrates that the heuristic is an excellent approximation in practice and is also much faster. Indeed, this heuristic also minimizes tree height on all but two of his test problems.

The motivation for this work is to reduce the complexity of computing the Jess and Kees reordering. By representing the chordal graph as a *clique tree*, we reduce both the time and storage required for the reordering algorithm to terms linear in the sum of the sizes of the maximal cliques of the filled graph. In terms of sparse matrix data structures, our algorithm has complexity bounded by the size of the array of subscripts for L after mass elimination compression. Empirically, the cost of our Jess and Kees implementation is essentially the same as the cost of Liu's heuristic. This cost is small enough so that finding an equivalent ordering with a shortest tree can be viewed as merely another small part of the recordering process.

The organization of our paper is as follows. We begin in § 2 by describing the Jess and Kees method and show how the *maximal clique* structure of the filled graph can be used to design an efficient implementation. This provides the reader with an overview of our algorithm. The graph-theoretical tools (maximal cliques and clique trees) required by our algorithm are not commonly used in sparse matrix applications. The next three sections, therefore, develop the details of these tools, emphasizing their relationships with more standard representations of sparse matrices. In addition, we extend the power and efficiency of these tools by adapting them to the specific context of sparse matrix factorization. The three sections of mathematical tools are followed by a detailed description of our algorithm. In summary, these four sections present the details of an efficient practical implementation that is obtained from the compressed row subscript lists of the sparse Cholesky factor.

The first tool we develop is a description of the maximal cliques of the filled graph; an algorithm for identifying the maximal cliques from the subscript lists of the Cholesky factor is given in § 3. In § 4 we extend the maximal clique algorithm to arrange the maximal cliques in a *clique tree* and present some important properties of the clique tree. Section 5 discusses how the maximal clique structure and the clique tree change as nodes are eliminated and, in particular, how clique trees simplify determining these changes. At this point all necessary tools are in place for an efficient parallel reordering algorithm, which is presented in § 6. This section also includes a complexity analysis showing that our algorithm is linear in the size of the compressed version of the row subscripts.

Following the detailed development of the algorithm, we present experimental results on our reordering algorithm in § 7, including empirical comparisons with the Liu and Mirzaian algorithm and Liu's heuristic algorithm. Section 8 contains some concluding observations on clique trees and directions for further work. Section 9 is an appendix in which the proofs of the major results in § 5 are given.

We use the following notation. We denote the adjacency graph of F, the *filled graph*, by G_F. The number of nodes in G_F is the order n of the matrix A. The size of the adjacency lists of G_F (twice the number of edges in G_F, also twice the number of off-diagonal nonzero entries in L), we denote by $\eta(F)$. The number of maximal cliques in G_F will be denoted by m. Let $\{K_1, K_2, \cdots, K_m\}$ be the set of maximal cliques of G_F. The total size of the maximal cliques of the graph will be denoted by q and is defined by $q = \sum_{i=1}^{m} |K_i|$. The monotone adjacency set $Madj(v)$ consists of all nodes adjacent to v and numbered higher than v, and the monotone degree $Mdeg(v)$ is the size of this set.

2. The Jess and Kees method. In this section we describe the Jess and Kees method for computing a good ordering for parallel sparse factorization and provide an overview of our clique tree algorithm that implements this method.

We begin with a matrix PAP^T, where P is a permutation matrix chosen to preserve sparsity, and we assume that a symbolic factorization has been computed. Note that the ordering $\{1, 2, \cdots, n\}$ is a perfect elimination (no-fill) ordering for the *filled matrix* $F = L + L^T$. We want to find a reordering that is more suitable for the parallel factorization of A. If we find a good parallel ordering Q from among the perfect elimination orderings of F, then we incur no additional fill (compared to PAP^T) when A is ordered by the composition of the two orderings, QP.

We now seek to motivate the Jess and Kees method. In the elimination graph model, a node v in G_F can be eliminated without causing any fill only if its adjacency set $adj(v)$ is a clique; such a node is called *simplicial* or *nondeficient*. To obtain a perfect elimination ordering of F appropriate for parallel factorization, we could attempt to eliminate all the simplicial nodes of G_F in one step. However, this is not possible in general; we are not allowed to eliminate simultaneously two nodes that are connected, even if both are simplicial.

To see this, consider two simplicial nodes j and k that share a common edge. Assume by symmetry that $j < k$. The common edge corresponds to $L_{kj} \neq 0$. By definition of the Cholesky factors, the diagonal term $L_{kk} = \sqrt{A_{kk} - \sum_{i=1}^{k-1} L_{ki}^2}$. One of the nonzero terms in the summation is L_{kj}^2, so L_{kk} cannot be computed prior to L_{kj}. However, $L_{kj} = (A_{kj} - \sum_{i=1}^{j-1} L_{ji} L_{ki}) / L_{jj}$ cannot be computed until L_{jj} is known. Thus, L_{kk} depends on L_{jj} and cannot be computed at the same step in which L_{jj} is computed.

In the graph model of the factorization, two nodes j and k can be eliminated at the same step only if no edge joins them, or equivalently, only if the nodes j and k

are *independent.* Therefore a perfect elimination ordering of G_F suitable for parallel factorization generally cannot eliminate all of its simplicial nodes in a single step, but only an independent subset of the simplicial nodes. The Jess and Kees method is a typical greedy algorithm: it eliminates a *maximum independent set* of simplicial nodes at each step, as described below.

repeat
    choose a maximum independent set of simplicial nodes;
    number these nodes consecutively and eliminate them from the graph
until
    all nodes are eliminated.

Consider an example: If F is a tridiagonal matrix, then G_F corresponds to a chain graph, and the only simplicial nodes are the two nodes at either end of the graph. The Jess and Kees method would eliminate both of these nodes in a single step, which results in a smaller chain graph. This results in the "burn at both ends" algorithm, which has the smallest number of parallel steps for factoring a tridiagonal matrix with no fill.

A fast implementation of the Jess and Kees method requires an efficient way to detect simplicial nodes and to identify a maximum independent set of simplicial nodes. As noted earlier, Jess and Kees did not specify how these operations should be performed. In the Liu and Mirzaian implementation [20], a simplicial node v is identified by a test involving the degrees and monotone degrees of nodes. These degree measures must be maintained throughout the elimination process, which requires updating these quantities for all uneliminated neighbors of each node when it is eliminated. Thus the cost of detecting simplicial nodes in their implementation is $\mathcal{O}(n+\eta(F))$. Some of this work can be eliminated by considering the degree measures as functions of cliques of simplicial nodes [22]; however, the worst-case complexity does not change. In [20] the filled graph is represented explicitly by the adjacency lists of the nodes, requiring $\eta(F)$ storage. Again, some reduction is possible. However, a better approach is to use the maximal clique structure of the chordal graph directly to identify simplicial nodes.

We exploit the maximal clique structure of the chordal filled graph and its clique tree representation to obtain a more efficient implementation of the Jess and Kees method. Recall that a clique is a set of nodes that are all pairwise adjacent. A *maximal clique* is a clique to which no other node in the graph can be added while preserving the pairwise adjacency property. Our algorithm is based on the following key properties of the maximal clique structure of chordal graphs, each of which will be verified later:

- a node is simplicial if and only if it is contained in exactly one maximal clique (§ 3, [6], [25]);
- a simplicial node is independent of the simplicial nodes in other maximal cliques, but not independent of simplicial nodes in the same maximal clique (§ 3);
- the maximal cliques can be organized into a clique tree structure (§ 4, [2], [3], [11], [21], [25], [26]);
- the elimination of simplicial nodes changes the maximal clique structure of the reduced graph; the clique tree of the reduced graph can be obtained by making simple local changes to the clique tree of the unreduced graph (§ 5).

The first of these properties is the basis for detecting simplicial nodes. It is clear from the second property that a *maximum* independent set of simplicial nodes can be found by choosing one simplicial node from each maximal clique that contains any simplicial nodes. Thus, both independence and simpliciality of nodes can be

characterized from the maximal clique structure. The use of clique trees will enable us to track the clique structure as it evolves during the elimination process. In addition, the clique tree representation of the filled graph is much more compact than its adjacency list representation; indeed, sometimes it is even more compact than the adjacency list representation of the original graph.

Our fast implementation of the Jess and Kees method is outlined below.

repeat

choose one simplicial node from each maximal clique that contains any simplicial nodes;
eliminate this maximum independent set of simplicial nodes;
update the clique tree

until

all nodes are eliminated.

By recognizing and maintaining the maximal clique structure, we reduce the tests for simpliciality and independence from operations that occur on the nodes and edges of the filled graph to operations that occur on the maximal cliques. The fundamental tools we use are first, maximal cliques, as discussed in § 3; second, the clique tree, which is described in § 4; and third, the evolution of the clique tree as it changes during the elimination, considered in § 5. Finally, in § 6, we use these results to state our parallel reordering algorithm in detail.

We assume the following context in the next three sections. A sparse symmetric matrix has been ordered by a fill-reducing procedure such as the minimum degree algorithm. To simplify notation, we assume that the matrix A has been renumbered in the fill-reducing ordering. Under this assumption, the natural ordering is a perfect elimination ordering for the filled matrix F. In addition, we assume that an efficient symbolic factorization procedure has been used to compute the row subscripts of the nonzeros in the columns of the Cholesky factor L, and that the row subscripts for each column are stored in ascending order. For simplicity, we assume that the adjacency graph of A is connected; otherwise, the generalizations from clique trees to clique forests are straightforward.

3. The maximal cliques of filled graphs. In this section we establish some fundamental properties of the maximal cliques of chordal graphs. We then present an algorithm for identifying and representing the maximal cliques using the row subscript lists obtained from a symbolic factorization. We begin with the characterization of a simplicial node in terms of maximal cliques, which exhibits the fundamental connection between the two concepts. This fundamental characterization has previously been used by Duff and Reid [6] and Tarjan and Yannakakis [25], without proof in both cases. We include the proofs of the first two propositions in this section for completeness.

PROPOSITION 1. *A node is simplicial if and only if it is contained in only one maximal clique.*

Proof. If a node v is simplicial, then, by definition, $K = adj(v) \cup \{v\}$ is a clique. K is a maximal clique because any $z \notin K$ is not adjacent to v and so $\{z\} \cup K$ is not a clique. K is the only maximal clique containing v, since any other maximal clique containing v must contain some $z \notin K$. Since z and v are not adjacent, they cannot be in the same clique.

Conversely, suppose v is not simplicial. We show that v must lie in at least two maximal cliques. By definition, $adj(v)$ is not a clique, so there exist two nodes u and $w \in adj(v)$ that are not mutually adjacent. Since $\{u, v\}$ is a clique, we can choose some maximal clique K_u containing $\{u, v\}$. Similarly there exists some maximal clique K_w

containing $\{v, w\}$. Since u and w are not adjacent, $u \notin K_w$ and $w \notin K_u$, and so K_u and K_w are distinct maximal cliques, both containing v. □

To make this characterization a practical tool, we must have an efficient way to find and represent the maximal cliques. We proceed to show that the maximal cliques are already represented in the subscript lists produced by the symbolic factorization. The following proposition was first obtained by Fulkerson and Gross [10] (see also Golumbic [13]).

PROPOSITION 2. *If K is a maximal clique in G_F and v is the lowest-numbered node in K, then $K = \{v\} \cup Madj(v)$.*

Proof. The set $K' = \{v\} \cup Madj(v)$ is a clique because the numbering is a perfect elimination ordering for G_F. Consider some node z in $K - K'$. By the choice of v, z must be numbered higher than v. But z cannot then be adjacent to v, else it belongs to K'. This contradiction shows that $K \subseteq K'$. Equality follows because K is maximal. □

Proposition 2 implies that the maximal cliques of G_F can be represented as $K(v_1)$, $K(v_2), \cdots, K(v_m)$, where $K(v)$ specifies the maximal clique $\{v\} \cup Madj(v)$ generated by the lowest-numbered or *representative* node v. If a node is not the representative of *any* maximal clique, we call it a *nonrepresentative* node. In Fig. 1, a perfect elimination ordering of a chordal graph is shown. The reader can verify that its maximal cliques are $K(1) = \{1, 2, 3\}$, $K(3) = \{3, 4, 6, 7\}$, $K(5) = \{5, 6\}$, and $K(6) = \{6, 7, 8\}$.

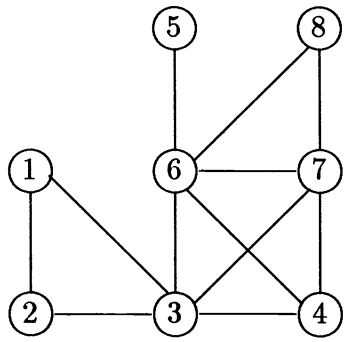


FIG. 1. *A perfect elimination ordering of a chordal graph.*

The set $Madj(v)$ is precisely the row indices of the nonzero entries in column v of L. It follows that the maximal cliques of G_F are given implicitly by the subscript lists. Each maximal clique consists of its representative node together with the subscripts in the column corresponding to the representative. Thus the symbolic factorization procedure exhibits the maximal cliques, if we identify which of the $n-1$ lists correspond to maximal cliques and which do not. Only the subscript lists corresponding to representative nodes are maximal. The problem is to recognize which nodes are representatives; the solution is to recognize the nodes that are not representatives.

PROPOSITION 3. *A node v is not a representative node for any maximal clique if and only if there exists a representative node z numbered lower than v, with $\{v\} \cup Madj(v) \subset \{z\} \cup Madj(z)$.*

Proof. If such a node z exists, then the clique $\{z\} \cup Madj(z)$ is a proper superset of $\{v\} \cup Madj(v)$ and hence $\{v\} \cup Madj(v)$ is not maximal.

Conversely, suppose there is no node z numbered lower than v with $\{v\} \cup Madj(v) \subset \{z\} \cup Madj(z)$. By Proposition 2, any maximal clique containing $\{v\} \cup Madj(v)$ has the form $\{w\} \cup Madj(w)$, where w is its representative node. By assumption, $w \geqq v$, from which it follows that v must be the lowest-numbered node in any maximal clique containing $\{v\} \cup Madj(v)$. Hence by Proposition 2, v is the representative node that generates the maximal clique $K(v) = \{v\} \cup Madj(v)$. □

The following Proposition provides an efficient test of whether $\{v\} \cup Madj(v) \subset \{z\} \cup Madj(z)$, using only the sizes of the subscript lists. The relevant quantities can be easily obtained since $Madj(z)$ is the subscript list for column z, and the monotone degree $Mdeg(z)$ is the size of this list.

PROPOSITION 4. *Let $K(v)$ be a maximal clique, with representative v and nodes $v < w_1 < w_2 \cdots < w_p$. Then w_i is not a representative node for any maximal clique if $Mdeg(w_i) = p - i$.*

Proof. Since the nodes are in a perfect elimination order,

$$\begin{aligned} Madj(v) &\subseteq w_1 \cup Madj(w_1) \\ &\subseteq \{w_1, w_2\} \cup Madj(w_2) \\ &\cdots \\ &\subseteq \{w_1, \cdots, w_p\} \cup Madj(w_p). \end{aligned}$$

This implies that $Mdeg(v) \geqq Mdeg(w_1) + 1 \geqq Mdeg(w_2) + 2 \geqq \cdots \geqq Mdeg(w_p) + p$. Assume $Mdeg(v) - i = p - i = Mdeg(w_i)$. It follows that equality holds for every set inclusion in the chain from $\{w_1, \cdots, w_i\} \cup Madj(w_i)$ down to $Madj(v)$, so that these two sets are equal. Then by Proposition 3, w_i is not a representative node. □

Although Proposition 4 is stated as a test on a single node w_i, it should be clear from the proof that when $Mdeg(w_i) = p - i$, in addition to w_i, the nodes $w_1, \cdots, w_{i-1}$ are also nonrepresentative nodes. On the other hand, if we take w_j to be the lowest-numbered node in $K(v)$ satisfying $Mdeg(w_j) > p - j$, then $Mdeg(w_k) > p - k$ for each k, $j \leqq k \leqq p$. Thus Proposition 4 divides the nodes $\{w_1, \cdots, w_p\}$ into two classes: $w_1, \cdots, w_{j-1}$ are identified as nonrepresentative, while the status of $w_j, \cdots, w_p$ cannot be determined by examining $K(v)$ alone.

Our algorithm for determining the maximal cliques of G_F does so by identifying which nodes are representatives and which nodes are not. Initially we mark all nodes as possible representatives, and process nodes in the given perfect elimination order. If a node v has not been marked as nonrepresentative by the time it is processed, no earlier node can satisfy the role of z in Proposition 3 and so we accept v as a representative. We then examine $Madj(v)$ and mark nodes satisfying the degree test in Proposition 4 as nonrepresentative nodes, stopping when a node fails the degree test. An informal algorithm follows.

```
mark all nodes as possible representatives;
for v = 1 to n do
    if v is still a possible representative then
        accept v as a representative;
        mark the nodes in Madj(v) in order as nonrepresentative
        until a node fails the degree test.
    endif
endfor
```

This informal algorithm is an obvious use of Proposition 4 to identify the nonrepresentative nodes. However, the marking procedure is redundant because each nonrepresentative node w is marked as such in every maximal clique containing it,

when the conditions of Proposition 4 are satisfied. We obtain a more efficient procedure by terminating the marking procedure as soon as a node already marked as a nonrepresentative node is encountered. The following Proposition is the basis for showing that this termination rule will not fail to mark any nodes as nonrepresentatives.

PROPOSITION 5. *Suppose that* $w_1 < w_2 < \cdots < w_{j-1}$ *are identifiable in* $K(v)$ *as nonrepresentative nodes by Proposition* 4. *Suppose that for some* $i \leqq j-1$, w_i *is also identifiable as a nonrepresentative node in* $K(u)$, *where* $u < v$. *Then* $w_{i+1}, \cdots, w_{j-1}$ *are also identifiable in* $K(u)$ *as nonrepresentative nodes.*

Proof. Let $K(u) = \{u\} \cup \{z_1, \cdots, z_{l-1}\} \cup \{z_l, \cdots, z_t\}$, where the nodes are listed in increasing perfect elimination order. Let $w_i = z_k \in \{z_1, \cdots, z_{l-1}\}$ be identifiable as a nonrepresentative node in $K(u)$. Then, as we saw in the proof of Proposition 4 and the discussion following it,

$$Madj(u) = \{z_1, \cdots, z_{k-1}, w_i\} \cup Madj(w_i),$$

$$Madj(w_i) = \{w_{i+1}, \cdots, w_{j-1}\} \cup Madj(w_{j-1}),$$

from which it follows that

$$Madj(u) = \{z_1, \cdots, z_{k-1}\} \cup \{w_i, w_{i+1}, \cdots, w_{j-1}\} \cup Madj(w_{j-1}).$$

Hence, w_{j-1} is identifiable as a nonrepresentative node in $K(u)$. The same argument obviously holds for $\{w_{i+1}, \cdots, w_{j-2}\}$ as well. □

In the more efficient procedure, we stop the search for nonrepresentative nodes as soon as we find either a node that fails the degree test or a node that has already been marked as nonrepresentative. We now show that a nonrepresentative node w will be marked exactly once—while processing the first maximal clique $K(v)$ in which w can be identified as nonrepresentative. To see this, consider the marking procedure for the first such maximal clique $K(v)$. By Proposition 4, nodes in $K(v)$ numbered lower than w cannot fail the degree test. If the marking stops earlier, it stops only because a node in $K(v)$ numbered less than w was marked during the processing of $K(z)$, with $z < v$. But then Proposition 5 implies that w is identifiable as nonrepresentative in $K(z)$, while we assumed that v was the lowest-numbered node generating such a maximal clique. The contradiction means that w must be marked at step v.

The modified algorithm for finding the representative nodes imposes the following structure on the maximal clique $K(v)$, where w_l is the last node examined in $K(v)$:

$$new(K(v)) = \{v, w_1, \cdots, w_{l-1}\}$$

$$anc(K(v)) = \{w_l, \cdots, w_p\}$$

$$first_anc(K(v)) = w_l.$$

The notation adopted for these sets will become more meaningful when we use these sets to organize the maximal cliques into a clique tree. The set $new(K(v))$ consists of the representative v together with the nodes marked as nonrepresentative while examining $K(v)$. The set $anc(K(v))$ (read "ancestor set of $K(v)$") consists of the nodes in $K(v)$ that either fail the degree test or were marked as nonrepresentative in some other maximal clique $K(u)$ with $u < v$. The node $first_anc(K(v))$ (read "first ancestor node in $K(v)$"), the last node examined at $K(v)$, is the lowest-numbered node in $anc(K(v))$. It will play a crucial role in the clique tree structure.

Again it might be helpful to the reader to consider the *new*, *anc* partition for the maximal cliques in Fig. 1. These sets are as follows:

$$new(K(1)) = \{1, 2\} \qquad anc(K(1)) = \{3\}$$
$$new(K(3)) = \{3, 4\} \qquad anc(K(3)) = \{6, 7\}$$
$$new(K(5)) = \{5\} \qquad anc(K(5)) = \{6\}$$
$$new(K(6)) = \{6, 7, 8\} \qquad anc(K(6)) = \varnothing.$$

The partition of a maximal clique into the sets $new(K)$ and $anc(K)$ has two important properties that we will use later in defining a clique tree from the maximal cliques. First, the first ancestor node of K splits K into the ordered subsets, $new(K)$ and $anc(K)$; all nodes in $new(K)$ are numbered lower than the first ancestor, and the other nodes in $anc(K)$ are numbered higher than the first ancestor. Second, the sets $\{new(K(v_1)), new(K(v_2)), \cdots, new(K(v_m))\}$ form a partition of the nodes of the chordal graph, since every node is chosen as a representative or marked as a nonrepresentative exactly once.

The *Maximal Cliques Algorithm*, shown in Fig. 2, finds the representative nodes and first ancestor nodes for each maximal clique in G_F, using the modified marking scheme. To emphasize the connection between the maximal cliques and the subscripts of the factor L, we use the notation L_{*v} to denote the subscripts of the nonzero entries in the vth column of L, i.e., $Madj(v)$. This procedure also computes other quantities that will be used later to generate the clique tree for the graph G_F. The maximal cliques

```
clique_num := 0;
for v := 1 to n do
    new_in_clique(v) := nil;
    clique_count(v) := 0;
endfor
{nodes will be marked by setting new_in_clique non-nil}

for v := 1 to n do
    if new_in_clique(v) = nil then
        {v is a representative node, K(v) is a maximal clique}
        clique_num          := clique_num + 1;
        clique_size         := |L_{*v}|;
        new_in_clique(v)    := clique_num;
        clique_count(v)     := clique_count(v) + 1;
        #_new(clique_num)   := 1;
        first_anc(clique_num) := nil;
        for w ∈ L_{*v} in increasing order do
            clique_count(w) := clique_count(w) + 1;
            if first_anc(clique_num) = nil then
                clique_size := clique_size - 1;
                if clique_size = |L_{*w}| and new_in_clique(w) = nil then
                    #_new(clique_num) := #_new(clique_num) + 1;
                    new_in_clique(w)  := clique_num;
                else
                    first_anc(clique_num) := w;
                endif
            endif
        endfor
    endif
endfor
```

FIG. 2. *Maximal Cliques Algorithm.*

are numbered from 1 to m, using the variable *clique_num*. The number of maximal cliques containing v, *clique_count*(v), will enable us to identify simplicial nodes. #_*new*(K), the number of nodes in *new*(K), will be used in the fast Jess and Kees method in a test on merging cliques. The mapping *new_in_clique*(v) = K means that $v \in$ *new*(K). This mapping will be essential in computing the clique tree.

The complexity of the algorithm is easily analyzed. The initializations require $\mathcal{O}(n)$ time. The processing of the nodes, excluding the examination of the *Madj*(v) sets, also cost $\mathcal{O}(n)$ time. Since the *Madj*(v) sets are examined only for representative nodes, and the cost of the operations associated with a node $w \in$ *Madj*(v) is a constant, the total cost of the latter operations is proportional to the size of the maximal cliques, which is $\mathcal{O}(q)$. Thus the overall complexity is $\mathcal{O}(n+q)$.

4. Clique trees. In this section we demonstrate how the maximal cliques of a chordal graph can be joined together in a meaningful tree structure. The concept of a *clique tree* has previously appeared in [2], [21], [25], and proofs that a clique tree can be constructed if and only if the graph is chordal can be found in the first two references. However, we require a more restrictive definition of the clique tree to obtain a linear time clique tree Jess and Kees algorithm. We describe this restriction at the end of this section. Our work also differs from the previous authors' in taking advantage of the additional information available to us: a perfect elimination ordering of the filled graph and a compact representation of its maximal cliques.

Our goals here are, first, to demonstrate that a clique tree can be constructed from the *Maximal Cliques Algorithm*, and, second, to elucidate some properties of clique trees. We will use these properties in the following section to show how clique trees express the changing clique structure as simplical nodes are eliminated from the filled graph by the Jess and Kees algorithm.

Recall that the sets $\{$*new*(K_1), *new*(K_2), $\cdots$, *new*(K_m)$\}$, defined by the *Maximal Cliques Algorithm*, partition the nodes of the graph because every node is either chosen as a representative or marked as nonrepresentative exactly once. As a result, the following *parent* function is well defined if *anc*(K) $\neq \emptyset$:

$$parent(K) = \text{the clique } P(K) \text{ such that } first_anc(K) \in new(P(K)).$$

The *clique tree* has a node corresponding to each maximal clique, and an edge joining each maximal clique to its parent. The root of the clique tree has *anc*(K) = $\emptyset$. We let $P(K)$ denote the parent of a maximal clique K. Let *first_anc*(K) = w. Then $w \in$ *new*($P(K)$) implies that $w <$ *first_anc*($P(K)$). Hence there cannot be any cycles of the form K, $P(K)$, $P(P(K))$, $\cdots$, K, which shows that the *parent* function does indeed define a tree.

The clique tree corresponding to the perfect elimination order of the chordal graph in Fig. 1 is shown in Fig. 3. The partition of each maximal clique into the *new* and *anc* sets is shown by writing the nodes in *anc*(K) above the nodes in *new*(K). The first ancestor node of each maximal clique is the first listed in *anc*(K), and the reader can verify that the first ancestor node of each maximal clique is a *new* node in its parent clique.

We now prove some elementary relationships between a child clique and its parent.

PROPOSITION 6. *Let w be the first ancestor node in a maximal clique K and let $P(K)$ be the parent of K. Then*

- $w \in$ *new*($P(K)$),
- *anc*(K) $\subset P(K)$,
- *new*(K) $\cap P(K) = \emptyset$.

Proof. The first statement follows immediately from the definition of a parent

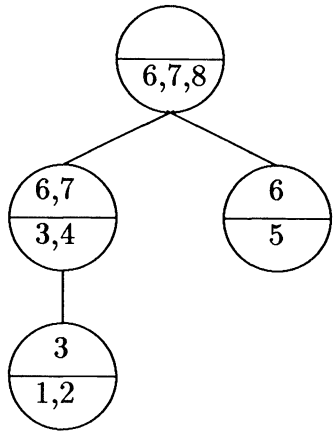


FIG. 3. *The clique tree of the chordal graph with perfect elimination order shown in Fig.* 1.

clique. To prove the second, note that by definition of the first ancestor node we have $anc(K) \subset \{w\} \cup Madj(w)$. Because $w \in new(P(K))$, either w is the representative of $P(K)$ or w was marked as a nonrepresentative in $P(K)$ by the *Maximal Cliques Algorithm.* In either case $\{w\} \cup Madj(w) \subseteq P(K)$. Thus $anc(K) \subset P(K)$.

To prove the third statement, suppose there exists a node $y \in new(K) \cap P(K)$. Since $y \in new(K)$, and w is the first ancestor of K, $y < w$ by the definition of the sets $new(K)$ *and* $anc(K)$. Now $w \in new(P(K))$ and $y < w$ imply that $y \in new(P(K))$, again by the definition of the sets $new(P(K))$, and $anc(P(K))$. Then $y \in new(P(K)) \cap new(K)$, and this contradicts the fact that the *new* sets partition the nodes. □

The *Maximal Cliques Algorithm* partitions each maximal clique into the sets $anc(K)$ and $new(K)$. The previous result shows that the clique tree also splits each maximal clique K into the same two sets: The nodes in $new(K)$ are not in the parent; each node in $anc(K)$ is also in $P(K)$.

Proposition 6 gives a local characterization of the *new, anc* partition. The following theorem presents a global interpretation of this partition—viewed from the root R of the tree, the *new* nodes appear first in the clique that marked them as *new,* and the *anc* nodes appear first in some ancestor of this clique.

THEOREM 7. *Given a maximal clique* K *and the partitioning* $K = new(K) \cup anc(K)$,

- *a node* $u \in new(K)$ *belongs to another maximal clique* D *only if* D *is a descendant of* K *in the clique tree;*
- *a node* $u \in anc(K)$ *belongs to new* (A), *where* A *is some ancestor of* K *in the clique tree, and* u *also belongs to all maximal cliques on the path between* K *and* A.

Proof. For the first part, assume that there is some maximal clique $J \neq K$ that is not a descendant of K and yet contains a node u in $new(K)$. Choose J to be a maximal clique at the least distance from the root R satisfying this property. The node u cannot be a member of $new(J)$, lest we contradict the partitioning principle. However, $u \notin new(J)$ implies $u \in anc(J)$, which, by Proposition 6, implies that u is in J's parent, $P(J)$. But $P(J)$ is then closer to the root than J, contradicting the choice of J. (The special case where $J = R$ leads to a contradiction because $anc(J) = \emptyset$.)

To prove the second part, let $u \in anc(K)$. Since the *new* sets partition the nodes, $u \in new(A)$ for some maximal clique A. It follows from the first part of the theorem that K must be a clique in the subtree rooted at A. Thus A is an ancestor of K. It remains only to prove that u belongs to all the intermediate ancestors $P(K) = I_1$,

$P(P(K)) = I_2, \cdots, I_p = A$. Suppose not. By Proposition 6, $u \in P(K) = I_1$. Hence, there must exist a least index $2 \leqq j < p$ such that $u \in I_{j-1}$ and yet $u \notin I_j$. By the same Proposition, $u \in I_{j-1}$ implies either $u \in new(I_{j-1})$ or $u \in P(I_{j-1}) = I_j$. The assumption that $u \notin I_j$ then requires u to belong to $new(I_{j-1})$, which again violates the fact that the *new* sets form a partition. □

We now consider previous approaches to defining clique trees. Buneman [3], Gavril [11], and Walter [26] (see also [13, Thm. 4.8]) were concerned with characterizing chordal graphs as intersection graphs of subtrees of a tree. Let $\mathcal{K}$ be the set of maximal cliques of a chordal graph. They proved that it is possible to construct a tree $T = (\mathcal{K}, \mathcal{E})$ connecting the maximal cliques with the property that each induced subgraph $T_{\mathcal{K}_v}$ is connected, where $\mathcal{K}_v$ consists of all the maximal cliques containing a vertex v of the chordal graph. Theorem 7 shows that our definition of a clique tree satisfies the requirements of a clique tree in the sense of these authors. In the database theory literature [2], [25], a clique tree (called a *join tree* in this context) can be defined in terms of one of several equivalent properties that characterize acyclic hypergraphs.

Our definition is more restrictive than either of these approaches. We require the first ancestor node w of a clique K to be a *new* node in its parent clique $P(K)$. Since $w \in new(P(K))$, w does not belong to any other ancestor clique of K. Thus, by our construction of a clique tree, the parent $P(K)$ is the *only* ancestor clique that contains the set $anc(K)$; all other ancestors of K contain only a proper subset of $anc(K)$. This property will lead to simple, local updating rules for changing the clique tree as nodes are eliminated in the Jess and Kees method. Previous definitions of a clique tree do not share this property, since they require a clique tree to satisfy only the last two conditions in Proposition 6.

5. Reduced clique trees. The preceding sections describe the maximal clique structure of a filled graph and its clique tree representation. This section shows how to update the clique tree to correspond to the reduced graphs created as nodes are eliminated during the Jess and Kees method. Technically this is the most demanding section of the paper. To keep this section to a reasonable length and to move the reader forward to our goal, the clique tree Jess and Kees algorithm, we present the major results without proof. The proofs are given in the Appendix.

The Jess and Kees method produces another perfect elimination ordering of the filled graph in a sequence of steps. At each step a maximum independent set of simplicial nodes (and the edges incident on them) are removed from the graph, producing a sequence of reduced (elimination) graphs. Since we delete only simplicial nodes, no new edges are created in the elimination graphs. It is well known that each of the reduced graphs is also chordal.

OBSERVATION 1. *Let G^* be the reduced subgraph obtained by applying a single step of the Jess and Kees method to a chordal graph G. Then*

(1) *a clique of G^* is also a clique of G;*

(2) *G^* is a chordal graph.*

Thus, the Jess and Kees method generates a sequence $G_F \equiv G^{(0)}$, $G^{(1)}$, $G^{(2)}, \cdots, G^{(h)} \equiv \varnothing$ of chordal graphs, where h is the total number of steps in the method. The goal of this section is to demonstrate that there is a simple set of transformations that can be applied to generate a corresponding sequence of clique trees. This will also give an efficient method for identifying new simplical nodes in the elimination graphs. Throughout this section we will describe the stepwise transformation of the clique tree representing a chordal graph $G^{(i)}$ into a clique tree representing its successor $G^{(i+1)}$. For simplicity, we shall drop the superscripts and let G denote

an arbitrary member (other than the last) of the sequence of chordal graphs and let G^* denote its successor.

Consider the effect of the reduction from G to G^* on the maximal cliques. Let K be a maximal clique in G and define its *residual clique* in G^* by the following:

$$R(K) = \begin{cases} K & \text{if } K \text{ contains no simplicial nodes in } G, \\ K - \{u\} & \text{where } u \text{ is the eliminated simplicial node of } K \text{ in } G. \end{cases}$$

We call the cliques from which a simplicial node is eliminated *shrinking cliques.* It is evident that the nonshrinking cliques are unchanged and are maximal in G^* as well as G.

All of the action occurs with the shrinking cliques. In particular, $R(K) = K - \{u\}$ need not be a maximal clique in G^*. When this occurs, there will be some other maximal clique J of G such that $R(K)$ is contained in a maximal clique $R(J)$ of G^*. We say that $R(K)$ has been *absorbed* by $R(J)$ and we call $R(J)$ an *absorbing clique.* However, it should be borne in mind that no nodes are added to the absorbing clique; we shall show that the only changes made to the absorbing clique are in its relationships to its neighbors in the clique tree. We shall also show that the clique tree provides the means for finding the absorbing clique $R(J)$ at very little cost.

A clique K of G_F is naturally associated with a unique clique $K^{(1)} = R(K)$ in $G^{(1)}$, $K^{(2)} = R(R(K))$ in $G^{(2)}, \cdots, K^{(i)} = R^{(i)}(K)$ in $G^{(i)}$. For ease of notation, we will associate the same name K with the cliques $K^{(1)}, K^{(2)}, \cdots, K^{(i)}$ until such point that $R^{(i)}(K)$ becomes a nonmaximal clique, in which case we say that the clique (and the name) K disappear.

An understanding of how the maximal clique structure changes, when one maximal clique shrinks to the point of being absorbed by another clique, is crucial for a fast implementation of the Jess and Kees algorithm. These changes are described in the following theorem.

THEOREM 8. *Let G be a chordal graph with clique tree T. Let K be a shrinking clique in G, with residual clique $R(K)$ in G^*.*

- *$R(K)$ is nonmaximal only if K contained only one simplicial node.*
- *If $R(K)$ is nonmaximal, and K is a leaf clique in T, then $R(K)$ is contained in the residual clique of its parent $P(K)$.*
- *If $R(K)$ is nonmaximal, and K is an interior clique in T, then $R(K)$ is contained in the residual clique of one of its children, specifically a child C whose ancestor set is largest.*

Identifying newly nonmaximal cliques is central to the Jess and Kees algorithm, since it is only when cliques become nonmaximal that the clique count for any node can decrease. It is the mechanism by which nodes become simplicial. Corollary 9 gives us a simple test to identify nonmaximal cliques at each step of the algorithm.

COROLLARY 9. *Consider a shrinking clique K whose last simplicial node is eliminated. Then $R(K)$ is nonmaximal if and only if either $|new(R(K))| = 0$ or a child C of K with the largest ancestor set size satisfies $|anc(C)| = |R(K)|$.*

The next goal is to build a clique tree for G^*. To avoid a possible ambiguity that arises from eliminating a maximum independent set of simplicial nodes simultaneously, we arbitrarily order the simplicial nodes that are eliminated from G at this step as $u_1, u_2, \cdots, u_t$. We will build the clique tree for G^* from the clique tree for G in t minor steps, where the ith step removes u_i from $G - \{u_1, \cdots, u_{i-1}\}$ and properly alters the clique tree. Note that in a given minor step in which a node u_i is removed, u_i belongs to exactly one clique K; for all other cliques $\tilde{K}$, it follows that $R(\tilde{K}) = \tilde{K}$. The rules for transforming the tree at each step are described in the following Theorem.

THEOREM 10. *Let G be a chordal graph with a clique tree T. Let K be a maximal*

clique containing a simplicial node u. Then a clique tree for $G-\{u\}$ can be constructed by the following rules:

- *If $R(K)$ is maximal, make no changes.*
- *Otherwise,*
  - *— if K is a leaf of T, delete K from T.*
  - *— if K is an interior clique of T, let C be a child with the largest ancestor set among the children of K. Then*
    - ** assign all nodes in new(K) to new(C), removing them from anc(C);*
    - ** attach C as a child of P(K), with first_anc(C) reset to first_anc(K);*
    - ** attach children of K other than C as children of C;*
    - ** delete K from the tree.*

We need to check the maximality of a clique only when a clique loses its last simplicial node. Clearly, maintaining a count of the number of *new* nodes suffices for leaf cliques. For an interior clique we need to find a child that can absorb it, but we would prefer not to have to search through all of its children. This is easy to do in the initial clique tree if we sort the children of each clique by ancestor set size. (This sorting can be done in $O(n)$ time by a bucket sort.) Then, the first child can absorb its nonmaximal parent. However, this becomes a subtle issue for a clique tree at some intermediate stage in the Jess and Kees method. During the elimination process, absorbing cliques may acquire new children, and the child lists may no longer be ordered by decreasing ancestor set size unless we re-sort the child list of each clique that acquires new children.

The next two results show that such re-sorting is not necessary. The initial child ordering by ancestor size may be destroyed by interior clique absorptions, but a careful treatment of the child lists can maintain an ordering in which only the first child needs to be considered as a candidate for absorbing a parent clique. These results will turn out to be crucial in obtaining a linear time bound for our implementation of the Jess and Kees method.

PROPOSITION 11. *Let C_1 and C_2 be siblings in the clique tree for G_F, with $|anc(C_1)| \geqq |anc(C_2)|$. Then $|anc(C_1)| \geqq |anc(C_2)|$ holds at all later stages of the method as long as C_1 and C_2 continue to be siblings.*

The gist of Proposition 11 is that the original children of K remain in the same relative order, even if some of the children are eliminated from the tree. Thus, the elimination of a leaf of the clique tree requires no changes in how its siblings are listed. When an interior clique is absorbed, a simple ordering of the revised child lists will enable us to continue testing only the first child for absorbing its parent clique.

Our rule for arranging the revised child lists when an interior clique is absorbed is as follows. Let C_1 be the first child of K, so that $R(C_1)$ absorbs $R(K)$. Let the other children of K be ordered $C_2, \cdots, C_i$, and let the children of C_1 be ordered $D_1, \cdots, D_j$. After the update of the tree, arrange the children of C_1 in the order $D_1, \cdots, D_j$, $C_2, \cdots, C_i$. This requires only the minimal cost of appending one list of children to the end of another list. Figure 4 illustrates this rule. The next theorem shows that this rule can be applied whenever an interior clique is absorbed during elimination.

THEOREM 12. *Assume that the lists of children in the clique tree are initially sorted by decreasing ancestor set size, and that the child lists are revised by the above rule when an interior clique is absorbed. Then, at any future step in the elimination, a nonmaximal interior clique can be absorbed by its first child.*

This result would not be true if any of the more general definitions of the clique tree were used; the proof (provided in the appendix) makes vital use of the fact that the first ancestor node of a clique is a *new* node in its parent clique.

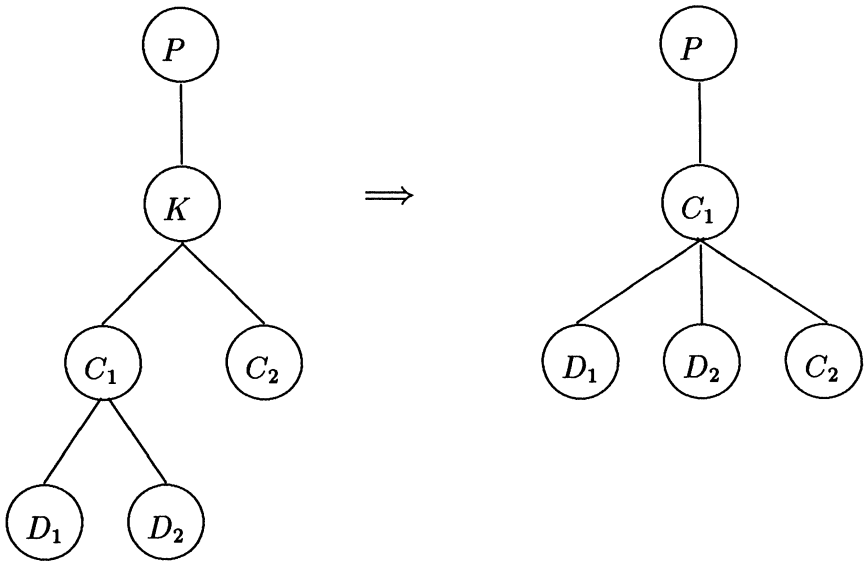


FIG. 4. *Updating the clique tree when an interior clique K is absorbed by its first child,* C_1.

We conclude this section with a discussion of the operations of downdating clique memberships counts and of finding new simplicial nodes. In both cases the proper view simplifies the data structures required to make the identification.

The count of clique membership for a node changes only when a clique containing it is absorbed by another clique. The rule for changing membership counts is the following: When a clique $R(K)$ is absorbed, reduce the clique count by one for each node in $R(K)$. In the case of a leaf clique, this set of nodes is $anc(K)$. When an interior clique is absorbed, this set is $anc(C)$, where C is the absorbing child. In both cases, the set of nodes is a contiguous trailing segment of the list of row subscripts for a representative column of L and can be obtained easily from the array of row subscripts.

Which nodes can be simplicial at the next step of Jess and Kees? Which cliques will be the shrinking cliques at the next step? Any node that was simplicial and was not eliminated remains simplicial. Thus, any shrinking clique whose set of simplicial nodes was not exhausted remains a shrinking clique. The only nodes that can become newly simplicial are those whose clique membership counts become one when downdated. Such nodes belong to a clique that became nonmaximal and also to the clique that absorbed the nonmaximal clique. Hence, only a maximal clique that absorbed another clique can possibly become a new shrinking clique. Thus we have proved the corollary, which follows.

COROLLARY 13. *Let* G^* *be the graph resulting from elimination of an independent set of simplicial nodes from G. Then*

- *a node can be simplicial in* G^* *only if it was simplicial in G or it lies in a clique that absorbed another clique during this elimination*;
- *a clique of* G^* *can be a shrinking clique only if it was a shrinking clique for G or it absorbed some other clique during this elimination.*

6. A parallel reordering algorithm. We are now ready to describe our parallel reordering algorithm. We assume that the initial clique tree T has been constructed, that the maximal cliques are numbered from 1 to m, and that the children of each maximal clique are ordered by decreasing ancestor set size. We represent each maximal

clique K in the clique tree by the list of nodes in $anc(K)$. Our algorithm does *not* require the list of nodes in $new(K)$—it will suffice to maintain the number $|new(K)|$ of such nodes. As noted previously, the subscript lists for the Cholesky factor L suffice as the list of *anc* nodes.

In addition, we assume that we are given for each node v, $clique_count(v)$, the number of maximal cliques to which v belongs, and $new_in_clique(v)$, the maximal clique in which v is a *new* node. We also require the list $\mathscr{L}$ of shrinking cliques, and the list of simplicial nodes $S(K)$ in each shrinking clique K. The list $\mathscr{L}$ is maintained as a linked list, and each list $S(K)$ is maintained as a queue. $R(K)$ will denote the residual set of nodes immediately after the removal of a simplicial node from K.

To update the clique tree when a clique becomes nonmaximal, we may require the parent of a leaf clique, or the parent and children of an interior clique. Hence we represent the clique tree in terms of both the parent relationship and the child relationship. The latter representation is explicitly maintained by means of the *first_child* and *right_sibling* lists. The *right_sibling* list is maintained as a doubly linked list to permit constant time deletions of any clique in the list.

We do not represent the parent relationship explicitly by means of the *parent* vector; it does not seem possible to update the *parent* vector explicitly and obtain a linear algorithm. Instead, we maintain the parent relationship implicitly by means of the vectors *first_anc* and *new_in_clique*. If $first_anc(K) = v$, then v is a *new* node in K's parent P. Hence $new_in_clique(v) = P$. Thus the *first_anc* and *new_in_clique* vectors together enable us to identify the parent clique in constant time. We note again that the restricted definition of the clique tree is necessary to update the implicit representation of the parent in linear time.

The *Parallel Ordering Algorithm*, shown in Fig. 5, computes the parallel reordering from the clique tree, numbering nodes from 1 to n. Each iteration of the **while** loop corresponds to one step of the Jess and Kees method. In each step, one simplicial node is eliminated from each shrinking clique in $\mathscr{L}$. After the elimination of a simplicial node v from a shrinking clique K, the clique tree and other data structures are updated as necessary. The list $\mathscr{L}_{new}$ is the list of new shrinking cliques. At the end of each step, these new shrinking cliques are added to the existing list of shrinking cliques.

An auxiliary procedure, *Downdate*, shown in Fig. 6, is used to decrement $clique_count(v)$ for each node v in a clique K that becomes nonmaximal; to update the queue $S(J)$ of simplicial nodes in the clique J that absorbs K; and to add, if necessary, the absorbing clique J to the list $\mathscr{L}_{new}$. This procedure is shown separately.

The operations associated with a nonmaximal interior clique K require some comment. When such a clique K is deleted, the set of nodes in it, $R(K)$, is precisely the set $anc(C)$ for its first child C. After the absorption of K, $first_anc(C)$ needs to be reset to $first_anc(K)$, and the partition $new(C)$, $anc(C)$ needs to be updated. By our rules, all nodes in $new(R(K))$ are removed from $anc(C)$ and added to $new(C)$. We now discuss how this is accomplished, since we do not explicitly maintain the set $new(R(K))$, but only its size.

We assume that the symbolic factorization procedure used orders the indices in each subscript list in ascending order; thus each set $anc(C)$ is a contiguous list of nodes in ascending order. Since $R(K) = anc(C)$, and nodes in $new(R(K))$ are numbered lower than nodes in $anc(R(K))$, the nodes in $new(R(K))$ are the first $|new(R(K))|$ nodes in $anc(C)$. For these nodes, we reset $new_in_clique(v)$ to be the clique C, and then $first_anc(C)$ is the node immediately following these nodes in $anc(C)$. We perform the *Downdate* operation first, then update $first_anc(C)$, and thereby implicitly, the set $anc(C)$.

In the remainder of this section we verify our claims regarding the complexity of

```
Make L_new the empty list;
while L is not empty do
    for each K ∈ L do
        number the first simplicial node v ∈ S(K);
        remove v from S(K) and decrement |new(K)|;
        if S(K) is empty then
            delete K from L;
            if K is a leaf then {the residual clique R(K) is anc(K)}
                delete K from T;
                Downdate(anc(K), P(K));
            else
                C := first_child(K);
                if |R(K)| = |anc(C)| then {the residual clique R(K) is anc(C)}
                    v := first_anc(C);
                    for i := 1 to |new(K)| do
                        new_in_clique(v) := C;
                        v := next node in anc(C);
                    endfor
                    Downdate(anc(C), C);
                    first_anc(C) := v;
                    update |new(C)| and |anc(C)|;
                    delete K from T and replace K by C as child of P(K);
                    append other children of K to end of the child list of C;
                endif
            endif
        endif
    endfor
    Add the shrinking cliques in L_new to L;
    Make L_new the empty list;
endwhile
```

FIG. 5. *Parallel Ordering Algorithm.*

```
procedure Downdate(U, J)
{downdate clique_count(u) for the set of nodes U in an absorbing clique J, and identify new simplicial
nodes and new shrinking cliques}
for each u ∈ U do
    clique_count(u) := clique_count(u) − 1;
    if clique_count(u) = 1 then
        {identify u as a new simplicial node}
        if S(J) is empty then
            set the queue S(J) := {u};
            Add J to the list L_new;
        else
            Add u to the queue S(J);
        endif
    endif
end for
```

FIG. 6. *Downdate procedure.*

the algorithm. Initially, we can sort the children of all the maximal cliques with a careful bucket sort in $\mathcal{O}(n)$ time. Clearly, in the *Parallel Ordering Algorithm* as written, the only nonconstant work in deleting a nonmaximal clique comes in the $\mathcal{O}(|R(K)|)$ operations in the call to *Downdate* and resetting the *new_in_clique* values. Let $\overline{R(K)}$ be the residual clique of K when it becomes nonmaximal. Each call of *Downdate* requires $\mathcal{O}(|\overline{R(K)}|)$ time, and hence the total cost of all calls of *Downdate* is

$$\sum_{K \in T} \mathcal{O}(|\overline{R(K)}|) \leq \sum_{K \in T} \mathcal{O}(|K|) = \mathcal{O}(q).$$

We now compute the complexity of the rest of the algorithm. The cost of eliminating simplicial nodes, without considering clique tree updates, is $\mathcal{O}(n)$. There are m cliques initially, so the constant work of removing a clique takes $\mathcal{O}(m) \leqq \mathcal{O}(n)$ time overall. Thus the algorithm has time complexity $\mathcal{O}(n+q)$.

7. Empirical results. In this section we present experimental results on the parallel reordering algorithm, and compare the performance of our algorithm with the Liu and Mirzaian algorithm and Liu's composite rotations heuristic.

We have written Fortran 77 codes for the *Maximal Cliques Algorithm* and the *Parallel Ordering Algorithm.* An additional procedure was written to create the data structures required by the latter algorithm. Each maximal clique K was represented by the nodes in $anc(K)$; this set of nodes is already represented in the subscript lists for L, if we maintain a pointer to the first ancestor node of each maximal clique. The subscript lists for L were generated by a symbolic factorization of A.

We wrote an experimental program for two purposes. One is to verify empirically the complexity analysis of our algorithm. The other is to provide a comparison with Liu's composite rotations heuristic, previously the only practical choice. Unfortunately Liu's code was not made available to us, so the data available for comparison is limited to the data published by Liu.

Liu used eleven problems from the Harwell-Boeing test set [5], [4] to test his algorithm; we obtained results on the same eleven problems. These are described in Table 1. The storage statistics for the Cholesky factorizations of these matrices, after ordering with the multiple minimum degree heuristic [16], are given in Table 2. These statistics give the primary terms in the complexities of each of the three algorithms. Liu's heuristic requires time almost linear in the number of nonzeros in A; the Liu and Mirzaian algorithm runs in time linear in the number of nonzeros in $L+L^T$; and our algorithm is linear in the number of elements in the mass-elimination compressed subscript array. It is clear from Table 2 that the size of the representation of the original matrix A and the size of the compressed subscript representation for L are similar and much smaller than the size of L. Table 2 also lists the number of maximal cliques in the filled graph for each problem.

Our experiments were performed on a Sun 3/260 workstation running Sun Unix 3.3, and we used the f77 compiler with the optimization turned on. Because of the coarse granularity of the timing subroutines on this machine, we obtain the times by averaging over several hundred repetitions. In Table 3, we report the CPU times required by the four steps in computing the parallel ordering. First is the fill-reducing

TABLE 1
Test matrices from the Harwell-Boeing sparse matrix collection.

Key	Description	Order
BCSPWR09	Western US power grid	1723
BCSPWR10	Eastern US power grid	5300
BCSSTK08	TV studio structure	1074
BCSSTK13	Fluid Flow Stiffness Matrix	2003
BCSSTM13	Fluid Flow Mass Matrix	2003
BLCKHOLE	Geodesic Dome	2132
CAN 1072	Aircraft structure	1072
DWT 2680	Naval destroyer structure	2680
LSHP3466	L-shaped regular grid	3466
GR 40×40	40-by-40 square grid	1600
GR 80×80	80-by-80 square grid	6400

TABLE 2
Fill statistics from the multiple minimum degree ordering, subscript compression only by mass elimination.

Key	Order	Number of nonzero entries in lower triangle of			Number of maximal cliques
		A	Compressed subscripts	L	
BCSPWR09	1723	2394	3995	4592	1621
BCSPWR10	5300	8271	16843	22764	4846
BCSSTK08	1074	5943	10845	29973	756
BCSSTK13	2003	40940	28027	269668	597
BCSSTM13	2003	9970	6972	43491	1540
BLCKHOLE	2132	6370	13053	51012	1150
CAN 1072	1072	5686	7091	19332	620
DWT 2680	2680	11173	18410	50072	1550
LSHP3466	3466	10215	21270	83116	1845
GR 40×40	1600	6162	10299	33304	845
GR 80×80	6400	25122	44222	183301	3285

TABLE 3
Time to compute a parallel ordering (CPU seconds) *on a* Sun 3/260.

Key	Min degree ordering	Symbolic factorization	Setup costs	Parallel reordering
BCSPWR09	0.48	0.08	0.08	0.07
BCSPWR10	1.96	0.31	0.26	0.23
BCSSTK08	2.78	0.12	0.06	0.05
BCSSTK13	4.56	0.44	0.11	0.10
BCSSTM13	1.03	0.13	0.08	0.04
BLCKHOLE	0.69	0.15	0.09	0.08
CAN 1072	0.69	0.09	0.05	0.04
DWT 2680	1.78	0.23	0.13	0.11
LSHP3466	1.12	0.25	0.15	0.13
GR 40×40	0.51	0.13	0.07	0.06
GR 80×80	2.06	0.53	0.29	0.25

minimum degree ordering step, second is the symbolic factorization step, next the procedure that computes the initial clique tree and other data structures (setup costs), and finally the *Parallel Ordering Algorithm.* These results show that the minimum degree ordering time dominates the other three steps. Also note that the time required to create the initial clique tree and the required data structures is comparable to the time taken by the parallel recordering algorithm. Both the *Maximal Cliques Algorithm* and the *Parallel Ordering Algorithm* are $\mathcal{O}(n+q)$ in complexity, and hence these results are in accord with our complexity analyses.

The second most expensive step is the symbolic factorization. The procedure we used is a modification of the Fortran program `smbfct` in [12]. We take greater advantage of mass elimination compression in the initial tests to detect such compression without creating new subscript lists. We removed the limited capability for finding accidental compression from the procedure. Both changes improve the speed of the procedure; in addition, the latter has the advantage of exhibiting precisely the size of the representa-

tion of the maximal cliques of the filled graph. As a result, our symbolic factorization is somewhat faster than `smbfct`, yet requires negligibly more storage for subscripts (0.4 percent on average).

Liu's codes were not available to us for direct comparison. However, Liu has published timing results for his algorithms applied to the problems in Table 1, using a SUN 3/50 workstation. This computer is closely related to the SUN 3/260 we used, but it is not as powerful. However, Liu has used his multiple minimum degree ordering code [16] for the first step, and we also used this code to find the fill-reducing ordering. The common minimum degree ordering code consistently ran approximately four times faster on our workstation. To attempt to make our results comparable to Liu's results, we normalize the statistics published by Liu [18] to a cost relative to that of the common minimum degree ordering. For all three approaches, we report the ratio of the CPU time required for the postprocessing to the CPU time required for the minimum degree ordering on the same machine. In our case the postprocessing requires a symbolic factorization; computation of the initial clique tree and other data structures; and the *Parallel Ordering Algorithm*, all of which together we refer to as the Clique Tree Jess and Kees algorithm. The total time for all of these steps in the Clique Tree Jess and Kees algorithm is reported in Table 4.

From the relative performances in Table 4, it is clear that the Liu and Mirzaian algorithm is much more expensive than the other two. Between our algorithm and Liu's composite rotations algorithm, we see a significant difference on only three of the eleven problems. The difference in performance on the remaining eight problems is smaller than the variation in relative times we have seen with our own code at various levels of compiler optimization.

TABLE 4
Performance relative to minimum degree ordering time.

Key	Comparative performance		
	Composite rotations	Clique tree Jess & Kees	Liu & Mirzaian Jess & Kees
BCSPWR09	0.21	0.48	0.54
BCSPWR10	0.18	0.41	0.55
BCSSTK08	0.06	0.08	0.53
BCSSTK13	0.26	0.14	1.74
BCSSTM13	0.27	0.24	1.30
BLCKHOLE	0.38	0.47	1.93
CAN 1072	0.19	0.26	0.98
DWT 2680	0.21	0.26	0.90
LSHP3466	0.42	0.48	2.04
GR 40×40	0.38	0.50	1.66
GR 80×80	0.49	0.52	2.42

The three problems with extremal behavior distinguish between the codes in a predictable way. The electric power problems, BCSPWR09 and BCSPWR10, represent a class of problems where very low fill is typical and the number of maximal cliques is close to n. For these problems, essentially all operations are $\mathcal{O}(n)$, and the use of maximal cliques makes only relatively small improvements on the Liu and Mirzaian

implementation of Jess and Kees. The largest realistic structural engineering problem, BCSSTK13, is at the other extreme. This is a large finite-element model with a number of degrees of freedom associated with each physical node. Mass elimination compression has an enormous effect, similar to what we usually see on much larger problems, which means that the maximal clique representation is very compact. The compressed subscripts lists are smaller than the representation of the original matrix A; the complexity bounds correctly predict that our algorithm is faster than Liu's heuristic.

Unfortunately our comparison is constrained to the problems for which Liu has published results; the limited sizes of the problems in Table 4 and the use of only one fill-reducing ordering algorithm (multiple minimum degree) make it difficult to extrapolate the behavior to larger problems and to other orderings. The previous paragraph contains the best analysis of expected timing comparison we can make at this point. We are also limited in our ability to compare the performance of the composite rotations heuristic at reducing tree height by the lack of any theoretical analysis of the suboptimality of the heuristic. We can only say at this time that we expect our algorithm to have similar running time to Liu's heuristic and to give results always as good as or better than the heuristic.

After the parallel ordering is computed, we will require the structure of the reordered L to compute the numeric factorization. The equivalence of the subscript lists and the maximal cliques used to create the initial clique trees for the Clique Tree Jess and Kees algorithm may serve to further advantage. The maximal cliques are the same for the parallel ordering as for the sequential ordering; the uneliminated nodes in a clique K from which a node u is eliminated in the *Parallel Ordering Algorithm* comprise the subscript list for u. A symbolic factorization for the parallel ordering based on this observation has been written, and it proved to be somewhat faster than the modified `smbfct` procedure. In contrast, the tree rotations algorithm must be followed by a general symbolic factorization step. Since symbolic factorization accounts for about half the postprocessing time in our algorithm, the comparison of the relative performances of these algorithms appears favorable to the tree rotations algorithm. But it remains to be seen whether the completed symbolic factorization provides an advantage in practice.

8. Concluding observations: Clique trees in other contexts. The applications of chordal graphs in sparse matrix computations have long been known (Rose [24]), but Peyton's thesis [21] is the only previous work in the sparse matrix literature (to the best of our knowledge) to make use of the clique tree.

The concept of a clique tree has previously appeared in the graph theory literature and in the context of acyclic relational database schemes. Buneman [3], Gavril [11], and Walter [26] have considered the clique tree representation of chordal graphs, and Golumbic [13] contains a good discussion of several properties of chordal graphs. In the context of database theory, an acyclic hypergraph captures the relations in acyclic database schemes, and a clique tree can be used to represent this hypergraph. Beeri et al. [2] and Tarjan and Yannakakis [25] have considered the clique tree in this context and call it a *join tree.* In his thesis, Peyton [21] discusses some applications of clique trees to sparse linear systems, including the design of a multifrontal Cholesky factorization algorithm based on the clique tree.

As we noted in § 3, our definition of a clique tree is more restrictive than the definitions employed in other contexts. We require the first ancestor node of a clique to be a *new* node in its parent clique. Our clique trees then have the property that the parent P is the *only* ancestor of a maximal clique K that contains all nodes in the set

$anc(K)$; other ancestors of K can contain only proper subsets of the nodes in $anc(K)$. This property is necessary for updating the clique tree solely by local changes during elimination, and is at the heart of the linear time bounds in our parallel reordering algorithm. A careful reading of the proof of Theorem 12 will show that it makes use of this property. Further, as described in § 6, it does not seem possible to update an explicit *parent* representation in the required time bounds; our restricted definition of the clique tree is essential to our implicit parent function utilizing the first ancestor node of a clique and the clique in which a node is *new*.

The method we use to compute a clique tree differs from the approaches employed in the graph theory literature and in database theory in taking full advantage of more information about the chordal filled graph. Specifically, we have a perfect elimination order for the filled graph from the fill-reducing ordering, and also a compact representation of the maximal cliques from the compressed row subscripts generated in the symbolic factorization step. By using this information, the *Maximal Cliques Algorithm* generates a clique tree efficiently with regard to both storage and time. Indeed, this algorithm computes the clique tree in $\mathcal{O}(n)$ time if we do not compute the *clique_count* information not needed for the clique tree.

Algorithms that simultaneously compute a perfect elimination order and a clique tree cannot be as efficient. Tarjan and Yannakakis [25] first renumber the nodes by a generalized maximum cardinality search of the hypergraph and then compute a clique tree from this ordering in $\mathcal{O}(n+q)$ time. Their algorithm requires node-clique incidence lists in addition to the node lists of the maximal cliques. Peyton [21] has designed an $\mathcal{O}(n+\eta(F))$ algorithm to compute simultaneously the maximum cardinality ordering of a chordal graph and a clique tree when the graph is represented by adjacency lists.

The algorithm by Tarjan and Yannakakis [25] for selectively reducing acyclic hypergraphs can be viewed as the computation of another perfect elimination ordering of the chordal graph, also through the process of eliminating simplicial nodes and deleting nonmaximal cliques. Their algorithm does not work with a clique tree; instead it uses a generalized maximum cardinality search on the hypergraph. The ordering that results is a perfect elimination ordering, but not a Jess and Kees ordering.

Other tree structures in sparse matrix computations. The elimination tree [19] is a nodal explanation of the factorization dependencies; it ignores the cliques in the chordal filled graph. The cliques can be used to great advantage in the design of high performance sparse matrix codes. Consequently, a number of researchers [7], [8], [9] have developed trees representing the cliques of filled graphs. The clique tree is similar to these previous structures because the interrelationship of the cliques is governed by the chordal graph. However, the clique tree has a number of minimality properties not shared by these other models because it comprises only the *maximal* cliques. This is clearly required in the present context for the simpliciality test on nodes. In addition, clique trees also provide explanations of certain optimality questions, as discussed below. A more detailed exposition of the relationship of the clique tree to these other trees of cliques will be made in a future paper.

Other applications of clique trees. The clique tree representation of the filled graph has consequences for several other sparse matrix problems. The clique tree helps to clarify the notion of a supernode in vectorized numeric factorization [1]. It explains how to compute a multifrontal factorization in the minimum number of frontal elimination steps: a postorder traversal of the clique tree results in exactly as many fronts as maximal cliques. It also appears that a multifrontal algorithm based on

traversing the clique tree may often have smaller working storage requirements than a multifrontal algorithm based on an elimination tree. The clique tree aids in the understanding of how elimination tree restructuring algorithms change the height of an elimination tree. It is useful in the context of sparse orthogonal factorization algorithms also. The notion of a "best" clique tree in a particular context needs careful study. We intend to explore several of these issues in future work.

9. Appendix: Proofs of results in § 5. In this appendix, we provide the proofs of the major results in § 5. For ease of reading, we restate each result before proving it.

THEOREM 8. *Let G be a chordal graph with clique tree T. Let K be a shrinking clique in G, with residual clique $R(K)$ in G^*.*

- *$R(K)$ is nonmaximal only if K contained only one simplicial node.*
- *If $R(K)$ is nonmaximal, and K is a leaf clique in T, then $R(K)$ is contained in the residual clique of its parent $P(K)$.*
- *If $R(K)$ is nonmaximal, and K is an interior clique in T, then $R(K)$ is contained in the residual clique of a child C, specifically a child whose ancestor set is largest.*

Proof. The first statement follows from the fact that a simplicial node belongs to exactly one maximal clique. If K contained more than one simplicial node, only one was eliminated in the reduction, and so $R(K)$ retains at least one simplicial node from K. This node is not in any other maximal clique, and so $R(K)$ cannot be a subclique of any other maximal clique. Thus a clique can become nonmaximal only when its last simplicial node is eliminated.

To prove the second statement, observe that the simplicial nodes of a leaf are precisely its *new* nodes. By Proposition 6, the *anc* nodes of any clique also belong to its parent, and are not simplicial; hence only the *new* nodes in a clique are ever simplicial. By Theorem 7, the *new* nodes lie only in the leaf and its empty set of descendants; hence all *new* nodes of a leaf are simplicial. Thus a leaf clique becomes nonmaximal only when its last *new* node is eliminated. In this case $R(K) = anc(K)$, which, by Lemma 6, is a subset of the parent clique $P(K)$. As *anc* nodes, these nodes in $R(K)$ were not simplicial in G, and hence they belong to $R(P(K))$ also.

Note that, unlike a leaf, an interior clique need not become nonmaximal when its last simplicial node is eliminated, for it has nonsimplicial *new* nodes. The third condition in the theorem applies only in those cases where the elimination of the last simplicial node of K causes it to become nonmaximal. We prove the result in three steps.

First, we show that if K becomes nonmaximal, any clique that absorbs $R(K)$ must be the residual clique of one of the descendants of K. Because K is an interior clique, it has at least one child clique C. The first ancestor node u of C is a *new* node in K and is not simplicial in G. Hence $R(K)$ also contains u. But by Theorem 7, u can belong only to descendants of K, hence only to their residual cliques, and thus only a descendant's residual clique can possibly absorb $R(K)$.

Second, we show that if $R(D)$ absorbs $R(K)$ for some descendant D of K, then $R(C)$ absorbs $R(K)$ for some child C of K. If $R(D)$ absorbs K, we must have $R(K) \subseteq anc(D)$. Suppose that D is not a child of K. By Theorem 7, any nodes that appear in a clique's ancestor set must appear as new nodes in some ancestor and in all cliques on the path in between. All of the nodes in $R(K)$ are *new* nodes in K or one of its ancestors. Hence if they belong to D, they belong also to all cliques lying on the path from D to K in the clique tree, and in particular, to a child of K. Thus, it suffices to consider only the children of K to find out if an absorbing clique exists.

Finally, we show that only a child C with the largest ancestor set has a residual set that can absorb $R(K)$. On the one hand, we know that $anc(C) \subseteq K$; since none

of these nodes is simplicial, we know that for all children C, $anc(C) \subseteq R(K)$. Hence, $|anc(C)| \leqq |R(K)|$. On the other hand, no node in K or $R(K)$ is found in $new(C)$; hence $R(K) \subseteq R(C)$ only if $anc(C) = R(K)$. Thus, such a child C has the largest possible ancestor set among the children of K. Note the stronger result: if any child of maximum ancestor set size can absorb K, so could any other child of maximum ancestor set size. □

COROLLARY 9. *Consider a shrinking clique K whose last simplicial node is eliminated. Then $R(K)$ is nonmaximal if and only if either $|new(R(K))| = 0$ or a child C of K with the largest ancestor size satisfies $|anc(C)| = |R(K)|$.*

Proof. First, observe that the two conditions are mutually exclusive: the first applies to a leaf clique and the second to an interior clique. If the first condition holds, K must be a leaf of T, since we showed in the proof above that an interior clique must have a nonsimplicial *new* node. Conversely, if the first condition fails, then K was not a leaf because all *new* nodes of leaves are simplicial, but at least one of K's *new* nodes was not simplicial.

The proof is then a straightforward extension of the proof of Theorem 8. If $R(K)$ is nonmaximal, and K is a leaf, then we have already shown that $R(K) = anc(K)$, and hence $new(R(K)) = \varnothing$. If $R(K)$ is nonmaximal and K is an interior clique, then again from the proof of Theorem 8, $R(K) = anc(C)$.

Conversely, if $|new(R(K))| = 0$, then K is a leaf and $R(K) = anc(K) \subseteq P(K)$. Then $R(K)$ is nonmaximal. If the second condition applies, we have $anc(C) = R(K)$, since we know that $anc(C) \subseteq R(K)$. Hence $R(K) \subseteq R(C)$, and again $R(K)$ is not maximal. □

THEOREM 10. *Let G be a chordal graph with a clique tree T. Let u be a simplicial node, belonging to a maximal clique K, that is eliminated from G. Then a clique tree for $G - \{u\}$ can be constructed by the following rules:*

- *If $R(K)$ is maximal, make no changes.*
- *Otherwise,*
  - *if K is a leaf of T, delete K from T.*
  - *if K is an interior clique of T, let C be a child with the largest ancestor set among the children of K. Then*
    - *assign all nodes in $new(K)$ to $new(C)$, removing them from $anc(C)$;*
    - *attach C as a child of $P(K)$, with $first_anc(C)$ reset to $first_anc(K)$;*
    - *attach children of K other than C as children of C;*
    - *delete K from the tree.*

Proof. We must show that the changes preserve a correct *new*, *anc*, *first_anc* labeling of each clique. Note that u appeared only in K, and hence all other cliques remain the same after the elimination of u.

The result is trivial when $R(K)$ remains maximal. The only change made to the tree is to remove a node from $new(K)$ that appeared only in K. Thus, there are no structural changes in the clique tree.

There are two cases to consider when $R(K)$ is not maximal, corresponding to whether K was a leaf of the clique tree or an interior clique. If K was a leaf, u was the last node in $new(K)$, which now becomes empty. This node appeared only in K, so the partition of nodes elsewhere is unaffected by removing K from the tree.

In the case of an interior clique, we need to check that the revised partition satisfies the required conditions for a parent and its child. For ease of notation, let $G' \equiv G - \{u\}$. We use primed quantities to refer to updated variables in G'. Note that when K is an interior clique absorbed by a child C, $R(K) = anc(C)$. Also, $anc(C)$ is an ordered subset of C consisting of all nodes numbered higher than or equal to C's first ancestor.

By our rules,

$$new'(C) = new(C) \cup (new(K) - \{u\}),$$
$$anc'(C) = anc(C) - (new(K) - \{u\}) = anc(K).$$

Thus our rules update C by moving nodes numbered lower than the first ancestor of K from the *anc* set of C to the *new* set of C. We now show that the revised partition $new'(C)$, $anc'(C)$ of C is correct, and then that the parent-child relationship in the new clique tree are correct.

If a node w is included in $new'(C)$ by these rules, either it was in $new(C)$ or in $new(R(K))$. In the former case, $Madj(w) \subseteq C$ before the elimination of u, and this relationship is unaffected by the elimination of u. In the latter case, $Madj(w) \subseteq K$ before the elimination of u implies that $Madj'(w) \subseteq R(K) = anc(C) \subset C$. In both cases, by Proposition 4, the *Maximal Cliques Algorithm* applied to the graph G' would mark w as a *new* node in C.

Similarly, if w is included in $anc'(C)$, then $w \in anc(K)$. Therefore, by Proposition 4, $Madj(w) \not\subseteq K$, and thus $Madj'(w) \not\subseteq R(K)$. Each node in $Madj(w)$ is numbered higher than w, which in turn is numbered higher than each node in $new(C)$ since $w \in anc(C)$. Therefore,

$$Madj'(w) \not\subseteq R(K) \cup new(C) = anc(C) \cup new(C) = C.$$

Thus the *Maximal Cliques Algorithm* applied to G' would not mark w as a *new* node in C.

$P(K)$ is the correct choice as the parent of C because $first_anc'(C) = first_anc(K)$, a node in $new(P(K))$. To demonstrate that the other children of K have been properly reassigned in T', consider any such child J. The eliminated node u was simplicial, and therefore did not appear in $anc(J)$. Hence $anc'(J) = anc(J) \subseteq R(K) = C$. The node $first_anc(J)$ was in $new(K)$ and so becomes a member of $new'(C)$. C is then the correct choice for the parent of J. □

PROPOSITION 11. *Let C_1 and C_2 be siblings in the clique tree for G_F, with $|anc(C_1)| \geqq |anc(C_2)|$. Then $|anc(C_1)| \geqq |anc(C_2)|$ holds at all later stages of the method as long as C_1 and C_2 continue to be siblings.*

Proof. Let $|anc(C_1)| \geqq |anc(C_2)|$ in the initial clique tree. No updating operations increase the size of the ancestor sets, and ancestor nodes are never simplicial, so a change in this relationship can be caused only by nodes being transferred from $anc(C_1)$ to $new(C_1)$ or by the disappearance of either C_1 or C_2. In the latter case, they are trivially no longer siblings. The former case is nearly as trivial because it can only occur if C_1 absorbs its parent, in which case all of its siblings become its children. Obviously, C_2 will then no longer be a sibling of C_1. □

THEOREM 12. *Assume that the lists of children in the clique tree are initially sorted by decreasing ancestor set size, and that the child lists are revised by the rule in § 5 when an interior clique is absorbed. Then, at any future step in the elimination, a nonmaximal interior clique can be absorbed by its first child.*

Proof. We prove that the rule for arranging child lists when an interior clique is absorbed preserves the property that a nonmaximal clique can be absorbed by its first child. First, we introduce the concept of a *cohort*: we group the children of a maximal clique into ordered sets called cohorts.

We define cohorts inductively. If K is a maximal clique in the original clique tree, its children form a *cohort* $\mathcal{H}$. When K becomes nonmaximal and is absorbed by some child C_l, the cohort structures change as follows: Let P be the parent of K; the children of K be ordered $C_1, \cdots, C_l, \cdots, C_i$; and let the children of C_l be ordered $D_1, \cdots, D_j$.

In the clique tree, C_l takes the place of K as a child of P; hence the clique C_l is removed from the cohort to which it belonged, and joins the cohort to which K belonged. The cliques $C_1, \cdots, C_{l-1}, C_{l+1}, \cdots, C_i$ become new children of C_l, listed after $D_1, \cdots, D_j$; if the latter cliques form r cohorts ordered as $\mathcal{H}_1, \cdots, \mathcal{H}_r$, and the former cliques form $t-r$ cohorts ordered as $\mathcal{H}_{r+1}, \cdots, \mathcal{H}_t$, then after the tree update, the children of C_l form t cohorts ordered as $\mathcal{H}_1, \cdots, \mathcal{H}_r, \mathcal{H}_{r+1}, \cdots, \mathcal{H}_t$. Note that the relative ordering of the cohorts within each group of children is preserved. We will prove that the absorbing child C_l can always be taken to be the first child C_1.

We use the concept of a cohort only in this proof; in our algorithms we do not need to maintain the cohort structures at parent cliques.

Let J be a maximal clique in the clique tree at some intermediate stage in the elimination, with its children grouped into *cohorts* $\mathcal{H}_1, \cdots, \mathcal{H}_t$. Except for the uneliminated children of J from the original clique tree, other cohorts are acquired when J absorbs a parent clique.

We claim that our rule for merging child lists preserves the following two properties of the cohort structure at any clique J:

(1) Within each cohort $\mathcal{H}_s$, the cliques are listed in decreasing order of ancestor size.

(2) Each child in a cohort $\mathcal{H}_s$ contains some node in $new(J)$ that is found in no clique in the cohorts $\mathcal{H}_{s+1}, \cdots, \mathcal{H}_t$.

We prove the claim by induction on the number of interior clique absorptions. It holds trivially before the first such absorption because each clique has only one cohort, its original children, which are ordered by ancestor size. For the inductive step, assume the properties hold before the absorption of a nonmaximal clique K by its first child, C_1. As before, denote the parent of K by P, the other children of K by $C_2, \cdots, C_i$, and the children of C_1 before the clique tree update by $D_1, \cdots, D_j$. Let $R(K)$ denote the set of nodes in K when it is absorbed, and denote the ancestor and *new* sets of C_1 after the update by $anc'(C_1)$, and $new'(C_1)$, respectively; note that the former set is equal to $anc(R(K))$, and similarly the latter set is $new(C_1) \cup new(R(K))$. Assume also that C_1's children $D_1, \cdots, D_j$ before the absorption form r cohorts, $\mathcal{H}_1, \cdots, \mathcal{H}_r$, and that its new children $C_2, \cdots, C_i$ form $t-r$ cohorts, $\mathcal{H}_{r+1}, \cdots, \mathcal{H}_t$.

Note that the only cliques whose child lists are modified by the absorption of K are the two cliques P and C_1. It is only for these two cliques that we need to prove that the properties hold.

The child list for P is modified only by replacing K with C_1, and the relative order of the cohorts is unchanged. Also, $anc'(C_1) = anc(R(K))$, so the ancestor sets of the children of P are unchanged. Since only the relative ordering of the cohorts and the ancestor sets are addressed by both properties, they continue to hold for P.

It remains to examine the children of C_1. The first property holds trivially, since within a cohort we do not change the relative order of the cliques. Hence we prove the second property.

Consider the situation before the absorption of K by C_1. For $k = 1, \cdots, j$, let the first ancestor of D_k be v_k. These nodes are not necessarily distinct; let $V = \bigcup_{k=1}^{j} \{v_k\}$ be the corresponding set of nodes. Each node in V belongs to $new(C_1)$, but not to $C_2, \cdots, C_i$, since the latter cliques are siblings and not descendants of C_1. This result continues to hold after the clique tree update, when these cliques become new children of C_1.

Thus, we have shown the second property with respect to the two groups of cohorts, $\mathcal{H}_1, \cdots, \mathcal{H}_r$, and $\mathcal{H}_{r+1}, \cdots, \mathcal{H}_t$. By the induction hypothesis, this property held within each group of cohorts before the update. It holds after the update for the

first group trivially. It holds for the second group because $new'(C_1)$ contains all nodes in $new(R(K))$ included in $C_2, \cdots, C_i$. This completes the proof of the claim.

The consequence of the second property for the Jess and Kees method is the following: For each child D in its first cohort and for every child C in later cohorts, there exists some node v in both $new(J)$ and D, but not in C. Thus the child clique C cannot absorb J until this node v has been eliminated. But before v is eliminated, it must become simplicial, which requires that D become nonmaximal and be deleted from the clique tree. Thus, only a child belonging to the first cohort can possibly absorb J.

Within each cohort, the children are listed in decreasing order of ancestor set size; in particular, this is true for the first cohort of J. From Theorem 8 and from Proposition 11, only a child of largest ancestor set size in the first cohort can absorb J. The result is then that the structure induced by our child list merging rule ensures that only the first child needs to be tested as a candidate for absorption. □

Acknowledgments. Preliminary versions of some results in this paper were included in [15] and [22]. Alex Pothen's work on this paper was begun while visiting Oak Ridge National Laboratory and the University of Tennessee, Knoxville, during their special year in numerical linear algebra. He thanks Alan George for the invitation to offer a graduate course on Sparse Matrix Algorithms, which provided the stimulus for this research; he also thanks other colleagues at both institutions for their hospitality and encouragement.

REFERENCES

[1] C. ASHCRAFT, R. GRIMES, J. LEWIS, B. PEYTON, AND H. SIMON, *Progress in sparse matrix methods for large linear systems on vector supercomputers*, Internat. J. Supercomputer Appl., 1 (1987), pp. 10-30.

[2] C. BEERI, R. FAGIN, D. MAIER, AND M. YANNAKAKIS, *On the desirability of acyclic database schemes*, J. Assoc. Comput., 30 (1983), pp. 479-513.

[3] P. BUNEMAN, *A characterization of rigid circuit graphs*, Discrete Math., 9 (1974), pp. 205-212.

[4] I. S. DUFF, R. GRIMES, AND J. LEWIS, *Sparse Matrix Test Problems*, Tech. Report. CSS 191, Harwell Laboratory, Didcot, Oxon, England, 1987; ACM Trans. Math. Software, 15 (1989), pp. 1-14.

[5] I. S. DUFF, R. GRIMES, J. LEWIS, AND W. POOLE, JR., *Sparse matrix test problems*, SIGNUM Newsletter, 17, 2 (1982), p. 22.

[6] I. S. DUFF AND J. K. REID, *A note on the work involved in no-fill sparse matrix factorization*, IMA J. Numer. Anal., 3 (1983), pp. 37-40.

[7] ———, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302-325.

[8] S. EISENSTAT, M. SCHULTZ, AND A. SHERMAN, *Applications of an element model for Gaussian elimination*, in Sparse Matrix Computations, J. Bunch and I. Duff, eds., Academic Press, New York, 1976, pp. 85-96.

[9] ———, *Software for sparse Gaussian elimination with limited core storage*, in Sparse Matrix Proceedings, I. Duff and G. W. Stewart, eds., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1978, pp. 135-153.

[10] D. R. FULKERSON AND O. A. GROSS, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835-855.

[11] F. GAVRIL, *The intersection graphs of subtrees are exactly the chordal graphs*, J. Combin. Theory, Ser. B, 16 (1974), pp. 47-56.

[12] A. GEORGE AND J. W. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs, NJ, 1981.

[13] M. GOLUMBIC, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[14] J. JESS AND H. KEES, *A data structure for parallel L/U decomposition*, IEEE Trans. Comput., C-31 (1982), pp. 231-239.

[15] J. LEWIS AND B. PEYTON, *A fast implementation of the Jess and Kees algorithm*, Tech. Report ETA-TR-90, Boeing Computer Services, Seattle, WA, May 1988.

[16] J. W. LIU, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141-153.

[17] ———, *Computational models and task scheduling for parallel sparse Cholesky factorization*, Parallel Comput., 3 (1986), pp. 327-342.

[18] ———, *Reordering sparse matrices for parallel elimination*, Tech. Report CS-87-01, Computer Science Department, York University, North York, Ontario, Canada, 1987, Par. Comput., to appear.

[19] ———, *The role of elimination trees in sparse factorization*, Tech. Report CS-87-12, Computer Science Department, York University, North York, Ontario, Canada, 1987, SIAM J. Matrix Anal. Appl., (to appear).

[20] J. W. LIU AND A. MIRZAIAN, *A linear reordering algorithm for parallel pivoting of chordal graphs*, Tech. Report CS-87-02, Computer Science Department, York University, North York, Ontario, Canada, 1987; SIAM J. Discrete Math., 2 (1989), pp. 100-107.

[21] B. PEYTON, *Some applications of clique trees to the solution of sparse linear systems*, Ph.D. thesis, Clemson University, Clemson, SC, 1986.

[22] A. POTHEN, *Simplicial cliques, shortest elimination trees, and supernodes in sparse Cholesky factorization*, Tech. Report CS-88-13, Computer Science Department, Pennsylvania State University, University Park, PA, April 1988.

[23] ———, *The complexity of optimal elimination trees*, Tech. Report CS-88-16, Computer Science Department, Pennsylvania State University, University Park, PA, April 1988.

[24] D. ROSE, *A graph-theoretic study of the numerical solution of sparse positive definite systems of equations*, in Graph Theory and Computing, R. Read, ed., Academic Press, New York, 1972, pp. 183-217.

[25] R. TARJAN AND M. YANNAKAKIS, *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM J. Comput., 13 (1984), pp. 566-579.

[26] J. R. WALTER, *Representations of chordal graphs as subtrees of a tree*, J. Graph Theory, 2 (1978), pp. 265-267.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 6, pp. 1174-1185, November 1989
© 1989 Society for Industrial and Applied Mathematics
007

HIGH PERFORMANCE PRECONDITIONING*

HENK A. VAN DER VORST†

Abstract. The discretization of second-order elliptic partial differential equations over three-dimensional rectangular regions, in general, leads to very large sparse linear systems. Because of their huge order and their sparseness, these systems can only be solved by iterative methods using powerful computers, e.g., vector supercomputers. Most of those methods are only attractive when used in combination with a so-called preconditioning matrix. Unfortunately, the more effective preconditioners, such as successive over-relaxation and incomplete decompositions, do not perform very well on most vector computers if used in a straightforward manner. In this paper it is shown how a rather high performance can be achieved for these preconditioners.

Key words. preconditioning, ICCG, block bidiagonal systems, vector computers, parallel computers

AMS(MOS) subject classifications. 65F05, 65F10, 65F50, 65N20

1. Introduction.

Among the methods that have been used successfully for solving the large linear systems that stem from discretized three-dimensional partial differential equations (PDEs) are:

—Preconditioned conjugate gradients for the symmetric positive definite case [18].

—Projection-type methods for the general nonsymmetric case, such as, e.g., Orthodir, Orthomin, and Orthores [33], GMRES [21], LSQR [20], Bi-CG [7], and CGS [24].

—Preconditioned adaptive Chebychev iteration [16], [26].

The success of these methods usually depends quite critically on the choice of preconditioner that has the effect of improving the spectrum of the linear operator involved.

Vector supercomputers, and parallel versions of them, play an important role in solving these large systems. Since the basic components of the methods, mentioned above, are operations like vector updates, innerproducts, and sparse matrix vector products, they are in principle quite easily implementable on vector computers with a high performance code as a result. This, however, is in general not the case for the most popular preconditioning techniques. Since they usually contain recurrence relations, they typically do not vectorize or parallelize very well in a standard way and therefore many authors have suggested better vectorizable alternatives [9], [11]-[15], [17].

In this contribution we will consider vector and parallel aspects of the so-called incomplete LU decompositions [18], [19] that are among the most frequently used preconditioners. Since the problems with respect to vector or parallel execution are similar for both the symmetric and the nonsymmetric case and since the preconditioners do not depend on the basic iteration method, we will study, for the sake of simplicity, the most simple case, namely, the preconditioned conjugate gradients algorithm for symmetric positive definite systems. Note, however, that most of the ideas outlined here can be carried over immediately to the other situations mentioned earlier (including successive over-relaxation (SSOR) and Gauss-Seidel preconditioning or smoothing).

We will show that, especially in the three-dimensional case, the preconditioned algorithm can be implemented by vectorized code that takes about as much CPU-time per iteration step as the unpreconditioned version. As preconditioning usually reduces

* Received by the editors October 5, 1988; accepted for publication (in revised form) February 10, 1989.

† Faculty of Technical Mathematics and Informatics, Delft University of Technology, P.O. Box 356, 2600 AJ Delft, the Netherlands.

significantly the number of iteration steps, this reduction then immediately translates to a similar reduction in CPU-time. With respect to vector computers we have in mind machines such as, e.g., CRAY X-MP, CYBER 205, ETA10, and for parallelism we aim, for the moment, for systems with only a moderate number of powerful processors, such as, e.g., CRAY X-MP/4, CRAY Y-MP/8, ETA10-G, or IBM 3090-600E (VF).

2. A sketch of the problem and a realistic goal. Let the linear system $Ax = b$ be the result of standard seven-point finite difference discretization of an elliptic second-order PDE, such as, e.g., $(au'_x)'_x + (bu'_y)'_y + (cu'_z)'_z + du = f$, over a rectangular grid imposed over a rectangular region.

To avoid questions with respect to mathematical aspects, including convergence properties and existence of the incomplete decompositions, we will further assume that A is a symmetric positive definite M-matrix.

Such a discretization matrix has a very regular structure when we number the unknowns, corresponding to the gridpoints, lexicographically such that the pointer in x-direction runs fastest, followed by the pointer in y-direction. Gridpoints with equal y and z coordinates define a line in the grid, those with equal z define a plane.

The matrix of the resulting linear system can be viewed as a block tridiagonal matrix of which the off diagonal elements are diagonal matrices, defining the relations between the unknowns in successive planes. The diagonal blocks by themselves can be seen as block tridiagonal matrices, corresponding to two-dimensional problems over each z-plane and, finally, the diagonal blocks of the latter are tridiagonal matrices defining the relations between unknowns along a line.

For the solution of this linear system of order N we consider the preconditioned conjugate gradient method. It is well known that the unpreconditioned cg method vectorizes very well, but the preconditioned version is much more troublesome. In fact, it would be very attractive to find a preconditioner that satisfies the following requirements:

(1) It should lead to a substantial reduction in iteration steps;

(2) The amount of work per iteration step should be roughly the same as for unpreconditioned cg; and

(3) The computational speed for each iteration step should have the same order of magnitude as the unpreconditioned cg process.

We will show in this paper that these requirements can all be met, for systems with the above described structure, for some well-known vector supercomputers.

For comparison purposes we start with a discussion of the unpreconditioned cg method. This method has the following form [8]:

x_0 is an initial guess for the solution x of $Ax = b$

$r_0 = b - Ax_0, \qquad p_0 = r_0$

for $i = 0, 1, 2, \cdots$ until the method is stopped:

$$\alpha_i = \frac{(r_i, r_i)}{(p_i, Ap_i)},$$

$$x_{i+1} = x_i + \alpha_i p_i,$$

$$r_{i+1} = r_i - \alpha_i A p_i,$$

$$\beta_i = \frac{(r_{i+1}, r_{i+1})}{(r_i, r_i)},$$

$$p_{i+1} = r_{i+1} + \beta_i p_i.$$

Per iteration step the cg-method comprises the following computational work: two innerproducts; three vector-updates; and one matrix-vector multiplication. All these basic operations perform very well on vector machines such as, e.g., the CRAY X-MP and CYBER 205. In three-dimensional problems almost any discretized linear system is rather large, i.e., it leads to vector lengths close to those for which limit performances for the basic operations are practically observed.

For situations in which the matrix was represented by seven (long) diagonals, rather than a blockwise structure, we observed computational speeds of about 180 Mflops on a single processor CRAY X-MP and about 105 Mflops on a two-pipe CYBER 205.

3. The (M)ICCG algorithm. A very successful preconditioning matrix is obtained by constructing a so-called incomplete triangular decomposition of the matrix A. When applied to positive definite systems, the correspondingly preconditioned cg method is usually referred to as ICCG [18], [19] or MICCG [10]. In this section we will discuss computational aspects of these methods as well as their mutual relation.

For ICCG we construct an incomplete Choleski decomposition of A. If A is written as $A = L + \text{diag}(A) + L^T$, where L is the strict lower triangular part of A, then the standard incomplete Choleski decomposition can be represented by

$$K = (L+D)D^{-1}(D+L^T), \tag{3.1}$$

in which D is a diagonal matrix. Its elements follow from the relation

$$\text{diag}(K) = \text{diag}(A). \tag{3.2}$$

When the diagonal elements of A are denoted by $a_{i,1}$ (where i refers to the row) and the other three nonzero diagonals in L^T by $a_{i,2}$, $a_{i,3}$, and $a_{i,4}$, respectively, then the typical form for the relation of the elements d_i of D is

$$d_i = a_{i,1} - \frac{a_{i-1,2}^2}{d_{i-1}} - \frac{a_{i-nx,3}^2}{d_{i-nx}} - \frac{a_{i-nxny,4}^2}{d_{i-nxny}} \tag{3.3}$$

(nx and $nxny$ denote the distances of the respective diagonals to the main diagonal; nx = number of gridpoints in the x-direction, $nxny = nx * ny$ and ny = number of gridpoints in the y-direction).

From the discussion in § 4 it will be clear that the construction of D (for ICCG as well as MICCG), by relations such as (3.3), can be vectorized.

MICCG differs from ICCG by a different construction of D. It follows from the requirement

$$\text{Rowsum}(K) = \text{Rowsum}(A). \tag{3.4}$$

Since the preconditioning matrix in MICCG has the same form as the one in ICCG, the vectorization problems in both methods are exactly the same. These problems will be discussed in § 4.

We will now comment briefly on the convergence behavior of ICCG and MICCG. It has been observed that for (academic) model problems MICCG often converges much faster than ICCG. However, in many realistic (industrial) problems ICCG turns out to be the winning method. To study the relation between ICCG and MICCG, we include a parameter α in the relation (3.4) so that the relations for the d_i typically

take the form

$$
\begin{aligned}
d_i = a_{i,1} &- a_{i-1,2} * (a_{i-1,2} + \alpha a_{i-1,3} + \alpha a_{i-1,4})/d_{i-1} \\
&- a_{i-nxny,3} * (\alpha a_{i-nx,2} + a_{i-nx,3} + \alpha a_{i-nx,4})/d_{i-nx} \\
&- a_{i-nxny,4} * (\alpha a_{i-nxny,2} + \alpha a_{i-nxny,3} + a_{i-nxny,4})/d_{i-nxny}.
\end{aligned} \tag{3.5}
$$

For $\alpha = 1$ the standard MICCG-method results and note that for $\alpha = 0$ we obtain the ICCG-method. By varying α we can smoothly shift from ICCG to MICCG. There is another way of looking at this. MICCG can be seen as a modification to ICCG in which the approximation errors in the incomplete decomposition are lumped to the diagonal. With α we can control the degree of lumping [2], [3].

If we plot (as in Fig. 1) the number of iterations for the preconditioned cg process as a function of α, then typically this number decreases for α increasing from zero (ICCG) toward one (MICCG), but it almost always suddenly jumps upward for α very close to one. For the earlier-mentioned model-problems the jump may be only small, but in most cases it is there.

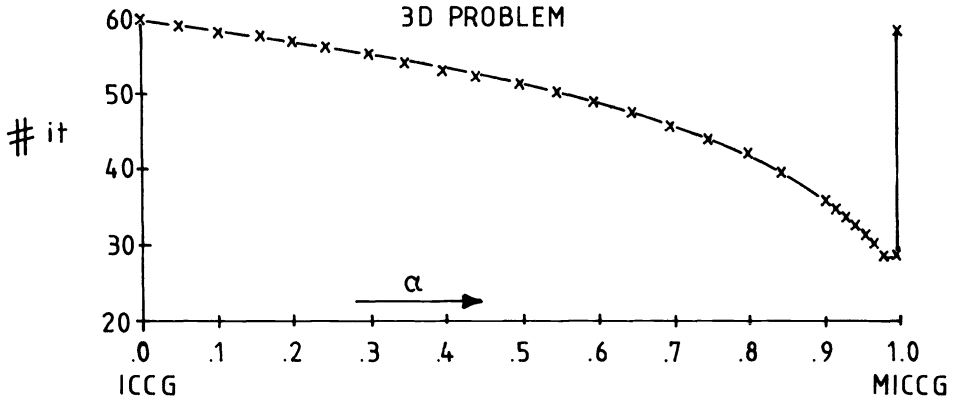


FIG. 1. *Three-dimensional problem.*

Similar experiences have been observed in, e.g., [1], [25], and [32]. The explanation given in [32] is that values of α very close to one lead to an early loss of orthogonality in the cg-iterations that spoils the convergence behavior. Many experiments carried out so far indicate that $\alpha = .95$ is a safe cure for this problem, and it is observed that MICCG ($\alpha = .95$) leads to a very substantial reduction in the number of iteration steps with respect to the unpreconditioned cg-process.

We will now consider some general implementation aspects of the (M)ICCG-process.

Once the diagonal D of the incomplete decomposition (3.1) has been computed, it is of advantage to scale the matrix A:

$$\tilde{A} = D^{-1/2} A D^{-1/2}. \tag{3.6}$$

The incomplete decomposition of $\tilde{A} = \tilde{L} + \text{diag}(\tilde{A}) + \tilde{L}^T$ can now be represented as

$$\tilde{K} = (\tilde{L} + I)(I + \tilde{L}^T), \tag{3.7}$$

so that scaling has led to a computationally cheaper preconditioner. The earlier

described cg algorithm is now applied to the preconditioned system

$$(\tilde{L}+I)^{-1}\tilde{A}(I+\tilde{L}^T)^{-1}\tilde{x}=(\tilde{L}+I)^{-1}\tilde{b}, \tag{3.8}$$

where $\tilde{b}=D^{-1/2}b$.

From the solution $\tilde{x}$ of (3.8) the solution x of $Ax=b$ is obtained via

$$x=D^{-1/2}(I+\tilde{L}^T)^{-1}\tilde{x}. \tag{3.9}$$

Note that the matrices $(\tilde{L}+I)^{-1}$ and $(I+\tilde{L}^T)^{-1}$ need not be formed explicitly (nor would it be advisable to do so), since these matrices occur throughout the cg process only in matrix-vector products. For example, the vector $y=(\tilde{L}+I)^{-1}z$ is, for given vector z, computed by solving y from the sparse lower triangular system

$$(\tilde{L}+I)y=z. \tag{3.10}$$

The form of the linear system (3.8) suggests that the amount of computational work per cg-iteration is considerably increased by preconditioning, since we have to solve two sparse triangular systems in each step. The amount of work for this is roughly the same as for the computation of the matrix vector product Ay.

Eisenstat [6] has shown that much of the extra work can be avoided by noting that the vector $(\tilde{L}+I)^{-1}\tilde{A}(I+\tilde{L}^T)^{-1}\tilde{p}_i$ (see the scheme in § 2) can be written as

$$(\tilde{L}+I)^{-1}\tilde{A}(I+\tilde{L}^T)^{-1}\tilde{p}_i=(I+\tilde{L})^{-1}[\tilde{p}_i+(\operatorname{diag}(\tilde{A})-2I)\tilde{t}]+\tilde{t}, \tag{3.11}$$

in which $\tilde{t}=(I+\tilde{L}^T)^{-1}\tilde{p}_i$.

Hence the expression at the right-hand side of (3.11) can be computed at about the cost of solving two triangular systems. For our class of three-dimensional model problems this comes down to $\approx$15 N flops (N is the order of the system), whereas Ap_i in the unpreconditioned process takes $\approx$13 N flops. From this we conclude that (M)ICCG can be carried out with about the same number of flops per iteration as unpreconditioned cg. Note that we now have fulfilled two of the requirements put on the preconditioner in § 2. In the next section we will consider the third requirement, on the performance of the preconditioned method that is a problem since, unfortunately, we have in fact replaced the fastest part of the unpreconditioned algorithm (i.e., Ap_i) by the more or less problematic solution of triangular systems.

4. The vectorized solution of the triangular systems. In this section we will discuss several approaches for achieving high computational speeds for the solution process of y from $(\tilde{L}+I)y=z$ (the same approaches hold, of course, for the upper triangular system).

Of course the computational work can be vectorized, or be carried out in parallel, when the order of the unknowns is changed, e.g., by red-black ordering. In general this leads to a different preconditioner, so it essentially changes the method, and experiments conducted by Duff and Meurant [34] indicate that for most reordering schemes the number of iterations doubles (or worse). However, for the given preconditioner there exist ordering schemes that do not change the preconditioner and that lead to exactly the same iterates as does the lexicographical ordering. The most obvious one, or at least the easiest programmable one, is the *plane-diagonal-wise ordering* that we will explain now.

If we indicate the unknowns, as well as the matrix coefficients by three indices, i, j, k, so that i refers to the index of the corresponding gridpoint in the x-direction, j to the index in the y-direction, and k in the z-direction, then the typical statement

for the computation of the elements of the vector y takes the form

$$y_{i,j,k} = z_{i,j,k} - a_{i-1,j,k,2}\, y_{i-1,j,k} - a_{i,j-1,k,3}\, y_{i,j-1,k} - a_{i,j,k-1,4}\, y_{i,j,k-1} \tag{4.1}$$

($a_{i,j,k,2}$ represents the matrix coefficient referring to the next unknown in the x-direction and so forth).

First we accumulate all contributions to y from the previous, already computed, plane in the grid:

$$\bar{z}_{i,j,k} = z_{i,j,k} - a_{i,j,k-1,4}\, y_{i,j,k-1} \quad \text{for all } i, j, k \text{ fixed.} \tag{4.2}$$

This is a vector operation. Now we define grid-diagonal m as the collection of gridpoints (i, j, k), for which $i+j=m$, and k is fixed.

Then we note that the unknowns corresponding to grid-diagonal m in a plane (k fixed) can be computed independently (or in vector mode) from the unknowns corresponding to the (previous) grid-diagonal $m-1$.

This approach leads to vector operations of length $O(n_x)$, n_x being the number of gridpoints in the x-direction. Therefore the vector lengths are rather short. This is not too bad for computers with a low $n_{1/2}$-value for this type of vector operation such as, e.g., the CRAY X-MP.

If the unknowns are not explicitly reordered in memory, then the diagonal-wise computation is typically carried out with stride $n_x - 1$ and this may lead to memory bank conflicts. For computers such as the CYBER 205 a stride $\neq 1$ must be avoided anyway.

As an illustration we present some results obtained for the CRAY-2 for two-dimensional problems. For three-dimensional problems for which the number of unknowns in x and y direction are similar to this two-dimensional case, we expect a little bit better Mflops-rates. We report here on the two-dimensional situation only since we have not actually tested this algorithm in the three-dimensional case. For a $100 * 100$ grid the CPU-times and computational speeds for a certain PDE are:

	CPU-time	Mflops-rate
cg (with diagonal scaling)	0.76 sec	82 Mflops
MICCG ($\alpha = .95$, Eisenstat-trick, unknowns explicitly reordered to avoid strides)	0.23 sec	53 Mflops

For a $200 * 200$ grid (and a different PDE) we obtain

	CPU-time	Mflops-rate
cg (with diagonal scaling)	4.13 sec	76.5 Mflops
MICCG (as above)	0.84 sec	58 Mflops

The interesting thing about these figures is not so much the CPU-times, since they depend on the PDE and the required solution, but the Mflops-rates, that are independent of those factors. We recall that the performance of unpreconditioned cg (that is equal to the performance for cg on the diagonally scaled system) is mainly determined by highly vectorizable components. The matrix vector product is the most CPU-time consuming one of these. Precisely this operation in our implementation of MICCG has been replaced by the solution of the more cumbersome triangular systems. Nevertheless, the performance (per iteration) drops only from ~80 Mflops to ~60 Mflops. For the CYBER 205 equally impressive results have been obtained. For computers for

which strides are not so much a problem (e.g., CRAY X-MP) the performance of MICCG with either explicit reordering or the original ordering does not differ so much, and in both cases is quite high (~100–120 Mflops).

In the three-dimensional situation there are more possibilities for vectorization. Instead of considering grid-diagonals we might as well consider hyperplanes, defined by all triples (i, j, k) having an equal sum. If we denote the hyperplane, formed by all gridpoints for which $i+j+k=m$ by H_m, then it is obvious from (4.1) that all unknowns corresponding to H_m can be computed independently from those corresponding to H_{m-1}.

This approach leads to rather long vector lengths ($O(N^{2/3})$), but the problem here is that the unknowns are not located equally spaced in memory, so that normally indirect addressing is required to identify those unknowns. On some vector supercomputers indirect addressing sort of spoils the performances (CYBER 205, NEC SX-2), on others the performances drop to about half the performance for the constant stride situation (CRAY X-MP, FUJITSU VP200). This explains that the performance per iteration of MICCG ($\alpha = .95$, Eisenstat-trick, hyperplane ordering) is about half the performance of the iteration of the unpreconditioned cg method on a CRAY X-MP. For the FUJITSU VP200 and the HITACHI S810/20 this is about 30 percent [28]. Hence on these machines we loose a factor of about three in the performance of MICCG, compared with cg, due to indirect addressing. However, for some of these computers the performance of the hyperplane ordering can be significantly improved, as we will show in the next section.

5. Implementation aspects of the hyperplane ordering. For some vector computers the performance reducing effects of indirect addressing, involved in the hyperplane ordering approach, can be largely reduced or even be undone by a careful implementation. To illustrate this we consider in more detail the implementation for the CYBER 205 (and ETA-10).

We will focus again on solving the lower triangular system (3.10), for which (4.1), with respect to the hyperplane ordering, is replaced by

(5.1)
for $l = 4, 5, \cdots, n_x + n_y + n_z$ *do*

(a) $\{y_{ijk} = z_{ijk} - a_{ijk-14}\, y_{ijk-1}$, for all $(i, j, k) \in H_l\}$
(b) $\{y_{ijk} = y_{ijk} - a_{ij-1k3}\, y_{ij-1k}$, for all $(i, j, k) \in H_l\}$
(c) $\{y_{ijk} = y_{ijk} - a_{i-1jk2}\, y_{i-1jk}$, for all $(i, j, k) \in H_l\}$

end l

The implementation for the three parts (a), (b), and (c) is quite similar and we will discuss only part (c). This part (c) may be written as

(5.2)
for $i = \max(2, l-n_y-n_z), \cdots, \min(n_x, l-2)$ *do*
  for $j = \max(1, l-i-n_z), \cdots, \min(n_y, l-i-1)$ *do*
    $k = l-i-j$,
    $y_{ijk} = y_{ijk} - a_{i-1jk2}\, y_{i-1jk}$
  end j
end i

This double loop defines our ordering of the (i, j, k) over H_l for each value of l. The obvious way to implement (5.2) is to store y_{ijk} and a_{i-1jk2} in the order in which they are used over H_l. This has to be done only once at the beginning of the cg process and, since we apply Eisenstat's trick, there is no need to store the matrix elements both in lexicographical and hyperplane ordering.

Having the elements of y and $a_{,2}$ reordered, there is still indirect addressing required for the elements y_{i-1jk} that are stored correspondingly to the ordering for H_{l-1}. For the CYBER 205 this leads to the following scheme for (5.2):

(5.3)
- (i) GATHER the required elements y_{i-1jk} into an array V_l, in the order in which they are required to update the y_{ijk} over H_l. The "gaps" in V_l, when H_l is larger than H_{l-1}, are left zero.
  This amounts for the CYBER 205 to $\approx 1.4 n_x n_y n_z$ clock cycles for all values of l together.
- (ii) The elements of V_l are multiplied by the a_{i-1jk2}: $\approx \frac{1}{2} n_x n_y n_z$ clock cycles for all l together.
- (iii) The result from step (ii) is subtracted element-wise from the y_{ijk}: $\approx \frac{1}{2} n_x n_y n_z$ clock cycles for all l together.

Hence the total execution time for step (5.1)(c) is, apart from startup overhead, equal to $\approx 2.4 n_x n_y n_z$ clock cycles. This leads to a total time for solving both triangular subsystems of roughly $14.4 n_x n_y n_z$ clock cycles. If we compare this with the time for computing Ay, where A has the structure as in § 1 (with $\operatorname{diag}(A) = I$) that is about $6 n_x n_y n_z$ clock cycles, then we see that the preconditioning part has been vectorized, but it is still about 2.5 times as expensive as, e.g., the matrix vector multiplication.

However, the two steps (i) and (ii) in scheme (5.3) can be implemented much more efficiently on the CYBER 205 (and the ETA-10), when using one single sparse vector instruction that comprises the following actions:

(5.4)
- (1) EXPAND the input data vectors under control of their associated bit vectors; the "holes" are filled up with a specified broadcast value: zero for addition/subtraction and one for multiplication.
- (2) Perform the required floating point operation on the expanded input vectors.
- (3) Carry out a specified logical operation on the input bit vectors that gives an output bit vector.
- (4) COMPRESS the output data vector of step (2) under control of the output bit vector of step (3).

In our situation we associate with the expanded y_{i-1jk}-vector V_l a bit vector that has value one in positions where the i-index of the V_l element is less than n_x, and we insert a zero when this index equals n_x (since such an element does not contribute to the y_{ijk} over H_l). The output bit vector in step (3) of (5.4) should contain exclusively $1-s$, so that the COMPRESS in step (4) is not effective.

With such a sparse vector instruction the CPU time for step (c) in (5.1), for all l together, is now reduced to $\approx n_x n_y n_z$ clock cycles. Hence the total CPU-time for solving both triangular factors is about $6 n_x n_y n_z$ clock cycles.

This means that solving *both* triangular systems takes about the same CPU-time as the matrix vector multiplication Ay itself, a quite remarkable result.

Note that the construction of the (modified) incomplete decomposition of A can be vectorized along the same lines as sketched above, which then also leads to a process that runs at near optimal speed on the CYBER 205. For more details on implementations for the CYBER 205, see [22]. Our analysis shows that the preconditioned cg process can be carried out, at least on the CYBER 205, with almost the same computational speed as the scaled cg-process. This implies that we also have succeeded in realizing our third requirement made in § 2. We believe that similar high performances for the

hyperplane approach can be achieved for other vector computers too, possibly with the help of assembler coding.

As an illustration we present for a certain problem the CPU-times for different methods on a CYBER 205 (the $34 * 34 * 34$ problem has been taken from [28, problem 2]:

Method	CPU-time
straightforward ICCG(0)	11.2 sec
cg with diagonal scaled A	1.7 sec
MICCG ($\alpha = .95$, Eisenstat-trick, hyperplane-approach)	0.36 sec

For results, obtained for some of these approaches on Japanese supercomputers, see [27] and [28].

6. Parallelism in the solution of the triangular systems. In [29] it is shown how the idea of twisted factorization of a triangular matrix [4], [30] can be extended to the construction of a twisted incomplete factorization of block tridiagonal linear systems of the type described in § 2. This approach can be used for the construction of a high performance preconditioner on, e.g., a CRAY X-MP/2. This has actually been carried out by constructing an incomplete factorization that is twisted only in the z-direction and optimizing the recursions over each of the two-dimensional-subblocks by the grid diagonal-wise approach discussed in § 4.

This approach has as an interesting side effect that it seems to reduce the number of (preconditioned) cg-steps slightly. Since it can be implemented using also the earlier explained Eisenstat-trick [6], a speed-up of slightly more than two could be obtained on a two-processor CRAY X-MP in comparison with the code on one processor described in § 4.

A further increase in the degree of parallelism, suitable for more parallel/vector processors, is obtained by following the twisted approach also for the two-dimensional subblocks or by neglecting certain off-diagonal blocks in the incomplete twisted factorization, similar to the approach for the regular incomplete decomposition as suggested by Seager [23]. If the twisted factorization for the two-dimensional subblocks is done in a straightforward manner as in [29], then the remaining recursions are no longer vectorizable. However it can be done in such a way that nothing of the vectorizability (obtained by the diagonal-wise approach in two dimensions) is lost. This can be achieved by numbering the gridpoints for each two-dimensional subgrid as follows:

$$\begin{bmatrix} & & & 22 & 14 & 8 & 4 & 2 \\ 19 & & & & & 16 & 10 & 6 \\ 11 & 17 & & & & & 18 & 12 \\ 5 & 9 & 15 & & & & & 20 \\ 1 & 3 & 7 & 13 & 21 & & & \end{bmatrix}.$$

It is clear that the computation for the unknowns on the diagonals with odd numbered gridpoints can be carried out in parallel with the unknowns on the diagonals with the even numbered gridpoints (except for the diagonal right in the middle that must wait for completion until all the other diagonals have finished). Indeed, the computation over each of the diagonals themselves in this ordering scheme is completely vectorizable, as we have already seen in § 4.

This approach can be used to further optimize the performance on the CRAY X-MP/2, since the two-dimensional block in the middle of the twisted factorization, when carried out as in the beginning of the section, degraded the performance slightly. Using the twisted diagonal approach the computation can be done entirely in parallel. Obviously, this approach can also be used to carry out the computation almost entirely in four parallel (and each of them vectorizable) parts.

For computers that allow for an efficient implementation of the hyperplane approach, additional parallelism may also be obtained by constructing a twisted factorization over the hyperplane blocks. The parallel variants and their interesting effects on the convergence behavior are currently the subject of further research.

7. Some other ideas and remarks. Other preconditioners such as, e.g., incomplete decompositions with more fill-in [19], or polynomial preconditioners, can also be vectorized [1], but they cannot be implemented at similar low costs as the standard (modified) incomplete Choleski preconditioners. Moreover, they usually require more memory to store the triangular factors.

When the matrix of the linear system has a less regular structure, as in finite-element discretizations, it is often possible to construct an incomplete decomposition and to use some of the earlier mentioned projection type methods to solve the preconditioned linear system. Though this can be done using Eisenstat's trick, the main problem is now that the recursions cannot easily be vectorized as in the regular structured case. One approach to vectorize at least part of the computation is to use a truncated Neumann series approach for subblocks of the matrix, as suggested in [5] and [31].

We will illustrate the effects of this approach by applying the technique to the regular structured block tridiagonal systems described in § 2. If we partition the matrix in subblocks so that the subblocks along the diagonal correspond to the recursions over a gridline in the x-direction, then the recursions over the y- and z-directions can be vectorized trivially. The recursions for the remaining subblocks can be solved formally by writing

$$y_{jk} = (I + B_{jk})^{-1} z_{jk}. \tag{7.1}$$

B_{jk} represents the off-diagonal matrix coefficients, in the (j, k)th subblock, referring to previous gridpoints on the gridline (with fixed j and k). The vectors y_{jk} and z_{jk} represent the parts of y and z over the (j, k)th gridline. The matrix B_{jk} in this case consists of only one nonzero diagonal, but in general situations the B_{jk} can, of course, be more complicated.

In the truncated Neumann-series approach $(I + B_{jk})^{-1}$ is approximated and y_{jk} is replaced by $\hat{y}_{jk}$:

$$\hat{y}_{jk} = I - B_{jk}(I - B_{jk}) z_{jk}. \tag{7.2}$$

This is more expensive in terms of flops, but it offers more possibilities with respect to vectorization. The approach is more successful when $\| B_{jk} \|_\infty$ gets smaller and becomes much smaller than one. For an analysis, see [31].

Example. For a nicely structured block tridiagonal three-dimensional system, coming from a 35 * 35 * 35-grid, standard ICCG(0) took 60 iterations to get the desired result, while the above sketched truncated Neumann series approach required 61 iterations.

On a CRAY X-MP the standard ICCG(0)-approach took 3.1 CPU-seconds (~30 Mflops), and the Neumann series approach took 0.91 seconds (~115 Mflops). This would be quite an improvement if there was no other vectorization technique available (as there is for this type of matrix).

For the CYBER 205 the figures are:

standard ICCG(0): 11.2 sec (~8 Mflops)
the Neumann approach: 2.0 sec (~50 Mflops).

Although matrices with this nice structure lend themselves to better performances, as shown in § 4, the example indicates that the Neumann series approach might be useful in situations where the other approaches fail. This will also be the subject of further research.

REFERENCES

[1] C. ASHCRAFT 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] I. BABUŠKA, *Numerical stability in problems of linear algebra*, SIAM J. Numer. Anal., 9 (1972), pp. 53–77.

[5] P. F. DUBOIS, A. GREENBAUM, AND G. H. RODRIGUE, *Approximating the inverse of a matrix for use in iterative algorithms on vector processors*, Computing, 22 (1979), pp. 257–268.

[6] S. C. EISENSTAT, *Efficient implementation of a class of preconditioned conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 1–4.

[7] R. FLETCHER, *Conjugate Gradient Methods for Indefinite Systems*, Lecture Notes in Mathematics 506, Springer-Verlag, Berlin, Heidelberg, New York, 1976, pp. 73–89.

[8] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, North Oxford Academic, Oxford, 1983.

[9] A. GREENBAUM AND G. RODRIGUE, *The incomplete Choleski conjugate gradient method for the* STAR *(5-point operator)*, Report UCID-17574, Lawrence Livermore Laboratory, Livermore, CA, 1977.

[10] I. GUSTAFSSON, *A class of 1st order factorization methods*, BIT, 18 (1978), pp. 142–156.

[11] O. G. JOHNSON AND G. PAUL, *Optimal parametrized incomplete inverse preconditioning for conjugate gradient calculations*, Res. Report RC 8644, IBM-Research Center, Yorktown Heights, NY, 1981.

[12] T. L. JORDAN, *A guide to parallel computation and some* CRAY-1 *experiences*, in Parallel Computations, G. Rodrigue, ed., Academic Press, New York, 1982.

[13] D. S. KERSHAW, *The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations*, J. Comput. Phys., 26 (1978), pp. 43–65.

[14] ———, *The solution of single linear tridiagonal systems and vectorisation of the* ICCG *algorithm on the* CRAY-1, Report UCID-19085, Lawrence Livermore Laboratory, Livermore, CA, 1981.

[15] J. R. KIGHTLEY AND I. P. JONES, *A comparison of conjugate gradient preconditionings for three-dimensional problems on the* CRAY-1, Harwell Report CSS 162, AERE Harwell, UK, 1984.

[16] T. A. MANTEUFFEL, *The Tchebychev iteration for nonsymmetric linear systems*, Numer. Math., 28 (1977), pp. 307–327.

[17] G. MEURANT, *Numerical experiments for the preconditioned conjugate gradient method on the* CRAY X-MP/2, Report LBL-18023, University of California, Berkeley, CA, 1984.

[18] 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.

[19] ———, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, J. Comput. Phys., 44 (1981), pp. 134–155.

[20] C. C. PAIGE AND M. A. SAUNDERS, LSQR: *An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71.

[21] 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.

[22] J. J. F. M. SCHLICHTING AND H. A. VAN DER VORST, *Solving* 3D *block bidiagonal linear systems on vector computers*, Report NM-R8819, CWI, Amsterdam, 1988.

[23] M. K. SEAGER, *Parallelizing conjugate gradient for the* CRAY X-MP, Parallel Computing, 3 (1986), pp. 35–47.

[24] P. SONNEVELD, CGS, *a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 36–52.

[25] Y. USHIRO, *Vector computer version of* ICCG *method*, in: M. Mori, Parallel Numerical Computational Algorithms and Related Topics, RIMS Report 514, Kyoto University, 1984 (in Japanese).

[26] H. A. VAN DER VORST, *Iterative solution methods for certain sparse linear systems with non-symmetric matrix arising from* PDE-*problems*, J. Comput. Phys., 44 (1981), pp. 1-19.

[27] ———, (M)ICCG *for* 2D *problems on vector computers*, in Supercomputing, A. Lichnewsky and C. Saguez, eds., North-Holland, Amsterdam, 1988, pp. 321-334.

[28] ———, ICCG *and related methods for* 3D *problems on vector computers*, Report No. A-18, Kyoto University, Kyoto, Japan, 1987.

[29] ———, *Large tridiagonal and block tridiagonal linear systems on vector and parallel computers*, Parallel Comput., 5 (1987), pp. 45-54.

[30] ———, *Analysis of a parallel solution method for tridiagonal linear systems*, Parallel Comput., 5 (1987), pp. 303-311.

[31] ———, *A vectorizable variant of some* ICCG *methods*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 86-92.

[32] ———, *The effect of* MICCG *in the presence of rounding errors*, in progress.

[33] D. M. YOUNG AND K. C. JEA, *Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl., 34 (1980), pp. 159-194.

[34] I. S. DUFF AND G. A. MEURANT, *The effect of ordering on preconditioned conjugate gradient*, Report CSS 221, AERE Harwell, UK, 1988.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 6, pp. 1186–1199, November 1989
© 1989 Society for Industrial and Applied Mathematics
008

APPLICATION OF SPARSE MATRIX SOLVERS AS EFFECTIVE PRECONDITIONERS *

DAVID P. YOUNG†, ROBIN G. MELVIN†, FORRESTER T. JOHNSON‡, JOHN E. BUSSOLETTI‡, LAURENCE B. WIGTON§, AND SATISH S. SAMANT§

Abstract. In this paper the use of a new out-of-core sparse matrix package for the numerical solution of partial differential equations involving complex geometries arising from aerospace applications is discussed. The sparse matrix solver accepts contributions to the matrix elements in random order and assembles the matrix using fast sort/merge routines. Fill-in is reduced through the use of a physically based nested dissection ordering. For very large problems a drop tolerance is used during the matrix decomposition phase. The resulting incomplete factorization is an effective preconditioner for Krylov subspace methods, such as GMRES. Problems involving 200,000 unknowns routinely are solved on the Cray X-MP using 64MW of solid-state storage device (SSD).

Key words. sparse matrices, preconditioning, transonic flow, incomplete factorization

AMS(MOS) subject classifications. 65, 76

1. Introduction.

The work on sparse matrix methods presented in this paper is motivated by the desire to develop reliable solution techniques for difficult partial differential equation (PDE) boundary value problems for complex geometries. Aerospace engineers routinely encounter such problems during the design of aircraft.

In the past much effort has been devoted to the construction of iterative schemes for solving these problems, including time marching, line relaxation, and alternating direction implicity method (ADI). Often these methods are used in conjunction with multigrid acceleration. Despite many notable successes, we have found that even state-of-the-art iterative schemes tend to be too fragile when faced with the rigors of the aerospace design environment. They often slow down or fail altogether when the grid is refined, or when special operators are implemented to account for complex geometries. Moreover, iterative schemes often rely on various parameters that must be carefully tuned on a case-by-case basis to achieve convergence. This is particularly undesirable in the design environment that requires performing hundreds of computer analysis runs in a short period of time.

The difficulties involved in constructing truly reliable iterative schemes have encouraged us to investigate direct solvers. Direct solution of the matrices arising in aerospace engineering problems through the use of sparse matrix solvers has been made even more attractive by recent advances in computer hardware. The Cray X-MP, for example, includes hardware gather/scatter and the large memory provided by the solid-state storage device (SSD). Even with these hardware features, it is commonly thought that direct solution techniques based on Gaussian elimination are far too costly for large problems. However, by using numerical techniques such as physically based nested dissection ordering [1], dynamic drop tolerances for incomplete factorizations [2], and acceleration schemes such as GMRES [3],[4], we have found that direct solution preconditioning is feasible for very large three-dimensional problems and provides the robustness and reliability we desire.

* Received by the editors October 5, 1988; accepted for publication March 1, 1989.
† Boeing Computer Services, Mail Stop 7L-21, PO Box 24346, Seattle, Washington 98124-0346.
‡ Boeing Advanced Systems, Mail Stop 7K-06, PO Box 3707, Seattle, Washington 98124-2207.
§ Boeing Commercial Airplanes, Mail Stop 7K-06, PO Box 3707, Seattle, Washington 98124-2207.

In this paper we will describe a new out-of-core sparse matrix solver and its application to realistic two- and three-dimensional problems arising in aerospace engineering.

2. Description of the sparse matrix solver. The sparse matrix solver package SPSLIB has evolved from the real and complex sparse matrix libraries RSLIB and CSLIB described in [5]. SPSLIB takes full advantage of the hardware features of the Cray X-MP, including gather/scatter, vector compress, and the large memory afforded by the SSD. Considerable attention has been devoted to making all phases of setting up and solving matrix problems convenient and efficient. A brief description of these phases including ordering the matrix elements to minimize the fill-in, matrix assembly, matrix decomposition, and the forward and backward substitution now will be presented.

2.1. Ordering the matrix. Commercially available matrix packages such as SPARSPAK [6] provide ordering schemes such as minimum degree or automated nested dissection to help minimize fill-in during the decomposition phase. Unfortunately, while it is possible to vectorize the computation of the matrix decomposition, the ordering is still done in scalar mode. Thus, on modern vector computers, such as the Cray X-MP, the ordering schemes offered by SPARSPAK can take as much CPU time as is required to decompose the matrix. We have even found instances when the ordering time was 10 times that required to decompose the matrix. Also these ordering methods depend on examining the graph of the matrix and therefore would be difficult to implement out of core.

Fortunately, when solving PDEs on reasonably structured grids, one can easily implement a physically based nested dissection ordering [1]. This ordering is determined by examining the grid on which the PDEs are being solved. The width of the dissectors depends on the local operator stencil and the number of unknowns at each gridpoint. Extension of the physically based nested dissection algorithm to locally refined grids is discussed in §3.2. The physically based nested dissection ordering must be performed in scalar mode, but the operation count is so small that it can be performed in a small fraction of the time required to decompose the matrix. Moreover, in terms of reducing fill-in during the decomposition phase, the physically based nested dissection algorithm proves to be slightly more effective than the minimum degree algorithm built into SPARSPAK and considerably more effective than the automated nested dissection algorithm.

The physically based nested dissection algorithm does not require the graph of the matrix and in fact can be used before the matrix is collected and assembled. Thus the permutation vector computed by the physically based nested dissection algorithm is available to adjust the row and column indices of the matrix entries during the collection process.

2.2. Matrix collection and assembly. SPSLIB allows the user to easily and efficiently collect and assemble the matrix without being familiar with the data structures employed by SPSLIB. The user can input contributions to matrix elements to SPSLIB in random order. This is particularly convenient when, for example, in a finite element method, we want to assemble the global stiffness matrix from the element stiffness matrices. These matrix element contributions along with their corresponding row and column indices are buffered to SSD. Buffering is required because the SSD is only efficient when large contiguous blocks of data are transferred. The input/output cost is virtually eliminated through use of unblocked random data sets

manipulated through the Cray-supplied routines READDR and WRITDR.

After all the matrix element contributions are collected, they are sorted according to row and column index. Elements with identical row and column indices are added together. The matrix is then ready for the decomposition phase. In cases where there are many duplicate row and column indices, SSD storage can be reduced by invoking the sort/add subroutines during the collection process at the expense of additional CPU time.

The sort is accomplished through the use of an in-core radix sort that sorts the biggest blocks of data that can be held in core at one time, followed by an out-of-core merge that merges these blocks together. These sorting techniques are discussed in Knuth [7]. The time required to perform the radix sort is $O(N)$, while the time required to perform the merge is $O(N \ln(B))$, where N is the number of matrix element contributions and B is the number of blocks. The SPSLIB sort routines are considerably faster than those used by SPARSPAK or even the Cray-supplied sort routine ORDERS. However, in cases where there is heavy duplication of row/column indices, the sort/add time can be significant compared to the time required to perform the decomposition (particularly if drop tolerances are used to reduce the decomposition cost). Therefore we are actively pursuing additional efficiency improvements to our sort/add package.

2.3. Matrix decomposition. During the decomposition phase, the matrix is stored in a row format. For the purposes of transferring information between main memory and the SSD, the matrix is partitioned into row blocks. For each element of the matrix, SPSLIB uses two storage locations, one to store the element and the other to store its column index. Optionally, SPSLIB can pack the matrix element and the column index into one word of storage. In this mode, the 64-bit storage location devotes 43 bits to the matrix element and 21 bits to its column index. Although not used in any of the examples in this paper, word packing reduces the SSD storage required to hold the matrix decomposition. Moreover, the CPU time required to perform the packing and unpacking is compensated for by the reduced time spent referencing memory. As a result, the word-packed version of SPSLIB actually runs slightly faster than the unpacked version.

Decomposition of the matrix is accomplished by Gaussian elimination. (For our problems no pivoting has yet been found necessary for numerical stability.) Each element in the lower triangle is eliminated in turn through a sparse SAXPY (SPAXPY) operation with the appropriate row. The multiplier becomes the corresponding element of the L matrix, and the appropriate U matrix row is modified by the SPAXPY operation. Each row block is decomposed, and when finished, used to eliminate corresponding lower triangular elements from all subsequent row blocks. As fill-in occurs the row blocks must be repartitioned and new row blocks added. An input/output package has been developed that does this automatically so that formally the code need only fetch or store any given row. When all lower triangular elements of a given row have been eliminated, small elements in the upper triangular part of the row can be dropped if they are small relative to the current row or column diagonal for that element. The criterion chosen depends on whether the problem is deemed well scaled row-wise or column-wise. The lower triangular elements are dropped in a similar fashion as they are eliminated, obviating the need for a SPAXPY operation.

Unlike conventional sparse matrix solvers, SPSLIB does not rely on a symbolic factorization to facilitate the matrix decomposition. Instead SPSLIB explicitly searches for the nonzero elements created during the matrix decomposition. This strategy al-

lows SPSLIB to easily implement drop tolerances as required for the large problems discussed in the following sections.

As pointed out in the sparse matrix literature (for example, [8]), searching for nonzero elements would be prohibitively expensive on a normal scalar machine. Fortunately, the hardware vector mask and vector compress feature on the Cray X-MP allows us to perform the searches efficiently. For the largest problems solved to date involving extensive use of drop tolerances (those problems incurring the maximum penalty for nonzero searching), about 40 percent of the total decomposition cost is taken by the searches for the next nonzero element below the diagonal to be eliminated. Another 40 percent of the cost of the decomposition is taken by the SPAXPY operations which would be required whether or not a symbolic factorization was available. The final 20 percent is taken by searching for the nonzero elements in the upper part of the matrix created by the SPAXPY operations. (The number of searches of this type depends on the size of blocks that can be held in core relative to the size of the LU decomposition. We think of this 20 percent as being the out-of-core penalty.) Thus in the worst case, for the moderately sparse matrices we encounter in our applications, the searches for nonzero elements increase the cost by about a factor of two. This cost is more than offset by the ability to introduce a drop tolerance.

2.4. Forward/backward substitution. The final phase is the forward/backward substitution. For problems requiring solutions for many right-hand sides, the substitution phase can be more expensive than the decomposition phase. Therefore, it is important to minimize the cost of the forward/backward substitution phase. For the short vector lengths characteristic of sparse matrix operations, we have found that on the Cray X-MP, a sparse vector dot product takes twice as long as a SPAXPY. To take advantage of the relative speed of SPAXPY operations during the solution phase, instead of solving $Ax = b$, we solve the equivalent system $x^t A^t = b^t$. Specification of the transposed problem is easily accomplished by transposing the row and column indices as they are collected by SPSLIB.

3. Applications and results. The algorithm described in the previous section has been applied to solve a number of interesting engineering problems. These include both linear potential flow and nonlinear full potential flow about realistic aerodynamic configurations, Euler flow about two-dimensional airfoils, acoustic scattering from some generic objects, and electromagnetic scattering from a variety of both generic and realistic configurations. There are some significant differences in the character of each of these applications. These differences are due in part to the fundamental mathematical differences in the partial differential equations that describe the physical phenomena and in part due to some implementation differences in the codes built for these applications. Before presenting results and timings for some of these applications, it is useful to review some of these differences, since they provide evidence that the use of sparse solvers is quite effective for a wide range of PDE problems and engineering applications.

3.1. Two-dimensional multi-element airfoil calculations. The first application of SPSLIB was to the solution of inviscid Euler flow around two-dimensional multi-element airfoil configurations. In this application the sparse solver was used to obtain a direct solution to the linearized problems associated with Newton's method. A detailed discussion may be found in [5].

A typical grid for an inviscid multi-element airfoil calculation is shown in Fig. 1. One family of curves in the grid coincide with streamlines associated with inviscid

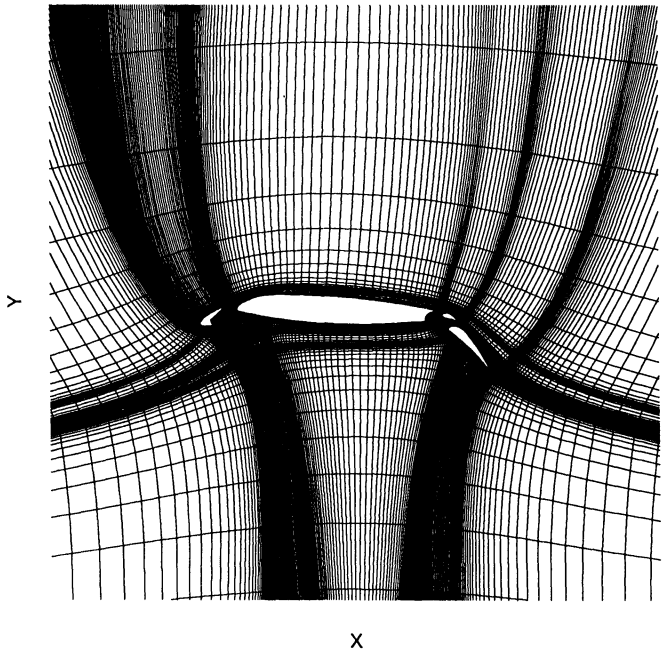


FIG. 1. *Typical computational grid for two-dimensional multi-element Euler solver.*

incompressible flow. The other family of curves nearly coincide with the corresponding equipotential lines, thus producing a nearly orthogonal grid.

The formulation of the Euler equations in this example involved three unknowns per gridpoint with each equation spanning a nine-point stencil. The FORTRAN code required to compute the entries of the Jacobian matrix (a single vectorizable do loop with 850 statements) was written with the aid of a MACSYMA program. SPSLIB was used to solve the linearized problems associated with Newton's method. For two-dimensional problems, a direct decomposition of the resulting sparse matrix is quite inexpensive, particularly with the physically-based nested dissection ordering. By using suitable damping strategies based on the physics of the problem [5], the Newton iteration itself can be made to converge in just a few iterations, thus making the entire computation quite viable.

3.2. Three-dimensional full potential flow. TRANAIR is a computer code for solving the transonic full potential equation about arbitrary aerodynamic configurations [9]–[12]. We use a rectangular grid with local adaptive refinement superimposed on the boundary configuration defined by networks of surface panels. The hierarchical local refinement is described in an oct-tree data structure [13] that may be held in central memory. Figure 2 shows two planar cuts through the locally refined grid used to compute the flow about the F-16 fighter.

The full potential equation

$$\vec{\nabla} \cdot \rho \vec{\nabla} \Phi = 0 \tag{1}$$

is a nonlinear second-order partial differential equation as the density ρ is a nonlinear function of potential Φ.

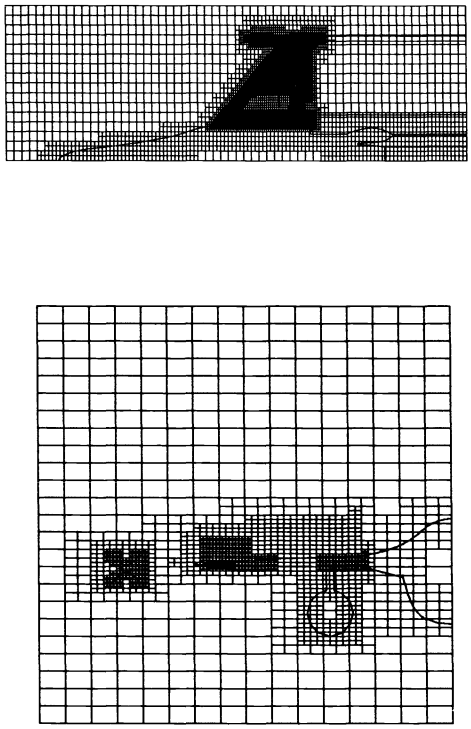


FIG. 2. *Cuts through grid used for* TRANAIR *analysis of* F-16 *configuration.*

A discretization is generated using trilinear finite elements parameterized by the potential Φ at the nodes. Away from the boundary, it is straightforward to define a discrete approximation to the governing partial differential equation. Special operators are defined using finite elements in grid boxes intersected by boundary panels.

The resulting discrete nonlinear set of equations

$$L(\Phi) = f \tag{2}$$

is solved using a damped Newton method. The linear problem generated at each iteration is solved using the preconditioned GMRES algorithm [3],[4]. It is not necessary to compute the full Jacobian matrix for the nonlinear system but only the action of the Jacobian on each search direction used by GMRES. The linear GMRES algorithm, applicable to nonsymmetric systems, minimizes the l_2 norm of the residual of the linear system over a Krylov subspace. For large systems of equations, powerful preconditioning is essential to ensure rapid convergence. A combination of a fast Poisson solver and a direct sparse solver provides excellent preconditioning in the TRANAIR formulation.

The Poisson solver arises from inverting the linear Prandtl–Glauert operator

$$\mathcal{T}\Phi = (1 - M_\infty^2)\Phi_{xx} + \Phi_{yy} + \Phi_{zz} \tag{3}$$

where M_∞ is the freestream Mach number. The Prandtl–Glauert equation is a linearization of the full potential equation about $\vec{V}_\infty$ (the uniform onset flow velocity) and is a good approximation to the nonlinear full potential equation in the regions of subsonic flow away from the boundary [9],[11],[12]. However, this is not a good preconditioner near the boundary and in regions of supersonic flow where the solution is highly nonlinear. The second preconditioner is a sparse solver used on a reduced set of equations. We define the reduced set for the problem to be the set of unknowns located at the corners of boxes that are either refined, cut by the boundary, or contain supersonic flow. For most configurations the reduced set is a thin layer of points surrounding the boundary surface plus a block of points in the supersonic region. The system is closed by the addition of a Dirichlet condition at the boundary of the reduced set except at the downstream face where a Neumann condition is applied. The Jacobian matrix is computed only for the equations involving unknowns in the reduced set.

Ordering the system has a big impact on the fill-in that occurs during the decomposition. Finding a good ordering is complicated by the unstructured local grid refinement and by the fact that the matrix is stored out of core. The ordering is obtained using a physically based nested dissection algorithm derived from the oct-tree data structure. Based on the coordinates of the nodes in the reduced set, candidate "cutting planes" (normal to each of the three coordinates axes) are computed. These cutting planes split the reduced set into two components of approximately equal size. The oct-tree is interrogated to find all grid boxes intersecting the cutting planes and nodes from these boxes are used to determine a dissecting set of nodes. The actual set of nodes in the dissector depends on the local refinement pattern and local operator stencil. The minimum length dissector is chosen from the candidates and the process repeated recursively on the newly formed components of the reduced set.

Typically, the matrices have between 30,000 and 200,000 rows with 20–30 nonzeros per row in subsonic flow. In regions of supersonic flow, there are 100–120 nonzeros per row. This increase is due to the larger operator stencil necessary to include the upwinding needed to rule out expansion shocks [14]. Without the use of a drop tolerance, significant fill-in occurs during the decomposition yielding a decomposed matrix with 10–30 times the number of nonzero entries in the original matrix. For many large problems, the memory required by a full decomposition would be too large for the SSD. Because the matrix is used as a preconditioner to solve the problem iteratively, it is not essential that the decomposition be exact. During the decomposition we drop elements when they are less than a specified fraction of the diagonal element (the drop tolerance). With a suitable choice of drop tolerance, CPU time and memory use for the decomposition are reduced by up to an order of magnitude with only a slight degradation in convergence rate.

3.3. Three-dimensional electromagnetic scattering. EM-TRANAIR is a computer code for the solution of electromagnetic scattering problems in three dimensions [11],[15]. It solves Maxwell's equations for plane wave excitations in the presence of arbitrary isotropic material objects:

$$L\vec{E} = \vec{E} - \alpha\vec{\nabla} \times \beta\vec{\nabla} \times \vec{E} = 0 \tag{4}$$

where α and β characterize the material properties of the local medium and may vary (discontinuously) from point to point, and $\vec{E}$ represents the electric field. The continuous set of equations is discretized in a manner very similar to the TRANAIR fluid

dynamics code. A configuration is defined by specifying two-dimensional surfaces in a three-dimensional space that define boundaries between regions of uniform material properties (α and β). A Cartesian grid is overlaid on the configuration and a discrete set of equations is defined using finite elements with a vector bilinear element [15]. The discrete operator L is a sparse operator with at most 43 points in its stencil. Far-field boundary conditions are satisfied by using an exterior Helmholtz solver in a manner completely analogous to the TRANAIR fluid dynamics code. The system of equations can be quite large. We have applied the code to problems involving about 10^6 field unknowns with more than 10^5 current sources. We solve the system iteratively, using the GMRES algorithm.

A sparse matrix N is defined over a reduced set of the field points

$$N\delta\vec{E} = R \tag{5}$$

where R represents the residuals at the current iterative step. The reduced set is defined by the collection of points in a neighborhood of the boundary, plus some closure points. The equation imposed at the closure points is a discrete outgoing radiation condition. In principle, this is a global condition and would destroy the sparse nature of N. Since we require a good preconditioner, and not an exact inverse for our iterative solution process, we approximate the global relationship with a local radiation condition.

In many scattering problems the reduced set is essentially a two-dimensional surface. In these problems, nested dissection is a particularly effective ordering scheme. The dissection procedure involves selecting a planar cut through a block of the field grid (normal to one of the three coordinate axes) that approximately bisects the number of points in the reduced set. Points that lie in this plane are ordered last, and the process continues on the remaining subblocks until a single point remains in a subblock. The bisection plane is selected to minimize the number of equations in the bisector.

Most of the applications of this approach have used a direct decomposition of the operator N. However, as an option, a drop tolerance may be introduced into the factorization procedure. When a drop tolerance is used, the resulting system can become unstable without some added dissipation. We introduce the dissipation by adding an imaginary component to the wavenumber in the N operator, this simulates a lossy material in which waves are damped with distance, that prevents resonance. To date, we have done only limited experimentation with the drop tolerance/loss combination. We have confirmed that it can significantly reduce the size of the decomposition at the cost of increasing the number of iterations required to solve a problem. With no drop tolerance, convergence to machine error is achieved in less than 20 iterations. With the combinations of drop tolerance and loss that we have explored to date, convergence to the same level requires about 100 iterations. Because of the significant reduction in the size of the factored matrix (typically a factor of 10–20), the decomposition and forward and back substitution steps are considerably less expensive when a drop tolerance is used.

3.4. Results and performance. The approach described in previous sections has been implemented on Cray X-MP machines with supporting SSD storage. It has been tested on problem sizes ranging from tens of thousands of unknowns to hundreds of thousands of unknowns.

Computer times for decomposing the Jacobian matrix for the Euler equations around a multi-element configuration are shown in Table 1. For this two-dimensional

TABLE 1

Performance characteristics for decomposition of nonsymmetric Jacobian matrix for two-dimensional Euler solver (no drop tolerance). Banded refers to natural grid ordering of equations. Sparse refers to physically based nested dissection ordering.

Grid	Equations	Matrix nonzeros	Method	Decomp (MW)	CPU sec	Operations (millions)
160 × 40	18,805	320,000	Banded	9	26	283
			Sparse	4	12	107
240 × 56	39,479	685,000	Banded	27	88	1,180
			Sparse	9	30	290

class of problems it is possible to form a complete decomposition of the matrix and no linear iteration step is necessary. We compare SPSLIB using a physically based nested dissection ordering with SPSLIB using the natural (banded matrix) ordering. We list the number of equations (roughly three times the number of gridpoints for this particular problem), the number of elements in the matrix and in its LU decomposition, the CPU time in seconds (on the Cray X-MP) required to decompose the matrix, and finally the number of floating point operations required to decompose the matrix (an add, multiply pair is counted as one operation). We see that the computer time and memory requirements are quite reasonable, particularly if we use the nested dissection ordering.

For very large problems, particularly those associated with three-dimensional applications, sparse solvers operating over the whole field are not practical, even with the benefits of nested dissection ordering. However, we have found that sparse solvers with a drop tolerance, operating over a reduced set, are extremely effective for such large three-dimensional problems, particularly when used as one of several preconditioners in an iterative solution process, driven by GMRES.

Tables 2–4 give the computer times for decomposition of the reduced set matrix for several representative cases run in EM-TRANAIR without a drop tolerance and TRANAIR with and without a drop tolerance as well as the computer time required for each GMRES iteration. The storage required for the decomposition (using no word packing) is also shown and is equal to twice the number of nonzero entries in the decomposition.

TABLE 2

Performance characteristics for three-dimensional electromagnetics solver (linear complex nonsymmetric indefinite system) with no drop tolerance. Convergence to machine error requires 10–20 *iterations.*

Total equations	Matrix equations	Decomp CPU sec	CPU sec per iteration	Decomp (MW)
21,675	10,502	200	2	17
107,163	29,920	250	4	18
397,953	48,675	600	8	40
1,013,643	103,895	1,550	28	80
536,769	158,927	2,000	22	100

The performance data from Tables 1–4 is summarized in Figs. 3 and 4. Here, the ordinate is the number of unknowns actually given to the sparse solver. (For the TRANAIR problems, this is the number of unknowns in the reduced set. For the

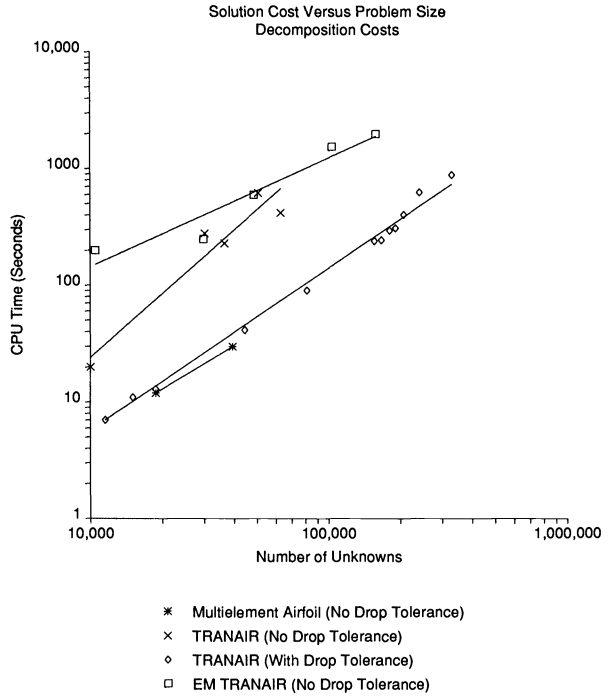


FIG. 3. *Decomposition* CPU *costs for* TRANAIR, EM-TRANAIR, *and multi-element airfoil code.*

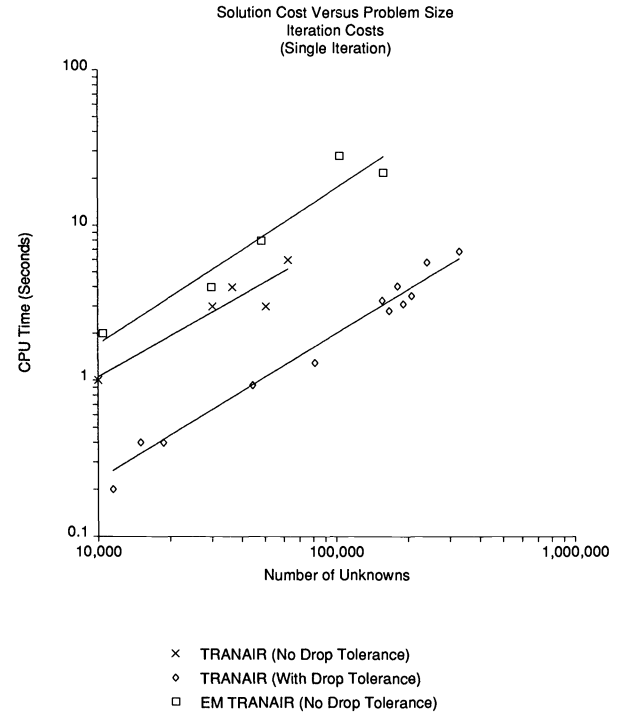


FIG. 4. CPU *costs for* TRANAIR *and* EM-TRANAIR *for a single iteration.*

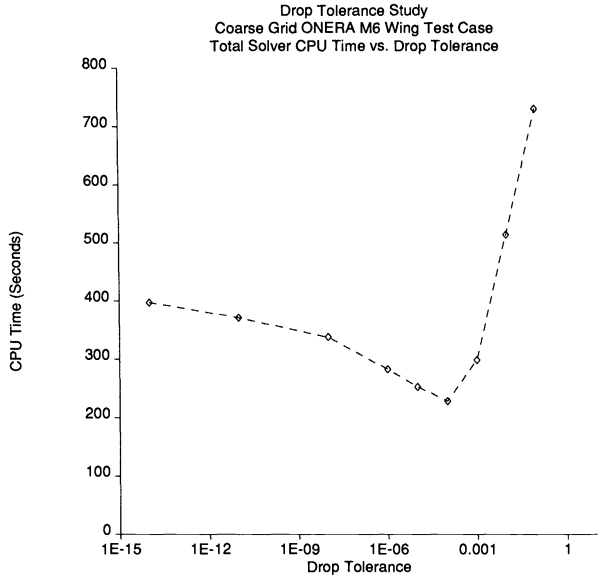


FIG. 5. *Cost versus drop tolerance for the* ONERA M6 TRANAIR *solution.*

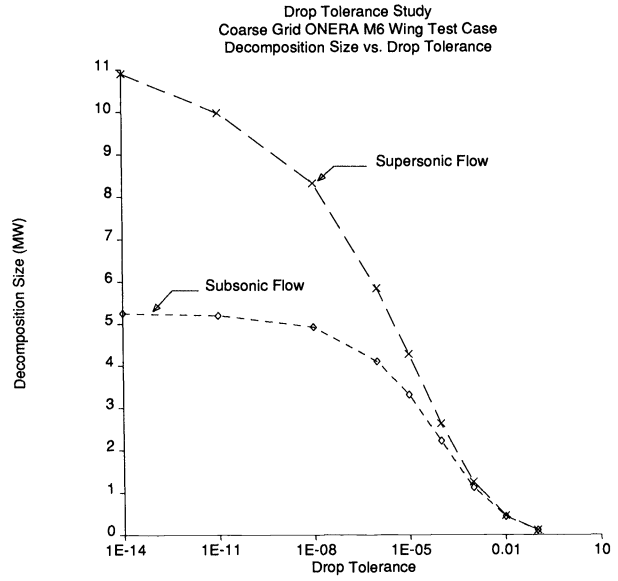


FIG. 6. SSD *storage versus drop tolerance for the* ONERA M6 TRANAIR *solution.*

TABLE 3

Performance characteristics for three-dimensional transonic full potential solver (real nonlinear nonsymmetric system) with no drop tolerance. Ten to twenty nonlinear Newton steps are required for each solution. Each linearized solution requires about 10 GMRES *iterations.*

Matrix equations	Decomp CPU sec	CPU sec per iteration	Decomp (MW)
10,018	20	1	3
30,267	280	3	11
36,627	230	4	16
50,655	620	3	29
63,069	420	6	26

TABLE 4

Drop tolerance for three-dimensional transonic solver (real nonlinear nonsymmetric system) with drop tolerance. Each linearized solution requires about 15–30 GMRES *iterations.*

Total equations	Matrix equations	Drop tolerance	Decomp CPU sec	CPU second per iteration	Decomp (MW)
18,376	11,514	0.0010	7	0.2	1.5
24,304	15,051	0.0010	11	0.4	2.2
37,702	18,731	0.0010	13	0.4	2.4
61,863	44,537	0.0010	41	0.9	6.7
124,878	81,216	0.0005	91	1.3	12.6
236,970	167,500	0.0010	246	2.8	26.4
247,703	156,659	0.0008	241	3.3	28.6
268,301	192,238	0.0010	311	3.1	30.8
284,052	181,802	0.0008	297	4.1	30.6
288,333	207,827	0.0010	405	3.5	40.0
373,813	241,474	0.0010	631	5.8	54.0
480,907	330,857	0.0008	882	6.8	64.6

multi-element airfoil calculations, it is the total number of unknowns in the problem.) Figure 3 gives the CPU time required to perform the decomposition of the sparse matrix. Figure 4 gives the CPU time required per iteration in the TRANAIR context where the number of equations in the reduced set is often substantially less than the number of degrees of freedom in the underlying discrete PDE problem and an iterative method must be used. Tables 2 and 4 give both the number of equations in the reduced set (the size of the problem given to the sparse solver) and the total number of degrees of freedom. The size of the sparse matrix varies from about 10,000 unknowns to around 330,000 unknowns. The maximum decomposition time (a full factorization of a complex matrix with 150,000 unknowns) is under 2,000 seconds. The lower slope for the EM-TRANAIR results in Fig. 4 is due to the effectiveness of nested dissection ordering in this case where the reduced set is often a two-dimensional surface. This is not the case for TRANAIR, except in very special circumstances due to the presence of grid refinement and regions of supersonic flow. Note that with the use of a drop tolerance, the decomposition costs are reduced by nearly an order of magnitude. The costs per iteration (Fig. 4) are also reduced by about a factor of two. Each linear problem requires 10–20 iterations to converge the solution when no drop tolerance is used. By adding a drop tolerance we find that those numbers are increased to 15–30 iterations. However, since the cost per iteration has been reduced

by a factor of two, this helps to compensate for the increase in number of iterations.

Memory limitations make it impossible to test the effect of a full range of drop tolerances for a large problem. However, for a small problem with 28,050 finite elements we have explored the drop tolerance feature in greater detail. Figure 5 illustrates total solution costs as a function of drop tolerance for a fluid dynamics problem (an ONERA M6 wing in transonic flow). Figure 6 illustrates the SSD resource requirements. It is clear that there is a minimum in the total cost for drop tolerances in the range of 10^{-4}. When higher drop tolerances are used, the decomposition costs continue to decrease, but the iterative costs begin to increase due to the larger number of iterations required to reach a given level of convergence. For this configuration, a drop tolerance in the range of 10^{-3} produces a slight increase in CPU time over the optimal value, but (Fig. 6) significantly reduces the amount of SSD storage required (by more than a factor of two). CPU time must be balanced against the SSD storage requirements. Thus some experimentation may be required to determine an appropriate value of drop tolerance. For larger problems, the amount of SSD storage may become the critical limiting factor. The reduction in SSD storage for larger problems can be as high as a factor of 20 with an appropriate choice of drop tolerance.

4. Conclusions. We have demonstrated that out-of-core sparse solvers can be used effectively in the solution of large two-dimensional problems (using a direct sparse matrix solution) and even large three-dimensional PDE problems (using a sparse solver on a reduced set as a preconditioner). On Cray X-MP systems with at least 64 MW of SSD space available, it is possible to solve PDEs involving up to a million unknowns in well under one hour of CPU time. The out-of-core approach that permits the solution of these problems is critically dependent on the availability of sufficient storage on the secondary storage device. Indeed, in the process of executing these performance tests, it rapidly became clear that on Cray X-MP systems the limiting problem size was determined by the amount of SSD space available, rather than by CPU time or by the amount of available central memory. Use of the drop tolerance feature of the sparse solver can reduce the SSD storage requirements for large problems by more than an order of magnitude.

REFERENCES

[1] A. GEORGE AND J. W. H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs, NJ, 1981.

[2] Z. ZLATEV, J. WAZNIEWSKI, AND K. SCHAUMBURG, *Comparison of two algorithms for solving large linear systems*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 486–501.

[3] Y. SAAD AND M. H. SCHULTZ, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, Res. Report YALEU/DCS/RR-254, August 1983; SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[4] L. B. WIGTON, N. J. YU, AND D. P. YOUNG, GMRES *acceleration of computational fluid dynamics codes*, AIAA Paper 85-1494, American Institute of Aeronautics and Astronautics, New York, July 1985.

[5] L. B. WIGTON, *Application of* MACSYMA *and sparse matrix technology to multielement airfoil calculations*, AIAA Paper 87-1142-CP, American Institute of Aeronautics and Astronautics, New York, June 1987.

[6] A. GEORGE, J. W. H. LIU, AND E. NG, *User guide to* SPARSPAK: *Waterloo sparse linear equations package*, Res. Report CS-78-30, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, June 1979.

[7] D. E. KNUTH, *The Art of Computer Programming, Volume* 3, *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[8] S. PISSANETZKY, *Sparse Matrix Technology*, Academic Press, New York, 1984.

[9] D. P. YOUNG, R. G. MELVIN, M. B. BIETERMAN, F. T. JOHNSON, S. S. SAMANT, AND J. E. BUSSOLETTI, *A locally refined rectangular grid finite element method*, Applied Mathematics Tech. Report SCA-TR-108, Boeing Computer Services, Seattle, WA, 1989.

[10] S. S. SAMANT, J. E. BUSSOLETTI, F. T. JOHNSON, R. G. MELVIN, AND D. P. YOUNG, *Transonic analysis of arbitrary configurations using locally refined grids*, in Lecture Notes in Physics 323, D. L. Dwoyer, M. Y. Hussaini, and R. G. Voigt, eds., Springer-Verlag, New York, 1989, pp. 518–522.

[11] P. E. RUBBERT, J. E. BUSSOLETTI, F. T. JOHNSON, K. W. SIDWELL, W. S. ROWE, S. S. SAMANT, G. SENGUPTA, W. H. WEATHERILL, R. H. BURKHART, B. L. EVERSON, D. P. YOUNG, AND A. C. WOO, *A new approach to the solution of boundary value problems involving complex configurations*, Symposium on Future Directions in Computational Mechanics, Anaheim, CA, December 10–13, 1986.

[12] S. S. SAMANT, J. E. BUSSOLETTI, F. T. JOHNSON, R. H. BURKHART, B. L. EVERSON, R. G. MELVIN, D. P. YOUNG, L. L. ERICKSON, M. D. MADSON, AND A. C. WOO, TRANAIR: *A computer code for transonic analyses of arbitrary configurations*, AIAA Paper 87-0034, American Institute of Aeronautics and Astronautics, New York, 1987.

[13] H. SAMET, *The quadtree and related hierarchical data-structures*, Comput. Surveys, 16 (1984), pp. 187–260.

[14] M. HAFEZ, J. SOUTH, AND E. MURMAN, *Artificial compressibility methods for numerical solutions of transonic full potential equation*, AIAA J., 17 (1979), pp. 838–844.

[15] EM-TRANAIR: *A computer program for the solution of Maxwell's equations in three dimensions: Volume 1, theory manual*, AFFDL-TR-87-3082, Flight Dynamics Laboratory, Wright Patterson Air Force Base, OH, December 1987.

SIAM J. SCI. STAT. COMPUT.
Vol. 10, No. 6, pp. 1200–1232, November 1989
© 1989 Society for Industrial and Applied Mathematics
009

KRYLOV SUBSPACE METHODS ON SUPERCOMPUTERS*

YOUCEF SAAD†

Abstract. This paper presents a short survey of recent research on Krylov subspace methods with emphasis on implementation on vector and parallel computers. Conjugate gradient methods have proven very useful on traditional scalar computers, and their popularity is likely to increase as three-dimensional models gain importance. A conservative approach to derive effective iterative techniques for supercomputers has been to find efficient parallel/vector implementations of the standard algorithms. The main source of difficulty in the incomplete factorization preconditionings is in the solution of the triangular systems at each step. A few approaches consisting of implementing efficient forward and backward triangular solutions are described in detail. Then polynomial preconditioning as an alternative to standard incomplete factorization techniques is discussed. Another efficient approach is to reorder the equations so as to improve the structure of the matrix to achieve better parallelism or vectorization. An overview of these ideas and others is given in this article, as well as an attempt to comment on their effectiveness or potential for different types of architectures.

Key words. large linear systems, Krylov subspace methods, iterative methods, preconditioned conjugate gradient, polynomial preconditioning, incomplete LU preconditioning.

AMS(MOS) subject classification. 65F10

1. Introduction.

Scientific problems that are tackled today are perhaps a few orders of magnitude more complex than those dealt with just one decade ago. This trend is likely to accelerate because of the rapid improvement in computational power that will result from parallel processing. However, a characteristic of the current trend in supercomputing is that the easy gains in speed are over: big improvements are only possible by radical changes in architecture as well as software and algorithms.

In this paper we consider the impact of modern supercomputers on the design of iterative methods for solving large linear systems of equations. Because of the increased importance of three-dimensional models, iterative methods are again playing a major role. There is a general consensus that for problems arising from partial differential equations in three-dimensional domains, direct methods alone are too costly, both in terms of storage and computation. For example, a $50 \times 50 \times 50$ grid with five degrees of freedom per grid point, such as Euler's equation in fluid dynamics, will lead to matrices of size $N = 625,000$ and of bandwidth $m \approx 25000$. Even though reordering techniques can be used to exploit the sparsity of the matrix, the complexity of the problem is still very high. For a survey of the impact of supercomputing on aerodynamics research, where these types of problems arise, see Peterson [93]. The main attraction of iterative methods in an example like this one is their low storage requirement. In fact in many cases the matrix need not be stored entirely because it consists of a small number of blocks that are repeated many times.

Krylov subspace techniques, of which the conjugate gradient (CG) method is an example, have increasingly been viewed as general purpose iterative methods, especially since the discovery and popularization of preconditioning techniques [76]. Although these techniques may fail for matrices that are not M-matrices, they are

* Received by the editors October 5, 1988; accepted for publication (in revised form) February 10, 1989. This research was supported by the National Science Foundation under grants US NSF-MIP-8410110 and US NSF DCR85-09970, the US Department of Energy under grant DOE DE-FG02-85ER25001, and by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

† Research Institute for Advanced Computer Science, Mail Stop 230-5, NASA Ames Research Center, Moffett Field, California 94035.

effective for the large class of problems arising from partial differential equations of the elliptic type. For extensions and modifications of point incomplete factorizations see [73], [121], [83]. An important gap in the literature concerns the development of truly general purpose iterative solvers that could replace direct methods with minimum risk of failure.

It is interesting to observe that Krylov subspace methods became popular at almost the same time the first vector computers appeared in the marketplace in the mid-1970s. As a result, considerations for vector implementations were given early on [49], [33], [125], [61], [60]. The survey paper of Ortega and Voigt [91] gives an exhaustive bibliography for research done before 1985 in the general area of solution of partial differential equations on supercomputers. Much still remains to be done since many new techniques for vectorizing and parallelizing standard preconditioners were considered only recently. Along the way several alternative methods have emerged that turned out to be effective even on sequential computers. We will not describe these in this paper but we should mention two such techniques which show tremendous potential. The first is the class of domain decomposition methods. The reader is referred to the recent survey by Chan [27] and to the excellent paper on parallel implementations by Keyes and Gropp [64]. The attraction of domain decomposition methods is that they have been implemented naturally in many engineering applications in the past and, as a result, we may readily benefit from this experience.

The second class of methods that show promise is that of hierarchical multigrid techniques, in the context of finite element discretizations. Briefly, these techniques amount to using hierarchical basis functions, i.e., a basis that consists not only of the nodal functions at the finest grid, but also of the coarser basis functions from which those fine grid functions have been obtained. Thus the function space is identical but its basis has changed. The remarkable property shown by Yserentant [134] is that for two-dimensional problems, the coefficient matrix arising from the discretization of elliptic partial differential equations with such bases has a condition number of $O((-\log\ h)^2)$ instead of the usual $O(h^{-2})$. This has been exploited by Greenbaum, Li, and Chao [51] and Adams and Ong [4] in the context of preconditioned conjugate gradient methods. The idea here is that there is no need to precondition such matrices, except possibly to use a simple diagonal scaling, since their condition numbers are so favorable.

In the brief overview presented in this paper we will mostly consider the problem of implementing preconditioned conjugate gradient type methods for solving linear systems. The framework is that of large sparse linear systems that are not necessarily well structured. We will attempt to address the implementation issues for parallel machines with shared as well as distributed memory, and with a small number of processors as well as a large number of processors.

Before concluding this introduction we would like to comment that the volume of publications on iterative methods is so large that survey papers can no longer be exhaustive. For additional reading we recommend other survey papers by Ortega and Voigt [91], by Axelsson [13], [14] and the recent book by Ortega [90].

The organization of the paper is as follows. In Section 2 we will give an overview of Krylov subspace methods without consideration to parallel implementations. Then we will address the questions of parallel / vector implementations of two specific examples of such methods: the conjugate gradient method for symmetric problems and GMRES for nonsymmetric problems. The effective implementation of the forward and backward solves in preconditioned Krylov subspace methods is important enough

that we will devote a separate section to it, namely, Section 4. In Section 5 we will describe an alternative to incomplete factorization techniques, based on polynomial preconditionings, and give some comparisons with the standard approach. Then we will look at other preconditioners in Section 6, and reordering techniques in Section 7. We will make some concluding remarks in Section 8.

2. An overview of Krylov subspace methods. Given an initial guess x_0 to the linear system

$$Ax = b, \tag{1}$$

a general *projection method* seeks an approximate solution x_m from an affine subspace $x_0 + K_m$ of dimension m by imposing the Petrov-Galerkin condition

$$b - Ax_m \perp L_m, \tag{2}$$

where L_m is another subspace of dimension m. A Krylov subspace method is a method for which the subspace K_m is the Krylov subspace

$$K_m(A, r_0) = \text{span}\,\{r_0, Ar_0, A^2r_0, \cdots, A^{m-1}r_0\}, \tag{3}$$

in which $r_0 = b - Ax_0$. When there is no ambiguity we will denote $K_m(A, r_0)$ by K_m. The different versions of Krylov subspace methods arise from different choices of the subspaces K_m and L_m and from the ways in which the system is preconditioned. The most popular choices of K_m and L_m are the following:

1. $L_m = K_m = K_m(A, r_0)$. This is the orthogonal projection or Galerkin case. The conjugate gradient method is a particular instance of this method when the matrix is symmetric positive definite. Another method in this class is the full orthogonalization method (FOM) [102] which is closely related to Arnoldi's method for solving eigenvalue problems [7]. Also in this class is ORTHORES [57], a method that is mathematically equivalent to FOM. Axelsson [10] also derived an algorithm of this class for general nonsymmetric matrices.

2. $L_m = AK_m; K_m = K_m(A, r_0)$. With this choice of L_m, it can be shown, see e.g., [108], that the approximate solution x_m minimizes the residual norm $\|b - Ax\|_2$ over all candidate vectors in $x_0 + K_m$. In contrast, there is no similar optimality property known for methods of the first class when A is nonsymmetric. Because of this, many methods of this type have been derived for the nonsymmetric case [11], [57], [38], [109]. The conjugate residual method [28] is the analogue of the conjugate gradient method that is in this class. The GMRES algorithm [109], which we will describe in Section 3.5, is an extension of the conjugate residual method to nonsymmetric problems.

3. $L_m = K_m(A^T, r_0); K_m = K_m(A, r_0)$. Clearly, in the symmetric case this class of methods reduces to the first one. In the nonsymmetric case, the biconjugate gradient method (BCG) due to Lanczos [68] and Fletcher [44] is a good representative of this class. There are various mathematically equivalent formulations of the biconjugate gradient method [103], some of which are more numerically viable than others. An efficient variation on this method, called CGS (conjugate gradient squared), was proposed by Sonneveld [118], [95].

4. $L_m = K_m = K_m(A^TA, A^Tr_0)$. This is nothing but the conjugate gradient method applied to the normal equations $A^TAx = A^Tb$, often referred to as CGNR [38]. The condition number of the normal equations is likely to be too large for most

problems to make this approach competitive with the approaches 1 to 3, except possibly for indefinite problems, i.e., problems for which the symmetric part is not positive definite. The LSQR algorithm of Paige and Saunders [92] is an implementation that is somewhat less sensitive to large condition numbers. Moreover, for least squares problems with nonsquare matrices, one must either explicitly or implicitly use an approach based on the normal equations. We put in this category also the method of conjugate gradients applied to $AA^Ty = b$, whose solution y is trivially related to x by $x = A^Ty$. This is often referred to as CGNE, or Craig's method. If we express the Galerkin conditions in terms of the y variable, then, clearly, $K_m = K_m(AA^T, r_0)$ and $L_m = K_m$. Using the relationship $x = A^Ty$ between the x and y variables, we can translate the Galerkin condition that y satisfies in terms of the x variable to find that for the variable x Craig's method corresponds to taking $K_m = K_m(A^TA, A^Tr_0)$ and $L_m = A^{-T}K_m$. Moreover, the main difference between CGNR and CGNE is that the first minimizes the residual norm over K_m while the second minimizes the error norm over K_m.

Note that the methods recently presented in [112], although related to Krylov subspace methods, do not belong to one of the above classes.

An important factor in the success of conjugate gradient-like methods is the preconditioning technique. This typically consists of replacing the original linear system (1) by, for example, the equivalent system

$$M^{-1}Ax = M^{-1}b. \tag{4}$$

In the classical case of the incomplete LU preconditionings, the matrix M is of the form $M = LU$, where L is a lower triangular matrix and U is an upper triangular matrix such that L and U have the same structure as the lower and upper triangular parts of A, respectively. In the general sparse case, the incomplete factorization is obtained by performing the standard LU factorization of A and dropping all fill-in elements that are generated during the process. This is referred to as $ILU(0)$, or $IC(0)$, in the symmetric case. There are more elaborate factorizations that allow a limited amount of fill-in to take place. In particular in the structured case one can let some fill-in appear along specific diagonals to get incomplete factorizations, denoted by $ICCG(k)$, which more closely approximate A [76]. In the general sparse case this technique has been generalized by introducing the notion of level of fill-in [128]: the level of a fill-in element is defined as one plus the sum of the levels of the L and U elements from which it is spawned in the elimination process. Initially, all elements have level of fill-in equal to 0. $ILU(k)$ is then defined as the incomplete factorization that is obtained by dropping all fill-in elements whose level exceeds k. An interesting question that comes to mind immediately is to see whether the more accurate factorizations will perform better in the context of parallel processing. The rationale is that we will deal with denser matrices and therefore the iterative technique will benefit from better vectorization and data locality. Unfortunately, the process of updating the levels is extremely expensive and sequential in nature. Experiments with such techniques reveal that the preprocessing phase of computing the incomplete LU factorization with any level of fill exceeding one dominates the computing time on multivector processors and was rarely competitive with the simpler $ILU(0)$ or $ILU(1)$ preconditioners [5], [116].

Another class of successful preconditioning techniques based on block factorizations was popularized by Concus, Golub, and Meurant [31]. For reasons of space we will only briefly give an overview of these in Section 6, but we should mention that

their performance has been well documented in the literature; see Meurant [82], [78], Eisenstat et al. [36], and Axelsson [13]. There exist many other ways of defining incomplete factorizations of a given matrix, most of which are based on some form of diagonal dominance. For instance, many of the standard point or line relaxation techniques such as Gauss-Seidel, *SOR*, *SSOR*, or *ADI*, see [126], can be used as preconditioners.

The generalized conjugate gradient (GCG) method, introduced by Concus and Golub [30] and Widlund [129] can be viewed as a particular case of a preconditioned conjugate gradient method where the preconditioning matrix is $M = A + A^T$. With this special choice of the preconditioning matrix M, there is a three-term recurrence similar to that of the conjugate gradient method. On the other hand, each step of GCG requires the solution of a linear system with the matrix M, which may be uneconomical.

For the standard *ILU/IC* preconditioners, solving a linear system with the matrix M requires performing a forward and a backward triangular system solution at every step of the Krylov subspace method. This may constitute the main bottleneck in iterative methods if not carefully implemented on supercomputers and will be discussed in detail later.

3. Preconditioned conjugate gradient.

3.1. The algorithm. By far the most popular Krylov subspace method for solving symmetric positive definite linear systems is the preconditioned conjugate gradient method, a version of which is described below. Here M represents the preconditioning matrix.

Algorithm: Preconditioned CG

1. *Preprocess:* Compute preconditioner M.
2. *Start:* $r_0 := b - Ax_0$, $p_0 := z_0 := M^{-1}r_0$.
3. *Iterate:* Until convergence do
   (a) $w := Ap_i$
   (b) $\alpha_i := (r_i, z_i)/(w, p_i)$
   (c) $x_{i+1} := x_i + \alpha_i p_i$
   (d) $r_{i+1} := r_i - \alpha_i w$
   (e) $z_{i+1} := M^{-1}r_{i+1}$
   (f) $\beta_i := (r_{i+1}, z_{i+1})/(r_i, z_i)$
   (g) $p_{i+1} := z_{i+1} + \beta_i p_i$

The above algorithm is nothing but the conjugate gradient method applied to the linear system $M^{-1}Ax = M^{-1}b$ in which the standard Euclidean inner product is replaced by the inner product $(x, y)_M = (Mx, y)$. It has the property of computing an approximate solution whose preconditioned residual vector $M^{-1}(b - Ax_i)$ is M-orthogonal to all the previous preconditioned residual vectors. Therefore it is a Krylov subspace method of the first type (Galerkin) with the matrix A replaced by the preconditioned matrix $M^{-1}A$ and the standard dot product replaced by the M dot product.

Concerning supercomputer implementation, one observes that the main operations in the above algorithm are the following:

1. Setting up of the preconditioner;
2. Matrix vector multiplications (3-a);
3. Vector updates (3-c, 3-d, and 3-g);
4. Dot products (3-b and 3-f);

5. Preconditioning operations (3-e).

In the above list the potential bottleneck is in setting up the preconditioner (1) and in the solution of linear systems with M, i.e., operation (5). Because of their importance, we will address the problem of efficient implementations of the operations in a separate section. Also potentially time consuming are matrix by vector products, which deserve particular attention. The rest of the algorithm consists essentially of dot products and vector updates. Since the dot products may be the source of bottlenecks on some machines, particularly those with a large number of processors, they should be carefully examined. This issue will also be discussed separately.

3.2. Matrix by vector products. Matrix by vector multiplications are relatively easy to implement efficiently on most supercomputers. The first observation that has been made in this context is that this operation can be performed by diagonals when the matrix is regularly structured, i.e., when it consists of a few diagonals [62]. The matrix can be stored in a rectangular array $DIAG(1:n, 1:ndiag)$ and the offsets of these diagonals from the main diagonal may be stored in a small integer array $IOFF(1:ndiag)$.

After initializing the vector y to zero, the main loop for computing $y = Ax$ is as follows:

```
      DO 10 J=1, NDIAG
         JOFF = IOFF(J)
         DO 20 I=1, N
            Y(I) = Y(I) + DIAG(I,J)*X(JOFF+I)
20       CONTINUE
10    CONTINUE
```

This can be implemented efficiently and thus excellent megaflops rates can be reached on vector machines when the matrix is large enough.

For general sparse matrices there have been several attempts to obtain similar performances by either generalizing the diagonal storage scheme [89], [133] or by reordering the matrix so as to obtain a diagonal structure [3], [97]. We will discuss only the first approach here. This approach is of interest only for matrices whose maximum number of nonzeros per row, $jmax$, is small. We then store the entries of the matrix in a real array $COEFF(1:n, 1:jmax)$ together with an integer array $JCOEFF(1:n, 1:jmax)$ that stores the column numbers of each entry of $COEFF$. We refer to this as the ITPACK/ELLPACK format. The above FORTRAN loop then becomes

```
      DO 10 J=1, JMAX
         DO 20 I=1, N
            Y(I) = Y(I) + COEFF(I,J)*X(JCOEFF(I,J))
20       CONTINUE
10    CONTINUE
```

The main difference between this loop and the previous one is the presence of indirect addressing in the innermost computation. Note that if the number of nonzeros per row varies substantially, then many zero elements must be stored unnecessarily, and this scheme may become inefficient.

When considering matrices from real applications we observe that there is always some structure to them. In particular most matrices are such that a large percentage of their elements belong to a few diagonals. We can therefore extract a small number of diagonals, store them as was described above for structured matrices and we put the rest on the elements in a general sparse matrix storage. The complexity of a code that does this conversion is on the order of the nonzero elements and is therefore

affordable. The payoff could be quite high, especially on vector machines that do well on long vectors. Clearly, here again many zero elements may have to be added to fill the diagonals and so we must be careful when assessing performances.

All the above storage schemes are specialized to some degree to certain types of matrices. These can perform well in many instances but their lack of generality is a serious limitation. Unfortunately, as is often the case, there is a conflict between generality and efficiency. One of the most general schemes for storing sparse matrices is the compressed sparse matrix storage scheme described next. The data structure consists of three arrays. First, a real array $A(1 : nnz)$ stores the nonzero elements of the matrix rowwise, i.e., the elements of a given row are stored contiguously. Then an integer array $JA(1 : nnz)$ stores the column positions of the elements in the real array A. Finally, an integer array $IA(1 : n+1)$ is a pointer array, in that its ith entry points to the beginning of the ith row in the arrays A and JA. This data structure is often referred to as the general sparse format, or the A, JA, IA format. With this storage scheme each component of the resulting vector y can be easily computed independently as the dot product of the ith row of the matrix with the vector x. In FORTRAN 8-X, we can write this as

```
      DO 10 I=1, N
            K1 = IA(I)
            K2 = IA(I+1)-1
            Y(I) = DOTPRODUCT( A(K1:K2) , X(JA(K1:K2)) )
10    CONTINUE
```

From the implementation point of view, an important observation is that the outer loop can be performed in parallel. On a machine like the Alliant FX-8, the synchronization of this outer loop is inexpensive and the performance of the above program can be excellent.

On distributed memory machines the above loop can be split and a number of its steps will be executed in each processor. The splitting may be done in such a way that roughly the same amount of work is performed in each processor, taking other parts of the CG algorithm into consideration. The part of the matrix that is needed is loaded in each processor initially. However, interprocessor communication will be needed to get necessary parts of the vector x that do not reside in a given processor. For general sparse matrices, it may not be easy to find the mapping that achieves the best overall time.

The indirect addressing involved in the second vector in the dot product loop is handled by a special hardware instruction called a *Gather* operation. The vector $X(JA(k1 : k2))$ is first gathered from memory into a vector of contiguous elements. The dot product is then carried out as a standard dot product operation between two dense vectors.

In case the matrix is stored by columns instead of rows, we can use the following program to compute $y = Ax$:

```
      DO 10 J=1, N
            K1 = IA(J)
            K2 = IA(J+1)-1
            Y(JA(K1:K2)) = Y(JA(K1:K2)) + X(J) * A(K1:K2)
10    CONTINUE
```

Clearly, the above code also computes the product of the transpose of a matrix by a vector, when the matrix is stored rowwise in the A, JA, IA format. Normally, the vector $Y(JA(k1 : k2))$ is gathered and the SAXPY operation is performed in vector mode. Then the resulting vector is "scattered" back into the positions $JA(*)$, by what

is called a *Scatter* operation. However, a major difficulty with the above FORTRAN program is that it is intrinsically sequential. First, the outer loop is not parallelizable as it is, but this may be remedied, as will be described shortly. Second, the inner loop involves writing back results of the right-hand side, into memory positions that are determined by the indirect address function JA. To be correct $Y(JA(1))$ must be copied first and then $Y(JA(2))$, etc. However, if it is known that the mapping $JA(\mathrm{i})$ is one-to-one then the order of the assignments no longer matters. Since compilers are not capable of deciding whether this is the case, a compiler directive from the user is necessary for the Scatter to be invoked. Going back to the outer loop, we can split it in p distinct parts and compute p subsums into p temporary (full) vectors, which are added after completion to get the result vector y. This last part constitutes additional work but it is highly vectorizable and parallelizable.

The first vector machines that appeared did not perform too well on sparse computations because they were not equipped with special instructions for Gather and Scatter. The beneficial impact of hardware Scatter and Gather on vector machines has been discussed in [69].

For vector machines the previous two techniques are likely to perform poorly because they involve vectors that are usually very short. For example, for a typical two-dimensional problem with one unknown per grid point, the number of nonzeros per row is at most 5, when finite differences are used. One option is to use one of the schemes based on diagonal or generalized banded format described above. However, the following scheme, related to the stripe structure of Melhem [77], is a more general alternative. We start from the A, JA, IA data structure and reorder the rows of the matrix according to their number of nonzeros, decreasingly. Then, a new data structure is built by constructing what we call "jagged diagonals" (j-diagonals). We store as a dense vector the leftmost element from each row, together with an integer vector containing the column positions of each element. This is followed by the second jagged diagonal consisting of the elements in the second position from the left. As we build more and more of these diagonals, their length decreases. The number of j-diagonals is equal to the number of nonzero elements of the first row, i.e., to the largest number of nonzero elements per row. To multiply a matrix by a vector using this scheme we can proceed as follows, where we denote by $IDIAG(j)$ the pointer to the beginning of the jth jagged diagonal, and by $JDIAG(k)$ the column position of the element stored in $A(k)$:

```
      DO 10 J=1, NDIAG
            K1 = IDIAG(J)
            K2 = IDIAG(J+1)-1
            LEN = K2-K1+1
            Y(1:LEN) = Y(1:LEN) + A(K1:K2)*X(JDIAG(K1:K2))
10    CONTINUE
```

On one processor of the Cray-2, the asymptotic speed of the above code is around 39 Mflops [94]. Note that since we assume that the rows of the matrix A have been permuted, the above code will compute a permutation of the vector Ax for the unpermuted matrix A. It is possible to permute the result back to the original ordering after the execution of the above program. One can also postpone this operation until the final solution has been computed, so that only two permutations on the solution vector are needed, one at the beginning and one at the end. For preconditionings that require a different ordering of the unknowns to achieve a good efficiency, it will be necessary to perform a permutation before or within each call to the preconditioning

TABLE 1
Megaflop rates for five matrix by vector multiplication kernels on an Alliant FX-80.

Grid size	Method				
	1	2	3	4	5
20 x 20	6.56	0.19	8.04	9.43	7.11
20 x 20 x 10	7.06	0.21	7.013	10.13	4.05
30 x 30	6.7	0.19	8.98	10.83	8.19
30 x 30 x 10	5.42	0.21	6.48	9.69	3.32

subroutines.

As an illustration we show in Table 1 the performance of the following five different ways of multiplying a matrix by a vector:

1. Rowwise storage (sparse dot product form);
2. Columnwise storage (sparse saxpy form);
3. Diagonal storage (triad form);
4. ITPACK format;
5. Jagged diagonal format.

The test was done on an Alliant FX-80, in double precision arithmetic, using 5-point and 7-point matrices for two-dimensional and three-dimensional rectangular grids.

Notice the wide differences in performance obtained between these five kernels that perform the same operation. On the Alliant FX-80, method 2, using the columnwise storage, is the worst performer. It is important to stress that relative performance is machine dependent: the dot product scheme which performs well here would not do well on vector machines. Also of interest, and to some extent disturbing, is the variation in performance obtained for different matrices *with the same kernel.* These discrepancies are especially noticeable in kernel 5, using the jagged diagonal format. Here the large degradation in performance when passing from two-dimensional to three-dimensional grids is due to the fact that the amount of data that is used is so large that it does not fit in the 128 KB cache. For the 30 x 30 x 10 matrix, the number of double precision words needed to hold the matrix plus the input and output vectors is roughly 7 x N + 2 x N = 9 x N, i.e., 81K words, while the cache size is only 64K words. At this point, execution time is dominated by memory traffic. Every time a j-diagonal is used, i.e., for each outer loop, the above code sweeps itself through the j-diagonal array $A(k1 : k2)$ itself plus the integer arrays for the indirect addressing $JDIAG(k1 : k2)$ and finally through the vectors x and y, with a very high cache-miss ratio. This situation might be remedied by unrolling the j loop to avoid unnecessary reloading of the vectors x and y.

One problem that seems not to have been studied in the literature is that of performing simple operations with general sparse matrices on SIMD machines like the MPP or the Connection machine. Much of what has been accomplished on such machines are tests with the symmetric conjugate gradient method for easy model problems [26], [24]. The difficulty with the more realistic general sparse problems is the apparent necessity of resorting to indirect addressing, a difficult operation on these architectures. Hammond and Law [55] propose a hardware solution based on systolic arrays. This challenging problem must be solved before SIMD machines can be considered real contenders to MIMD machines in the race for practical supercomputers.

3.3. The problem of the dot products. It has been observed that the dot products in the conjugate gradient algorithm constitute a bottleneck on many parallel or vector machines. This is because when all the vectors in the algorithm are split equally among the processors dot products require global communication. However, this need not be a problem unless the number of processors becomes large.

Another minor difficulty caused by the dot products is the fact that they constitute two synchronization points in the algorithm. The dot products must be completed before anything else can be done and no other computation can be done while they are being computed. As was observed independently in [59] and [105] this can be partially overcome by exploiting the orthogonality of the vectors r_{i+1} and r_i from which we can derive the equality,

$$\|r_{i+1}\|_2^2 + \|r_i\|_2^2 = \alpha_i^2 \|Ap_i\|_2^2. \tag{5}$$

As a result β_{i+1} can be computed from α_i and Ap_i, leading to the following formulation, when $M = I$ (no preconditioning). Note that the algorithm can easily be extended to the preconditioned case; see [81], [80].

Algorithm: CG, version 2

Start: $p_0 := r_0 := b - Ax_0$ and $\rho_0 := \|r_0\|_2^2$

Iterate: For $i = 0, 1, \cdots$ until convergence do:

1. Compute $w := Ap_i$, (Ap_i, r_i), and $\|Ap_i\|_2^2$;
2. Compute the scalars

$$\begin{aligned} \alpha_i &:= \rho_i/(Ap_i, r_i) \quad , \quad \cos^2\theta_i := \frac{(Ap_i, r_i)^2}{\rho_i \|Ap_i\|_2^2}, \\ \beta_i &:= \tan^2\theta_i = 1/\cos^2\theta_i - 1 \quad , \quad \rho_{i+1} := \beta_i \rho_i. \end{aligned}$$

3. Compute
   - $x_{i+1} := x_i + \alpha_i p_i$
   - $r_{i+1} := r_i - \alpha_i w$
   - $p_{i+1} = r_{i+1} + \beta_i p_i$

Unfortunately, the above version of Algorithm CG is unstable and there is a simple analysis to understand the difficulty and to remedy it. To simplify the notation we introduce the quantities

$$c_i \equiv \cos^2\theta_i \quad , \quad s_i \equiv \sin^2\theta_i \quad , \quad t_i \equiv \tan^2\theta_i \quad , \quad \tau_i \equiv c_i\rho_i = \frac{(Ap_i, r_i)^2}{\|Ap_i\|_2^2}. \tag{6}$$

Then we have the following result on the relative error on ρ_{i+1}:

$$\frac{\delta\rho_{i+1}}{\rho_{i+1}} \approx \frac{\delta\rho_i}{\rho_i} + \frac{\delta t_i}{t_i}. \tag{7}$$

We have similarly for t_i,

$$\frac{\delta t_i}{t_i} \approx -\frac{\delta c_i}{c_i^2}\frac{1}{t_i} = -\frac{\delta c_i}{c_i^2(1/c_i - 1)} = \frac{-\delta c_i}{c_i s_i} \tag{8}$$

and finally for c_i,

$$\frac{\delta c_i}{c_i} \approx \frac{\delta\tau_i}{\tau_i} - \frac{\delta\rho_i}{\rho_i}. \tag{9}$$

From (7), (8) and (9) we get

$$\frac{\delta\rho_{i+1}}{\rho_{i+1}} \approx \frac{\delta\rho_i}{\rho_i} - \frac{1}{s_i}\left[\frac{\delta\tau_i}{\tau_i} - \frac{\delta\rho_i}{\rho_i}\right] = (1+\frac{1}{s_i})\frac{\delta\rho_i}{\rho_i} - \frac{1}{s_i}\frac{\delta\tau_i}{\tau_i}. \tag{10}$$

The above formula suggests that there is a danger of catastrophic error when s_i gets close to zero. In fact even if s_i is not small, the above expression tells us that the accumulated error may grow exponentially. To simplify our model a little further, let us assume that $\tau_i = (Ap_i, r_i)^2/\|Ap_i\|_2^2$, a quantity that is computed locally and independently of ρ_i has no error in it, i.e., that $\delta\tau_i = 0$. Then the above formula yields

$$\frac{\delta\rho_{i+1}}{\rho_{i+1}} \approx (1+\frac{1}{s_i})\frac{\delta\rho_i}{\rho_i}. \tag{11}$$

The meaning of the above relation is that even under the strong condition where there are no *local* errors introduced, an initial error on ρ_0 will affect the predicted value of ρ_{i+1} by an error that grows like the product of the factors $1 + 1/s_i$. The resulting algorithm will be unstable in general.

A simple remedy would be to keep track of the growth of the estimated error and recompute occasionally the residual norm by the usual formula. In fact Meurant [81], [80] recomputes it at every step and shows that the additional dot product is worth the cost on a vector machine like the Cray X-MP. Note that the scalar s_i on which the error estimate is based is available at every step for free. The above analysis can be extended to the preconditioned conjugate gradient method, by using the M-inner product instead of the Euclidean inner product.

We can take the above idea of postponing inner products one step further and ask whether it is possible to derive a version of the conjugate gradient method that has as few synchronization points as possible. One idea is to try to execute m steps of the conjugate gradient algorithm at once [29]. However, one can expect to encounter the same stability difficulties as before. Another simple and reliable way to reduce bottlenecks due to inner products is to use polynomial preconditioning as will be described in Section 5.

There are some difficult issues when implementing algorithms on parallel/vector machines pertaining to the best use of the available hardware and software. We have mentioned the delicate problem of optimizing the use of vector registers, the hardware gather and scatter operations, and the disastrous effect that cache memories may have on sparse computations. Work by Meurant on a Cray X-MP-48 [80], Seager [115], and Koniges [66] addressed the question of how to implement conjugate gradient methods and best exploit multitasking and microtasking. Efficient implementations of different preconditioning techniques is more difficult on multivector processor machines such as the Alliant FX-8 because of the complicated side effects of the memory hierarchy [56], [75]. Thus, it seems that there are no all-purpose preconditioning techniques.

3.4. Preconditioned GMRES. GMRES [109] is an effective conjugate gradient-like algorithm for solving general large sparse linear systems of equations of the form

$$Ax = b. \tag{12}$$

Assuming a preconditioner M is used on the left, see [38], we will be solving, instead of (12), the preconditioned linear system

$$M^{-1}Ax = M^{-1}b. \tag{13}$$

A brief description of the preconditioned GMRES method follows. Details can be found in [109].

Algorithm : Preconditioned GMRES

1. *Start:* Choose x_0 and a dimension m of the Krylov subspaces.
2. *Arnoldi process:*
   - Compute $r_0 = M^{-1}(b - Ax_0)$, $\beta = \|r_0\|$ and $v_1 = r_0/\beta$.
   - For $j = 1, 2, \cdots, m$ do:

$$\begin{aligned} h_{i,j} &= (M^{-1}Av_j, v_i), \quad i = 1, 2, \cdots, j, \\ \hat{v}_{j+1} &= M^{-1}Av_j - \sum_{i=1}^{j} h_{i,j}v_i, \\ h_{j+1,j} &= \|\hat{v}_{j+1}\|, \\ v_{j+1} &= \hat{v}_{j+1}/h_{j+1,j}. \end{aligned} \tag{14}$$

   Define H_m as the $(m+1)$ x m upper Hessenberg matrix whose nonzero entries are the coefficients h_{ij}.
3. *Form the approximate solution:*
   - Find the vector y_m which minimizes the function $J(y) = \|\beta e_1 - H_m y\|$ where $e_1 = [1, 0, \cdots, 0]^T$, among all vectors of R^m.
   - Compute $x_m = x_0 + V_m y_m$
4. *Restart:* If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.

Each outer loop of the above algorithm, i.e., the loop consisting of steps 2, 3, and 4, is divided in two main stages. The first stage is an Arnoldi step and consists of building a basis of the Krylov subspace K_m. The second consists of finding in the affine space $x_0 + K_m$ the approximate solution x_m which minimizes the residual norm. This is found by solving the least squares problem of size $m + 1$ of step 3, whose coefficient matrix is the upper Hessenberg matrix H_m.

Note that in practice, the least squares solution of the $(m + 1) \times m$ problem in step 3 is solved by a QR factorization of the matrix H_m which is updated at each step of the Arnoldi process in step 2. With this implementation we can obtain at no additional cost the residual norm of the corresponding approximate solution x_k without having to actually compute it; for details see [109]. This allows us to stop at the appropriate step.

When the preconditioned matrix is positive real, then GMRES is theoretically equivalent to the generalized conjugate residual (GCR) method [38] and to ORTHODIR [53] and it is known to converge. Moreover, it is less costly both in terms of storage and arithmetic [109]. It can be shown that, in exact arithmetic, the method cannot break down although it may be very slow or even stagnate in cases when the matrix is not positive real. For details on stagnation and breakdown behaviors of Arnoldi and GMRES methods, see the recent analysis by Brown [23].

Consider the implementation of the above algorithm on a vector or parallel machine. As before we start by enumerating the main kernels of the algorithm:

1. Setting up the preconditioner;

2. Matrix by vector multiplication;
3. Orthogonalizing a vector against a set of orthogonal vectors;
4. Vector updates;
5. Preconditioning operation.

The work involved in solving the small least squares problem in step 3 of the algorithm is negligible for large linear systems.

The new operation here with respect to the conjugate gradient method is the orthogonalization of the vector Av_i against the previous v's. The usual way of accomplishing this is via the modified Gram–Schmidt process, which is basically a sequence of subprocesses of the form

- Compute $\alpha = (y, v)$,
- Compute $\hat{y} := y - \alpha v$,

consisting of orthogonalizing a vector y against another vector v of norm one. Thus the outer loop of the modified Gram–Schmidt is not parallelizable, but the inner loop, i.e., each subprocess, can be parallelized by dividing the inner product and SAXPY operations among processors. Although this constitutes a perfectly acceptable approach for a small number of processors, the elementary subtasks may be too small for this approach to be efficient on a large number of processors. In that case one solution is to use a standard Gram–Schmidt process with reorthogonalization. This would replace the previous sequential orthogonalization process by a matrix operation of the form $\hat{y} = y - VV^T y$, i.e., BLAS-1 kernels are replaced by BLAS-2 kernels. Further, we should point out that BLAS-3 type kernels that usually allow extremely high performances on machines with caches or local memories [47] cannot be used here because at every step we have only one vector to orthogonalize against all previous ones. To some extent this may be remedied by using block methods considered next. An alternative technique for the orthogonalization process in GMRES was recently proposed by Walker [127]. This technique, based on the Householder process, has superior numerical properties, and can easily be parallelized.

Radicati and Robert [98] compare the performance of several Krylov subspace methods on an IBM 3090 VF. In particular they discuss good strategies for implementing the main kernels on this machine and provide an exhaustive set of experiments. The paper focuses on iterative solvers rather than preconditioners and concludes that although it is difficult to make any definite statements as to an overall best method, on the average GMRES and CGS did perform better than the other iterative methods tested.

3.5. Block Krylov subspace methods. The idea of using a block of vectors instead of a single vector in methods such as the conjugate gradient algorithm and the Lanczos algorithm has been suggested by several authors as a simple means for increasing parallelism [107], [106], [85], [87]. The main idea can be explained by assuming that we have to solve a linear system of the form $AX = B$ where the right-hand side is no longer a single vector but an $N \times p$ matrix. Then a natural modification of the CG algorithm consists of replacing all the operations with single vector by operations with blocks of p vectors. After convergence, the block conjugate gradient algorithm would have solved the p systems simultaneously. If only one linear system must be solved then additional artificial right-hand sides must be created. In absolute terms this approach is not efficient, i.e., the total number of arithmetic operations is likely to be much higher than with the standard single vector method. Note that there are many problems involving linear systems with several right-hand sides in which case a block method becomes very attractive even on scalar machines.

A well-known attraction of the block methods is that they tend to reduce the number of accesses to secondary storage for out-of-core problems. Thus a block Lanczos algorithm was found preferable to the single vector Lanczos algorithm in the context of eigenvalue calculations [52], [70]. Many of the modern supercomputers emphasize a hierarchical organization of the memory using principles similar to those used for secondary storage in traditional computers, except that the number of levels in the hierarchy is higher. One example of this architecture is the CEDAR multiprocessor of the University of Illinois [67]. Access to data from the global shared memory in CEDAR is more expensive than access to data within each local memory (cluster memory). For this reason, block methods may be important since they will help reduce interprocessor communication, in the same way they have been used in the past to reduce the number of accesses to secondary storage [52]. Block methods can also be combined with the other alternatives described in this paper, such as reordering, and they offer an excellent potential for shared memory machines with a hierarchical organization of memory.

4. Solution of sparse triangular systems. Each step of a preconditioned iterative method involves computing

$$z = M^{-1}y.$$

In typical preconditioners M is the product of a lower and an upper triangular matrix, often having the same sparsity pattern as the lower and the upper triangular parts of the original matrix. We consider in this section different ways of performing this operation, which is critical to the performance of the preconditioned conjugate gradient method. We only consider lower triangular systems of the form

$$Lx = b. \tag{15}$$

Without loss of generality we will assume that L is unit lower triangular. If not, the matrix can be scaled before the CG iteration is started so as to save N multiplications per CG step.

If we assume that the matrix L is stored rowwise in a general sparse format, using the standard sparse storage scheme,

- AL : nonzeros of L, stored by rows,
- JAL: column numbers for each element of AL,
- IAL: IAL(i) points to the start of row i in AL, JAL,

the sequential elimination sweep is as follows:

Algorithm: Forward elimination for a sparse triangular system

```
x(1)  =  b(1)
do i  =  2, N
         x(i)  =  b(i)
         do j= ial(i), ial(i+1)-1
               x(i)  =  x(i) - al(j) * x(jal(j))
         enddo
enddo
```

The outer loop corresponding to the variable i is sequential. The j loop is essentially a sparse dot product of the ith row of L and the dense vector x. This dot product can be split among the processors and the partial results will then be added at the end. This is what the Alliant FX-8 compiler would do if the proper optimization options are invoked. However, if the length of the dot product is very short, the

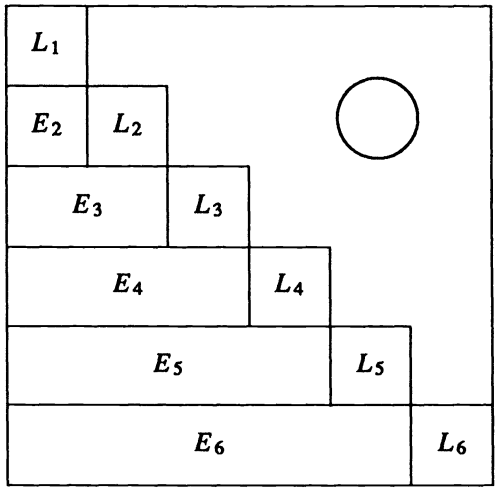


FIG. 1. *Block partitioning for L .*

synchronization overhead and additional operations involved will make this approach inefficient. Consequently, on a machine such as the Alliant FX-8, it is often better not to use the optimized version of the compiler for the above code. As an alternative we may store the matrix by columns and use a column-oriented algorithm. However, the difficulties are identical and we omit the details.

We briefly describe two distinct approaches for breaking the sequential nature of the above implementation; for details see [6], [5].

4.1. Blocking. In the blocking approach, the right-hand side b and the solution x of (15) are partitioned into subvectors $b_1, b_2, \cdots, b_m$ and $x_1, x_2, \cdots, x_m$, respectively. According to this partitioning, the matrix L will have the structure shown in Fig. 1.

If we denote by $\hat{x}_j$ the vector consisting of the subvectors $x_1, x_2, \cdots, x_j$, then the algorithm for computing the solution of the linear system (15) can be written as follows:

Algorithm: Block forward elimination
$x_1 = L_1^{-1} b_1$
Do $i = 2, \cdots, m$:

$$x_i = L_i^{-1}(b_i - E_i \hat{x}_{i-1}). \tag{16}$$

Each step, except the first, consists of two basic operations: first a multiplication of the vector $\hat{x}_{i-1}$ by E_i and then a solution of a triangular system with the matrix L_i. The multiplication $E_i \hat{x}_{i-1}$, a sparse matrix times vector operation, causes no difficulty because all the inner products of rows of E_i with $\hat{x}_{i-1}$ can proceed in parallel. The other significant operation, computing $x_i = L_i^{-1} y$ by solving a lower triangular system involving L_i, is the same problem with which we started, although on a smaller scale.

The first option is simply to solve this smaller system by the usual sequential algorithm using forward elimination. Clearly, this suffers from the same sequential nature as the initial algorithm, but it does have the advantage of preserving the sparsity of the original problem and there is the potential benefit that the sequential

bottleneck is now shorter at every step since at least the matrix by vector products are done in parallel. The experiments in [6] reveal that the performance of this alternative is not too different from that of the sequential algorithm; see also the experiments reported at the end of this section. The second option consists of computing the inverse of each matrix L_i at the beginning and storing it as a dense matrix. Then the triangular system solution in (16) can be replaced by the multiplication of a triangular matrix by a vector. The attraction is that multiplying a vector by a matrix is highly parallelizable and vectorizable and despite the additional arithmetic, could be cost-effective when solving a large number of systems with the same matrix. Note that an intermediate option for preconditioners is to compute a sparse approximate inverse for the triangular matrix L_i instead of a full inverse. We can afford to make such a substitution because we are only interested in using the inverse of L to precondition the original system. This approach is essentially to the one used in [125] for the diagonally structured problems.

4.2. Level scheduling. In the block forward elimination, the main bottleneck is in solving each of the subsystems (16). If the matrices L_i were diagonal matrices the main difficulty would be removed and (16) would be a fully parallelizable operation of the order of the block size. For many sparse matrices it is possible to obtain such a block triangular system simply by reordering the rows and columns of the coefficient matrix. We will describe a simple ordering called *level scheduling* [6], [111], [130], whose objective is to obtain a block lower triangular system like the one in Fig. 1, where the E_i's are sparse rectangular matrices and the L_i's are diagonal blocks.

The idea of *forward scheduling* is as follows. Consider the following formula to compute the ith unknown,

$$x_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j<i \ , \ l_{ij} \neq 0} l_{ij} x_j \right). \tag{17}$$

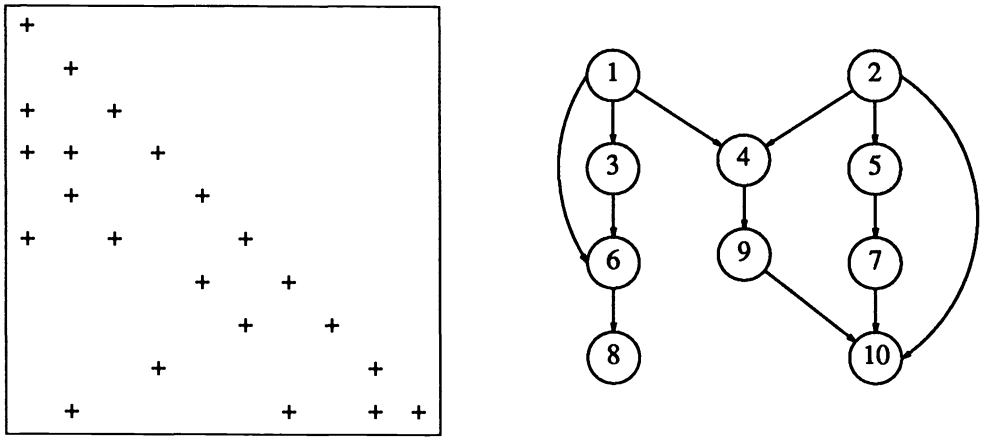
This can be executed after all the components x_j needed in the sum have been computed. The idea is to look at the adjacency graph of the matrix at the outset, and determine groups of equations that can be solved at the same time [6], [111], [130]. Recall that a node of the graph corresponds to a row of L or, equivalently, a component of x, and there is a directed edge from node j to node i if and only if $l_{ij} \neq 0$, which indicates that x_j must be known to solve for x_i. Since L is lower triangular, the adjacency graph is a directed acyclic graph. Fig. 2 shows the digraph for a small sample matrix.

Thus, the first step of the solution algorithm consists of determining x_1 and any other unknowns for which there are no predecessors in the graph, i.e., all those unknowns x_i for which the off-diagonal elements of row i of L are zero. These unknowns will constitute the elements of the first level. The next step will determine in parallel all those unknowns that will have the nodes of the first level as their (only) predecessors in the graph.

More generally, we can define a root node as an imaginary node with links to the nodes having no predecessors, and the *depth* of a node as the maximum distance of that node from the root [130]. The introduction of the root node ensures that the depth of each node is defined from the same point. The depth of each node can be computed with one pass through the structure of the coefficient matrix L by

$$depth(i) = \begin{cases} 1 & \text{if } l_{ij} = 0 \text{ for all } j < i, \\ 1 + \max_{j<i}\{depth(j) \ , \ l_{ij} \neq 0\} & \text{otherwise.} \end{cases} \tag{18}$$

Before reordering



After reordering by levels

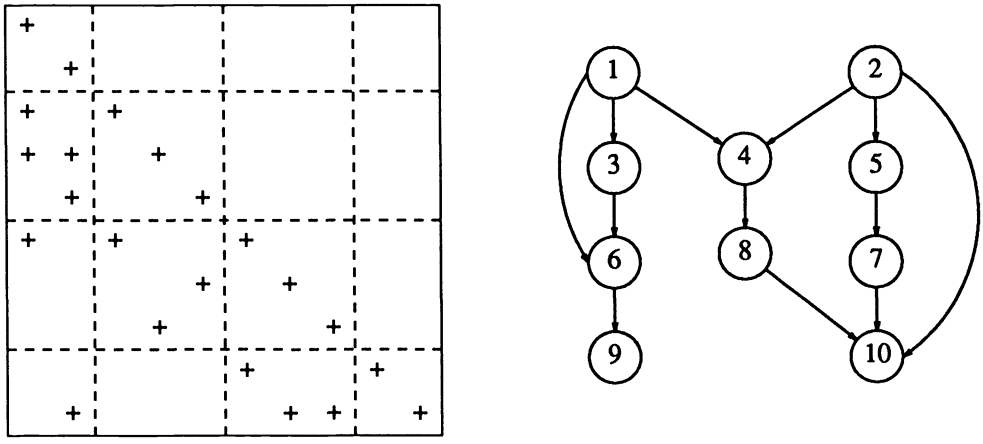


FIG. 2. *A sparse lower triangular matrix and its level structure.*

A *level* of the graph is, by definition, the set of nodes with the same depth. Thus the depth of a node is the same as the block number of its row in the block algorithm, and the nodes of a level define the set of rows in a block. Let us assume that we now define a data structure for the levels: a permutation $q(1:n)$ defines the new ordering and $level(i), i = 1, \cdots, nlev + 1$ points to the beginning of the ith level in that array. Then the algorithm for solving the triangular systems can be written as follows:

Algorithm: Forward elimination with level scheduling

```
do lev=1, nlev
j1 = level(lev)
j2 = level(lev+1)-1
     do k = j1, j2
       i = q(k)
       do j= ial(i), ial(i+1)-1
           x(i) = x(i) - al(j) * x(jal(j))
       enddo
      enddo
 enddo
```

The k loop can be executed in parallel. The idea of forward level scheduling is a natural one for finite difference matrices on rectangles and several authors suggested it independently, [124], [123], [50], [110], [9]. It is also interesting to note that for the computer scientist the idea in the general context of irregularly structured matrices is a textbook example of scheduling for parallel processing. In fact the level scheduling approach described here is a "greedy" algorithm and is unlikely to be optimal. There is no reason why one should solve an equation as soon as it is possible and it may be preferable to use a *backward scheduling* [5] which consists of defining the levels from bottom up in the graph. Thus the last level consists of the leaves of the graph, the previous level consists of their predecessors, etc.

Another possibility is to use dynamic scheduling as opposed to static scheduling. The main difference is that the level structure is not preset; rather the order of the computation is determined at run-time. The clear advantage over prescheduled triangular solutions is that it allows processors to always execute a task as soon as its predecessors have been completed thus reaching a better load balancing. On loosely coupled distributed memory machines this approach may be the most viable since it will dynamically adjust to irregularities in the execution and communication times that can cause a lock-step technique to become inefficient. On the other hand, for those shared memory machines in which hardware synchronization is available and inexpensive, such as the Alliant FX-8, dynamic scheduling would have some disadvantages since it requires managing queues and generating explicitly busy waits. Both approaches have been tested and compared in [21] where it was concluded that on the Encore Multimax dynamic scheduling is usually preferable except for problems with few synchronization points and large amount of parallelism. In [54] a combination of prescheduling and dynamic scheduling was found to be the best approach on a Sequent balance 21000. There seems to have been no comparison of these two approaches on distributed memory machines or on shared memory machines with microtasking, or hardware synchronization features.

To illustrate the performance of the solvers described in this section we show in Table 2 speed-ups of the various methods considered here over the sequential version Lsol on a number of matrices. The experiments have been performed on an Alliant

TABLE 2
Speed-up over the sequential method of four triangular system solvers, on an Alliant FX-8.

Matrix			Speedup over Lsol			
Name	N	NNZ	blksla	blkslb	levsl	jagsl
STEAM2	600	7180	1.15	2.68	5.15	2.43
ORSIRR-2	886	3428	0.97	2.55	4.96	2.79
JPWH-991	991	3529	1.33	2.43	5.06	3.05
SHERMAN1	1000	2375	0.87	1.94	4.46	4.07
SHERMAN4	1104	2445	0.85	1.75	4.87	4.37
ORSREG-1	2205	8169	1.10	2.50	5.51	5.36
SHERMAN5	3312	11571	0.88	1.93	4.70	2.72
SHERMAN3	5005	12519	0.97	1.96	5.03	6.68
BCSSTK06	420	4140	0.91	2.84	2.58	0.43
BCSSTK19	817	3835	0.89	2.88	1.41	0.45
BCSSTK09	1083	9760	1.13	2.79	3.96	0.98
BCSSTK27	1224	28675	1.33	3.51	1.73	0.22
BCSSTM12	1473	10566	1.13	2.74	3.92	0.95
BCSSTM13	2003	11973	0.99	1.92	2.71	0.44
BCSSTK23	3134	24156	1.08	2.82	4.74	1.38
BCSSTK21	3600	15100	1.20	2.70	5.83	5.87
20x20x1	400	1160	0.96	2.15	3.24	2.27
20x20x10	4000	15200	1.12	2.46	5.67	7.25
20x20x20	8000	30800	1.12	2.54	5.57	8.29
20x20x30	12000	46400	1.13	2.54	5.51	8.87
30x30x1	900	2640	0.97	2.10	3.84	2.88
30x30x10	9000	34500	1.13	2.50	5.55	8.27
30x30x20	18000	69900	1.14	2.51	5.33	9.29
30x30x30	27000	105300	1.14	2.54	5.29	9.77

FX-8, with 8 processors, using double precision arithmetic. The meanings of the labels are as follows:

- blksla : blocking method with sequential solve;
- blkslb : blocking method with dense inverse solve;
- levsl : level scheduling solve;
- jagsl : level scheduling using the jagged diagonal format for the matrix-vector products $E_i\hat{x}_{i-1}$ in (16).

The first two sets of matrices are from the sets RUA and RSA of the Harwell-Boeing collection of sparse matrices [35]. The last set consists of 5-point or 7-point matrices on rectangular domains, with the numbers referring to the grid sizes.

In the table n represents the size of the matrix while nnz is its total number of nonzero elements. The numbers shown in the last four columns are the speed-ups of the four techniques they refer to, as compared to the times obtained for Lsol, the sequential method whose program is shown at the beginning of this section. Recall that the number of arithmetic operations performed by blksla, levsl, and jagsl is identical with that of the sequential solve, while blkslb requires more operations. The block size used in blksla and blkslb is always 16. Note that the speed-up of blksla may be less than one and is otherwise negligible. blkslb on the other hand can reach a speed-up that exceeds three in some cases despite the additional work involved.

The speed-ups achieved by levsl are more consistent than the other techniques and they are generally lower for the denser matrices. Jagsl performed extremely well on the three-dimensional grid problems, sometimes reaching speed-ups that exceed the number of processors due to a better vectorization. The performance on some of the matrices could also be very poor. These matrices are those that have a large number of very short jagged diagonals [6].

In [22], [21] and [5], [6] a number of additional experiments are presented to study the performance of level scheduling within the context of preconditioned conjugate gradient methods.

5. Polynomial preconditioning. Polynomial preconditioning consists of choosing a polynomial s and replacing the original linear system by

$$s(A)Ax = s(A)b \tag{19}$$

which is then solved by a conjugate gradient type technique. There have been several recent publications on the use of such preconditioners motivated mostly by their potential on vector computers [8], [58], [105], [60], [115], [131]. However, the idea of using polynomial preconditioning is an old one and has been suggested by Stiefel [119] for eigenvalue calculations. Later, Rutishauser [100] adapted Stiefel's method for linear systems. The idea of polynomial preconditionings reappeared with the paper by Dubois, Greenbaum, and Rodrigue [33] which was spurred by the potential of these techniques for vector machines. Their motivation was that many matrices have a diagonal structure, in which case matrix by vector multiplications can be performed at near peak speed of the machine. Several papers discussing better polynomials or performance issues followed [58], [105], [60], [80], [37], [131]. However, there are often doubts surrounding the usefulness of the method on supercomputers. Before discussing the advantages and disadvantages of the method we begin with an overview.

To be efficient, polynomial preconditionings require the determination of an optimum polynomial s. The preconditioned matrix $s(A)A$ should be as close as possible to the identity matrix in some sense. One possible criterion is to make the spectrum of the preconditioned matrix as close as possible from that of the identity. For example, denoting by $\sigma(A)$ the spectrum of A, and by Π_k the space of polynomials of degree not exceeding k, we may wish to solve

$$\begin{aligned} &\text{Find } s \in \Pi_k \text{ that minimizes} \\ &\max_{\lambda \in \sigma(A)} |1 - \lambda s(\lambda)|. \end{aligned} \tag{20}$$

Unfortunately, this problem involves all the eigenvalues of A and is harder to solve than the original problem. What is usually done is to replace problem (20) by the problem obtained from replacing the set $\sigma(A)$ by some continuous set E that encloses it:

$$\begin{aligned} &\text{Find } s \in \Pi_k \text{ that minimizes} \\ &\max_{\lambda \in E} |1 - \lambda s(\lambda)|. \end{aligned} \tag{21}$$

Thus, polynomial preconditioning techniques start with the assumption that we have a rough idea of the spectrum of the matrix A. We will come back to this problem later.

Consider first the particular case where A is symmetric positive definite in which case E can be taken to be the interval $[\lambda_{\min}, \lambda_{\max}]$ containing the eigenvalues of A.

The best residual polynomial $1 - \lambda s(\lambda)$ in this case is a shifted and scaled Chebyshev polynomial of the first kind, and its three-term recurrence results in a simple three-term recurrence for the approximate solution [53]. An alternative considered by Johnson et al. in [58] is to use a least squares polynomial on the interval instead of the infinity norm polynomial. In other words, we need to solve

$$\begin{aligned} &\text{Find } s \in \Pi_k \text{ that minimizes} \\ &\|1 - \lambda s(\lambda)\|_w, \end{aligned} \tag{22}$$

where w is some weight function on the interval $(\lambda_{\min}, \lambda_{\max})$, and $\|.\|_w$ is the L_2-norm associated with the corresponding inner product. Because of the fact that the distribution of eigenvalues matters more than condition numbers for the preconditioned conjugate gradient method, it has been observed in [58] that least squares polynomials tend to perform better than those based on the uniform norm, in that they lead to a better overall clustering of the spectrum. Moreover, as was already observed by Rutishauser [100], in the symmetric case there is no need for accurate eigenvalue estimates: it suffices to use the simple bounds that are provided by Gershgorin's theorem. In [105] it was also observed that in some cases the least squares polynomial over the Gershgorin interval may perform as well as the infinity norm polynomial over $[\lambda_{\min}, \lambda_{\max}]$. Note that this is only a minor advantage of least squares polynomials since effective adaptive procedures exist to compute $\lambda_{\min}, \lambda_{\max}$; see [53] for symmetric problems and [72], [41] for nonsymmetric problems. We should add that the observations made in [58] and in [105], and the simplicity of a method that bypasses eigenvalue estimates, have made least squares polynomials more popular for polynomial preconditionings. For the simple case of the model 5-point Laplace matrix, the best polynomials up to the degree 10 are listed in [105] and are readily usable.

We now consider the more general nonsymmetric case. Given a set of approximate eigenvalues of A, we can construct a region E in the complex plane which ideally would contain the eigenvalues of the matrix A. There are several choices for E. The first idea is to use an ellipse E [74], [72] that encloses an approximate convex hull of the spectrum. Then the shifted and scaled Chebyshev polynomials are optimal when the foci of the ellipse are on the real axis and nearly optimal in other situations and the use of these polynomials leads again to an attractive three-term recurrence. A second alternative is to use a polygon H that contains $\sigma(A)$ [117], [104]. A notable advantage of using polygons is that they may better represent the shape of an arbitrary spectrum. The polynomial is not explicitly known but it may be computed by a Remez algorithm. As in the symmetric case an alternative is to use an L_2-norm instead of the infinity norm, i.e., to solve (22) where this time w is some weight function defined on the boundary of H. Smolarski and Saylor used a discrete norm on the polygon. Their original algorithm is unstable [117] but an improved alternative is proposed in [113]. In [104] we used an L_2-norm associated with Chebyshev weights on the edges of the polygon and expressed the best polynomial as a linear combination of Chebyshev polynomials associated with the ellipse of smallest area containing H. If the contour of H consists of μ edges each with center c_i and half-length d_i, then the weight on each edge is defined by

$$w_i(\lambda) = \frac{2}{\pi} \, |d_i - (\lambda - c_i)^2|^{-1/2}. \tag{23}$$

With these weights, or any other Jacobi weights on the edges, there is a finite procedure to compute the best polynomial *that does not require numerical integration*;

for details see [104]. The particular case where A is symmetric but indefinite was examined in detail in [101].

Still in the context of using polygons instead of ellipses, yet another attractive possibility proposed by Fischer and Reichel [43] is to avoid the problem of best approximation altogether and interpolate the function $1/z$ with a polynomial at the Fejer points of E, i.e., the points $e^{2ji\pi/k}, j = 0, 1, \cdots, k$ that are conformally mapped from the unit circle to H. This is known to be an asymptotically optimal process. There are numerous publications related to this approach and the use of Faber polynomials; see the references in [43]. However, we should point out that these papers are more concerned with the approximation theory problem than with the actual linear algebra problem. Often the tests performed are contrived into problems for which the spectrum is known to lie in nice and known regions. Real problems do not have nice spectra enclosed in rectangles that happen to be well separated from the origin.

We also mention that although we have only discussed approaches based on the formulations (21), (22), there are other less known possibilities based on minimizing $\|1/\lambda - s(\lambda)\|_\infty$; see Freund [46] and the references therein.

The next question that arises is: how can we get the polygon H? In the symmetric positive definite case, where H reduces to an interval, this is not too difficult to do because of the relationship between the Lanczos algorithm and the conjugate gradient method; see, e.g., [32], [53]. This seems to have been proposed first by Rutishauser [100], who suggested using the QD algorithm for computing the eigenvalues of the tridiagonal matrix obtained from the CG/Lanczos procedure. In the nonsymmetric case, an adaptive procedure was proposed by Manteuffel [74], [72]. An improved technique that exploits Arnoldi's method was later developed in [41]. For illustration, we describe next an implementation based on a combination with GMRES [109]. Eigenvalue estimates can be computed from the Hessenberg matrices generated from GMRES. Thus, the idea is to use a certain number, say m_1, of steps of GMRES to get eigenvalue estimates and to improve the current solution. The eigenvalue estimates are used to improve the current polygon H and to compute the next least squares polynomial. A number of steps (say m_2) of GMRES are then performed for the preconditioned system $s(A)Ax = s(A)b$. This procedure is outlined below.

Algorithm: Polynomial preconditioned GMRES

1. *Restart:*
   Compute current residual vector $r := b - Ax$.
2. *Adaptive* GMRES *run:*
   Run m_1 steps of GMRES for solving $Ad = r$. Update x by $x := x + d$. Get eigenvalue estimates from the eigenvalues of the Hessenberg matrix.
3. *Compute new polynomial.* Refine H from previous hull H and new eigenvalue estimates. Get new best polynomial s_k.
4. *Polynomial iteration:*
   - Compute the current residual vector $r = b - Ax$.
   - Run m_2 steps of GMRES applied to $s_k(A)Ad = s_k(A)r$. Update x by $x := x + d$.
   - Test for convergence. If solution converged then stop else goto 1.

We now discuss some of the advantages and disadvantages of polynomial preconditionings, starting with the advantages. The main attraction of polynomial preconditioning is that the only operations involving the matrix are products with vectors. As a result parallelism of order N can be achieved for each step of the preconditioned

conjugate gradient method. As a consequence of this, portability is facilitated because we only need to concentrate on optimizing matrix by vector multiplications and a few other basic kernels. A second advantage is that we need fewer dot products than with the nonpreconditioned conjugate gradient method to solve a linear system. The dot products can be bottlenecks for large number of processors but may not cause any difficulty otherwise. Thus, the main attraction of polynomial preconditioning is when the number or processors is very large. In this situation the usual incomplete LU factorization described in Section 3 will not be very helpful except for matrices with special structures. In many cases one may reorder the matrix to obtain a highly parallel LU solve, e.g., by using a red-black ordering as is described in Section 7. Polynomial preconditioning can be used when this is not possible or in combination with reordering techniques. Another important point is that polynomial preconditioning can be combined with a subsidiary relaxation-type preconditioning such as $SSOR$ [3], [37]. Finally, polynomial preconditionings can be very useful in solving some special linear systems such as complex linear systems arising from the Helmholtz equation [45] and those arising from the biharmonic equations [132].

The main disadvantage of polynomial preconditionings is their poor performance on sequential machines or parallel machines with small number of processors. In the numerical experiments reported in [5] on matrices with sizes ranging from $N = 434$ to $N = 2205$, polynomial preconditioning compares favorably only in a few cases with the parallel implementation of ILU preconditioning, on an Alliant FX-8. On the optimistic side, these numbers may look better if we think of the potential speed-ups that might be achieved on a machine with a much larger number of processors, since typically there is a parallelism of order N (the size of the matrix) for polynomial preconditioning but a parallelism of order $\sqrt{N}$ for ILU preconditioning. Unfortunately, theoretical comparisons of the two approaches are difficult. One such comparison was proposed by Axelsson [13] for the simple case of the diagonally scaled Neumann series approach [33], applied to a symmetric positive definite matrix. It was concluded that in this case polynomial preconditioning was rarely competitive with the nonpreconditioned conjugate gradient method. More precisely, polynomial preconditioning can outperform the standard conjugate gradient method only when the cost of the matrix by vector multiplication is less than half the cost of the other operations in a CG step. This is clearly a very stringent condition, which is likely to be satisfied only for machines with a very large number of processors, i.e., when the dot products start dominating the cost of a CG step. The model used in [13] is restricted to the simplest polynomial preconditioning, and it is not known whether a similar conclusion might be proved for the more sophisticated preconditioners.

A second disadvantage of polynomial preconditioning is that there are often alternatives that may do better. For example, for the case of a small number of processors, the parallel implementations of the standard ILU preconditioning discussed earlier will be difficult to outperform. For machines with large numbers of processors, reordering techniques have the potential of being superior to polynomial preconditioning, although extensive comparisons on such machines are still lacking.

It is often argued that one weakness of these methods is the requirement for eigenvalue estimates. In fact we found this to be a rather minor drawback because of the possibility of combining the process with an algorithm like GMRES or Arnoldi. We would also like to mention that there has been very little work on polynomial preconditioning or Krylov subspace methods for highly nonnormal matrices; see, however, the recent analysis in [120].

TABLE 3
Times for solving four linear systems on an Alliant FX/8.

Matrix	N	$ILU(0)$	Polyn.
JPWH_991	991	0.30	0.31
ORSIRR_1	1030	0.74	2.88
ORSREG_1	2205	1.74	5.77
SHERMAN5	3312	3.50	4.20

To give an idea of the performance of polynomial preconditionings as compared with ILU preconditioners, we show in Table 3 the time that it takes to solve a few linear systems with matrices from the Harwell-Boeing collection. The stopping criterion used was to stop as soon as the residual norm dropped by a factor of 10^{-7}. This was done on an Alliant FX-8. The polynomial preconditioning used here was combined with a diagonal or block diagonal scaling. The degree of the polynomials used was 10. It is worth pointing out that the time for the first linear system (JPWH_991) using the sequential version of ILU, was about 1.3 sec or roughly 4.3 times slower than with level scheduling. Similarly, for Sherman5 level scheduling was roughly 3.14 faster than with sequential forward and backward solves.

6. Block and other preconditioners. When vector machines first appeared, the preconditioned conjugate gradient method was just becoming popular as an iterative technique. As a result one of the first issues that this technique was facing concerned the vectorization of the preconditioner, the most popular of which was Meijerink and van der Vorst's incomplete Choleski factorization. The first idea in this context was to use cyclic reduction to solve the bidiagonal systems that arise during the lower triangular solves [63], [60]. Van der Vorst [125] suggested replacing the inverse of each of the bidiagonal matrices by a simple Neumann series. This introduced a parallelism of order $n = \sqrt{N}$ for a problem originating from an $n \times n$ grid. The performance of these alternatives were, as expected, better than the sequential algorithm. However, it appeared that the vectorizable variant was not as reliable as the original $ICCG$. Moreover, often the speed-ups obtained were not satisfactory because of the short vector lengths that these methods involve.

Block incomplete factorizations were popularized independently by Axelsson, Brinkkemper, and Ill'n [15] and by Concus, Golub, and Meurant [31]. See, however, a complete bibliography regarding these methods in [13], which indicates early work by Underwood dating back to 1976. Given a block tridiagonal matrix $A = tridiag(B_i^T, A_i, B_{i+1})$, the basic idea is derived from the standard block Gaussian elimination process, in which A is factored as

$$A = (D - L)D^{-1}(D - L)^T \tag{24}$$

where L is the negative of the strict lower triangular part of A, and D is a block diagonal matrix $D = diag(D_i)$, defined through the recurrence,

$$D_i = A_i - B_i D_{i-1}^{-1} B_i^T, \quad i = 2, 3, \cdots, n. \tag{25}$$

The first observation made here [15], [31] is that even though the blocks D_i are dense, a sparse approximation to them can easily be found. For example, one technique is to use a banded approximation Γ_i to the inverse D_i^{-1}. If we denote the resulting approximations to D_i by Δ_i, then the recurrence (25) becomes

$$\Delta_i = A_i - B_i \Gamma_{i-1} B_i^T, \quad i = 2, 3, \cdots, n. \tag{26}$$

Then the preconditioning matrix would be

$$M = (\Delta - L)\Delta^{-1}(\Delta - L)^T. \tag{27}$$

Solving a linear system with M is not fully satisfactory on vector machines since the forward and backward solutions involve sequential banded solutions.

When block preconditioners emerged as contenders to the scalar incomplete factorizations [31], several variants were quickly derived and tested as vectorizable options [82], [78], [12], [13]. The simplest such option proposed in [78] is to exploit the fact that both Δ_i and Γ_i are available, and therefore any multiplication by Δ_i^{-1} can be replaced by a multiplication by Γ_i. The "inverse-free" factorization proposed by Axelsson is to write M as

$$M = (I - L\Delta^{-1})\Delta(I - L\Delta^{-1})^T. \tag{28}$$

Then, if Γ is block diagonal matrix approximating the inverse of Δ, the idea is to approximate the inverse of the factor $(I - L\Delta^{-1})$ by the so-called Euler expansion

$$(I - L\Gamma)^{-1} = \prod_{k=0}^{p} \left[I + (L\Gamma)^{2^k}\right] \tag{29}$$

where the integer p is carefully chosen. A detailed analysis of block preconditioners including the one just outlined was proposed by Axelsson and Polman [18].

There have been a number of extensions and related techniques proposed. For example, Meurant [80] suggested an alternative consisting of performing the LU factorization not only from top to bottom, but also from bottom to top, at the same time. The resulting decomposition is called the twisted incomplete factorization and is related to the WZ factorization [42]. The motivation here was to take advantage of multitasking on Cray machines. The gain in performance that was reported on a Cray X-MP was limited. Using a slightly different viewpoint, Rodrigue and Wolitzer [99] have argued for a form of incomplete cyclic reduction to derive incomplete factorizations of block tridiagonal matrices.

The more recent effort in preconditioning methods has been mainly in three directions. The first is in deriving preconditioners from domain decomposition techniques [65], [79], [27], [17]. The second direction is in hierarchical basis methods [4], [19], [51], [134], whereby the finite element basis is chosen in a such a manner as to yield well-conditioned matrices. Finally, there is now increasing interest in methods that will take advantage of the special nature of three-dimensional problems [16], [122]. Some early comparisons of domain decomposition preconditioners and standard block preconditioners reported in [79] seem to indicate that for a large number of processors the former might be better.

We should also mention work related to domain decomposition methods which consists of multisplitting iterative techniques [88], and element-by-element (EBE) preconditioning techniques [84]. A collection of recent papers on domain decomposition methods may be found in the proceedings [48] and [25].

7. Reordering for parallelism. A convenient and simple way to achieve a high level of parallelization is to reorder the equations and the unknowns. For example, equations that originate from 5-point matrices can be reordered according to the red-black ordering to yield systems of the form

$$\begin{pmatrix} D_1 & E \\ E^T & D_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{30}$$

where D_1 and D_2 are diagonal matrices. The triangular system to be solved at each step of incomplete *ICCG* are highly parallelizable due to the structure of the above matrix. However, one might be suspicious that the rate of convergence of the algorithm is not as good as that for the original ordering. Experience with model problems suggest that the deterioration of the convergence rate rarely exceeds a factor of two [114], [71]. However, for more realistic and difficult problems convergence behavior for reordered matrices may be unpredictable [116]. As was mentioned in Section 4, in order to avoid bottlenecks in the forward and backward sweeps, we want to have a matrix with a block structure in which each diagonal block is diagonal. We refer to this structure as diagonal diagonal block (DDB) structure. The L matrix will then have the form of Fig. 1, where each L_i is a diagonal matrix. There are many ways of achieving such structures; see, e.g., [71]. For matrices whose adjacency graphs are planar, a simple strategy is to start by setting up the level structure of the graph. This consists of using a Cuthill-McKee ordering starting from some initial point; see, e.g., [34]. The levels are then colored alternately in red and black. The next step is to group a certain number of levels of the same color in the same set. The number of levels chosen may be varied and will depend on the desired size of the diagonal blocks. For example, maximum parallelism may be achieved by putting all the red levels together in one set and all the blacklevels in another set. For matrices with property A, this will lead to the usual red-black ordering. Since each level set may consist of connected vertices, coloring may again be needed within each set. This technique was suggested in [71]. Interestingly, experiments for model problems reveal that the number of iterations needed for convergence does not vary too much as the block size increases, deteriorating by a factor of two in the worst case.

More generally, multicoloring is a general term that refers to reordering the unknowns to obtain a matrix that has the DDB structure mentioned above. Several authors have considered multicoloring techniques and the various effects on convergence of either the preconditioned conjugate gradient or the underlying relaxation technique [1], [86], [2], [39], [97], [96].

For matrices having property A, once the matrix is reordered in the form (30), an alternative to using *PCG* on the corresponding system directly is to solve the reduced system which involves only the black unknowns, namely,

$$(D_1 - ED_2^{-1}E^T)x_2 = b_1 - ED_2^{-1}b_2. \tag{31}$$

The above system is again a sparse linear system involving only half of the unknowns. Moreover, setting up the preconditioner is an inexpensive and fully parallel process even for unstructured problems [20]. The resulting system can again be preconditioned by incomplete Choleski factorization with level scheduling as described before. However, as revealed by experiments on various model problems, an excellent alternative is to use the diagonal of the reduced matrix as a preconditioner. This was observed for both well-structured [75] and unstructured problems [20]. In [20] it was found that for the reduced system, *ICCG* with level scheduling outperformed diagonal preconditioning, only for very large problems because the overhead in setting up the level structure and the preconditioner is difficult to offset given that diagonal scaling performs so well. A detailed analysis by Elman and Golub [40] on model problems indicates that it is often more effective to solve the reduced system by iterative methods than the original system, a fact which was observed empirically [38], [36]. The reduced system approach appeared to be the best option on vector or multivector processors in experiments reported in many papers. It also has excellent potential for

machines with a large number of processors of SIMD type. Its only limitation is that it does not generalize to matrices that do not have property A.

8. Conclusion. We have presented an overview of numerical techniques to solve realistic large linear systems by Krylov subspace methods on supercomputers. For machines with a small number of processors, the standard preconditionings can be efficiently implemented and they often constitute the most efficient approach. The advantage of this over using less conventional techniques is that these preconditioners are reliable and their behavior is well understood from experience on standard scalar machines. For machines with a very large number of processors, these techniques will not be sufficient to achieve satisfactory speed-ups. We have discussed two possible alternatives for this case. The first consists of using polynomial preconditioning and the second consists of reordering the equations by multicoloring techniques. Although these alternatives offer a good potential, they are not sufficiently well understood for general problems. For example, reordering affects the convergence rate of the conjugate gradient method in a way that is difficult to predict for general sparse problems, while polynomial preconditioning may not be competitive against rival techniques such as the unpreconditioning conjugate gradient method or the multicolored incomplete Choleski/LU.

These observations suggest that it is crucial that researchers experiment with existing massively parallel machines in order to better understand the effects that are difficult to predict from theory, such as impact of communication costs, rates of convergence of new algorithms, ways of mapping the data, etc.

Research in the direction of domain decomposition and hierarchical basis preconditioners, two natural ideas for parallel processing, is very active. It is interesting to observe that the successful ideas in parallel numerical methods have often been derived from existing techniques that are either adapted or slightly modified. This is the case of domain decomposition methods that have been extensively used in structural analysis. The search for parallelism has forced researchers to take a second look at many old techniques, sometimes resulting in remarkable success. Ortega and Voigt concluded in [91] that there has been very few "truly parallel algorithms" invented, as opposed to modifications of existing algorithms. This has been even more pronounced for iterative methods, perhaps because there are many known algorithms that do offer a large degree of parallelism or that can be simply modified to improve their parallelism. However, a common difficulty is that those iterative methods that are intrinsically highly parallel, such as the Jacobi iteration, often converge much more slowly than those with a more sequential nature, such as the Gauss-Seidel iteration.

We have not discussed issues related to software development and hardware. Although the trend towards parallelism is clear, the question as to which type(s) of architecture(s) will prevail in the long term remains difficult to answer. A consequence is that in the near term at least, developers of numerical software will face challenging problems. They must not only select and develop different algorithms for different machines but also acquire a good understanding of the machines and the software that runs them in order to be able to produce reasonable performance. Moreover, porting a program from one machine to another usually requires substantial reprogramming effort which may involve changing languages, completely remapping the data, and extensive testing to tune the code. Given the variety of machines that are available these efforts are increasingly viewed as intolerably time consuming and

non-cost-effective. Thus, the need for better programming environments and languages for parallel machines is becoming critical and as iterative methods are gaining importance, they should be considered from the perspective of these other important issues.

REFERENCES

[1] L. Adams. M-step preconditioned conjugate gradient methods. *SIAM J. Sci. Statist. Comput.*, 6:452–463, 1985.

[2] L. Adams and H. Jordan. Is SOR color-blind? *SIAM J. Sci. Statist. Comput.*, 6:490–506, 1985.

[3] L. M. Adams. *Iterative algorithms for large sparse linear systems on parallel computers.* PhD thesis, Applied Mathematics, University of Virginia, Charlottsville, VA, 22904, 1982. Also NASA Contractor Report 166027.

[4] L. M. Adams and E. G. Ong. A comparison of preconditioners for GMRES on parallel computers. In A. Noor, editor, *Parallel Computations and their Impact on Mechanics*, pages 171–186, Amer. Soc. Mech. Engr., Dec. 1987.

[5] E. C. Anderson. *Parallel implementation of preconditioned conjugate gradient methods for solving sparse systems of linear equations.* Technical Report 805, CSRD, University of Illinois, Urbana, IL, 1988. MS Thesis.

[6] E. C. Anderson and Y. Saad. *Solving sparse triangular systems on parallel computers.* Technical Report 794, University of Illinois, CSRD, Urbana, IL, 1988.

[7] W. E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.

[8] S. F. Ashby. *Polynomial preconditioning for conjugate gradient methods.* PhD thesis, Computer Science Dept., University of Illinois, Urbana, IL, 1987. Available as Technical Report 1355.

[9] C. C. Ashcraft and R. G. Grimes. On vectorizing incomplete factorization and SSOR preconditioners. *SIAM J. Sci. Statist. Comput.*, 9:122–151, 1988.

[10] O. Axelsson. Conjugate gradient type-methods for unsymmetric and inconsistent systems of linear equations. *Linear Algebra Appl.*, 29:1–16, 1980.

[11] ———. A generalized conjugate gradient, least squares method. *Numer. Math.*, 51:209–227, 1987.

[12] ———. Incomplete block matrix factorization preconditionings. The ultimate answer? *J. Comp. Appl. Math.*, 17:3–18, 1985.

[13] ———. A survey of preconditionied iterative methods for linear systems of algebraic equations. *BIT*, 25:166–187, 1985.

[14] ———. *A survey of vectorizable preconditioning iterative methods for large scale finite element matrix problems.* Technical Report CNA-190, Center for Numerical Analysis, University of Texas, Austin, TX, 1984.

[15] O. Axelsson, S. Brinkkemper, and V. P. Ill'n. On some versions of incomplete block-matrix factorization iterative methods. *Linear Algebra Appl.*, 58:3–15, 1984.

[16] O. Axelsson and V. Eijkhout. *Vectorizable preconditioners for elliptic difference equations in three space dimensions.* Technical Report, Dept. of Math., University of Nijmegen, Nijmegen, The Netherlands, 1988.

[17] O. Axelsson and B. Polman. *Block preconditioning and domain decomposition methods.* Technical Report 8807, Dept. of Math., Catholic University, Toernooiveld, Nijmegen, The Netherlands, 1988.

[18] ———. On approximate factorization methods for block matrices suitable for vector and parallel processors. *Linear Algebra Appl.*, 77:3–26, 1986.

[19] O. Axelsson and P. S. Vassilevski. *Algebraic multilevel preconditioning methods, I.* Technical Report 8811, Department of Mathematics, Catholic University, Toernooiveld, Nijmegen, The Netherlands, 1988.

[20] C. L. Baucom. *Reduced systems and the preconditioned conjugate gradient method on a multiprocessor.* Technical Report, CSRD, University of Illinois, Urbana, IL, 1988.

[21] D. Baxter, J. Saltz, M. H. Schultz, and S. C. Eisenstat. *Preconditioned Krylov solvers and methods for runtime loop parallelization.* Technical Report 655, Computer Science, Yale University, New Haven, CT, 1988.

[22] D. Baxter, J. Saltz, M. H. Schultz, S. C. Eisenstat, and K. Crowley. *An experimental study of methods for parallel preconditioned Krylov methods.* Technical Report 629, Computer Science, Yale University, New Haven, CT, 1988.

[23] P. N. Brown. *A theoretical comparison of the Arnoldi and GMRES algorithms.* Technical Report UCRL-98630, Lawrence Livermore National Lab., Livermore, CA, 1988.

[24] O. A. McBryan. *The Connection Machine: PDE solution on 65,536 processors.* Technical Report LA-UR-86-4219, Los Alamos National Lab., Los Alamos, NM, 1986.

[25] T. F. Chan, R. Glowinski, J. Periaux, and O. Widlund. *Second international symposium on domain decomposition methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[26] T. F. Chan, C. C. Kuo, and C. Tong. *Parallel elliptic preconditioners: Fourier analysis and performance evaluation.* Technical Report 88-22, Computational and Applied Mathematics, UCLA, Los Angeles, CA, 1988.

[27] T. F. Chan and D. Resasco. A framework for the analysis of domain decomposition preconditioners. In *Proc. First Internat. Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 217–230, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[28] R. Chandra. *Conjugate Gradient Methods for Partial Differential Equations.* PhD thesis, Computer Science Dept., Yale University, New Haven, CT, 1978.

[29] A. Chronopoulos. *A class of parallel iterative methods implemented on multiprocessors.* PhD thesis, University of Illinois, Computer Science Dept., Urbana, IL, 1987.

[30] P. Concus and G. H. Golub. A generalized conjugate gradient method for nonsymmetric systems of linear equations. In R. Glowinski and J. L. Lions, editors, *Computing Methods in Applied Sciences and Engineering*, pages 56–65, Springer-Verlag, New York, 1976.

[31] P. Concus, G. H. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM J. Sci. Statist. Comput.*, 6:309–332, 1985.

[32] 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 James R. Bunch and Donald J. Rose, editors, *Sparse Matrix Computations*, pages 309–332, Academic Press, New York, 1976.

[33] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue. Approximating the inverse of a matrix for use on iterative algorithms on vectors processors. *Computing*, 22:257–268, 1979.

[34] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

[35] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Soft.*, 15:1–14, 1989.

[36] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Block preconditioned conjugate-gradient-like methods for numerical reservoir simulation. In *Proceedings of the SPE 1985 Reservoir Simulation Symposium*, pages 397–405, Society of Petroleum Engineers of AIME, Richardson, TX, 1985. Paper number 13534.

[37] S. C. Eisenstat, J. M. Ortega, and C. T. Vaughan. *Efficient polynomial preconditioning for the conjugate gradient method.* Technical Report RM-88-14, Dept. of Appl. Math., University of Virginia, Charlottesville, VA, 1988.

[38] H. C. Elman. *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations.* PhD thesis, Computer Science Dept., Yale University, New Haven, CT, 1982.

[39] H. C. Elman and E. Agron. *Ordering techniques for the preconditioning conjugate gradient method on parallel computers.* Technical Report UMIACS-TR-88-53, UMIACS, University of Maryland, College Park, MD, 1988.

[40] H. C. Elman and G. H. Golub. *Iterative Methods for cyclically reduced non-self-adjoint linear systems.* Technical Report CS-TR-2145, Dept. of Computer Science, University of Maryland, College Park, MD, 1988.

[41] H. C. Elman, Y. Saad, and P. Saylor. A hybrid Chebyshev Krylov subspace algorithm for solving nonsymmetric systems of linear equations. *SIAM J. Sci. Statist. Comput.*, 7:840–855, 1986.

[42] D. J. Evans. *Parallel Processing Systems, an Advanced Course.* Cambridge University Press, New York, 1982.

[43] B. Fischer and L. Reichel. A stable Richardson iteration method for complex linear systems. *Numer. Math.*, 54:225–241, 1988.

[44] R. Fletcher. Conjugate gradient methods for indefinite systems. In G.A. Watson, editor, *Proceedings of the Dundee Biennial Conference on Numerical Analysis 1974*, pages 73–89, University of Dundee, Scotland, Springer-Verlag, New York, 1975.

[45] R. Freund. *On conjugate gradient type methods and polynomial preconditioners for a class of complex non-Hermitian matrices.* Technical Report 88-44, RIACS, NASA Ames Research Center, Moffett Field, CA, 1989.

[46] R. Freund. On polynomial approximation to $f_a(z) = (z-a)^{-1}$ with complex a and some applications to certain non-Hermitian matrices. *Approximation Theory and its Applications*, 1989.

[47] 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.*, 8:1079–1084, 1987.

[48] R. Glowinski, G. H. Golub, G. A. Meurant, and J. Periaux. *First International Symposium on Domain Decomposition Methods for Partial Differential Equations.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[49] A. Greenbaum and G. H. Rodrigue. *The incomplete Choleski conjugate gradient for the STAR (5-point) operator.* Technical Report UCID 17574, Lawrence Livermore National Lab., Livermore, CA, 1977.

[50] A. Greenbaum. *Solving Triangular Linear Systems Using FORTRAN with Parallel Extensions on the NYU Ultracomputer Prototype.* Technical Report 99, Courant Institute, New York University, New York, NY, 1986.

[51] A. Greenbaum, C. Li, and H.Z. Chao. *Comparison of linear system solvers applied to diffusion type finite element equations.* Technical Report , Courant Institute, New York University, New York, NY, 1987.

[52] R. G. Grimes, J. G. Lewis, and H. D. Simon. *The implementation of a block Lanczos algorithm with reorthogonalization methods.* Technical Report ETA-TR-91, Boeing Computer Services, Seattle, WA, 1988.

[53] A. L. Hageman and D. M. Young. *Applied Iterative Methods.* Academic Press, New York, 1981.

[54] S. Hammond and R. Schreiber. *Efficient ICCG on a shared memory multiprocessor.* Technical Report 89.24, RIACS, NASA Ames Research Center, Moffett Field, CA, 1989.

[55] S. W. Hammond and K. H. Law. Architecture and operation of a systolic engine for finite element computations. *Computers and Structures*, 30:365–374, 1988.

[56] W. Jalby, U. Meier, and A. Sameh. *The behavior of conjugate gradient algorithms on a multivector processor with a hierarchical memory.* Technical Report 607, CSRD, University of Illinois, Urbana, IL, 1985.

[57] K. C. Jea and D. M. Young. Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods. *Linear Algebra Appl.*, 34:159–194, 1980.

[58] O. G. Johnson, C. A. Micchelli, and G. Paul. Polynomial preconditionings for conjugate gradient calculations. *SIAM J. Numer. Anal.*, 20:362–376, 1983.

[59] S. L. Johnsson. Highly concurrent algorithms for solving linear systems of equations. In G. N. Birkhoff and A. Schoenstadt, editors, *Elliptic Problem Solvers II, Proceedings of the Elliptic Problem Solvers Conference*, Monterey, CA, Jan 10-12 1983, pages 105–126, Academic Press, 1983.

[60] T. L. Jordan. Conjugate gradient preconditioners for vector and parallel processors. In G. N. Birkhoff and A. Schoenstadt, editors, *Elliptic Problem Solvers II, Proceedings of the Elliptic Problem Solvers Conference*, Monterey, CA, Jan 10-12 1983, pages 127–139, Academic Press, 1983.

[61] ———. A guide to parallel computation and some CRAY-1 experiences. In Garry Rodrigue, editor, *Parallel Computations*, pages 1–50, Academic Press, 1982.

[62] T. I. Karush, N. K. Madsen, and G. H. Rodrigue. *Matrix Multiplication by Diagonals on Vector/Parallel Processors.* Technical Report UCUD, Lawrence Livermore National Lab., Livermore, CA, 1975.

[63] D. Kershaw. Solution of single tridiagonal systems and vectorization of the ICCG algorithm on the CRAY-1. In Garry Rodrigue, editor, *Parallel Computations*, pages 85–89, Academic Press, 1982.

[64] D. E. Keyes and W. D. Gropp. A comparison of domain decomposition techniques for elliptic partial differential equations. *SIAM J. Sci. Statist. Comput.*, 8:s166–s202, 1987.

[65] ———. Domain decomposition techniques for nonsymmetric systems of elliptic boundary value problems: Examples from cfd. In T. F. Chan, R. Glowinski, J. Periaux, and O. Widlund, editors, *Proceedings of the second international symposium on domain decomposition methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989.

[66] A. Koniges. *Parallel processing of a preconditioned biconjugate gradient algorithm in CRAY supercomputers.* Technical Report, Lawrence Livermore National Lab, Livermore, CA, 1987.

[67] D. J. Kuck, E. S. Davidson, D. L. Lawrie, and A. H. Sameh. Parallel supercomputing today and the CEDAR approach. *Science*, 231:967–974, 1986.

[68] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards*, 49:33–53, 1952.

[69] J. G. Lewis and H. D. Simon. The impact of hardware scatter-gather on sparse Gaussian elimination. *SIAM J. Sci. Statist. Comput.*, 9:304–311, 1988.

[70] ———. *Numerical experience with the spectral transformation Lanczos.* Technical Report MM-TR-16, Boeing Computer Services, Seattle, WA, 1984.

[71] A. Lichnewski. Some vector and parallel implementations for preconditioned gradient algorithms. In J. Kowalik, editor, *Proceedings of the NATO Workshop on High Speed Computations*, pages 343–359, 1984.

[72] T. A. Manteuffel. Adaptive procedure for estimation of parameter for the nonsymmetric Tchebychev iteration. *Numer. Math.*, 28:187–208, 1978.

[73] ———. An incomplete factorization technique for positive definite linear systems. *Math. Comp.*, 34:473–497, 1980.

[74] ———. The Tchebychev iteration for nonsymmetric linear systems. *Numer. Math.*, 28:307–327, 1977.

[75] U. Meier and A. Sameh. The behavior of conjugate gradient algorithms on a multivector processor with a hierarchical memory. *J. Comput. Appl. Math.*, 24:, 1988.

[76] 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. *Math. Comp.*, 31(137):148–162, 1977.

[77] R. Melhem. Solution of linear systems with striped sparse matrices. *Parallel Comput.*, 6:165–184, 1988.

[78] G. Meurant. The block preconditioned conjugate gradient method on vector computers. *BIT*, 24:623–633, 1984.

[79] ———. Domain decomposition versus block preconditioning. In R. Glowinski, G. H. Golub, G. Meurant, and J. Periaux, editors, *Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1988.

[80] ———. Multitasking the conjugate gradient method on the CRAY X-MP/48. *Parallel Comput.*, 5:267–280, 1987.

[81] ———. *Numerical experiments for the preconditioned conjugate gradient method on the CRAY X-MP/2.* Technical Report LBL-18023, Lawrence Berkeley Lab, Berkeley, CA, 1984.

[82] ———. Vector preconditioning for the conjugate gradient on the CRAY-I and cdc CYBER 205. In J.L. Lions and R. Glowinski, editors, *Proceedings of the 6th International Conference on Computing Methods in Engineering and Applied Sciences, Versailles, France*, Dec. 12-16 1984, INRIA, North-Holland, Amsterdam, 1985.

[83] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method. *ACM. Trans. Math Software*, 6:206–219, 1980.

[84] B. Nour-Omid and B. N. Parlett. Element preconditioning using splitting techniques. *SIAM J. Sci. Statist. Comput.*, 6:761–770, 1985.

[85] D. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra Appl.*, 29:243–322, 1980.

[86] ———. Ordering schemes for parallel processing of certain mesh problems. *SIAM J. Sci. Statist. Comput.*, 5:620–632, 1984.

[87] ———. Parallel implementations of the block conjugate gradient algorithm. *Parallel Comput.*, 5:127–140, 1987.

[88] D. O'Leary and R. White. Multi-splittings of matrices and parallel solution of linear systems. *SIAM J. Algebraic Discrete Methods*, 6:630–640, 1986.

[89] T. C. Oppe and D. R. Kincaid. The performance of ITPACK on vector computers for solving large sparse linear systems arising in sample oil reservoir simulation problems. *Commun. in Appl. Numer. Methods*, 2:1–7, 1986.

[90] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York, 1988.

[91] J. M. Ortega and R.G. Voigt. Solution of partial differential equations on vector and parallel computers. *SIAM Rev.*, 27:149–240, 1985.

[92] C. C. Paige and M.A. Saunders. An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, 8:43–71, 1982.

[93] V. L. Peterson. Impact of computers on aerodynamics research and development. *Proceedings of the IEEE*, 72:68–79, 1984.

[94] B. Philippe and Y. Saad. Solving large sparse eigenvalue problems on supercomputers. In *Proceedings of International Workshop on Parallel Algorithms and Architectures*, Bonas, France, Oct. 3-6 1988, North-Holland, Amsterdam, 1989.

[95] S. J. Polak, C. Den Heijer, W. H. A. Schilders, and P. Markowich. Semiconductor device modelling from the numerical point of view. *Internat. J. Numer. Methods Engrg.*, 24:763–838, 1987.

[96] E. L. Poole. *Multi-color incomplete Choleski conjugate gradient methods for vector computers.* Technical Report 178117, NASA Langley Research, Hampton, VA, 1986.

[97] E. L. Poole and J. M. Ortega. Multicolor ICCG methods for vector computers. *SIAM J. Numer. Anal.*, 24:1394–1418, 1987.

[98] G. Radicati and Y. Robert. *Vector and parallel CG-like algorithms for sparse nonsymmetric linear systems.* Technical Report RR 681, IMAG, University of Grenoble, France, Grenoble, France, 1987.

[99] G. Rodrigue and D. Wolitzer. Preconditioning by incomplete block cyclic reduction. *Math. Comp.*, 42:549–565, 1984.

[100] H. Rutishauser. Theory of gradient methods. In *Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems*, pages 24–49, Institute of Applied Mathematics, Zurich, Birkhauser Verlag, Basel-Stuttgart, 1959.

[101] Y. Saad. Iterative solution of indefinite symmetric systems by methods using orthogonal polynomials over two disjoint intervals. *SIAM J. Numer. Anal.*, 20:784–811, 1983.

[102] ———. Krylov subspace methods for solving large unsymmetric linear systems. *Math. Comput.*, 37:105–126, 1981.

[103] ———. The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems. *SIAM J. Numer. Anal.*, 19:470–484, 1982.

[104] ———. Least squares polynomials in the complex plane and their use for solving sparse nonsymmetric linear systems. *SIAM J. Numer. Anal.*, 24:155–169, 1987.

[105] ———. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Statist. Comput.*, 6:865–881, 1985.

[106] Y. Saad, A. Sameh, and P. Saylor. Solving elliptic difference equations on a linear array of processors. *SIAM J. Sci. Statist. Comput.*, 6:1049–1063, 1985.

[107] Y. Saad and A. H. Sameh. A parallel block Stiefel method for solving positive definite systems. In M. H. Schultz, editor, *Proc. Elliptic Problem Solver Conf.*, pages 405–12, Academic Press, 1980.

[108] Y. Saad and M. H. Schultz. Conjugate gradient-like algorithms for solving nonsymmetric linear systems. *Math. Comput.*, 44(170):417–424, 1985.

[109] ———. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[110] ———. *Parallel Implementations of Preconditioned Conjugate Gradient Methods.* Research Report 425, Dept. Computer Science, Yale University, New Haven, CT, 1985.

[111] J. H. Saltz. *Automated Problem Scheduling and Reduction of Synchronization Delay Effects.* Technical Report 87-22, ICASE, Hampton, VA, 1987.

[112] M. A. Saunders, H. D. Simon, and E. L. Yip. Two conjugate-gradient type methods for unsymmetric linear equations. *SIAM J. Numer. Anal.*, 25(4):927–940, 1988.

[113] P. E. Saylor and D. C. Smolarski. Computing the roots of complex orthogonal kernel polynomials. *SIAM J. Sci. Statist. Comput.*, 9:1–13, 1988.

[114] R. Schreiber and W. P. Tang. Vectorizing the conjugate gradient method. In *Proc. Symposium CYBER 205 Applications*, Denver, CO, 1982.

[115] M. K. Seager. *Parallelizing conjugate gradient for the CRAY X-MP.* Technical Report, Lawrence Livermore National Lab, Livermore, CA, 1984.

[116] H. D. Simon. Incomplete LU preconditioners for conjugate gradient type iterative methods. In *Proceedings of the SPE 1985 reservoir simulation symposium*, pages 302–306, Society of Petroleum Engineers of AIME, Dallas, TX, 1988. Paper number 13533.

[117] D. C. Smolarski and P. E. Saylor. An optimum iterative method for solving any linear system with a square matrix. *BIT*, 28:163–178, 1988.

[118] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 10(1):36–52, 1989.

[119] E. L. Stiefel. Kernel polynomials in linear algebra and their applications. *U.S. Nat. Bur. Standards Applied Math. Series*, 49:1–24, 1958.

[120] L. N. Trefethen. *Approximation Theory and Numerical Linear Algebra.* Technical Report Numerical Analysis Report 88-7, Massachussetts Institute of Technology, Cambridge, MA, 1988.

[121] H. A. van der Vorst. *High performance preconditioning.* Technical Report 88-54, Delft University of Technology, Faculty of Technical Mathematics and Informatics, 1988.

[122] ———. ICCG and related methods for 3-D problems on vector computers. In D. Truhlar, editor, *Workshop on Practical Iterative Methods for Large Scale Computations*, Minneapolis, MN, Oct. 23-25 1988, Computer Physics Communication, Vol. 53, 1989.

[123] ———. Large tridiagonal and block tridiagonal linear systems on parallel and parallel computers. *Parallel Comput.*, 5:303–311, 1987.

[124] ———. The performance of FORTRAN implementations for preconditioned conjugate gradient methods on vector computers. *Parallel Comput.*, 3:49–58, 1986.

[125] ———. A vectorizable version of some ICCG methods. *SIAM J. Sci. Statist. Comput.*, 3:350–356, 1982.

[126] R. S. Varga. *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1962.

[127] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Statist. Comput.*, 9:152–163, 1988.

[128] J. W. Watts III. A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation. *Society of Petroleum Engineer J.*, 21:345–353, 1981.

[129] O. Widlund. A Lanczos method for a class of non-symmetric systems of linear equations. *SIAM J. Numer. Anal.*, 15:801–812, 1978.

[130] O. Wing and J. W. Huang. A computation model of parallel solution of linear equations. *IEEE Transactions on Computers*, C-29:632–638, 1980.

[131] Y. S. Wong. Solving large elliptic difference equations on CYBER 205. *Parallel Comput.*, 6:195–207, 1988.

[132] Y. S. Wong and H. Jiang. *Approximate polynomial preconditioning applied to biharmonic equations on vector computers.* Technical Report ICOMP-87-5, NASA Lewis Research Center, Inst. for Computational Mechanics in Propulsion, Cleveland, OH, 1987.

[133] D. M. Young, T.C. Oppe, D. R. Kincaid, and L. J. Hayes. *On the use of vector computers for solving large sparse linear systems.* Technical Report CNA-199, Center for Numerical Analysis, University of Texas at Austin, Austin, TX, 1985.

[134] H. Yserentant. On the multi-level splitting of finite element spaces. *Numer. Math.*, 49:379–412, 1986.